Kombinatorische Suche in Bäumen

Dissertation
zur
Erlangung des Doktorgrades
des Fachbereichs Mathematik
der Freien Universität Berlin

Vorgelegt von

Frank Recker

Mai 1997

Betreuer: Prof. Dr. Martin Aigner

Gutachter: Prof. Dr. Martin Aigner

Prof. Dr. Eberhard Triesch

Tag der Disputation: 1. Juli 1997

Das Graduiertenkolleg „Algorithmische Diskrete Mathematik“ wird gefördert durch die Deutsche Forschungsgemeinschaft (We 1265/2–1).

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die kombinatorische Suche ist eine junge mathematische Disziplin. Sie hat ihre Ursprünge in den 40er Jahren. Damals war das Problem, viele Personen auf eine Infektion mit Syphilis zu testen. Dazu wurde der Wassermann-Bluttest durchgeführt. Es gab dabei zwei Fakten, die diese neue mathematische Theorie motivierten. Erstens waren im allgemeinen von den getesteten Personen nur sehr wenige infiziert. Zweitens war es möglich, das Blut mehrerer Patienten zu mischen, ohne daß der Test bei dem verdünnten Blut versagte. War auch nur eine der Proben infiziert, so zeigte der Test dies an. Für eine weiterführende Beschreibung dieser geschichtlichen Entwicklung verweisen wir auf [DUH–93].

Damit stellte sich die Frage: „Wieviele Blutproben sollten vor dem ersten Test gemischt werden?“ und „Was ist zu tun, wenn der Test positiv[1] ausfiel?“

Wie bei den meisten Optimierungsproblemen ist die Frage leicht zu stellen. Die Lösung ist jedoch sehr schwierig und in der vollständigen Allgemeinheit bis heute ungelöst.

Für die mathematische Modellierung ist die obige Formulierung noch zu unklar. Bei der Analyse von Suchproblemen können wir uns zwei Spieler vorstellen. Das Ziel des Spiels ist es, in einer Menge S ein Element zu finden. Der erste Spieler $\mathcal{A}$ darf dazu bestimmte Fragen stellen. Wir werden diese Fragestrategie später Suchalgorithmus oder einfach Algorithmus nennen. Der zweite Spieler $\mathcal{O}$, das Orakel, beantwortet die Fragen. Das Spiel wird so lange gespielt, bis $\mathcal{A}$ das gesuchte Element eindeutig identifiziert hat.

[1] Also mindestens eine infizierte Blutprobe vorhanden war.

Verschiedene Spiele, die wir Suchprozesse nennen, unterscheiden sich dabei durch ihre Grundmenge S und durch die erlaubten Fragen. Bei dem Infektionstest besteht die Menge S aus allen Teilmengen der Blutproben. Erlaubte Fragen sind ebenfalls alle Teilmengen dieser Proben. Die Antwort besagt, ob die getestete Teilmenge mit der zu findenden einen nichtleeren Schnitt hat. Mengen, für die das zutrifft, werden in der Literatur häufig kontaminiert genannt, was auf die historische Entwicklung hinweist. Gefundene Elemente werden meist defekt genannt. Die Motivation dafür ist, daß die Gruppentests auch auf komplexe Maschinen oder elektronische Schaltungen angewendet werden. Wird ein Teil der Maschine bzw. Schaltung getestet, so funktioniert er nur, wenn er kein defektes Bauteil enthält.

Wir können die möglichen Fragealgorithmen nach verschiedenen Kriterien bewerten. Ein Kriterium ist die maximale Anzahl von Fragen, die ein Algorithmus stellen muß. Um diese Zahl zu bestimmen, stellen wir uns ein Orakel $\mathcal{O}$ vor, das sich zunächst auf gar kein Element festlegt. Es antwortet nach jeder Frage so, daß $\mathcal{A}$ noch möglichst viele weitere Fragen stellen muß. $\mathcal{O}$ muß dabei lediglich darauf achten, daß die Antworten noch mindestens ein Element aus S zulassen. Das Ziel der kombinatorischen Suche ist es, Algorithmen zu finden, die die Anzahl von maximal notwendigen Fragen minimieren. Dabei besteht der schwierige Teil meist in dem Beweis, daß ein Algorithmus optimal ist. Ein sehr wichtiges Hilfsmittel ist hierfür die informationstheoretische Schranke. Wir stellen sie in Kapitel 3 vor.

Wir haben oben einen Suchprozeß kennengelernt, in dem mehrere Elemente zu finden waren, und der Fragende die Anzahl der Elemente nicht kennt. In diesem Fall wird ein Algorithmus im allgemeinen mehr Fragen stellen müssen, als ein Algorithmus, der die Anzahl der gesuchten Elemente kennt. Mit dieser Fragestellung beschäftigt sich die kompetitive Suche.

Neben der „worst case"-Analyse ist es auch möglich, daß auf der Menge S eine Wahrscheinlichkeitsverteilung gegeben ist. In dem Fall können wir die erwartete Anzahl von Fragen eines Suchalgorithmusses betrachten. Für diese Analyse liefert die Entropie der Wahrscheinlichkeitsverteilung eine untere Schranke für den Erwartungswert. Wir verweisen hierzu auf [AIG–88].

Wie wir oben schon erwähnt haben, können Suchprozesse bei gleicher Menge S, aber unterschiedlichen erlaubten Fragen, zu verschiedenen optimalen Algorithmen führen. Gerade die Infektionstests bei Blutproben sind in den letzten Jahren in der Öffentlichkeit diskutiert worden. Die Annahmen, es können beliebig viele Blutproben zusammen gemischt und auf einmal getestet werden, erwies sich bei HIV-Infektionen als falsch. Es ist infolgedessen die Aufgabe von Medizinern festzustellen, ob es zumindest eine Konstante $k \in \mathbb{N}$ gibt, so daß der Test korrekt arbeitet, falls nicht mehr als k Blutproben zusammen gemischt werden. Diese Einschränkung wurde theoretisch bereits gelöst. Wir geben in Kapitel 3 die Suchkomplexität an.

Neben einer Vorgabe der Maximalgröße kommen auch andere Einschränkungen der Testmengen in Frage. Sie ergeben sich in der Anwendung häufig, wenn das gesuchte Element in einer Datenstruktur gespeichert ist. Wir werden als Datenstruktur die gerichteten Bäume betrachten. Wir nehmen dabei an, daß sich das eine oder die mehreren gesuchten Elemente unter den Blättern befinden. Als Testmengen sind dann alle Teilbäume erlaubt.

Als letztes ist etwas zum „Scheduling“ der Fragen zu sagen. Die beiden wichtigsten Fälle sind

- jeder Test wird erst dann ausgewählt, wenn die Ergebnisse der vorhergehenden Tests bekannt sind,

- alle Tests werden von vornherein festgelegt.

Die erste Methode benötigt häufig, aber nicht immer, weniger Fragen als die zweite. Die zweite Methode hat jedoch den Vorteil, daß sich die Tests parallel ausführen lassen. Falls die Zeit der kritische Faktor ist, ist diese Methode zu bevorzugen.

Wir werden neben den beiden extremen Vorgaben an den Fragenden auch untersuchen, wie sich die Suchkomplexität entwickelt, wenn die Fragen teilweise parallel gestellt werden. Eine Formalisierung dieser Idee werden wir in Kapitel 7 geben.

Die Arbeit ist folgendermaßen gegliedert: In Kapitel 2 geben wir die mathematische Definition von Suchproblemen. Wir stellen dabei verschiedene Sichtweisen des Problems dar. Es hat sich gezeigt, daß, je nach Suchprozeß, eine andere Zugangsweise zu einer Lösung führt.

In Kapitel 3 zeigen wir einige grundlegende Ergebnisse aus der bisherigen Forschung in dem Gebiet der kombinatorischen Suche. Insbesondere werden wir die sehr wichtige informationstheoretische Schranke vorstellen.

Kapitel 4 behandelt den Fall eines defekten Blattes in einem beliebigen Baum. Wir werden dabei, wie in der kombinatorischen Suche üblich, nicht spezielle Bäume betrachten, sondern zu gegebenem $n \in \mathbb{N}$ alle Bäume mit n Blättern untersuchen. Jeder dieser Bäume liefert dann einen Suchprozeß.

Wir fragen dann bei gegebenem n nach den Bäumen, die die Suchkomplexität aller dieser Suchprozesse maximieren.

In Kapitel 5 untersuchen wir die Suchkomplexität bei mehreren defekten Blättern in binären Bäumen. Jeder Suchalgorithmus muß sich dabei mit dem Individualtest vergleichen lassen. Dieser fragt bei n Blättern $n-1$ Blätter einzeln nacheinander. Danach ist von allen Blättern bekannt, ob sie defekt sind oder nicht.

Es wird sich zeigen, daß allgemeine Bäume immer eine sehr schlechte Suchkomplexität ergeben. Um dennoch für die Praxis nützliche Ergebnisse zu liefern, betrachten wir daher die vollständigen binären Bäume. Bei wenigen defekten Blättern braucht ein optimaler Suchalgorithmus weniger Fragen als der Individualtest.

Die erhaltenen Suchkomplexitäten vergleichen mit den Laufzeiten, welche kompetitive Algorithmen erreichen können. Wie oben schon erwähnt ist ein kompetitiver Algorithmus einer, der bei nicht bekannter Anzahl von defekten Elementen am Ende der Suche den Zustand von allen Elementen kennt.

Nach dieser, in der Praxis wichtigen, Einschränkung auf binäre Bäume, betrachten wir in Kapitel 6 vollständige reguläre q-Bäume. Dies sind Bäume, bei denen jeder innere Knoten genau q Söhne hat. Auch bei ihnen kann bei mehreren, aber nicht zu vielen, defekten Blättern ein Gruppentest gegenüber dem Individualtest Fragen sparen.

In Kapitel 7 untersuchen wir dann die Suchkomplexität von parallelen Algorithmen, also solchen, die alle Fragen gleichzeitig stellen. Da diese Suchprozesse wieder eine sehr hohe Suchkomplexität haben, erweitern wir das Modell auf solche Algorithmen, die die Fragen nur teilweise parallel stellen. Wir können damit die strikte Trennung zwischen sequentieller und paralleler Suche aufheben. Wir untersuchen in diesem Kapitel zunächst wieder die Suchkomplexität bei vollständigen binären Bäumen.

In Kapitel 8 verallgemeinern wir die Ergebnisse von Kapitel 7 auf vollständige reguläre q-Bäume. Mit den erweiterten Mitteln dieses Kapitels untersuchen wir die Laufzeit von parallelen Algorithmen bei beliebigen Bäumen.

In Kapitel 9 beschäftigen wir uns abschließend mit einer von Stanislaw Ulam eingeführten Erweiterung der kombinatorischen Suche. Es geht um Suchprozesse, bei denen das Orakel lügen darf. Im parallelen Fall führt dies bei allgemeinen Suchprozessen zu den fehlerkorrigierenden Codes. Wir werden bei den Bäumen die sequentiellen Suchalgorithmen bei einer erlaubten Lüge betrachten.

Ich möchte mich bei Prof. Dr. Martin Aigner für die motivierende Betreuung während meiner Zeit im Graduiertenkolleg „Algorithmische Diskrete Mathematik“ an der Freien Universität Berlin bedanken. Er gab mir nicht nur Anregungen zur Forschung, sondern verhinderte auch mehr als einmal, daß ich mich zu lange an einem Problem festbiß.

Ich bedanke mich weiterhin bei Frau Bocan und Frau Felsner. Sie waren Anlaufstelle für organisatorische Probleme aller Art und nahmen auch einer Reisekostenabrechnung den Schrecken.

Ebenfalls bedanke ich mich bei Claudia Siekmeier für ihre Unterstützung. Ihre Kenntnisse der deutschen Rechtschreibung waren mir eine große Hilfe.

Kapitel 2

Notation

In der Literatur ist bis heute nicht eindeutig definiert, welches die kleinste **natürliche Zahl** ist. Für die Menge $\{1, 2, 3, \ldots\}$ schreiben wir das Symbol $\mathbb{N}$. Die Menge $\{0, 1, 2, \ldots\}$ bezeichnen wir mit $\mathbb{N}_0$.

Ist T ein Baum, so setzen wir $|T|$ als die **Anzahl der Blätter** von T. Dies ist eine andere Zahl, als die graphentheoretische Anzahl der Knoten des Baumes. Da die von uns gesuchten Elemente aber immer unter den Blättern zu finden sind, ist dies die angemessene Größe.

Wir werden bei Bäumen immer annehmen, daß sie gerichtete Bäume sind. Diese haben immer eine ausgezeichnete Wurzel. Sie heißen daher auch **Wurzelbäume**. Die Orientierung der Kanten geht dabei von der Wurzel weg. Die Wurzel ist folglich der einzige Knoten, dessen Eingangsgrad 0 ist. Zu jedem anderen Knoten x existiert ein eindeutiger Knoten y, so daß im Baum eine gerichtete Kante yx existiert. Wir nennen dann y den **Vater**[1] von x und x den **Sohn** von y. Die **Vorfahren** eines Knotens x sind alle Knoten, von denen ein gerichteter Weg zu x führt. Die **Nachfolger** sind entsprechend alle Knoten, die x als Vorfahre haben. Haben zwei Knoten x und x' den gleichen Vater y, so ist x der **Bruder** von x'. Haben k Knoten $x_1, \ldots, x_k$ den gleichen Vater y, so sind die Knoten $x_1, \ldots, x_k$ **Geschwister**.

Die **Tiefe** der Wurzel setzen wir auf 0. Die Tiefe jedes anderen Knoten, ist die Tiefe seines Vaters plus 1. Die Tiefe eines Baumes ist die maximale

[1]Diese Bezeichnung ist wegen des männlichen Geschlechts nicht unumstritten. Sie ist jedoch in der deutschsprachigen Literatur üblich. Der Begriff kann auch nicht einfach durch Vorfahre oder Vorgänger ersetzt werden, da diese Begriffe schon eine andere Bedeutung haben.

Tiefe seiner Knoten.

Ist x ein Knoten in einem Wurzelbaum T, so besteht der Baum $\boldsymbol{T_x}$ aus dem Knoten x und allen Nachfolgern von x.

Sind mehrere Bäume $T_1, \ldots, T_q$ gegeben, so ist $\boldsymbol{(T_1, \ldots, T_q)}$ der Baum, dessen Wurzel q Söhne $x_1, \ldots, x_q$ hat, und für alle $i = 1, \ldots, q$ jeweils $T_i = T_{x_i}$ gilt. Abbildung 2.1 zeigt den Baum $(T_1, T_2, \ldots, T_q)$.

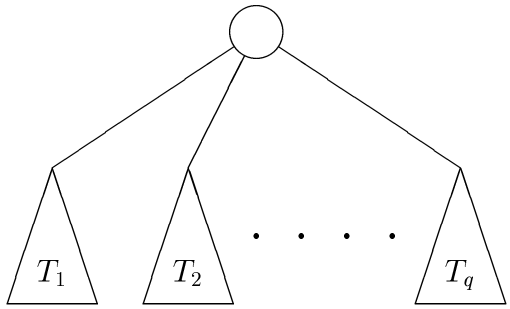

Abbildung 2.1: Der Baum $(T_1, T_2, \ldots, T_q)$

Ist T ein Baum und S ein Teilbaum, so ist $\boldsymbol{T - S}$ der von T übrigbleibende Baum, wenn alle zu S gehörenden Knoten von T entfernt werden. Sollte bei dem Baum $T - S$ ein Knoten entstehen, der nur einen Sohn hat, so können wir die Kante, die diese beiden Knoten verbindet, kontrahieren. An der weiter unten definierten Suchkomplexität ändert sich dabei nichts.

Wir werden daher häufig annehmen, daß die auftretenden Bäume niemals einen inneren Knoten haben, der nur einen Sohn hat. Damit kann es insbesondere nicht passieren, daß der Baum $T - S$ ein Blatt hat, das in T noch ein innerer Knoten war.

Für die Suchkomplexität übernehmen wir die Definitionen aus [AIG–88] und [DUH–93]. Gegeben seien $q \in \mathbb{N}$, $q \geq 2$, eine endliche Menge S und eine Familie $\mathcal{F}$ von **Testfunktionen** $f : S \to \{0, \ldots, q-1\}$. Das Tupel $(S, \mathcal{F})$ wird **Suchprozeß** genannt.

Sei nun in S ein Element x^* gegeben, welches dem Fragenden nicht bekannt ist. Ein **Suchalgorithmus** $\mathcal{A}$ ist eine Folge von Funktion $f_1, \ldots, f_l$ aus $\mathcal{F}$, so daß die Funktionswerte $f_1(x^*), \ldots, f_l(x^*)$ das Element x^* eindeutig bestimmen. l wird dann die **Suchlänge** von x^* genannt. $\mathcal{A}$ muß dabei jedes mögliche Element $x^* \in S$ finden können. Um dies zu betonen, wird so ein Algorithmus daher manchmal auch **korrekter Algorithmus** genannt.

Damit läßt sich die Suchlänge von $\mathcal{A}$ durch

$$L_{\mathcal{A}}(S, \mathcal{F}) = \max_{x^* \in S} l(x^*)$$

definieren. Abkürzend schreiben wir dafür auch $L(\mathcal{A})$.

Ist auf S eine Wahrscheinlichkeitsverteilung $p : S \to [0, 1]$ gegeben, so ist die erwartete Suchlänge von $\mathcal{A}$ definiert als

$$\overline{L}_{\mathcal{A}}(S, \mathcal{F}) = \sum_{x^* \in S} p(x^*) l(x^*).$$

Die Suchlänge oder **Suchkomplexität** des Suchprozesses ist dann definiert als

$$L(S, \mathcal{F}) = \min_{\mathcal{A}} L_{\mathcal{A}}(S, \mathcal{F}).$$

Suchalgorithmen, die diese minimale Laufzeit erreichen, nennen wir **optimal**.

Bei gegebenem p ist die erwartete Suchlänge oder Suchkomplexität entsprechend als

$$\overline{L}(S, \mathcal{F}) = \min_{\mathcal{A}} \overline{L}_{\mathcal{A}}(S, \mathcal{F})$$

definiert.

Als Erweiterung dieses Modells werden häufig mehrere Suchprozesse betrachtet und unter diesen das Maximum der Suchkomplexitäten untersucht. Als Beispiel kann die Suche von Kanten in Graphen betrachtet werden. Hier wird meist nicht die konkrete Graphenstruktur gegeben, sondern es werden nur Angaben über die Anzahl der Ecken und Kanten gemacht. Jeder Graph, der diese Eigenschaften erfüllt, stellt einen eigenen Suchprozeß dar. Bei gegebenen Parametern ist dann die maximale Suchlänge von all diesen Suchprozessen gefragt.

Wir werden zum Beispiel in Kapitel 4 alle Bäume betrachten, die n Blätter haben, und bei denen jeder Knoten maximal q Söhne hat.

Wir haben in der Einleitung schon die Begriffe defekt und kontaminiert erwähnt. Wenn der Algorithmus von einem Blatt festgestellt hat, daß dieses eines der gesuchten Blätter ist, so nennen wir dieses Blatt **defekt**. Hat der Algorithmus ermittelt, daß ein Teilbaum des gesamten Baumes mindestens ein defektes Blatt beinhalten muß, so nennen wir diesen Teilbaum **kontaminiert**. Ein nicht kontaminierter Teilbaum ist **rein**. Einen Knoten x nennen wir **kontaminiert**, wenn der zugehörige Teilbaum T_x kontaminiert ist. Nach dieser Definition ist ein defektes Blatt auch kontaminiert. Das ist zwar pathologisch, stört aber nicht.

Wir betonen noch einmal, daß ein Blatt erst dann defekt und ein Knoten erst dann kontaminiert ist, wenn der Suchalgorithmus dies zweifelsfrei festgestellt hat. Welche Blätter defekt sind, ist vor der Suche nicht festgelegt. Es ergibt sich erst durch die Antworten des Gegenspielers. Von diesem Gegenspieler müssen wir dabei annehmen, daß er immer die für den Fragenden ungünstigste Antwort wählt. Wenn also zum Beispiel bei einem zu findenden Blatt in einem Baum mit mehr als zwei Blättern die erste Frage ein Blatt ist, so ist die Antwort immer „Nein". Der Suchalgorithmus kann kein Glück haben und ein „Ja" als Antwort erhalten, da sich der Gegenspieler noch frei entscheiden kann, welches Blatt defekt ist.

Außer der oben stehenden Definition für die Suchkomplexität gibt es noch einige äquivalente Definitionen. Diese ermöglichen bei manchen Fragestellungen eine einfachere Analyse. So läßt sich ein Suchprozeß als Baum, Präfixcode oder Strategiespiel auffassen. Eine detaillierte Darstellung gibt es dazu in [AIG–88].

Die Baumdarstellung hat den Vorteil, daß der komplette Suchalgorithmus mit allen Entscheidungen zur Verfügung steht. Gerade bei der Analyse von Fragestrategien ist dies von Vorteil.

Die Darstellung als Präfixcode hat den Nebeneffekt, daß alle über diese Codes bekannten Sätze auf die Suchprozesse übertragen werden können. So liefert das Noiseless Coding Theorem von Shannon eine untere Schranke für die erwartete Suchlänge von Suchprozessen bei gegebener Wahrscheinlichkeitsverteilung.

Die Sichtweise als Strategiespiel hilft häufig, Schranken für die Suchkomplexität zu erhalten. Meist gibt es intuitiv gute Frage- und Antwortstrategien. Jede Fragestrategie liefert dann eine obere und jede Antwortstrategie eine untere Schranke für die Suchkomplexität. Schon allein durch die Trennung der Berechnung in obere und untere Schranke ist diese Definition sehr nützlich. Wir werden davon im weiteren Verlauf der Arbeit sehr häufig Gebrauch machen.

Daneben ist es für kleine Mengen S möglich, diese Strategiespiele auf einem Computer zu implementieren und so die Suchkomplexität zu berechnen. Diese Beispiele können einen Hinweis liefern, wie sich die Suchkomplexität bei wachsender Menge S entwickelt.

Bei diesen Strategiespielen nennen wir den antwortende Spieler **Orakel**. Das Orakel legt sich dabei zu Beginn des Spiels noch nicht fest, welche Elemente defekt sind. Es antwortet lediglich so, daß es noch mindestens eine Möglichkeit gibt, wie die defekten Blätter in dem Baum liegen. Orakel, die immer mindestens soviele Fragen erzwingen, wie die Suchkomplexität des Suchprozesses angibt, nennen wir **optimale Orakel**. Im Gegensatz zu den optimalen Suchalgorithmen ist diese Bezeichnung in der Literatur nicht üblich. Sie erweist sich aber bei Beweisen als nützlich.

Kapitel 3

Grundlagen

3.1 Allgemeine Suchprozesse

Bei Suchproblemen werden sehr häufig Abschätzungen über die Suchkomplexität benötigt. Die wichtigste untere Schranke ist hierfür die informationstheoretische Schranke.

Satz 3.1 (Informationstheoretische Schranke) *Sei $q \in \mathbb{N}$, $q \geq 2$ und $(S, \mathcal{F})$ ein Suchprozeß mit $\mathcal{F} \subseteq \{0, \ldots, q-1\}^S$. Dann gilt*

$$L(S, \mathcal{F}) \geq \lceil \log_q |S| \rceil .$$

Ist $\mathcal{F} = \{0, \ldots, q-1\}^S$, so gilt Gleichheit.

Der kanonische Beweis dieses Satzes verdeutlicht sehr gut die Beweisstrategie bei den unteren Schranken. Insbesondere zeigt er den Aufbau von optimalen Orakeln.

Beweis: Die erste Frage sei eine Funktion $f \in \mathcal{F}$. Das Orakel wählt eine Antwort aus $\{0, \ldots, q-1\}$, die die Menge der noch möglichen Elemente aus S maximiert. Nach dem Schubfachprinzip ist diese Anzahl mindestens $|S|/q$. Mittels vollständiger Induktion hat diese Menge nach l Fragen damit noch mindestens die Größe $|S|/q^l$. Damit ein Algorithmus mit l Fragen das Element finden kann, muß daher $|S|/q^l \leq 1$ gelten. Somit gilt $l \geq \log_q |S|$.

Stehen dem Fragenden alle Fragen zur Verfügung, so wählt er jeweils eine solche Testfunktion, die die restliche Menge möglichst gleichmäßig partitioniert. Damit erreicht der Fragende, daß nach l Fragen die noch übrigbleibende

Menge maximal die Größe $\lceil |S|/q^l \rceil \leq 1$ hat.

$\square$

3.2 Kombinatorische Gruppentests

Bei den Gruppentests besitzt die zu untersuchenden Menge mehrere defekte Elemente. Bezeichnen wir die Größe der Menge mit n und die Anzahl der defekten Elemente mit d, so ist $M(d, n)$ die Suchkomplexität, um diese Elemente zu identifizieren. Als Tests sind alle Teilmengen von S erlaubt. Die Antwort besagt jeweils, ob diese Testmenge kontaminiert ist, also mindestens ein defektes Element beinhaltet. Die informationstheoretische Schranke liefert damit

$$M(d, n) \geq \left\lceil \log_2 \binom{n}{d} \right\rceil .$$

Eine sehr wichtige Tatsache bei Gruppentests ist die aus [DUH–93] stammende Überlegung. Eine Menge A sei als kontaminiert erkannt. Im Laufe des Suchalgorithmusses wird von einer Menge $B \subseteq A$ erkannt, daß sie kontaminiert ist. Die Information, daß A kontaminiert ist, ist damit wertlos. Dadurch, daß B kontaminiert ist, ist A automatisch kontaminiert. Die Information, die ein Suchalgorithmus zu einem bestimmten Zeitpunkt hat, besteht also aus kontaminierten Mengen, bei denen nie eine Menge eine andere komplett beinhaltet.

Der Fall $d = 1$ ist einfach, und eine binäre Suche erreicht die informationstheoretische Schranke. Schon der Fall $d = 2$ ist jedoch schwierig, und die bekannten unteren und oberen Schranken liegen immer noch auseinander. Näheres steht dazu in [DUH–93], Seite 48–53. Für noch größere d ist für $d \geq \frac{8}{21}n$ bewiesen, daß $n - 1$ Fragen notwendig sind. Vermutet wird, daß dies bereits ab $d \geq \frac{1}{3}n$ gilt. Umgekehrt ist bekannt, daß für $d < \frac{1}{3}n$ die Suchkomplexität $M(d, n) < n - 1$ ist. Die Beweise hierfür stehen ebenfalls in [DUH–93].

Ist d dem Fragenden nicht bekannt, so wird dies *kompetitive Suche* genannt. Ein kompetitiver Suchalgorithmus $\mathcal{A}$ muß also ohne Kenntnis von d den Zustand aller Elemente von S erfragen. Falls alle Elemente von S defekt sind, braucht er dafür mindestens n Fragen. Der Individualtest, der alle Elemente von S einzeln fragt, ist also in diesem Sinne optimal.

Wenn allerdings n groß ist, so ist diese Suchstrategie bei kleinem d sehr schlecht. Um eine angemessene Bewertung von kompetitiven Suchalgorithmen zu erhalten, wird ihre Laufzeit mit $M(d,n)$ verglichen. Es sei $M_{\mathcal{A}}(d|n)$ die Laufzeit des kompetitiven Algorithmusses $\mathcal{A}$, wenn in der n-elementigen Menge S genau d defekte Elemente sind. Wir sagen für ein $c > 0$, daß der Algorithmus **c-kompetitiv** ist, wenn es eine Konstante $a > 0$ gibt, so daß für alle $n \in \mathbb{N}$ und $0 \leq d < n$ die Ungleichung $M_{\mathcal{A}}(d|n) \leq cM(d,n) + a$ gilt. Den Fall $d = n$ müssen wir ausschließen, da $M(n,n) = 0$ ist. Wie wir weiter oben bemerkt haben, ist $M_{\mathcal{A}}(n|n) \geq n$. Es gibt also für $d = n$ nie Konstanten c und a, die die geforderte Ungleichung erfüllen.

Bei dem Nachweis, daß ein Algorithmus c-kompetitiv ist, kommt als Problem dazu, daß der Wert von $M(d,n)$ für $2 \leq d < \frac{8}{21}n$ nicht exakt bekannt ist. Es sind lediglich aus der informationstheoretischen Schranke abgeleitete untere Schranken bekannt. Sollten bei der Berechnung von $M(d,n)$ Fortschritte erzielt werden, so wird dies auch bei der Konstruktion von kompetitiven Algorithmen hilfreich sein.

Schlaghoff und Triesch haben in [TRI–97] gezeigt, daß es zu jedem $\epsilon > 0$ einen $(1.5 + \epsilon)$-kompetitiven Algorithmus gibt.

Wir wollen noch erwähnen, daß es neben dieser Definition der Kompetitivität eine weitere, stärkere Definition gibt. Für die Details verweisen wir auf [DUH–93] und [TRI–97].

Wir haben in der Einleitung das in der Medizin sehr wichtige Problem der größenbeschränkten Testmengen erwähnt. Wir nehmen also an, daß in einer Menge S ein Element gefunden werden soll. Dabei sind lediglich Testmengen mit einer maximalen Größe k zugelassen. Ist $k \geq |S|/2$, so ist dies keine Einschränkung. Soll eine Teilmenge A gefragt werden, mit $|A| > k$, so testet der Suchalgorithmus statt dessen die Menge $B = S \backslash A$. Es gilt dann $|B| \leq k$, und die Antwort auf die Testmenge b ist gerade das Komplement der Antwort auf die Testmenge A.

Ist aber $k < |S|/2$, so steigt die Suchkomplexität gegenüber dem unbeschränkten Fall an. Wie wir aus [AIG–88], Seite 64 entnehmen, ist die Suchkomplexität in diesem Fall

$$t + \lceil \log_2(n - tk) \rceil,$$

wobei $t = \lceil \frac{n}{k} \rceil - 2$ ist.

Wir werden in der Arbeit als Grundmenge immer die Blätter eines Wurzelbaumes nehmen. Zulässige Testmengen sind dann jeweils alle zu den Knoten x des Baumes gehörenden Teilbäume T_x. Als Antworten kommen „Ja“ und „Nein“ in Frage. „Ja“ bedeutet, daß dieser Teilbaum kontaminiert ist. Bei „Nein“ ist er rein.

Der einfachste Fall ist ein vollständiger binärer Baum T mit einem defekten Blatt. Der Fragende kann hier mit jeder Frage die Anzahl der noch möglichen Blätter halbieren. Damit erreicht er die informationstheoretische Schranke $\log_2 |T|$.

Hiervon ausgehend werden wir einige Fragestellungen aus der kombinatorischen Suche und den kombinatorischen Gruppentests auf Bäume übertragen.

Kapitel 4

Ein defektes Blatt

4.1 Einleitung

Wir betrachten in diesem Kapitel die sequentielle Suche in allgemeinen Bäumen. Als Parameter für die untersuchten Bäume kommen dabei n, als die Anzahl der Blätter, und q, als die maximale Anzahl der Söhne eines Knotens, in Frage. Wir definieren darum wie in [AIG–88]:

Definition 4.1 ((n,q)-Baum) *Seien $n, q \in \mathbb{N}$ und $q \geq 2$. Ein* **Wurzelbaum** *T ist ein (n,q)-Baum, falls T genau n Blätter besitzt und jeder innere Knoten maximal q Söhne hat. T heißt* **regulär***, falls jeder innere Knoten genau q Söhne besitzt. Die Menge aller (n,q)-Bäume sei $\mathcal{T}_{n,q}$ und $\mathcal{T}_q = \cup_{n \in \mathbb{N}} \mathcal{T}_{n,q}$ die Menge aller q-Bäume.*

Wir untersuchen jetzt die Suchkomplexität für das Finden von einem Blatt. Als zulässige Testmengen erlauben wir dabei alle Teilbäume in dem Baum. Die Antwort besagt, ob dieser Teilbaum kontaminiert ist oder nicht.

Im folgenden nehmen wir dabei an, daß die Bäume *irreduzibel* sind, also jeder innere Knoten mindestens zwei Söhne besitzt. Wir haben bereits in Kapitel 2 erwähnt, daß dies die Suchkomplexität nicht verändert. Dies ist bei den erlaubten Testmengen der Fall. Hat ein Knoten x nur einen Sohn y, so haben x und y genau die gleichen Blätter als Nachfolger. Die Antworten sind also bei x und y immer die gleichen. Kontrahieren wir die Kante xy, so ist der neue Baum ebenfalls ein (n,q)-Baum.

Die Suchkomplexität bei einem Baum T sei $\boldsymbol{L(T)}$. Das Maximum der Suchkomplexität von allen (n,q)-Bäumen bezeichnen wir mit $\boldsymbol{C(q,n)}$.

Rivest hat in [RIV–77] den Fall $q = 2$, also binäre Bäume, untersucht. Bekanntlich hat jeder binäre Baum einen Teilbaum, der mindestens $\frac{1}{3}$ und maximal $\frac{2}{3}$ der Blätter besitzt. Eine triviale obere Schranke für die Suchkomplexität eines binären Baumes mit n Blättern ist somit $\lceil \log_{\frac{3}{2}} n \rceil$. Rivest konnte jedoch durch eine genaue Analyse folgendes zeigen:

Satz 4.2 *Sei für $i \in \mathbb{N}$ f_i die i-te Fibonaccizahl*[1]*. Dann gilt für alle $n \in \mathbb{N}$ mit $f_i \leq n < f_{i+1}$: $C(2, n) = i - 2$.*

Für die Fibonaccizahlen gilt

$$f_i = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^i - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^i .$$

Setzen wir $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$ (Goldener Schnitt), so wächst $C(2, n)$ asymptotisch wie $\log_\phi n$. Dies ist deutlich besser, als die triviale Schranke $\log_{1.5} n$.

Rivest konstruierte in seinem Artikel sogenannte Fibonaccibäume. Dies sind Bäume, die unter allen Bäumen mit vorgegebener Suchkomplexität eine minimale Blattanzahl haben. Wir werden weiter unten solche Bäume für allgemeines $q \in \mathbb{N}$, $q \geq 2$ bestimmen.

4.2 Minimale Bäume

Die Funktion $C(q, n)$ ist monoton wachsend in n. Zur Berechnung von $C(q, n)$ lösen wir das duale Problem. Zu gegebenem $k \geq 1$ berechnen wir das minimale n, so daß $C(q, n) \geq k$ ist. Nach Definition von C ist dies das minimale n, so daß es einen (n, q)-Baum T gibt mit $L(T) \geq k$.

Wir haben bereits die Größe $|T|$ als die Anzahl der Blätter von T definiert. Zusätzlich brauchen wir in diesem Kapitel die Größen $q(T)$ und $g(T)$. Dabei sei $q(T)$ die Anzahl der Söhne der Wurzel von T und

$$g(T) = \max\{|T_1| \,:\, T_1 \text{ ist echter Teilbaum von } T\}.$$

Sind $T_1, \ldots, T_l$ Bäume, so haben wir bereits in Kapitel 2 den Baum $(T_1, \ldots, T_l)$ definiert. Zwei Spezialfälle dieser Konstruktion für einen Baum $T = (T_1, T_2, \ldots, T_l)$ sind dabei

$$J(T) := (T, T_1)$$

[1] $f_1 = 1,\ f_2 = 1,\ f_3 = 2,\ f_4 = 3,\ f_5 = 5,\ f_6 = 8,\ \ldots$

und
$$N(T) := (T_1, T_1, T_2, \ldots, T_l).$$
Die Schreibweise J und N bezieht sich dabei auf die optimale Fragestrategie bei diesen Bäumen. Genaueres dazu folgt weiter unten.

Wir definieren auf den Bäumen eine partielle Ordnung. Für zwei Bäume S und T gelte $S \leq T$, falls $|S| \leq |T|$, $q(S) \leq q(T)$ und $g(S) \leq g(T)$ ist. Genaugenommen ist diese Ordnung also auf den Äquivalenzklassen von Bäumen, die in den drei Werten übereinstimmen, definiert. Bei einer Menge M von Bäumen können wir mit Hilfe der partiellen Ordnung über die minimalen Bäume sprechen, die wir mit $\min M$ bezeichnen.

Als nächstes definieren wir rekursiv Mengen von Bäumen, bei denen nach i Fragen noch k Blätter als defekt in Frage kommen können. Mit $T_1 \subseteq T$ ist dabei gemeint, daß T_1 ein Teilbaum von T ist.

Definition 4.3 *Seien $q, k \in \mathbb{N}$, $q \geq 2$ und $k \geq 2$. Dann sei*
$$\mathcal{B}_{k,q}^{(0)} = \{T \in \mathcal{T}_{n,q} : n \geq k\},$$
und für $i \in \mathbb{N}_0$ sei
$$\mathcal{B}_{k,q}^{(i+1)} = \left\{T \in \mathcal{T}_q : \forall\, T_1 \subseteq T \text{ gilt: } T_1 \in \mathcal{B}_{k,q}^{(i)} \text{ oder } T - T_1 \in \mathcal{B}_{k,q}^{(i)}\right\}.$$

Diese Mengen erfüllen die obige Behauptung. Es gilt nämlich

Satz 4.4 *Für alle $i, q, k \in \mathbb{N}_0$, $q, k \geq 2$ gilt*
$$\mathcal{B}_{k,q}^{(i+1)} \subseteq \mathcal{B}_{k,q}^{(i)}.$$
Es gilt genau dann
$$T \in \mathcal{B}_{k,q}^{(i)},$$
wenn zu jedem Algorithmus der Länge i ein Orakel existiert, so daß nach den Antworten noch mindestens k Blätter von T als defekt in Frage kommen.

Beweis: Wir führen vollständige Induktion über i.

$\boldsymbol{i = 0}$**:** Sei $T \in \mathcal{B}_{k,q}^{(1)}$. Wählen wir als T_1 den Baum T, so muß T_1 in $\mathcal{B}_{k,q}^{(0)}$ liegen, da $T - T_1 = \emptyset$ ist. Also hat T mindestens k Blätter. Daher gilt $T \in \mathcal{B}_{k,q}^{(0)}$. Da $T \in \mathcal{B}_{k,q}^{(1)}$ beliebig war, ist $\mathcal{B}_{k,q}^{(1)} \subseteq \mathcal{B}_{k,q}^{(0)}$. Die zweite Behauptung ist für $i = 0$ trivial. Da keine Fragen gestellt werden, kommen noch alle

Blätter in Frage. Ein Baum muß folglich für beide Behauptungen mindestens k Blätter besitzen. Sie sind daher äquivalent.

$\boldsymbol{i \to i+1}$**:** $\mathcal{B}_{k,q}^{(i+2)} \subseteq \mathcal{B}_{k,q}^{(i+1)}$ ergibt sich sofort aus der rekursiven Definition und der Induktionsvoraussetzung.

Ist $T \in \mathcal{B}_{k,q}^{(i+1)}$ und $\mathcal{A}$ ein beliebiger Suchalgorithmus, so sei T_1 der erste von $\mathcal{A}$ gefragte Teilbaum. Das Orakel antwortet so, daß der restliche Baum in $\mathcal{B}_{k,q}^{(i)}$ liegt. Nach Induktionsvoraussetzung können die restlichen i Fragen so beantwortet werden, daß noch k Blätter in Frage kommen.

Ist T ein Baum, der nicht in $\mathcal{B}_{k,q}^{(i+1)}$ liegt, dann gibt es einen Teilbaum T_1, so daß weder T_1 noch $T - T_1$ in $\mathcal{B}_{k,q}^{(i)}$ liegen. $\mathcal{A}$ fragt daher als erstes den Teilbaum T_1. Nach Induktionsvoraussetzung gibt es i weitere Fragen, so daß danach weniger als k Blätter übrigbleiben.

□

Korollar 4.5 *Seien $i, k, q \in \mathbb{N}_0$, $k, q \geq 2$ und $T \in \mathcal{B}_{k,q}^{(i+1)}$. Sei T' ein Baum, der nach Entfernen eines beliebigen Blattes von T entsteht. Dann gilt*

$$T' \in \mathcal{B}_{k,q}^{(i)}.$$

Beweis: Wäre $T' \notin \mathcal{B}_{k,q}^{(i)}$, so existiert nach Satz 4.4 ein Suchalgorithmus $\mathcal{A}$ der Länge i, bei dem maximal $k-1$ Blätter übrigbleiben. Wir stellen bei T die gleichen i Fragen und fragen dann das Extrablatt. Ist die Antwort „Nein“, so bleiben bei T die gleichen Blätter wie bei T' übrig, also maximal $k-1$. Ist die Antwort „Ja“, so bleibt nur ein Blatt übrig, was auch maximal $k-1$ ist. Nach Satz 4.4 lag T daher nicht in $\mathcal{B}_{k,q}^{(i+1)}$.

□

Dieses Korollar zeigt insbesondere folgenden Eigenschaft: Ist T_1 ein Teilbaum von T und gilt $T_1 \in \mathcal{B}_{k,q}^{(i)}$, so gilt auch $T \in \mathcal{B}_{k,q}^{(i)}$.

Wir zeigen als nächstes, wie ein Baum an der Wurzel verändert werden kann, ohne daß sich seine Zugehörigkeit zu einer dieser Mengen ändert (vergleiche Abbildung 4.1).

Lemma 4.6 *Seien i, k, q wie oben, $l \in \mathbb{N}$, $3 \leq l \leq q+1$ und $T_1, T_2, \ldots, T_l$ q-Bäume. Dann gilt*

$$A := (T_1, (T_2, T_3, \ldots, T_l)) \in \mathcal{B}_{k,q}^{(i)}$$

genau dann, wenn

$$B := (T_2, (T_1, T_3, \ldots, T_l)) \in \mathcal{B}_{k,q}^{(i)}$$

gilt. Ist $l \leq q$, so ist dies ebenfalls äquivalent dazu, daß

$$(T_1, T_2, T_3, \ldots, T_l) \in \mathcal{B}_{k,q}^{(i)}$$

gilt.

Beweis: $\boldsymbol{i = 0}$**:** Die Bäume liegen nach Satz 4.4 genau dann in $\mathcal{B}_{k,q}^{(0)}$, wenn

$$|T_1| + |T_2| + \ldots + |T_l| \geq k$$

gilt.

$\boldsymbol{i \to i+1}$**:** Wir zeigen als Beispiel, daß aus $A \in \mathcal{B}_{k,q}^{(i+1)}$ auch $B \in \mathcal{B}_{k,q}^{(i+1)}$ folgt. Sei also B_1 ein beliebiger Teilbaum von B. Wir haben zu zeigen, daß B_1 oder $B - B_1$ in $\mathcal{B}_{k,q}^{(i)}$ liegt. Ist B_1 ein Teilbaum von einem T_i, $i = 1, \ldots, l$, so ist B_1 auch ein Teilbaum von A, und die Behauptung folgt aus der Induktionsvoraussetzung für $A - B_1$ und $B - B_1$.

Ist $B_1 = (T_1, T_3, \ldots, T_l)$, so gilt $B - B_1 = T_2$. Da aber $A \in \mathcal{B}_{k,q}^{(i+1)}$ ist, liegt T_2 oder $A - T_2 = (T_1, (T_3, \ldots, T_l))$ in $\mathcal{B}_{k,q}^{(i)}$. Also liegt nach Induktionsvoraussetzung auch B_1 oder $B - B_1$ in $\mathcal{B}_{k,q}^{(i)}$.

Die anderen Richtungen des Beweises laufen analog.

□

Dieser Beweis zeigt auch gleich, wie sich optimale Suchalgorithmen bei solchen Transformationen verändern. Ist beim Baum A die erste gestellte Frage der Teilbaum $(T_2, T_3, \ldots, T_l)$, so wird bei B statt dessen der Teilbaum T_1 gefragt. Dies ist gleichwertig, da auch bei A die Fragen $(T_2, T_3, \ldots, T_l)$ und T_1 gleichwertig sind. Das Blatt kann nur in genau einem Teilbaum sein.

Wir warnen jedoch davor, daß eine Umordnung dieser Art an inneren Knoten im Baum seine Zugehörigkeit zu einer der Mengen verändern kann.

Da wir daran interessiert sind, wie klein Bäume sein können, die in einer der Mengen $\mathcal{B}_{k,q}^{(i)}$ liegen, reicht es aus, Bäume aus $\min \mathcal{B}_{k,q}^{(i)}$ zu kennen. Wir definieren nun rekursiv eine Menge von Bäumen, die sich als solche herausstellen werden.

Definition 4.7 *Für $k, q \in \mathbb{N}$, $k \geq 2$ und $q \geq 2$ bestehe $\mathcal{M}_{k,q}^{(0)}$ aus allen Bäumen $T \in \min \mathcal{B}_{k,q}^{(0)}$, die folgende zusätzliche Eigenschaft erfüllen: Ist*

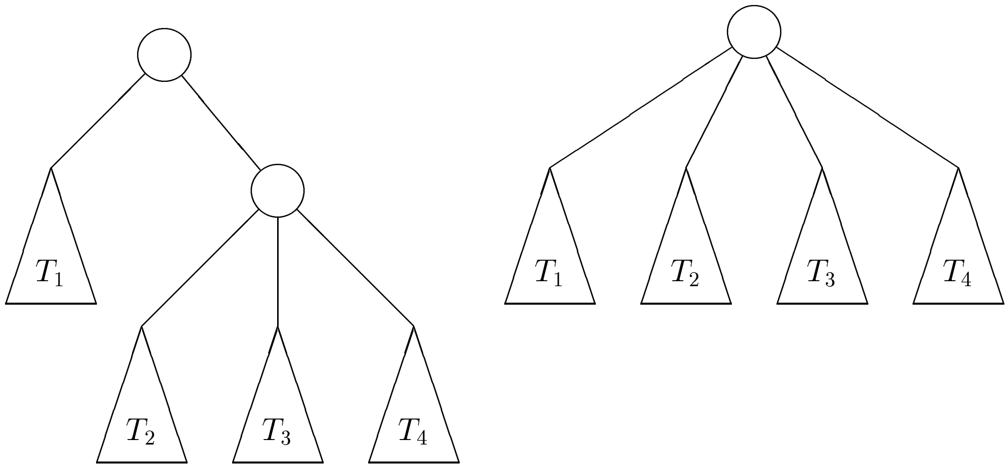

Abbildung 4.1: Zwei Bäume, die in den gleichen Mengen $\mathcal{B}_{k,q}^{(i)}$, $q \geq 4$ liegen.

$T = (T_1, \ldots, T_l)$, so enthält T_1 für alle $i \in \{2, \ldots, l\}$ einen zu T_i isomorphen Teilbaum.

Für $i \in \mathbb{N}_0$ sei

$$\mathcal{M}_{k,q}^{(i+1)} = \min \left\{ \left\{ J(T) \,|\, T \in \mathcal{M}_{k,q}^{(i)} \right\} \cup \left\{ N(T) \,|\, T \in \mathcal{M}_{k,q}^{(i)} \textit{ und } q(T) < q \right\} \right\}.$$

Daß diese Bäume den „worst case" darstellen, ist der Inhalt des folgenden Satzes.

Satz 4.8 *Für alle $i, k, q \in \mathbb{N}_0$, $k, q \geq 2$ ist*

$$\emptyset \neq \mathcal{M}_{k,q}^{(i)} \subseteq \left(\mathcal{B}_{k,q}^{(i)} \backslash \mathcal{B}_{k,q}^{(i+1)} \right).$$

Für alle $T \in \mathcal{B}_{k,q}^{(i)}$ existiert ein $S \in \mathcal{M}_{k,q}^{(i)}$ mit $S \leq T$.

Alle $T \in \mathcal{M}_{k,q}^{(i)}$ erfüllen die Bedingung (*): Ist $T = (T_1, \ldots, T_l)$, so enthält T_1 für alle $i \in \{2, \ldots, l\}$ einen zu T_i isomorphen Teilbaum.

Beweis: Wir führen vollständige Induktion über i.

$\boldsymbol{i = 0}$**:** Nach Definition 4.7 gilt für $i = 0$ die Bedingung (*) und

$$\mathcal{M}_{k,q}^{(0)} \subseteq \mathcal{B}_{k,q}^{(0)}.$$

Für alle Bäume $T \in \mathcal{M}_{k,q}^{(0)}$ gilt $|T| = k$. Andererseits gilt für alle $T \in \mathcal{B}_{k,q}^{(1)}$ $|T| > k$. Daher gilt

$$\mathcal{M}_{k,q}^{(0)} \cap \mathcal{B}_{k,q}^{(1)} = \emptyset.$$

Ist T ein beliebiger Baum aus $\mathcal{B}_{k,q}^{(0)}$, so wählen wir zunächst einen beliebigen Baum S' aus $\min \mathcal{B}_{k,q}^{(0)}$ mit $S' \leq T$. Sei $S' = (S'_1, \dots, S'_l)$. Ohne Einschränkung nehmen wir an, daß $|S'_1| = g(S')$ gilt. Wir setzen nun $S_1 = S'_1$. Für alle $i \in \{2, \dots, l\}$ setzen wir S_i als einen Teilbaum von S_1 mit $|S_i| = |S'_i|$. Für den Baum $S = (S_1, \dots, S_l)$ gilt dann $|S| = |S'| = k$, $q(S) = q(S')$ und $g(S) = g(S')$. Also gilt $S \leq T$. S erfüllt nach Konstruktion die Bedingung (*). S liegt also in $\mathcal{M}_{k,q}^{(0)}$.

Mit der Existenz von S haben wir insbesondere gezeigt, daß $\mathcal{M}_{k,q}^{(0)} \neq \emptyset$.

$\boldsymbol{i \to i+1}$**:** Da $\mathcal{M}_{k,q}^{(i)} \neq \emptyset$ ist, ist auch $\mathcal{M}_{k,q}^{(i+1)} \neq \emptyset$.

Ist $T \in \mathcal{M}_{k,q}^{(i+1)}$ beliebig, so existiert ein Baum $T' \in \mathcal{M}_{k,q}^{(i)}$ mit $T = J(T')$ oder $T = N(T')$. Nach Induktionsvoraussetzung erfüllt T' die Bedingung (*). Ist nun $T' = (T'_1, \dots, T'_l)$, so erfüllen sowohl $J(T') = (T', T'_1)$, als auch $N(T') = (T'_1, T'_1, \dots, T'_l)$ die Bedingung (*).

Wir zeigen nun, daß $T \in \mathcal{B}_{k,q}^{(i+1)}$ gilt. Nach Induktionsvoraussetzung gilt $\mathcal{M}_{k,q}^{(i)} \subseteq \mathcal{B}_{k,q}^{(i)}$. Also ist $T' \in \mathcal{B}_{k,q}^{(i)}$. Sei T_1 ein beliebiger echter Teilbaum von T. Für die Lage von T_1 und T' gibt es drei Möglichkeiten. Ist $T_1 = T'$, so gilt $T_1 \in \mathcal{B}_{k,q}^{(i)}$. Liegt T_1 außerhalb von T', so hat der Baum $T - T_1$ den Baum T' als Teilbaum. Daraus folgt $T - T_1 \in \mathcal{B}_{k,q}^{(i)}$. Liegt T_1 echt in T', so liegt T_1 in einem T'_o, $o \in 1, \dots, l$. Nach Bedingung (*) enthält T'_1 einen zu T'_o isomorphen Teilbaum. Das gilt natürlich auch für $o = 1$. Damit enthält $T - T'_o$ einen zu T' isomorphen Teilbaum. Also gilt $T - T'_o \in \mathcal{B}_{k,q}^{(i)}$. Aus $T_1 \subseteq T'_o$ folgt $T - T'_o \subseteq T - T_1$ und damit $T - T_1 \in \mathcal{B}_{k,q}^{(i)}$. Wir haben also in allen drei Fällen nachgewiesen, daß $T_1 \in \mathcal{B}_{k,q}^{(i)}$ oder $T - T_1 \in \mathcal{B}_{k,q}^{(i)}$ gilt. Da T_1 beliebig war, gilt $T \in \mathcal{B}_{k,q}^{(i+1)}$.

Wir zeigen jetzt

$$\mathcal{M}_{k,q}^{(i+1)} \cap \mathcal{B}_{k,q}^{(i+2)} = \emptyset.$$

Angenommen es existiere ein

$$T \in \mathcal{M}_{k,q}^{(i+1)} \cap \mathcal{B}_{k,q}^{(i+2)}.$$

Nach Korollar 4.5 können wir von einem größten Teilbaum von T ein Blatt entfernen und erhalten einen in $\mathcal{B}_{k,q}^{(i+1)}$ liegenden Baum T'. Es gilt also $T' < T$, was im Widerspruch zu $T \in \mathcal{M}_{k,q}^{(i+1)}$ steht, da $\mathcal{M}_{k,q}^{(i+1)}$ nur minimale Bäume aus $\mathcal{B}_{k,q}^{(i+1)}$ enthält.

Als letztes zeigen wir die Existenz eines Baumes $S \in \mathcal{M}_{k,q}^{(i+1)}$ mit $S \leq T$. Wir nehmen an, es gäbe einen Baum $T \in \mathcal{B}_{k,q}^{(i+1)}$, so daß für keinen Baum $S \in \mathcal{M}_{k,q}^{(i+1)}$ die Gleichung $S \leq T$ gilt. Wir wählen unter all diesen Bäumen einen, der bezüglich der partiellen Ordnung minimal ist.

Sei $T = (T_1, T_2, \ldots, T_k)$. Ohne Einschränkung nehmen wir an, daß für die Teilbäume $|T_1| \geq |T_2| \geq |T_k|$ gilt, da isomorphe Bäume die gleiche Suchkomplexität haben und eine andere Reihenfolge der Teilbäume die drei Werte $|T|$, $q(T)$ und $g(T)$ nicht ändert.

Da $T \in \mathcal{B}_{k,q}^{(i+1)}$ gilt, folgt $T_1 \in \mathcal{B}_{k,q}^{(i)}$ oder $T - T_1 \in \mathcal{B}_{k,q}^{(i)}$.

Ist $T - T_1 = (T_2, \ldots, T_k) \in \mathcal{B}_{k,q}^{(i)}$, so wählen wir ein nach Induktionsvoraussetzung existierendes $S' = (S_2, \ldots, S_k) \in \mathcal{M}_{k,q}^{(i)}$, so daß $|S_2| \geq \ldots \geq |S_k|$ und $S' \leq (T_2, \ldots, T_k)$ ist. Dann ist aber $S'' = J(S') \in \mathcal{B}_{k,q}^{(i+1)}$, und es gilt $S'' \leq T$. Damit existiert jedoch auch ein $S \in \mathcal{M}_{k,q}^{(i+1)}$ mit $S \leq T$. Die Annahme ist also widerlegt.

Ist $T_1 \in \mathcal{B}_{k,q}^{(i)}$, so wählen wir ein $S_1 \in \mathcal{M}_{k,q}^{(i)}$ und setzen $S = J(S_1)$. Wir zeigen wieder $S \leq T$, was wiederum die Annahme widerlegt.

Behauptung 1: Es gilt $|T_1| > |T|/2$. Es gilt $q(S) = 2$, da $S = J(S_1)$. Außerdem ist $|S| \leq 2|S_1|$. Aus $|T| \geq 2|T_1|$ folgt daher $S \leq T$, da $|S_1| \leq |T_1|$.

Behauptung 2: Es gilt $q(T) = 2$. Wäre $q(T) > 2$, so ließe sich durch Anwendung von Lemma 4.6 ein Baum T' konstruieren mit $|T'| = |T|$, $q(T') = 2$ und $g(T') = |T_1| = g(T)$, da $|T_1| > |T|/2$ ist. Dies steht im Widerspruch zur Minimalität von T.

Behauptung 3: Es gilt $g(T_1) \leq |T| - |T_1|$. Andernfalls würde Lemma 4.6 wiederum einen kleineren Baum liefern.

Behauptung 4: Im Gegensatz zur Annahme gilt doch $S \leq T$. Es gilt $q(S) = 2 \leq q(T)$, $g(S) = |S_1| \leq |T_1| = g(T)$ und

$$|S| = |S_1| + g(S_1) \leq |T_1| + g(T_1) \leq |T|.$$

$\square$

Der Beweis erklärt jetzt auch die Schreibweisen $J(T)$ und $N(T)$. Ein optimaler Suchalgorithmus würde in beiden Fällen den Teilbaum T fragen. Im ersten Fall wäre die Antwort des Orakels „Ja“, im zweiten Fall „Nein“.

Die Abbildungen 4.2, 4.3 und 4.4 zeigen für $i \in \{0, 1, 2\}$ die Mengen $\mathcal{M}_{2,3}^{(i)}$. Für $i = 2$ besteht diese Menge nur aus zwei Bäumen. Abbildung 4.5 zeigt den Baum aus der Menge $\mathcal{B}_{2,3}^{(2)}$, der nicht in der Menge $\mathcal{M}_{2,3}^{(2)}$ liegt. Dieser Baum hat 5 Blätter, sein größter Teilbaum hat 3 Blätter, und die Wurzel hat 2 Söhne. Er ist damit größer als der Baum mit vier Blättern.

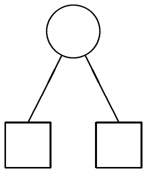

Abbildung 4.2: Die Menge $\mathcal{M}_{2,3}^{(0)}$

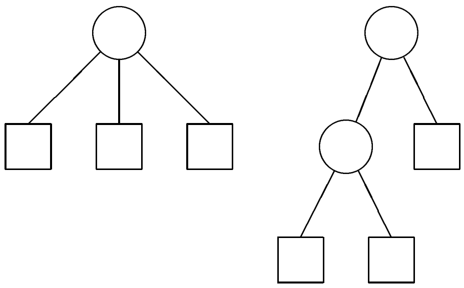

Abbildung 4.3: Die Menge $\mathcal{M}_{2,3}^{(1)}$

Nach Satz 4.8 brauchen wir nicht die komplette Struktur der Bäume kennen, sondern lediglich die drei Parameter $(|T|, g(T), q(T))$. Wir verwenden also auf $\mathbb{N}^3$ die gewöhnliche partielle Ordnung und setzen

$$m_{k,q}^{(i)} = \min \left\{ (|T|, g(T), q(T)) \mid T \in \mathcal{B}_{k,q}^{(i)} \right\}.$$

Dann gilt

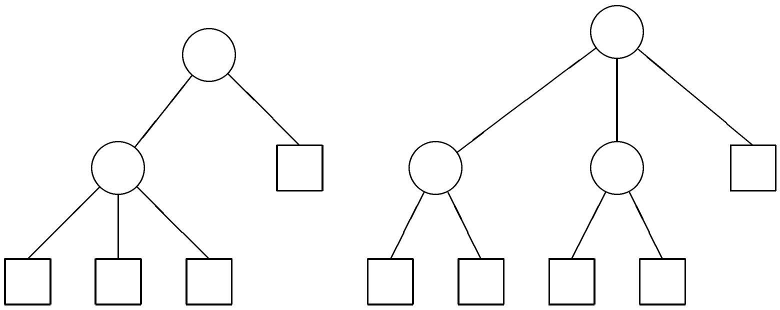

Abbildung 4.4: Die Menge $\mathcal{M}_{2,3}^{(2)}$

Korollar 4.9 *Seien $q, k \in \mathbb{N}$, $q \geq 2$ und $k \geq 2$. Dann ist*

$$m_{k,q}^{(0)} = \min\{(a,b,r) \mid a,b,r \in \mathbb{N}, 2 \leq r \leq q \text{ und } rb \geq a\},$$

und für $i \in \mathbb{N}_0$ ist

$$\begin{aligned} m_{k,q}^{(i+1)} = \quad & \min\Big\{\Big\{(a+b,a,2) \mid (a,b,r) \in m_{k,q}^{(i)}\Big\} \\ \cup \quad & \Big\{(a+b,b,r+1) \mid (a,b,r) \in m_{k,q}^{(i)} \text{ und } 2 \leq r \leq q-1\Big\}\Big\}. \end{aligned}$$

In Tabelle 4.1 sind die Tripel der minimalen Bäumen für $k = 2$ und $q = 2, 5$ angegeben. Der Grund für $k = 2$ wird in Satz 4.10 gegeben. Die Tripel sind dabei nach ihrer ersten Komponente sortiert. Die erste Komponente des ersten Tripels zeigt also die minimale Blattanzahl, die Bäume in der Menge $\mathcal{B}_{2,q}^{(i)}$ haben. Insbesondere erhalten wir für $q = 2$ die schon aus [RIV–77] bekannten Fibonaccizahlen.

4.3 Suchkomplexität

Wir kommen zur anfänglichen Fragestellung zurück: Wieviele Fragen sind maximal notwendig, um bei gegebener Blattanzahl n und maximaler Anzahl von Söhnen q ein defektes Blatt zu ermitteln?

Satz 4.10 *Seien $i, q \in \mathbb{N}_0$, $q \geq 2$. Ist T ein Baum mit $T \in \mathcal{B}_{2,q}^{(i)} \backslash \mathcal{B}_{2,q}^{(i+1)}$, so ist $L(T) = i + 1$.*

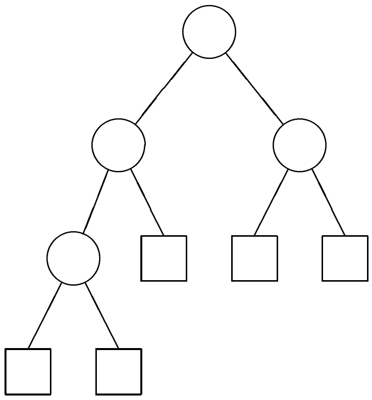

Abbildung 4.5: Der Baum aus $\mathcal{B}_{2,3}^{(2)} \backslash \mathcal{M}_{2,3}^{(2)}$

Beweis: $\boldsymbol{i = 0}$**:** T hat 2 Blätter. Also ist $L(T) = 1$.

$\boldsymbol{i \to i+1}$**:**

„$\leq$“: Da $T \notin \mathcal{B}_{2,q}^{(i+2)}$ gilt, gibt es einen Teilbaum T_1, so daß weder T_1 noch $T - T_1$ in $T \in \mathcal{B}_{2,q}^{(i+1)}$ liegen. Fragt man also zunächst T_1, so liegt der restliche Baum in $\mathcal{B}_{2,q}^{(i)} \backslash \mathcal{B}_{2,q}^{(i+1)}$. Nach Induktionsvoraussetzung reichen somit $i+1$ weitere Fragen.

„$\geq$“: Da $T \in \mathcal{B}_{2,q}^{(i+1)}$ gilt, läßt sich auf jeden möglichen Teilbaum T_1 eine Antwort geben, daß der übrigbleibende Baum in $T \in \mathcal{B}_{2,q}^{(i)}$ liegt. Für diesen Baum sind nach Induktionsvoraussetzung $i+1$ Fragen notwendig.

□

$C(q,n) = \max_{T \in \mathcal{T}_{n,q}} L(T)$ ist monoton wachsend in n. Wir können daher die **Sprungfunktion** $r(t,q)$ für $t \in \mathbb{N}$ durch

$$r(t,q) = \min\{n \in \mathbb{N} \,|\, C(q,n) \geq t\}$$

definieren. $r(t,q)$ ist also die minimale Größe eines q-Baumes, der die Suchkomplexität t hat. $r(t,q)$ läßt sich mit den bisherigen Ergebnissen berechnen. Es gilt

Satz 4.11 *Seien $t, q \in \mathbb{N}$, $q \geq 2$. Dann ist*

$$r(t,q) = \min\left\{|S| \,:\, S \in \mathcal{M}_{2,q}^{(t-1)}\right\}.$$

i	$m_{2,2}^{(i)}$	$m_{2,5}^{(i)}$
0	$(2,1,2)$	$(2,1,2)$
1	$(3,2,2)$	$(3,1,3),(3,2,2)$
2	$(5,3,2)$	$(4,1,4),(4,3,2),(5,2,3)$
3	$(8,5,2)$	$(5,1,5),(5,4,2),(7,2,4),(7,3,3)$
4	$(13,8,2)$	$(6,5,2),(9,2,5),(9,4,3),(10,3,4)$
5	$(13,8,2)$	$(11,5,3),(11,6,2),(13,3,5),(13,4,4)$
6	$(21,13,2)$	$(16,5,4),(16,11,2),(17,4,5),(17,6,3)$
7	$(34,21,2)$	$(21,5,5),(21,16,2),(23,6,4),(27,11,3)$

Tabelle 4.1: Beispiele für $m_{k,q}^{(i)}$

Beweis: „$\leq$“: Sei $S \in \mathcal{M}_{2,q}^{(t-1)}$ mit $n := |S|$ minimal. Dann gilt nach Satz 4.10 $L(S) \geq t$. Insbesondere ist also $C(q,n) \geq t$ und damit $r(t,q) \leq n$.

„$\geq$“: Sei $r(t,q) = n$. Da $C(q,n) \geq t$ ist, existiert ein Baum $T \in \mathcal{T}_{n,q}$ mit $L(T) \geq t$. Aus Satz 4.10 folgt, daß $T \in \mathcal{B}_{2,q}^{(t-1)}$ gilt. Wir wählen ein $S \in \mathcal{M}_{2,q}^{(t-1)}$ mit $S \leq T$. Insbesondere ist dann $|S| \leq |T| = n = r(t,q)$.

$\square$

Tabelle 4.2 zeigt die ersten Werte von $r(t,q)$.

q \ t	1	2	3	4	5	6	7	8	9	10	11	12	13
2	2	3	5	8	13	21	34	55	89	144	233	377	610
3	2	3	4	7	10	15	24	37	56	89	137	209	330
4	2	3	4	5	9	13	17	24	40	56	81	115	185
5	2	3	4	5	6	11	16	21	26	35	60	85	110
6	2	3	4	5	6	7	13	19	25	31	37	48	84
7	2	3	4	5	6	7	8	15	22	29	36	43	50
8	2	3	4	5	6	7	8	9	17	25	33	41	49
9	2	3	4	5	6	7	8	9	10	19	28	37	46
10	2	3	4	5	6	7	8	9	10	11	21	31	41

Tabelle 4.2: Die Sprungfunktion $r(t,q)$

4.4 Asymptotisches Wachstum

Wir untersuchen in diesem Abschnitt, wie die Größe $r(t,q)$ für $t \to \infty$ wächst. Es stellt sich die Frage, ob es für allgemeines q einen Faktor λ_q gibt, so daß $r(t,q)$ asymptotisch wie $(\lambda_q)^t$ wächst. Wie in der Einleitung erwähnt, existiert λ_2 und ist $\frac{1+\sqrt{5}}{2}$.

Für $i \in \mathbb{N}_0$ und $j \in \{2,\dots,q\}$ sei $t_i^{(j)}$ die Größe eines kleinsten[2] Baumes in $\mathcal{B}_{2,q}^{(i)}$, bei dem die Wurzel genau j Söhne hat. Weiterhin setzen wir

$$t_i := \min_{j\in\{2,\dots,q\}} t_i^{(j)}. \tag{4.1}$$

Nach Satz 4.10 ist t_i die minimale Größe eines Baumes mit Suchkomplexität $i+1$.

Um das Wachstum dieser Bäume zu untersuchen, brauchen wir zunächst einige Lemmata.

Lemma 4.12 *Für alle $i \in \mathbb{N}_0$ gilt $t_i^{(2)} \le t_{i+1}^{(2)}$.*

Beweis: Dies folgt aus $\mathcal{B}_{2,q}^{(i+1)} \subset \mathcal{B}_{2,q}^{(i)}$.

□

Lemma 4.13 *Für alle $i \in \mathbb{N}_0$, $i \ge q-2$ gilt $t_i \ge t_{i-q+2}^{(2)}$.*

Beweis: Sei $j \in \{2,\dots,q\}$ beliebig. Dann gilt nach der Rekursionsformel der $\mathcal{B}_{2,q}^{(i)}$

$$t_i^{(j)} \ge t_{i-1}^{(j-1)} \ge \dots \ge t_{i-j+2}^{(2)} \ge t_{i-q+2}^{(2)}.$$

Da j beliebig war, gilt daher

$$t_i = \min_{j\in\{2,\dots,q\}} t_i^{(j)} \ge t_{i-q+2}^{(2)}.$$

□

Wir zeigen nun, daß es reicht, einige dieser Werte zu betrachten, um das Verhalten der ganzen Folge zu bestimmen.

---

[2] bezüglich der Blattanzahl

Lemma 4.14 *Es gilt*

$$\liminf_{i\to\infty} \sqrt[i]{t_i} = \liminf_{i\to\infty} \sqrt[i]{t_i^{(2)}}$$

und

$$\limsup_{i\to\infty} \sqrt[i]{t_i} = \limsup_{i\to\infty} \sqrt[i]{t_i^{(2)}}.$$

Beweis: „$\leq$“: Dies folgt direkt aus Gleichung (4.1).

„$\geq$“: Nach Lemma 4.13 gilt für alle $i \geq q-2$

$$\begin{aligned}
\sqrt[i]{t_i} &\geq \sqrt[i]{t_{i-q+2}^{(2)}} \\
&= \exp\left(\frac{1}{i}\log\left(t_{i-q+2}^{(2)}\right)\right) \\
&= \exp\left(\frac{i-q+2}{i}\frac{1}{i-q+2}\log\left(t_{i-q+2}^{(2)}\right)\right) \\
&= \exp\left(\underbrace{\frac{i-q+2}{i}}_{\to 1}\log\left(\sqrt[i-q+2]{t_{i-q+2}^{(2)}}\right)\right).
\end{aligned}$$

Daraus folgt die Behauptung.

□

Nach Definition gilt immer $t_i \leq t_i^{(2)}$. Kombinieren wir die Lemmata 4.12, 4.13 und 4.14, so erhalten wir

Lemma 4.15 *Es gilt*

$$\liminf_{i\to\infty} \sqrt[i]{t_i} = \liminf_{i\to\infty} \sqrt[q*i]{t_{q*i}^{(2)}}$$

und

$$\limsup_{i\to\infty} \sqrt[i]{t_i} = \limsup_{i\to\infty} \sqrt[q*i]{t_{q*i}^{(2)}}.$$

Um eine obere Schranke für die Folge

$$\sqrt[q*i]{t_{q*i}^{(2)}}$$

zu gewinnen, reicht es, eine feste Rekursionsvorschrift für die Bäume aus $\mathcal{B}_{2,q}^{(i)}$ zu wählen. Sie muß nur der Bedingung genügen, daß nie mehr als $q-2$ mal der Grad der Wurzel erhöht wird.

Satz 4.16 *Für die oben definierte Folge* $(t_i)_{i\in\mathbb{N}}$ *gilt*

$$\limsup_{i\to\infty} \sqrt[i]{t_i} \leq \sqrt[q]{\frac{q+1+\sqrt{q^2+2q-3}}{2}}.$$

Beweis: Wir starten mit T_0 als dem binären Baum mit 2 Blättern. Seien die Bäume T_{kq}, $k = 0, \ldots, i$ schon definiert. Dann setzen wir

$$T_{(i+1)q} = J(J(N^{q-2}(T_{iq}))).$$

Damit liegt für jedes $i \in \mathbb{N}$ der Baum T_{iq} in $\mathcal{B}_{2,q}^{(iq)}$. Also ist $t_{iq}^{(2)} \leq |T_{iq}|$.

Um das Wachstum der Bäume zu ermitteln, stellen wir ihre Größe als Vektor in $\mathbb{N}^2$ dar. Für einen Baum T ist dies der Vektor

$$\begin{pmatrix} |T| \\ g(T) \end{pmatrix}.$$

Ist $S = N(T)$, so gilt mit der Matrix

$$\mathcal{N} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

die Gleichung

$$\begin{pmatrix} |S| \\ g(S) \end{pmatrix} = \mathcal{N} \begin{pmatrix} |T| \\ g(T) \end{pmatrix}.$$

Entsprechend gehört zu J die Matrix

$$\mathcal{J} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Damit entspricht der Abbildung $J^2\dot{N}^{q-2}$ die Matrix

$$\mathcal{Q} = \mathcal{J}^2\mathcal{N}^{q-2} = \begin{pmatrix} 2 & 2q-3 \\ 1 & q-1 \end{pmatrix}.$$

Diese besitzt die Eigenwerte

$$\lambda = \frac{q+1+\sqrt{q^2+2q-3}}{2} \quad \text{und} \quad \mu = \frac{q+1-\sqrt{q^2+2q-3}}{2}$$

mit zugehörigen Eigenvektoren

$$v = \begin{pmatrix} \frac{-q+3+\sqrt{q^2+2q-3}}{2} \\ 1 \end{pmatrix} \quad \text{und} \quad w = \begin{pmatrix} \frac{-q+3-\sqrt{q^2+2q-3}}{2} \\ 1 \end{pmatrix}.$$

Wir können also den Vektor $\begin{pmatrix} 2 \\ 1 \end{pmatrix}$ als Linearkombination $\alpha v + \beta b$ schreiben. Da $\begin{pmatrix} 2 \\ 1 \end{pmatrix}$ kein Eigenvektor der Matrix $\mathcal{Q}$ ist, ist $\alpha \neq 0$. Per vollständiger Induktion folgt für $t \geq 0$

$$Q^t \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \alpha \lambda^t v + \beta \mu^t w.$$

Da $|\lambda| > |\mu|$ ist und v nur positive Einträge besitzt, konvergiert die t-te Wurzel beider Komponenten demnach gegen λ. Die Behauptung folgt damit aus Lemma 4.15.

□

Tabelle 4.3 gibt die ersten 10 Werte an. Wir bemerken noch, daß diese Schranke für $q \to \infty$ gegen 1 konvergiert[3]. Die Effektivität einer einzelnen Frage wird somit beliebig schlecht, wenn der Verzweigungsgrad des Baumes wächst.

q	Schranke für λ_q
2	1.6180
3	1.5511
4	1.4795
5	1.4227
6	1.3782
7	1.3428
8	1.3140
9	1.2901
10	1.2699

Tabelle 4.3: Obere Schranke für beliebige Bäume

[3]Dies folgt aus $\lim_{q\to\infty} \sqrt[q]{q} = 1$.

Beim Beweis von Satz 4.16 liefert jede andere Rekursionsvorschrift ebenfalls eine Schranke für λ_q. Die dabei auftretenden 2×2-Matrizen haben nur positive Einträge. Quadratische Matrizen, die nur positive Einträge besitzen, haben immer einen positiven reellen Eigenwert, der betragsmäßig größer als alle anderen Eigenwerte ist und die Vielfachheit 1 besitzt (siehe [PUL–76]). Die 2×2-Matrizen besitzen also alle zwei Eigenwerte, wobei der betragsmäßig größere positiv ist.

Wir zeigen im folgenden, daß die naheliegende Vorschrift, die Bäume zu saturieren[4], asymptotisch schlechter ist. Betrachten wir die Matrix $\mathcal{J}\mathcal{N}^{q-2}$, welche solche Bäume erzeugt. Es gilt

$$\mathcal{J}\mathcal{N}^{q-2} = \begin{pmatrix} 1 & q-1 \\ 1 & q-2 \end{pmatrix}.$$

Die Eigenwerte dieser Matrix sind

$$\frac{q-1+\sqrt{q^2-2q+5}}{2}$$

und

$$\frac{q-1-\sqrt{q^2-2q+5}}{2}.$$

Um zu zeigen, daß diese Wachstumsvorschrift schlechter als die in Satz 4.16 gewählte ist, zeigen wir

Satz 4.17 *Für alle $q \in \mathbb{N}$, $q \geq 3$ ist*

$$\sqrt[q-1]{\frac{q-1+\sqrt{q^2-2q+5}}{2}} > \sqrt[q]{\frac{q+1+\sqrt{q^2+2q-3}}{2}}.$$

Beweis: Für $q = 3$ ist

$$\sqrt[2-1]{\frac{q-1+\sqrt{q^2-2q+5}}{2}} = \sqrt{1+\sqrt{2}} \geq 1.553$$

und

$$\sqrt[q]{\frac{q+1+\sqrt{q^2+2q-3}}{2}} = \sqrt[3]{2+\sqrt{3}} \leq 1.552.$$

[4]Jeder innere Knoten des Baumes hat q Söhne.

Sei also $q \geq 4$. Wir setzen

$$\alpha := \frac{q+1+\sqrt{q^2+2q-3}}{2}$$

und

$$\beta := \frac{q-1+\sqrt{q^2-2q+5}}{2}.$$

Dann gilt

$$\begin{aligned} & \sqrt[q-1]{\beta} > \sqrt[q]{\alpha} \\ \Longleftrightarrow \quad & \log \sqrt[q-1]{\beta} > \log \sqrt[q]{\alpha} \\ \Longleftrightarrow \quad & \frac{\log \beta}{q-1} > \frac{\log \alpha}{q}. \end{aligned}$$

Weiterhin gilt

$$\begin{aligned} & \log \alpha \\ = \quad & \log \frac{q+1+\sqrt{q^2+2q-3}}{2} \\ < \quad & \log \frac{q+1+\sqrt{q^2+2q+2}}{2} \\ = \quad & \log \frac{q+1+q+1}{2} \\ = \quad & \log(q+1) \end{aligned}$$

und

$$\begin{aligned} & \log \beta \\ = \quad & \log \frac{q-1+\sqrt{q^2-2q+5}}{2} \\ > \quad & \log \frac{q-1+\sqrt{q^2-2q+2}}{2} \\ = \quad & \log \frac{q-1+q-1}{2} \\ = \quad & \log(q-1). \end{aligned}$$

Wir zeigen also

$$\frac{\log(q-1)}{q-1} > \frac{\log(q+1)}{q}.$$

Nach dem Mittelwertsatz gibt es ein $\xi \in]q-1, q+1[$, so daß

$$\frac{\log(q+1) - \log(q-1)}{2} = \log'(\xi) = \frac{1}{\xi}$$

ist. Aus $\frac{1}{\xi} < \frac{1}{q-1}$ und $\log(q-1) \geq 1$ folgern wir

$$\begin{aligned}
& \frac{\log(q+1) - \log(q-1)}{2} < \frac{1}{q-1} \\
\Longrightarrow\quad & 2(q-1)\frac{\log(q+1) - \log(q-1)}{2} < 1 \\
\Longrightarrow\quad & 2(q-1)\frac{\log(q+1) - \log(q-1)}{2} < \log(q-1) \\
\Longrightarrow\quad & (q-1)\log(q+1) < q\log(q-1) \\
\Longrightarrow\quad & \frac{\log(q+1)}{q} < \frac{\log(q-1)}{q-1}.
\end{aligned}$$

$\square$

Wir vermuten, daß die in Satz 4.16 angegebene Schranke für den lim sup der Grenzwert der Folge $(t_i)_{i\in\mathbb{N}}$ ist. Daß dies tatsächlich für alle periodischen Wachstumsvorschriften mit Periode $q-1$ oder q der Fall ist, zeigt folgende Überlegung: Unter $q-1$ aufeinanderfolgenden Vorschriften muß mindestens einmal J auftreten. Sonst wäre der Grad des Baumes an der Wurzel größer als q. Unter q aufeinanderfolgenden Vorschriften sollten maximal zwei J auftreten. Andernfalls ließe sich das in der Mitte stehende J durch ein N ersetzen, was zu kleineren Bäumen führen würde. Diese beiden Fälle haben wir aber schon in Satz 4.17 untersucht. Es ist jedoch nicht klar, warum eine nichtperiodische Wachstumsvorschrift nicht zu kleineren Bäumen führen kann.

4.5 Zusammenfassung

Wir haben in diesem Kapitel die Suche nach einem defekten Blatt in einem beliebigen Baum untersucht. Es zeigte sich, daß die kleinsten Bäume, die eine vorgegebene Anzahl von Fragen erzwingen, eine rekursive Struktur haben. Wie die Tabelle 4.1 jedoch zeigt, ergeben sich die Bäume mit minimaler

Blattanzahl in $\mathcal{M}_{k,q}^{(i+1)}$ im allgemeinen nicht aus den kleinsten Bäumen in $\mathcal{M}_{k,q}^{(i)}$. Andernfalls wäre die aufwendige Konstruktion mittels partieller Ordnung auch gar nicht notwendig gewesen.

Wir haben eine obere Schranke für das Wachstum dieser Bäume angegeben. Diese Schranke ergab sich aus einer konkreten rekursiven Vorschrift für die Bildung von Bäumen aus den Mengen $\mathcal{B}_{2,q}^{(i)}$. Wir vermuten, daß für unendlich viele i diese Bäume in $\mathcal{M}_{2,q}^{(i)}$ liegen, also die Größe t_i haben. Diese Annahme hat sich bei Computerberechnungen nie als falsch erwiesen. Für jedes q ergab sich eine Periode für i, mit der die konstruierten Bäume in $\mathcal{M}_{2,q}^{(i)}$ lagen. Sollte diese Annahme stimmen, so wäre die in Satz 4.16 gezeigte Schranke der exakte Wert für den $\limsup$. Mit Lemma 4.15 oder einer leichten Verallgemeinerung ließe sich dann aber auch zeigen, daß diese Schranke der $\liminf$ ist. Es würde sich also ergeben, daß λ_q existiert und diesen Wert hat.

Als weiteres Indiz für diese Vermutung haben wir gezeigt, daß die regulären Bäume asymptotisch schneller wachsen, als die von uns konstruierten Bäume. Zu jedem $q \in \mathbb{N}$, $q \geq 2$ existiert also ein $t_q \in \mathbb{N}$, so daß für alle $t \geq t_q$ kein q-Baum T mit Suchkomplexität t und $r(t, q)$ Blättern regulär ist.

Kapitel 5

Mehrere Blätter sind defekt

5.1 Einleitung

Wir untersuchen in diesem Kapitel Suchprozesse, bei denen mehrere Blätter defekt sind. Wie in Kapitel 4 bezeichnen wir einen Teilbaum, der mindestens ein defektes Blatt beinhaltet, wieder als kontaminiert. Dem Fragenden stehen wieder alle Teilbäume zur Verfügung. Das Orakel antwortet dann, ob dieser Teilbaum kontaminiert ist.

Im Anschluß untersuchen wir, wie sich die Suchkomplexität erhöht, wenn dem Fragenden die Anzahl der defekten Blätter nicht bekannt ist.

Wir untersuchen zunächst allgemeine (n, q)-Bäume. Danach schränken wir uns aus gutem Grund auf vollständige binäre Bäume ein. Die Verallgemeinerung auf vollständige q-Bäume betrachten wir in Kapitel 6.

5.2 (n, q)-Bäume

Wie in Kapitel 4 betrachten wir zu gegebenen $q, n \in \mathbb{N}$, $q, n \geq 2$ wieder das Maximum der Suchkomplexität über alle (n, q)-Bäume. Für einen Baum T mit d defekten Blättern setzen wir $L(d, T)$ als die Suchkomplexität. Ein spezieller (n, q)-Baum wird in Abbildung 5.1 dargestellt.

Mit diesem Baum können wir die Suchkomplexität bestimmen.

Satz 5.1 *Sei T der Baum aus Abbildung 5.1 mit n Blättern und $d \in \mathbb{N}$ mit $2 \leq d \leq n-1$. Dann gilt $L(d, T) = n - 1$.*

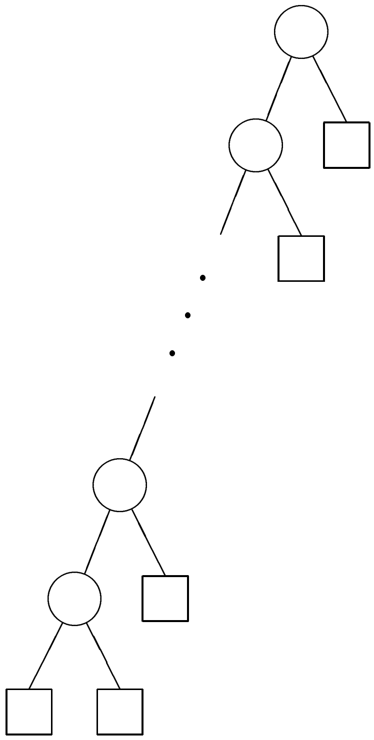

Abbildung 5.1: Extremer Baum für $d \geq 2$ defekte Blätter

Beweis: Der Individualtest benötigt genau $n-1$ Fragen. Wir geben ein Orakel an, das soviele Fragen erzwingt.

Wird ein innerer Knoten gefragt, so ist die Antwort immer „Ja".

Wird von den zwei am tiefsten liegenden Blättern zum ersten Mal eins gefragt, so ist die Antwort „Ja".

Bei allen anderen Blättern wird zunächst $n-1-d$ mal „Nein" und danach „Ja" geantwortet.

Dieses Orakel ist durchführbar. Bei mindestens einem der zwei am tiefsten liegenden Blättern wird „Ja" geantwortet. Die „Ja"-Antworten bei den inneren Knoten sind also korrekt. Außerdem antwortet das Orakel bei maximal $n-1-d$ Blättern „Nein". Es kommen also immer $d+1$ Blätter als defekt in Frage.

Hat der Fragende erst $n-2$ Fragen gestellt, so wurde bei maximal $n-1-d$

Blättern „Nein“ und maximal $n-2-(n-1-d)=d-1$ Blättern „Ja“ geantwortet. Damit sind noch zwei Blätter ungefragt, und genau eins von beiden ist defekt.

$\square$

Diese sehr hohe Suchkomplexität ist für Suchprozesse mit mehreren defekten Elementen nicht üblich. Bilden die zu findenden Elemente zum Beispiel eine Kante in einem Hypergraphen mit Rang r und e Kanten, so liegt die bisher beste bewiesene Schranke für die notwendige Anzahl an Fragen nur $r-1$ über der informationstheoretischen Schranke $\lceil \log_2 e \rceil$. Für Ergebnisse aus diesem Bereich verweisen wir auf [ALT–93, TRI–94, TRI–96, ALT–96].

Wir haben die kombinatorische Suche in Bäumen bei mehreren defekten Blättern theoretisch gelöst. Der Individualtest ist im „worst case“ ein optimaler Algorithmus. Dies ist für eine Anwendung allerdings ein sehr negatives Ergebnis. Unser Beispiel aus Abbildung 5.1 ist jedoch ein ziemlich entarteter Baum. In praktischen Anwendungen wird der Baum häufig regelmäßiger sein. Wir analysieren darum jetzt die vollständigen regulären Bäume.

5.3 Vollständige binäre Bäume

Wir betrachten in diesem Abschnitt vollständige binäre Bäume. Sie haben $2^m = n \geq 2$ Blätter. Wir werden immer annehmen, daß die beiden natürlichen Zahlen m und n in diesem Zusammenhang stehen.

Die Suchkomplexität bei d defekten Blättern ($1 \leq d \leq n$) bezeichnen wir mit $D(d,m)$.

Wir wiederholen zunächst eine allgemeine Überlegung über Suchalgorithmen für den Fall $d \geq 2$. Wenn zwei Knoten x und y kontaminiert sind und x Vorfahre von y ist, so ist die Information über x wertlos. Mit y sind automatisch auch alle Vorfahren von y als kontaminiert bekannt. Ein Algorithmus braucht sich also zu jedem Zeitpunkt von den kontaminierten Knoten nur die ohne kontaminierten Nachfolger zu merken. Solange diese Zahl kleiner als d ist, kann der Algorithmus nur die Blätter ausschließen, die in einem Teilbaum liegen, der mit „Nein“ beantwortet wurde. Kann der Algorithmus auf diese Weise $n-d$ Blätter ausschließen, so sind die d defekten Blätter gefunden.

Gibt es aber irgendwann d kontaminierte Knoten, die paarweise weder Vorfahren noch Nachfolger sind, so bilden sie die Wurzeln von d disjunkten Teilbäumen. Damit liegt in jedem dieser Bäume genau ein defektes Blatt.

Blätter, die in keinem dieser Bäume liegen, sind infolgedessen nicht defekt. Der Algorithmus kann von dem Moment an in jedem dieser Teilbäume einfache binäre Suche anwenden. Wir sagen im folgenden in dieser Situation, daß der Algorithmus die d defekten Blätter **separiert** hat.

5.3.1 Suchalgorithmen

Gegeben seien $d, m \in \mathbb{N}$ mit $1 \leq d \leq 2^m - 1$. Wir setzen wieder $n := 2^m$ und betrachten den vollständigen binären Baum mit n Blättern. Zu jedem $l \in \mathbb{N}$ definieren wir einen Suchalgorithmus $\mathcal{A}_l$:

- Frage in Tiefe l alle 2^l Knoten.
- Für jede Tiefe i von $l+1$ bis m frage von jedem kontaminierten Knoten
  - einen Sohn, falls die d defekten Blätter schon separiert sind,
  - beide Söhne, falls die d defekten Blätter noch nicht separiert sind.
- In Tiefe m wird der letzte Knoten nicht mehr gefragt, falls $m = l$ ist oder in Tiefe $m-1$ erst maximal $d-1$ Knoten kontaminiert waren.

Wir untersuchen jetzt die Laufzeit dieser Algorithmen. Dabei sehen wir auch, daß sie korrekte Suchalgorithmen sind.

Satz 5.2 *Die oben definierten Algorithmen $\mathcal{A}_l$ sind korrekt. Für die Laufzeit $L(\mathcal{A}_l)$ gilt*

$$L(\mathcal{A}_l) \leq \begin{cases} n-1, & \textit{falls } l = m \\ 2^l + d(m-l), & \textit{falls } l < m \textit{ und } d = 2 \\ 2^l + 2(d-1)(m-l) - 1, & \textit{falls } l < m \textit{ und } d > 2. \end{cases}$$

Beweis: Wir unterscheiden die Fälle $l = m$ und $l < m$.

Fall 1: $\boldsymbol{l = m}$. Der Algorithmus fragt in Tiefe m $2^m - 1 = n-1$ Blätter. Der Zustand des letzten Blattes kann dann aus den bisherigen Antworten geschlossen werden.

Fall 2: $\boldsymbol{l < m}$. Der Algorithmus fragt in Tiefe l genau 2^l Fragen. Damit ist der Zustand aller Knoten in Tiefe l bekannt. Sind in Tiefe i maximal $d-1$ Knoten bekannt, so werden in der Tiefe $i+1$ von jedem kontaminierten Knoten alle Söhne gefragt. Somit ist in dieser Tiefe wieder der Zustand aller Knoten bekannt. In Tiefe m braucht jedoch die letzte Frage nicht mehr gestellt werden, da der Zustand des Blattes klar ist. Sind bis dahin erst $d-1$ defekte Blätter gefunden worden, so muß das letzte Blatt defekt sein.

Sind in einer Tiefe i jedoch d Knoten kontaminiert, so braucht von jedem nur noch ein Sohn gefragt werden.

In Tiefe $l+1 \leq i \leq m-1$ werden also entweder maximal $2(d-1)$ oder d Fragen gestellt. In Tiefe m werden entweder maximal $2(d-1)-1$ oder d Fragen gestellt. Für $d=2$ liegt das Maximum damit jeweils bei 2. Für $d>2$ liegt das Maximum bei $2(d-1)$ beziehungsweise $2(d-1)-1$. Daraus folgt die Behauptung.

□

Wir könnten für gegebene d und m jetzt das l suchen, das die Laufzeit von $\mathcal{A}_l$ minimiert. Wir werden im folgenden aber gleich zeigen, welcher Algorithmus $\mathcal{A}_l$ optimal für den Suchprozeß ist. Damit haben wir auch das l gefunden, welches diese Schranke minimiert.

5.3.2 $d = 2$

Wir wenden Satz 5.2 mit von d abhängigem l an.

Satz 5.3 *Sei $m \in \mathbb{N}$. Dann gilt:*

$$D(2,m) \leq \begin{cases} 0, & \textit{falls } m=1 \\ 3, & \textit{falls } m=2 \\ 2m, & \textit{falls } m \geq 3. \end{cases}$$

Beweis: Für $m=1$ ist keine Frage notwendig, da der Baum nur 2 Blätter besitzt. Für $m \geq 2$ setzen wir in Satz 5.2

$$l = \begin{cases} 2, & \text{falls } m=2 \\ 1, & \text{falls } m \geq 3. \end{cases}$$

Damit folgt die Behauptung.

□

Wir zeigen jetzt, daß die in Satz 5.3 gezeigten Schranken die exakten Werte für $D(2,m)$ sind. Wir werden zunächst drei Suchkomplexitäten berechnen, wenn der Fragende über die zwei defekten Blätter schon mehr weiß. Wenn wir den zwei defekte Blätter besitzenden binären Baum an der Wurzel in zwei Teilbäume aufspalten, sind für uns drei Möglichkeiten von Interesse:

1. Jeder Teilbaum besitzt genau ein defektes Blatt,
2. von einem Teilbaum ist bekannt, daß er kontaminiert ist,
3. es ist nur bekannt, daß beide Teilbäume zusammen zwei defekte Blätter haben.

Wir werden in den folgenden Lemmata die Suchkomplexität in Abhängigkeit von dieser zusätzlichen Information berechnen. Fall 3 entspricht dann der eigentlichen Aufgabenstellung.

Wie immer bezeichnen wir die Suchkomplexität bei einem gegebenem Baum T mit $L(T)$. Dabei steht in den Lemmata jeweils, wieviele defekte Blätter der Baum T hat und ob der Fragende schon mehr über die Position der Blätter weiß. $L(T)$ ist dann die noch notwendige Anzahl von Fragen, wenn der Suchalgorithmus dieses Wissen ausnützt.

Lemma 5.4 *Seien $a, b \in \mathbb{N}_0$ und A und B vollständige binäre Bäume mit 2^a und 2^b Blättern. Sei $T = (A, B)$. Weiterhin enthalten sowohl A als auch B genau ein defektes Blatt. Dann ist $L(T) = L(A) + L(B) = a + b$.*

Beweis: „$\leq$“: Der Fragende wendet in beiden Bäumen binäre Suche an.

„$\geq$“: Für die beiden defekten Blätter gibt es $2^a * 2^b$ Möglichkeiten. Nach der informationstheoretischen Schranke sind daher mindestens

$$\lceil \log_2(2^{a+b}) \rceil = a + b$$

Fragen notwendig.

□

Lemma 5.5 *Seien $a, b \in \mathbb{N}_0$, $a \geq 1$ und A und B vollständige binäre Bäume mit 2^a und 2^b Blättern. Sei $T = (A, B)$. Weiterhin enthalte A mindestens ein defektes und T insgesamt zwei defekte Blätter. Dann ist $L(T) \geq a + b + 1$.*

Beweis: Beweis durch Induktion über die Tupel (a, b). Im Fall $(1, 0)$ hat T 3 Blätter. Daher sind nach der informationstheoretischen Schranke 2 Fragen notwendig.

Sei also $(a, b) > (1, 0)$. Damit gilt $a \geq 1$, $b \geq 0$ und $(a, b) \neq (1, 0)$.

Fall 1: Die erste Frage ist die Wurzel von *B*. Das Orakel antwortet „Ja“. Nach Lemma 5.4 sind weitere $a + b$ Fragen notwendig.

Fall 2: Die erste Frage ist im Baum B. Das Orakel antwortet „Nein“. Der Rest von B enthält einen vollständigen binären Baum der Größe 2^{b-1}. Nach Induktionsvoraussetzung sind somit weitere $a + (b - 1) + 1$ Fragen notwendig.

Fall 3: Die erste Frage ist im Baum A und $a = 1$. Das Orakel antwortet „Ja“. Der restliche Baum hat 2^b+1 Blätter, über die nichts bekannt ist. Aus der informationstheoretischen Schranke folgt, daß noch mindestens $b + 1$ Fragen notwendig sind.

Fall 4: Die erste Frage ist im Baum A und $a > 1$. Das Orakel antwortet „Nein“. Der von A übrigbleibende Teil enthält einen vollständigen Baum der Größe 2^{a-1}. Nach Induktionsvoraussetzung sind also noch $(a - 1) + b + 1$ Fragen notwendig.

□

Wir weisen darauf hin, daß diese Schranke bei großem a und kleinem b nicht mehr scharf ist. Ein optimales Orakel wird beide defekten Blätter in den Teilbaum A legen.

Lemma 5.6 *Seien $a, b \in \mathbb{N}_0$, $a \geq 2$ und A und B vollständige binäre Bäume mit 2^a und 2^b Blättern. Sei $T = (A, B)$. T habe zwei defekte Blätter. Dann ist $L(T) \geq a + b + 2$.*

Beweis: Wir führen Induktion über $a + b$.

Ist $a = 2$ und $b = 0$, so sind mindestens 4 Fragen notwendig, da T 5 Blätter hat und

$$\log_2 \binom{5}{2} > 3$$

ist.

Ist $a + b > 2$, so betrachten wir die erste Frage eines beliebigen Suchalgorithmusses.

Fall 1: Die erste Frage ist die Wurzel von Baum B und $b = 0$. Das Orakel antwortet „Nein“. Es bleibt der Baum A übrig, für den nach Induktionsvoraussetzung weitere $(a - 1) + (a - 1) + 2$ Fragen notwendig sind. Daraus folgt die Behauptung, denn es gilt $2a \geq a + 2 = a + b + 2$.

Fall 2: Die erste Frage ist die Wurzel von Baum B und $b \geq 1$. Das Orakel antwortet „Ja“. Nach Lemma 5.5 sind noch $a + b + 1$ Fragen notwendig.

Fall 3: Die erste Frage ist im Baum B. Das Orakel antwortet „Nein“. Nach Induktionsvoraussetzung sind noch $a + (b - 1) + 2$ Fragen notwendig.

Fall 4: Die erste Frage ist die Wurzel von Baum $\boldsymbol{A}$. Das Orakel antwortet „Ja“. Nach Lemma 5.5 sind weitere $a + b + 1$ Fragen notwendig.

Fall 5: Die erste Frage ist im Baum $\boldsymbol{A}$ und $\boldsymbol{a > 2}$. Das Orakel antwortet „Nein“. Nach Induktionsvoraussetzung sind weitere $(a-1)+b+2$ Fragen notwendig.

Fall 6: Die erste Frage ist im Baum $\boldsymbol{A}$, $\boldsymbol{a = 2}$ und $\boldsymbol{b \geq 2}$. Dieser Fall ist symmetrisch zum Fall 3.

Fall 7: Die erste Frage ist im Baum $\boldsymbol{A}$, $\boldsymbol{a = 2}$ und $\boldsymbol{b = 1}$. Der Baum T hat 6 Blätter. Ist die Frage in Tiefe 2, so antwortet das Orakel „Ja“, und es gibt noch 9 Möglichkeiten für beide defekten Blätter. Ist die Frage ein Blatt, so antwortet das Orakel „Nein“, und es gibt noch 10 Möglichkeiten. In beiden Fällen sind also nach der informationstheoretischen Schranke noch mindestens 4 Fragen notwendig.

□

Mit diesen Lemmata können wir die Suchkomplexität für zwei defekte Blätter angeben.

Satz 5.7 *Sei $m \in \mathbb{N}$. Dann gilt:*

$$D(2,m) = \begin{cases} 0, & \textit{falls } m = 1 \\ 3, & \textit{falls } m = 2 \\ 2m, & \textit{falls } m \geq 3. \end{cases}$$

Beweis: In Satz 5.3 haben wir die Werte als obere Schranken nachgewiesen. Wir zeigen jetzt „$\geq$“.

$\boldsymbol{m = 1}$**:** 0 ist eine untere Schranke für jede Suchkomplexität.

$\boldsymbol{m = 2}$**:** Es gilt $\log_2 \binom{4}{2} > 2$. Die Behauptung folgt daher aus der informationstheoretischen Schranke.

$\boldsymbol{m \geq 3}$**:** Dies folgt aus Lemma 5.6 mit $a = b = m - 1$.

□

5.3.3 $2 < d \leq n/4$

Das folgende Lemma und seine beiden Folgerungen sind für die Analyse der Suchkomplexität fundamental. Sie zeigen den großen Unterschied zwischen der Suche nach einem und der nach mehreren Blättern. Bei mehreren defekten Blättern schließt eine positive Antwort nicht automatisch alle Blätter aus anderen Teilbäumen aus.

Lemma 5.8 *Sei T ein vollständiger binärer Baum mit $n = 2^m \geq 2$ Blättern. Der Fragende weiß, daß die Anzahl der defekten Blätter d von T zwischen 1 und 2 liegt. Dann gibt es ein Orakel, so daß nach $2m-1$ Fragen noch $d = 1$ und $d = 2$ möglich ist. Insbesondere kennt der Fragende nach $2m-1$ Fragen noch nicht den Status aller Blätter.*

Beweis: Für $m = 1$ hat der Baum 2 Blätter. Auf die erste Frage wird immer „Ja“ geantwortet. Damit ist ein Blatt als defekt bekannt, und das andere kann sowohl defekt als auch nicht defekt sein.

Sei also $m \geq 2$.

Fall 1: Die erste Frage ist in Tiefe 1. Das Orakel antwortet „Ja“. In den zu diesem Knoten gehörenden Teilbaum A der Größe 2^{m-1} existiert nach Induktionsvoraussetzung ein Orakel, nach dessen Vorschrift geantwortet wird. Auf Fragen in dem restlichen Teilbaum $T - A$ wird „Nein“ geantwortet. Sind in A $2(m-1)-1 = 2m-3$ Fragen gestellt worden, so sind insgesamt mindestens $2m-2$ Fragen gestellt worden, und es ist sowohl $d = 1$ als auch $d = 2$ möglich. Sind bisher erst genau $2m-2$ Fragen gestellt worden, so ist sowohl ein Blatt vom Teilbaum A als auch der komplette Teilbaum $T - A$ unbekannt. Die nächste sinnvolle Frage kann also mit „Nein“ beantwortet werden, und es bleibt immer noch $d = 1$ und $d = 2$ möglich.

Fall 2: Die erste Frage ist ein Knoten a in Tiefe ≥ 2. Das Orakel antwortet „Nein“. Ist die zweite Frage ebenfalls in Tiefe ≥ 2 oder der Vater von a, so ist die Antwort wieder „Nein“. Der restliche Baum hat einen vollständigen Baum der Größe 2^{m-1} als Teilbaum. Daraus folgt die Behauptung. Ist die zweite Frage in Tiefe 1 und kein Vorfahre von a, so ist die Antwort „Ja“, und dieser Teilbaum hat die Größe 2^{m-1}. Damit folgt auch die Behauptung.

□

Wir wenden das Lemma 5.8 häufig an, wenn der Baum ein Teilbaum eines größeren Baumes ist. Durch Fragen in dem restlichen Baum kann es passieren, daß die Anzahl der defekten Blätter in dem Teilbaum auf einmal nicht mehr variabel ist. Damit befassen sich die beiden folgenden Lemmata. Wenn wir dabei sagen, daß sich das Orakel auf eine Anzahl von defekten Blättern festlegt, so meinen wir damit, daß der Fragende diese Zahl ab diesem Zeitpunkt kennt.

Lemma 5.9 *Das Orakel aus Lemma 5.8 habe erst maximal $2m-3$ Fragen beantwortet. Es kann sich dann auf 2 defekte Blätter festlegen und insgesamt*

$2m-1$ *Fragen erzwingen.*

Beweis: Wenn weniger als $2m-3$ Fragen gestellt worden sind, antwortet das Orakel unverändert bis nach der $2m-3$-ten Frage.

Da die Antworten des Orakels bis jetzt auch mit nur einem defekten Blatt konsistent waren, gibt es einen eindeutigen kleinsten kontaminierten Teilbaum. Für diesen gibt es zwei Fälle:

Fall 1: Dieser Teilbaum hat die Größe 1. Ein defektes Blatt ist demnach schon gefunden. Annahme: Es gibt eine Frage, nach der das zweite defekte Blatt entdeckt ist. Diese Frage könnte ein Suchalgorithmus auch dem Orakel aus Lemma 5.8 stellen. Da im Fall $d=2$ nur noch ein Blatt als defekt in Frage kommt, fragen wir dieses Blatt in der $2m-1$-ten Frage. Damit wissen wir, ob $d=2$ oder $d=1$ ist. Dies steht im Widerspruch zu Lemma 5.8.

Fall 2: Dieser Teilbaum hat mehr als ein Element. Es muß außerhalb dieses Teilbaumes noch mindestens ein unidentifiziertes Blatt geben, da sonst das Orakel aus Lemma 5.8 nur $2m-1$ Fragen erzwingen könnte. Folglich sind noch 2 Fragen notwendig.

□

Lemma 5.10 *Das Orakel aus Lemma 5.8 habe* $2m-2$ *Fragen beantwortet. Es gibt dann eine Möglichkeit des Orakels sich auf die Anzahl der defekten Blätter festzulegen, so daß noch mindestens eine weitere Frage notwendig ist.*

Beweis: Wie im vorherigen Beweis betrachten wir die Größe des kleinsten kontaminierten Teilbaumes. Ist diese 1, so muß es noch mindestens 2 unidentifizierte Blätter geben. Das Orakel kann sich also auf zwei defekte Blätter festlegen, und eine weitere Frage ist notwendig.

Hat der Baum mehr als ein Blatt, so legt sich das Orakel auf ein defektes Blatt fest. Um dieses zu identifizieren, ist eine weitere Frage notwendig.

□

Wir leiten aus diesem Orakel ein Orakel für den Fall zwei oder drei defekte Blätter ab.

Lemma 5.11 *Sei* T *ein vollständiger binärer Baum mit* $2^m \geq 4$ *Blättern. Der Fragende weiß, daß die Anzahl der defekten Blätter* d *von* T *zwischen* 2 *und* 3 *liegt. Dann gibt es ein Orakel, so daß nach* $4m-5$ *Fragen noch* $d=2$ *und* $d=3$ *möglich ist. Insbesondere kennt der Fragende nach* $4m-5$ *Fragen noch nicht den Status aller Blätter.*

Bis nach der $4m-7$-ten Antwort kann sich das Orakel auf drei defekte Blätter festlegen und insgesamt $4m-5$ Fragen erzwingen.

Nach der $4m-6$-ten Antwort gibt es für das Orakel noch eine Möglichkeit, sich festzulegen, um eine weitere Frage zu erzwingen.

Beweis: Existenz des Orakels: Das Orakel wendet in beiden Teilbäumen der Größe 2^{m-1} das Orakel aus Lemma 5.8 an. Wird in einem Teilbaum die $2(m-1)$-te Frage gestellt, so legt sich das Orakel in diesem Teilbaum auf ein defektes Blatt fest. Sind erst $4m-5$ Fragen gestellt worden, so sind in mindestens einem Teilbaum maximal $2(m-1)-1$ Fragen gestellt worden. Die Behauptung folgt damit aus Lemma 5.8.

Festlegung auf drei defekte Blätter: Sind bisher weniger als $4m-7$ Fragen beantwortet worden, so antwortet das Orakel zunächst unverändert. Sind nach $4m-7$ Fragen in einem Teilbaum bereits $2(m-1)$ Fragen gestellt worden, so sind im anderen erst $2(m-1)-3$ Fragen gestellt worden. Nach Lemma 5.9 kann sich das Orakel hier auf 2 Blätter festlegen und in diesem Teilbaum $2(m-1)-1$ Fragen erzwingen.

Sind stattdessen in einem Teilbaum T_1 erst $2(m-1)-1$ Fragen gestellt worden, so sind im anderen Teilbaum T_2 bereits $2(m-1)-2$ Fragen gestellt worden. Ist die nächste Frage in T_2, so antwortet das Orakel wie in Lemma 5.8 und kann sich noch aussuchen, ob das dritte defekte Blatt in T_1 oder T_2 ist. Ist die nächste Frage in T_1, so legt sich das Orakel nach Lemma 5.10 für die Anzahl der Blätter in T_1 so fest, daß in T_2 eine weitere Frage notwendig ist.

Festlegung nach $4m-6$ Fragen: Sind nach $4m-6$ Fragen in einem Baum bereits $2(m-1)$ Fragen gestellt worden, so sind im anderen Baum erst $2(m-1)-2$ Fragen gestellt worden. Die Behauptung folgt daher aus Lemma 5.10. Sind in beiden Teilbäumen $2(m-1)-1$ Fragen gestellt worden, legt sich das Orakel auf drei defekte Blätter fest. Es ist dann noch eine weitere Frage notwendig, um festzustellen, in welchem Teilbaum dieses Blatt ist.

□

Dieser Beweis erklärt nochmals die vor Lemma 5.9 genannte Notwendigkeit des Orakels, sich festzulegen. Der Fragende kann dies zwar nicht verlangen, aber ein übergeordnetes Orakel kann es.

Als nächstes brauchen wir eine Monotonieeigenschaft der Suchkomplexität. Diese Eigenschaft gilt dabei ganz allgemein für Gruppentests.

Lemma 5.12 *Sei $(S, \mathcal{F})$ ein Suchprozeß und $d \in \mathbb{N}$ mit $1 \leq d \leq |S| - 2$. Sei L die Suchkomplexität um d Elemente in S zu finden und L' die Suchkomplexität um $d+1$ Elemente zu finden. Dann gilt $L \leq L'$.*

Beweis: Sei $\mathcal{O}$ ein optimales Orakel für d defekte Elemente. Wir konstruieren ein Orakel $\mathcal{P}$ für $d+1$ defekte Elemente, das mindestens soviele Fragen wie $\mathcal{O}$ erzwingt: Wird eine Menge $A \subseteq S$ gefragt, so gibt es folgende Möglichkeiten:

1. $\mathcal{O}$ würde „Ja“ antworten: $\mathcal{P}$ antwortet „Ja“.
2. $\mathcal{O}$ würde „Nein“ antworten, und bisher hat $\mathcal{P}$ nur „Ja“ geantwortet: $\mathcal{P}$ antwortet „Ja“. Es wählt aus dieser Testmenge ein Element y aus.
3. $\mathcal{O}$ würde schon mindestens zum zweiten Mal „Nein“ sagen: Gilt $y \in A$, so antwortet $\mathcal{P}$ „Ja“, sonst „Nein“.

Das Orakel $\mathcal{P}$ erzeugt die d defekten Elemente, die $\mathcal{O}$ erzeugt und zusätzlich das Element y. Es werden also $d+1$ defekte Elemente erzeugt.

Würde auf die Fragen des Suchalgorithmusses das Orakel $\mathcal{O}$ mindestens einmal „Nein“ antworten, so legt das Orakel $\mathcal{P}$ das zusätzliche defekte Element y in eine beim Orakel $\mathcal{O}$ reine Teilmenge. Um die d anderen defekten Elemente zu finden, sind daher genauso viele Fragen notwendig wie bei dem $\mathcal{O}$.

Würde $\mathcal{O}$ jedoch nur „Ja“ antworten, so antwortet auch $\mathcal{P}$ nur mit „Ja“. Damit wird nie ein Element der Menge S als nicht defekt ausgeschlossen. Selbst nachdem die d defekten Elemente gefunden worden sind, ist die Lage des letzten defekten Elementes noch unbekannt, da $d+1 < |S|$ ist.

□

Wir berechnen jetzt die Suchkomplexität für $3 \leq d \leq n/4$.

Satz 5.13 *Sei $d, m \in \mathbb{N}$, $n = 2^m$ und $3 \leq d \leq n/4$. Dann ist*

$$D(d, m) = 2^l + 2(d-1)(m-l) - 1 \leq n - 3,$$

wobei $l = \lceil \log_2 d \rceil + 1$ ist.

Beweis: „$\leq$“: Da $d \leq 2^{m-2}$ ist, ist $l < m$. Die obere Schranke folgt also aus Satz 5.2.

„$\geq$“: Wir konstruieren ein Orakel, das soviele Fragen erzwingt. Für die Konstruktion des Orakels benötigen wir einige Ungleichungen. Aus $3 \leq d \leq \frac{n}{4}$

folgt $3 \leq l \leq m-1$. Aus $1+log_2 d \leq l < 2+\log_2 d$ folgt $2d \leq 2^l < 4d$. Damit sind $a := 2^{l-1} - d + 1$ und $b := d - 1 - 2^{l-2}$ nicht negativ. Außerdem gilt $a + b = 2^{l-2}$ und $a + 2b = d - 1$. In den ersten a Teilbäumen, die zu den Knoten auf Tiefe $l-2$ gehören, wird das Orakel aus Lemma 5.8 angewendet und in den anderen b Teilbäumen das aus Lemma 5.11. Wird in solch einem Teilbaum die $2(m-l+2)$-te bzw. die $4(m-l+2)-4$-te Frage gestellt, so entscheidet sich das Orakel zunächst immer für die geringere Anzahl an defekten Blättern. Erst im vorletzten Fall entscheidet es so, daß sich die Suchkomplexität im letzten Teilbaum nur um 1 verringert (Lemma 5.9, 5.10 und 5.11).

Aus $a + 2b = d - 1$ folgt, daß dieses Orakel mit der Vorgabe „d defekte Blätter“ konsistent ist und der Fragende solange keine Information über die Anzahl der defekten Blätter pro Teilbaum hat, bis nur noch ein Teilbaum übrig ist. Die Anzahl der Fragen, die das Orakel erzwingt, sind

$$a * 2(m-l+2) + b * (4(m-l+2)-4) - 1 = 2(m-l)(d-1) + 2^l - 1.$$

Setzen wir $d = n/4$ ein, so ist dieser Wert $n-3$. Aus Lemma 5.12 folgt daher, daß alle Werte $\leq n-3$ sind.

□

5.3.4 $n/4 < d \leq n-1$

Wir zeigen, daß in diesem Fall der Individualtest optimal ist.

Satz 5.14 *Seien $m, d \in \mathbb{N}$, $d \geq 3$, $n = 2^m$ und $n/4 < d \leq n-1$. Dann ist $D(d,m) = n-1$.*

Beweis: „$\leq$“: Der Individualtest erreicht diesen Wert.

„$\geq$“: Für $n = 1$ oder $n = 2$ ist die Behauptung klar. Sei also $n \geq 4$. Nach Lemma 5.12 reicht es, $d = n/4 + 1$ zu betrachten. Auf Fragen, bei denen der Knoten eine Tiefe $\leq m-2$ hat, antwortet das Orakel „Ja“. Bei Fragen in diesen Teilbäumen der Größe 4 antwortet das Orakel wie es Lemma 5.8 vorgibt. Bei der 4-ten Frage in so einem Teilbaum legt sich das Orakel auf ein defektes Blatt fest. Damit sind pro Teilbaum 4 Fragen notwendig. Nur beim letzten Teilbaum reichen 3 Fragen. Das sind insgesamt $n-1$ Fragen.

□

Für die kombinatorische Suche in vollständigen Bäumen ist die Fragestellung „Wann ist die Einzelabfrage optimal?“ vollständig gelöst. Dies ist bei $m \geq 3$ für $n/4 < d \leq n-1$ der Fall.

Die Suchkomplexität $n-2$ tritt interessanterweise für $m \geq 4$ nie auf.

Tabelle 5.1 gibt für die ersten d und m die Werte von $D(d,m)$ an.

m \ d	1	2	3	4	5	6	7	8	9	10
1	1									
2	2	3	3							
3	3	6	7	7	7	7	7			
4	4	8	11	13	15	15	15	15	15	15
5	5	10	15	19	23	25	27	29	31	31
6	6	12	19	25	31	35	39	43	47	49
7	7	14	23	31	39	45	51	57	63	67
8	8	16	27	37	47	55	63	71	79	85
9	9	18	31	43	55	65	75	85	95	103
10	10	20	35	49	63	75	87	99	111	121
11	11	22	39	55	71	85	99	113	127	139
12	12	24	43	61	79	95	111	127	143	157
13	13	26	47	67	87	105	123	141	159	175
14	14	28	51	73	95	115	135	155	175	193
15	15	30	55	79	103	125	147	169	191	211
16	16	32	59	85	111	135	159	183	207	229
17	17	34	63	91	119	145	171	197	223	247
18	18	36	67	97	127	155	183	211	239	265
19	19	38	71	103	135	165	195	225	255	283
20	20	40	75	109	143	175	207	239	271	301

Tabelle 5.1: $D(d,m)$ für vollständige binäre Bäume

5.4 Informationstheoretische Schranke

Wir untersuchen in diesem Abschnitt, wie nahe der Wert von $D(d, m)$ an der informationstheoretischen Schranke

$$I(d, m) := \left\lceil \log_2 \binom{n}{d} \right\rceil$$

liegt. Wir erinnern nochmal daran, daß wir immer $n = 2^m$ annehmen.

5.4.1 $d = 1$

In diesem Fall stimmen die Suchkomplexität und die informationstheoretische Schranke überein. Es gilt $D(1, m) = \lceil \log_2 \binom{n}{1} \rceil = I(1, m)$.

5.4.2 $d = 2$

Es gilt $D(2, 1) = 0$, $D(2, 2) = 3$ und für $m \geq 3$ ist $D(2, m) = 2m$. Für die informationstheoretische Schranke gilt

$$I(2, 1) = \left\lceil \log_2 \binom{2}{2} \right\rceil = \lceil \log_2 1 \rceil = 0,$$

$$I(2, 2) = \left\lceil \log_2 \binom{4}{2} \right\rceil = \lceil \log_2 6 \rceil = 3,$$

und für $m \geq 3$ ist

$$\begin{aligned} I(2, m) &= \left\lceil \log_2 \binom{n}{2} \right\rceil \\ &= \left\lceil \log_2 \frac{n(n-1)}{2} \right\rceil \\ &= \lceil \log_2 n + \log_2 (n-1) - 1 \rceil \\ &= 2m - 1. \end{aligned}$$

Für $m \geq 3$ beträgt der Unterschied also genau 1.

5.4.3 $3 \leq d \leq n/4$

Es gilt

$$\left(\frac{n}{d}\right)^d \leq \binom{n}{d} \leq (n-d+1)^d \leq n^d.$$

Also gilt

$$d(m - \log_2 d) \leq I(d,m) \leq dm.$$

Andererseits wächst $D(d,m)$ asymptotisch wie $2(d-1)m$. Damit wächst die Differenz asymptotisch mit $(d-2)m$.

5.4.4 $d > n/4$

Zur Abschätzung verwenden wir die Stirlingsche Formel.

Satz 5.15 (Stirlingsche Formel) *Sei $n \in \mathbb{N}$. Dann gibt es ein $\epsilon \in]0,1[$, so daß*

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{\epsilon}{12n}}$$

gilt.

Wir schätzen

$$\log_2 \binom{n}{d} = \log_2 n! - \log_2 d! - \log_2 (n-d)!$$

ab. Aus der Stirlingschen Formel folgt

$$\begin{aligned} \log_2 n! &> \log_2 \sqrt{2\pi n} + \log_2 \left(\frac{n}{e}\right)^n \\ &= \log_2 \sqrt{2\pi} + \frac{m}{2} + n\left(m - \log_2 e\right), \end{aligned}$$

$$\begin{aligned} \log_2 d! &< \log_2 \sqrt{2\pi d} + \log_2 \left(\frac{d}{e}\right)^d e^{\frac{1}{12d}} \\ &= \log_2 \sqrt{2\pi} + \frac{\log_2 d}{2} + d\left(\log_2 d - \log_2 e\right) + \frac{1}{12d}\log_2 e \end{aligned}$$

und

$$\begin{aligned}
\log_2(n-d)! &< \log_2\sqrt{2\pi(n-d)} + \log_2\left(\frac{n-d}{e}\right)^{n-d} e^{\frac{1}{12(n-d)}} \\
&= \log_2\sqrt{2\pi} + \frac{\log_2(n-d)}{2} + (n-d)\left(\log_2(n-d) - \log_2 e\right) \\
&\quad + \frac{1}{12(n-d)}\log_2 e.
\end{aligned}$$

Daher ist

$$\begin{aligned}
&\log_2\binom{n}{d} \\
> \; & -\log_2\sqrt{2\pi} + \frac{m - \log_2 d - \log_2(n-d)}{2} + nm - d\log_2 d \\
& -(n-d)\log_2(n-d) - \left(\frac{1}{12d} + \frac{1}{12(n-d)}\right)\log_2 e. \qquad (5.1)
\end{aligned}$$

Da die Suchkomplexität immer $n-1$ ist, ist die beste informationstheoretische Schranke für $d = n/2$ zu erwarten. Aus der Gleichung (5.1) folgt

$$\begin{aligned}
\log_2\binom{n}{n/2} &> -\log_2\sqrt{2\pi} + \frac{2-m}{2} + n - \frac{1}{3n}\log_2 e \\
&\geq n - \frac{1}{2}\log_2 n - \frac{0.49}{n} - 0.33.
\end{aligned}$$

Die Differenz zwischen der informationstheoretischen Schranke und der Suchkomplexität $D(d, m)$ wächst also für $d = n/2$ mit $\Omega(m)$.

5.4.5 Bewertung

Tabelle 5.2 faßt diese Differenzen noch einmal zusammen.

5.5 Kompetitive Suche

Wir betrachten in diesem Abschnitt die kompetitive Suche in vollständigen binären Bäumen. Wir haben die Fragestellung bereits im Kapitel 3 erläutert. Wir übertragen an dieser Stelle darum nur die Definition aus [DUH–93] für

d	$D(d,m)$	$I(d,m)$	Differenz
1	m	m	0
$2,\ m \geq 3$	$2m$	$2m-1$	1
$3 \leq d \leq n/4$	$\approx 2(d-1)m$	$\approx dm$	$\approx (d-2)m$
$d = n/2$	$n-1$	$n - \frac{1}{2}m$ $-\frac{0.48}{n} - 1.65$	$\approx 0.5m$

Tabelle 5.2: Bewertung der informationstheoretischen Schranke

die Suche in Bäumen. Wir erinnern daran, daß ein kompetitiver Algorithmus die Anzahl der defekten Elemente nicht kennt. Er muß so lange Fragen stellen, bis er den Zustand aller Blätter kennt.

Definition 5.16 *Sei $\mathcal{A}$ ein kompetitiver Suchalgorithmus für die Suche in vollständigen binären Bäumen. Wir setzen $D_{\mathcal{A}}(d|m)$ als die Suchlänge von $\mathcal{A}$ bei d defekten Blättern im vollständigen binären Baum mit $n = 2^m$ Blättern.*

$\mathcal{A}$ heißt c-kompetitiv mit $c \in \mathbb{R}$, falls es eine Konstante $a \in \mathbb{R}$ gibt, so daß für alle $m \in \mathbb{N}$ und $d \in \mathbb{N}_0$ mit $0 \leq d < n$ gilt

$$D_{\mathcal{A}}(d|m) \leq cD(d,m) + a.$$

Wie wir in Kapitel 3 schon bemerkt haben, existiert bei allgemeinen Gruppentests zu jedem $\epsilon > 0$ ein $(1.5 + \epsilon)$-kompetitiver Algorithmus. Ob es Algorithmen mit noch kleinerem kompetitiven Faktor gibt, ist bisher nicht bekannt.

Wir geben nun einen kompetitiven Algorithmus $\mathcal{A}$ für die Suche in vollständigen binären Bäumen an. Er lautet:

1. Frage die Wurzel.
2. Für jede Tiefe von 1 bis zur maximalen Tiefe frage jeweils alle Söhne von kontaminierten Knoten.

Der Algorithmus fragt in jeder Tiefe i maximal $\min\{2^i, 2d\}$ Knoten. Definieren wir für einen vollständigen binären Baum mit $n = 2^m$ Blättern wieder $l = \lceil \log_2 d \rceil + 1$, so ist $l \leq m+1$, und damit gilt

$$D_{\mathcal{A}}(d|m) \ = \ \sum_{i=0}^{m} \min\{2^i, 2d\}$$

$$
\begin{aligned}
&= \sum_{i=0}^{l-1} 2^i + \sum_{i=l}^{m} 2d \\
&= 2^l - 1 + 2d(m-l+1).
\end{aligned}
$$

Wir vergleichen diese Laufzeit mit den oben berechneten Suchkomplexitäten für den Fall, daß d bekannt ist. Für $d = 1$ gilt $D(1, m) = m$. Also ist

$$D_{\mathcal{A}}(1|m) = 2m + 1 = 2D(1, m) + 1.$$

Für $d = 2$ und $m \geq 3$ ist $D(2, m) = 2m$ und damit

$$D_{\mathcal{A}}(2|m) = 4m - 1 = 2D(2, m) - 1.$$

Für $3 \leq d \leq n/4$ gilt $D(d, m) = 2^l + 2(d-1)(m-l) - 1$. Also gilt

$$
\begin{aligned}
& D_{\mathcal{A}}(d|m) \\
=\ & 2^l - 1 + 2d(m-l+1) \\
=\ & D(d, m) + 2(m-l) + 2d \\
\leq\ & D(d, m) + 2(m-l) + 2^l \\
\leq\ & 2D(d, m) + 1.
\end{aligned}
$$

Der Baum hat insgesamt $2n - 1$ Knoten. $\mathcal{A}$ fragt also maximal $2n - 1$ Knoten. Für $n/4 < d < n$ ist $D(d, m) = n - 1$. Damit haben wir auch in diesem Fall

$$D_{\mathcal{A}}(d|m) \leq 2n - 1 = 2D(d, m) + 1.$$

Der Algorithmus $\mathcal{A}$ ist demnach 2-kompetitiv. Wir zeigen, daß es bei vollständigen binären Bäumen keinen besseren kompetitiven Faktor geben kann.

Satz 5.17 *Für jeden c-kompetitiven Algorithmus zum Suchproblem in vollständigen binären Bäumen gilt* $c \geq 2$.

Beweis: Es gilt $D(1, m) = m$. Wir wenden das Orakel aus Lemma 5.8 an. Nach $2m - 1$ Fragen ist für jeden kompetitiven Algorithmus $\mathcal{B}$ noch $d = 1$ und $d = 2$ möglich. Daher ist

$$D_{\mathcal{B}}(1|n) \geq 2m = 2D(1, m).$$

□

Der Grund, daß es keine besseren als 2-kompetitive Algorithmen geben kann, liegt also an der Suchkomplexität im Fall $d = 1$. Es ist jedoch interessant, wie gut kompetitive Algorithmen bei größeren d werden können. Ist $3 \leq d \leq n/4$, so antworten die Orakel aus Satz 5.13 auf die Weise, daß der Fragende immer nur $d-1$ disjunkte kontaminierte Teilbäume hat. Wenn der Fragende die Anzahl $d \geq 2$ der defekten Blätter nicht kennt, können wir das entsprechende Orakel aus Satz 5.13 für $d+1$ defekte Blätter anwenden. In diesem Fall legen sich die Teilorakel aber immer auf ein defektes Blatt fest. Damit spart der Fragende nicht einmal im letzten Teilbaum eine Frage. Es gilt also der folgende Satz:

Satz 5.18 *Sei $\mathcal{B}$ ein beliebiger kompetitiver Algorithmus und $2 \leq d < n/4$. Dann gilt*

$$\begin{aligned} & D_{\mathcal{B}}(d|m) \\ \geq\ & D(d+1,m)+1 \\ =\ & 2^l + 2d(m-l), \end{aligned}$$

wobei wir $l = \lceil \log_2 d \rceil + 1$ oder $l = \lceil \log_2 (d+1) \rceil + 1$ setzen können.

Beweis: Es ist nur noch zu zeigen, daß die beiden Möglichkeiten für l keine unterschiedlichen Werte ergeben. Der Wert von $D(d+1,m)$ folgt dann aus Satz 5.13.

Ist d keine Zweierpotenz, so sind beide Wahlmöglichkeiten gleich. Ist $d = 2^\kappa$, $\kappa \in \mathbb{N}$, so gilt mit $l = \lceil \log_2 d \rceil + 1 = \kappa + 1$ und $l' = \lceil \log_2 (d+1) \rceil + 1 = \kappa + 2$:

$$\begin{aligned} & 2^l + 2d(m-l) \\ =\ & 2^{\kappa+1} + 2^{\kappa+1}(m-\kappa-1) \\ =\ & 2^{\kappa+2} + 2^{\kappa+1}(m-\kappa-2) \\ =\ & 2^{l'} + 2d(m-l'). \end{aligned}$$

□

Diese untere Schranke für die Laufzeit von Suchalgorithmen, die die Anzahl der defekten Blätter nicht kennen, hat Damaschke bereits in [DAM–95] bewiesen. Weiterhin gibt Damaschke zu jedem d einen Suchalgorithmus an, der, sollte der Baum tatsächlich d defekte Blätter besitzen, diese Schranke erreicht.

In Tabelle 5.3 vergleichen wir den Algorithmus $\mathcal{A}$ mit den theoretischen unteren Schranken für kompetitive Algorithmen.

d	Untere Schranke	Algorithmus $\mathcal{A}$	Differenz
0	1	1	0
1	$2m$	$2m+1$	1
$2 \leq d < n/4$	$2d(m-l)+2^l$	$2d(m-l+1)+2^l-1$	$2d-1$
	$l = \lceil \log_2 d \rceil + 1$	$l = \lceil \log_2 d \rceil + 1$	
$d = n-1$	n	$2n-1$	$n-1$

Tabelle 5.3: Bewertung von $\mathcal{A}$

Als letztes untersuchen wir noch, wie sich das Verhältnis der Laufzeit des kompetitiven Algorithmusses zu $D(d,m)$ bei konstantem d entwickelt. Für $d=1$ gilt, wie oben bereits gezeigt,

$$\lim_{m\to\infty} \frac{D_{\mathcal{A}}(1|m)}{D(1,m)} = 2.$$

Für $d=2$ ergibt sich

$$\lim_{m\to\infty} \frac{D_{\mathcal{A}}(2|m)}{D(2,m)} = \lim_{m\to\infty} \frac{4m-1}{2m} = 2.$$

Ist $d \geq 3$, so gilt

$$\lim_{m\to\infty} \frac{D_{\mathcal{A}}(d|m)}{D(d,m)} = \lim_{m\to\infty} \frac{2dm-2dl+2d+2^l-1}{(2d-2)m-2dl+2l+2^l-1} = \frac{d}{d-1}.$$

Das Verhältnis der Suchlängen konvergiert für $d \to \infty$ demnach gegen 1.

5.6 Zusammenfassung

Wir haben in diesem Kapitel die Suchkomplexität bei mehr als einem defekten Blatt untersucht. Es zeigte sich, daß bei allgemeinen Bäumen der Individualtest optimal ist. Der Grund für dieses extreme Resultat liegt darin, daß ein entdecktes defektes Blatt nicht einfach bei den weiteren Tests ignoriert werden kann. Der Fragende kann nicht, wie bei allgemeinen Gruppentests, eine Testmenge beliebig zusammenstellen.

Durch die Einschränkung auf vollständige binäre Bäume haben wir in diesem wichtigen Spezialfall optimale Suchalgorithmen angeben können. Bei gegebener Blattanzahl $n = 2^m \geq 16$ ergeben sich für die Anzahl d der defekten Blätter vier verschiedene Bereiche.

Ist $d = 1$, so ist dies der klassische Fall einer binären Suche. Ein optimaler Algorithmus halbiert mit jeder Frage die Anzahl der noch als defekt in Frage kommenden Blätter.

Ist $d = 2$, so verdoppelte sich die Suchkomplexität. Bei optimalen Suchalgorithmen werden die beiden defekten Blätter am Ende verschiedene Vorfahren in Tiefe 1 haben.

Für $3 \leq d \leq \frac{n}{4}$ ist diese Aufteilung der Blätter nicht mehr möglich. Die Suchkomplexität ist daher geringer als dm.

Ist schließlich $d > \frac{n}{4}$, so ist der Individualtest optimal. Anders als bei allgemeinen Gruppentests ist die Fragestellung, ab wann der Individualtest optimal ist, damit vollständig beantwortet.

Anschließend haben wir einen 2-kompetitiven Suchalgorithmus vorgestellt. Wir konnten zeigen, daß dies optimal ist, es also keinen c-kompetitiven Algorithmus für $c < 2$ geben kann. Dies beruht auf der Tatsache, daß bei nur einem defekten Blatt in einem Baum mit 2^m Blättern jeder kompetitive Algorithmus mindestens $2m$ Fragen stellen muß. Eine solche untere Schranke für den besten möglichen kompetitiven Faktor ist bei allgemeinen Gruppentests bisher nicht bekannt.

Als letztes haben wir die Laufzeit des kompetitiven Algorithmusses für den Fall untersucht, daß der Baum eine Mindestanzahl $d \geq 2$ an defekten Blättern enthält. In diesem Fall verbessert sich sein kompetitiver Faktor auf den Wert $\frac{d}{d-1}$. Für $d = 2$ beträgt er also immer noch 2.

Kapitel 6

Vollständige q-Bäume

6.1 Einleitung

Wir betrachten in diesem Kapitel vollständige reguläre q-Bäume. Wie im Kapitel 5 bestimmen wir die Suchkomplexität bei d defekten Blättern. Es wird also darum gehen, die dort für $q = 2$ vorgestellten Methoden zu verallgemeinern. Dies gelingt bei den Suchalgorithmen, also den oberen Schranken, gut. Bei den unteren Schranken ist jedoch wesentlich mehr Arbeit erforderlich. Der Grund dafür liegt in der Tatsache, daß ein binärer Baum automatisch immer regulär ist, jeder innere Knoten also genau 2 Söhne hat. Ein q-regulärer Baum wird im Laufe eines Suchalgorithmusses aber nicht regulär bleiben. Ein Induktionsbeweis wird daher auch nichtreguläre Bäume betrachten müssen.

In diesem Kapitel seien $d, q, m \in \mathbb{N}$, $q \geq 2$, $n = q^m$ und $1 \leq d \leq n$. Mit $D(d, q, m)$ bezeichnen wir dann die Suchkomplexität, um in dem vollständigen regulären (n, q)-Baum d defekte Blätter zu finden. Da $D(n, q, m) = 0$ ist, nehmen wir im weiteren an, daß $d \leq n - 1$ ist.

6.2 Suchalgorithmen

Wie in Kapitel 5 definieren wir wieder eine Menge von Algorithmen, die sich durch einem Parameter l unterscheiden. Anschließend wählen wir l in Abhängigkeit von d und m. Dadurch erhalten wir wieder eine obere Schranke für die Suchkomplexität.

Sei also $l \in \mathbb{N}$, $1 \leq l \leq m$. Der Algorithmus $\mathcal{A}_l$ ist dann der folgende:

- Frage in Tiefe l alle q^l Knoten.
- Für jede Tiefe i von $l+1$ bis m frage von jedem kontaminierten Knoten
  - $q-1$ Söhne, falls die d defekten Knoten schon separiert sind,
  - alle Söhne, falls die d defekten Knoten noch nicht separiert sind.
- In Tiefe l brauchen im Fall $d = 1$ nur $q^l - 1$ Knoten gefragt werden.
- In Tiefe m braucht der letzte Knoten nicht mehr gefragt werden, falls $m = l$ ist oder in Tiefe $m-1$ erst maximal $d-1$ Knoten kontaminiert sind.

Für diese Verallgemeinerung des Suchalgorithmusses aus Kapitel 5 gilt erwartungsgemäß ein entsprechender Satz.

Satz 6.1 *Die oben definierten Algorithmen $\mathcal{A}_l$ sind korrekt. Für die Laufzeit $L(\mathcal{A}_l)$ gilt*

$$L(\mathcal{A}_l) \leq \begin{cases} n-1, & \text{falls } l = m \\ q^l + d(q-1)(m-l) - 1, & \text{falls } l < m \text{ und } d = 1 \\ q^l + d(q-1)(m-l), & \text{falls } l < m \text{ und } d \leq q \\ q^l + (d-1)q(m-l) - 1, & \text{falls } l < m \text{ und } d > q. \end{cases}$$

Beweis: Der Beweis ist die Verallgemeinerung vom Beweis von Satz 5.2. Der Algorithmus $\mathcal{A}_m$ ist wieder der Individualtest. Für $l < m$ fragt der Algorithmus in jeder Ebene nur dann nicht alle Söhne eines kontaminierten Knotens, wenn der Status des letzten Sohns bekannt ist, also entweder die d defekten Blätter in disjunkten Teilbäumen liegen oder es das letzte Blatt in der Tiefe m ist. Der Algorithmus ist wieder korrekt und stellt

$$q^l + \sum_{i=l+1}^{m-1} \max\{d(q-1), (d-1)q\} + \max\{d(q-1), (d-1)q - 1\}$$

Fragen. Eine Fallunterscheidung $d \leq q$ und $d > q$ liefert die Behauptung.

$\square$

Wie wir sehen, wird das Maximum für $d \leq q$ angenommen, wenn ab Tiefe l jeweils d kontaminierte Knoten vorhanden sind. Bei $d > q$ sollten statt dessen immer nur $d-1$ Knoten kontaminiert sein. Diese Idee werden wir später beim Beweis der unteren Schranke verwenden.

Wir werden in Satz 6.1 ein von n abhängiges l einsetzen und damit eine obere Schranken für $D(d,q,m)$ erhalten. Wir bemerken nochmal, daß wir immer $d < n$ voraussetzen. Falls $m = 1$ ist, ist also $d < q$.

Korollar 6.2 *Es gilt*

$$D(d,q,m) \leq \begin{cases} (q-1)m, & \textit{falls } d = 1 \\ n-1, & \textit{falls } 2 \leq d < q \textit{ und } m = 1 \\ n-1, & \textit{falls } d = q \textit{ und } m = 2 \\ q + d(q-1)(m-1), & \textit{falls } 2 \leq d \leq q \\ q^l + (d-1)q(m-l) - 1 & \\ \quad \leq n - q - 1, & \textit{falls } q < d \leq (q-1)q^{m-2} \\ n-1, & \textit{falls } (q-1)q^{m-2} < d, \end{cases}$$

wobei $l = \left\lceil \log_q \frac{d}{q-1} \right\rceil + 1$ *ist.*

Beweis: Im Fall $1 \leq d \leq q$ setzen wir in Satz 6.1 $l = 1$ ein. Im Fall $q < d \leq (q-1)q^{m-2}$ nehmen wir das im Satz angegebene l. Die Schranke $n-1$ für die Suchkomplexität gilt immer, da sie die Laufzeit vom Individualtest ist.

Um im Fall $q < d \leq (q-1)q^{m-2}$

$$q^l + (d-1)q(m-l) - 1 \leq n - q - 1$$

zu zeigen, verwenden wir wieder Lemma 5.12. Wir setzen also $d = (q-1)q^{m-2}$. Dann ist $l = m-1$, und es gilt

$$\begin{aligned} & q^l + (d-1)q(m-l) - 1 \\ = \; & q^{m-1} + ((q-1)q^{m-2} - 1)q - 1 \\ = \; & q^m - q - 1 \\ = \; & n - q - 1. \end{aligned}$$

$\square$

Im Korollar 6.2 sind die Fälle nicht immer disjunkt. Es gelten dann beide Schranken, wobei wir natürlich nur die kleinere als Suchkomplexität beweisen werden können.

6.3 Untere Schranken

Wir geben für die verschiedenen Fälle Orakel an. Zusammen mit Korollar 6.2 ergibt sich damit die Suchkomplexität. Bei den Orakeln werden wir auf Fragen, deren Antwort vorher feststeht, nicht eingehen. Wird zum Beispiel die Wurzel gefragt, so ist die Antwort „Ja“, da der Baum kontaminiert ist.

6.3.1 $d = 1$

Satz 6.3 *Es gilt* $D(1, q, m) = (q - 1)m$.

Beweis: Nach Korollar 6.2 ist $(q-1)m$ eine obere Schranke für $D(1, q, m)$. Für die untere Schranke führen wir vollständige Induktion über m.

$\boldsymbol{m = 1}$**:** Bei den q Blättern ist nur der Individualtest möglich.

$\boldsymbol{m \to m + 1}$**:** Das Orakel antwortet auf die ersten $q - 1$ Fragen „Nein“. Vom Baum mit q^{m+1} Blättern bleibt dann mindestens ein Baum mit q^m übrig. Damit sind insgesamt $(q - 1) + (q - 1)m = (q - 1)(m + 1)$ Fragen notwendig.

□

6.3.2 $2 \leq d \leq q$

Wir betrachten zunächst einige Spezialfälle. Der allgemeine Fall läßt sich dann durch vollständige Induktion lösen, wobei die Spezialfälle den Induktionsanfang bilden.

Wenn der Algorithmus einen Knoten x fragt, so können wir ohne Einschränkung annehmen, daß x nicht die Wurzel ist. Der Knoten x hat also in einem q-regulären Baum noch $q-1$ Geschwister. Von diesen hat der Algorithmus eventuell schon vorher einige Knoten gefragt. Wir setzen dann $c(x)$ als die Anzahl der gefragten Geschwister plus 1. Wir zählen also die Frage nach x mit. Es gilt damit bei einem gefragten Knoten x immer $c(x) \in \{1, \ldots, q\}$. $c(x)$ bekommt definitionsgemäß erst dann einen Wert, wenn der Algorithmus x fragt.

Zunächst betrachten wir den Fall, daß $d < q$ ist. In diesem Fall fängt unsere Induktion später schon bei $m = 2$ an.

Lemma 6.4 *Sei* $2 \leq d < q$ *und* $m = 2$. *Dann ist* $D(d, q, m) = q + d(q - 1)$.

Beweis: Das Orakel antwortet zunächst in Tiefe 2 immer „Nein". Wird aber ein Blatt x mit $c(x) = q$ gefragt, so sind bereits alle Geschwister von x gefragt worden, und das Orakel hat bei diesen Geschwistern immer „Nein" geantwortet. Der Suchalgorithmus könnte also auch den Vater von x fragen und hätte danach die gleiche Information. Wir nehmen daher ohne Einschränkung an, daß in diesem Fall nie alle Söhne eines Knotens in Tiefe 1 gefragt werden. Dadurch ersparen wir uns einige Sonderfälle bei dem Orakel.

In Tiefe 1 antwortet das Orakel bei den ersten $d - 2$ Knoten, also bei $d = 2$ gar nicht „Ja". Wird der $d - 1$-te Knoten in Tiefe 1 gefragt, so gibt es für diesen Knoten x zwei Möglichkeiten:

Fall 1: Von x sind erst maximal $q - 2$ Söhne gefragt worden. Das Orakel antwortet „Ja". Bei den Söhnen von x wird ausnahmsweise schon beim $q - 1$-ten Knoten „Ja" geantwortet. Der Zustand des letzten Knotens muß in diesem Baum also extra erfragt werden. In Tiefe 1 antwortet das Orakel auf die nächsten $q - d$ Fragen „Nein". Die defekten Blätter haben alle verschiedene Väter in Tiefe 1. Pro defektem Blatt sind somit $q - 1$ Fragen in Tiefe 2 notwendig. Sind in Tiefe 1 auch nur $q - 1$ Fragen gestellt worden, so ist ein Knoten in Tiefe 1 ungefragt, und es ist nur $d - 1$ mal „Ja" in Tiefe 1 gefallen. Das defekte Blatt kann dann der letzte ungefragte Sohn dieses Knotens sein oder der eine ungefragte Sohn von x. Es ist folglich noch eine Frage notwendig. Damit sind insgesamt $q + d(q - 1)$ Fragen gestellt worden.

Fall 2: Von x sind genau $q - 1$ Söhne gefragt worden. Das Orakel antwortet „Nein". Da $d < q$ ist und das Orakel in Tiefe 1 bisher nie „Nein" gesagt hat, kann das Orakel die d defekten Blätter so aufteilen, daß nie zwei den gleichen Vater haben. Es sind damit pro defektem Blatt wieder $q - 1$ Fragen in Tiefe 2 nötig. Dazu kommen die q Fragen bei x und seinen Söhnen. Es sind zusammen also wieder $q + d(q - 1)$ Fragen.

□

Wir betrachten jetzt den Fall $d = q$. Hier ist $m = 2$ ein Sonderfall. Die Induktion beginnt erst bei $m = 3$.

Lemma 6.5 *Sei $d = q$ und $m = 2$. Dann existiert ein Orakel mit folgenden Eigenschaften:*

1. *Auf Fragen in Tiefe 1 wird immer „Ja" geantwortet.*
2. *Für alle $k \in \{0, \ldots, q - 1\}$ existieren nach $kq + q - 1$ Fragen maximal*

k Blätter, die als defekt erkannt worden sind.

3. *Nach der q^2-2-ten Frage hat das Orakel bei maximal q^2-q-1 Blättern „Nein“ geantwortet.*

Beweis: Sei T der vollständige q-reguläre Baum der Tiefe 2. T hat q Teilbäume $T_1, \dots, T_q$, die jeweils q Blätter haben. Das Orakel beantwortet Fragen in Tiefe 1 mit „Ja“. Fragen in Tiefe 2, also Blätter, werden mit „Nein“ beantwortet, außer in zwei Spezialfällen. Gilt für ein gefragtes Blatt x $c(x) = q$, so ist die Antwort „Ja“.

Der andere Spezialfall ist dieser: Es wird ein Blatt x in einem Teilbaum T_j gefragt, es gilt $c(x) = q-1$, in allen $q-1$ anderen Teilbäumen T_i, $i \neq j$ sind schon mindestens $q-1$ Blätter gefragt worden und dies ist nicht die q^2-1-te Frage. Dann ist die Antwort auch „Ja“.

Wir zeigen, daß dieses Orakel zulässige Antworten gibt und die obigen Behauptungen erfüllt.

Zunächst wird in jedem Teilbaum spätestens beim q-ten Blatt „Ja“ geantwortet. Das „Ja“ in Tiefe 1 ist also immer korrekt. Behauptung 1 ist folglich erfüllt.

Die „Ja“, die bei $c(x) = q$ geantwortet werden, setzen immer $q-1$ vorangegangene Fragen in diesem Teilbaum voraus. Sind auf diese Art also k „Ja“ vom Orakel geantwortet, so sind mindestens qk Fragen gestellt worden. Die andere Art, ein defektes Blatt zu finden, ist, ein Knoten in Tiefe 1 und $q-1$ Söhne zu fragen. Das sind ebenfalls q Fragen.

Das eine „Ja“, das schon bei $c(x) = q-1$ kommen kann, setzt in allen $q-1$ anderen Teilbäumen $q-1$ Fragen voraus. Mit den $q-2$ Fragen im Teilbaum, in dem x liegt, waren damit vorher schon mindestens $(q-1)(q-1)+(q-2) = (q-1)q-1$ Fragen gestellt worden. Mit dieser Frage sind es also $(q-1)q$ Fragen. Behauptung 2 wäre nur verletzt, wenn dies das q-te „Ja“ wäre. Dann wären aber in jedem anderen Teilbaum q Fragen gestellt worden, insgesamt somit $(q-1)q+(q-1) = q^2-1$. Dies war bei dem zweiten Spezialfall jedoch nicht zugelassen. Behauptung 2 ist daher immer erfüllt.

Sind erst q^2-2 Fragen gestellt worden, so ist in maximal $(q-1)$ Teilbäumen bei $q-1$ Blättern „Nein“ geantwortet worden. In einem Teilbaum wurde nur maximal $q-2$ mal „Nein“ geantwortet, da bei der $q-1$-ten Frage der zweite Spezialfall eintritt. Insgesamt sind also maximal

$$(q-1)(q-1)+(q-2) = q^2-q-1$$

„Nein“ geantwortet worden. Damit stimmt auch Behauptung 3.

□

Der zweite Spezialfall ist gegen solche Suchalgorithmen gerichtet, die zunächst immer $q-1$ Blätter, die Geschwister sind, fragen. Würde das Orakel dann immer nur „Nein“ antworten, so wären danach nur noch q ungefragte Blätter übrig. Da es q defekte Blätter geben muß, wären sie damit gefunden. Das Orakel verhindert dies, in dem es bei der letzten Möglichkeit „Ja“ antwortet.

Das Orakel ist im Fall $m = 2$ für die Bestimmung der Suchkomplexität nützlich. Die vielen zusätzlichen Eigenschaften werden aber erst bei $m = 3$ benötigt.

Lemma 6.6 *Es gilt für $d = q$, $m = 2$: $D(d, q, m) = q^2 - 1 = n - 1$.*

Beweis: Wir wenden das Orakel aus Lemma 6.5 an. Sind erst $q^2 - 2$ Fragen gestellt worden, so sind maximal $q-1$ Blätter als defekt erkannt. Es ist aber erst bei maximal $q^2 - q - 1$ Blätter „Nein“ gesagt worden. Es gibt damit noch insgesamt $q+1$ Blätter, unter denen die q defekten sind. Der Suchalgorithmus kann also noch nicht anhalten.

□

Wir kommen als nächstes zum kompliziertesten Spezialfall. Es ist der Fall $d = q$ und $m = 3$. Beim Beweis werden wir einige Ideen aus dem Beweis von Lemma 6.4 wieder anwenden.

Lemma 6.7 *Sei $d = q$ und $m = 3$. Dann ist $D(d, q, m) = 2q^2 - q$.*

Beweis: Unser Orakel wendet zunächst in den Tiefen 1 und 2 das Orakel aus Lemma 6.5 an. In Tiefe 3, also bei einem Blatt x, antwortet es im Fall $c(x) \leq q - 1$ „Nein“. Ist $c(x) = q$, so antwortet das Orakel derart, als wenn der Vater von x gefragt würde. Wir können daher ohne Einschränkung annehmen, daß dieser Fall nicht eintritt.

Dies macht das Orakel bis das Orakel aus Lemma 6.5 den $q-1$-ten kontaminierten Knoten in Tiefe 2 erzeugen würde. Wir nennen diesen Knoten x. Nach Lemma 6.5 sind mit dieser Frage insgesamt mindestens $(q-1)q$ Fragen gestellt worden. Es gibt für x zwei Möglichkeiten.

Fall 1: Von x sind schon $q-1$ Söhne gefragt worden. Dann antwortet das Orakel statt dessen „Nein“. x ist also rein. Das Orakel antwortet so, daß die q defekten Blätter als Söhne von q verschiedenen Knoten in Tiefe 2 erscheinen. Das ist möglich, da nach Lemma 6.5 erst maximal $q^2 - q - 1$

Knoten, mit x also maximal $q^2 - q$ Knoten ausgeschlossen worden sind. Für jedes der q defekten Blätter sind in Tiefe 3 $q - 1$ Fragen notwendig. Mit den q Fragen im zu x gehörenden Teilbaum sind dann insgesamt

$$(q-1)q + q(q-1) + q = 2q^2 - q$$

Fragen gestellt worden.

Fall 2: Von x sind erst maximal $q - 2$ Söhne gefragt worden. Es wird, wie es das Orakel aus Lemma 6.5 verlangt, „Ja“ geantwortet. Werden später die Söhne von x gefragt, so wird schon bei dem $q - 1$-ten „Ja“ geantwortet. Der Status des letzten Sohns von x ist daher noch nicht bekannt. Um die q defekten Blätter zu finden, müssen von jedem kontaminierten Knoten in Tiefe 2 wieder mindestens $q - 1$ Söhne gefragt werden. Dies sind $q(q - 1)$ Fragen. Sind erst $2q^2 - q - 1$ Fragen gestellt worden, so sind in Tiefe 1 und 2 bisher maximal $q^2 - 1$ Fragen gestellt worden. Damit ist der q-te kontaminierte Knoten aber nicht durch ein „Ja“ gefunden worden, und bei seinen Söhnen wurde nur $q - 1$ mal „Nein“ geantwortet. Es ist also noch nicht klar, ob das defekte Blatt in diesem Teilbaum liegt oder der letzte ungefragte Sohn von x ist.

□

Wir beweisen jetzt die Suchkomplexität für allgemeines m.

Satz 6.8 *Für $2 \leq d \leq q$ gilt*

$$D(d,q,m) = \begin{cases} n-1, & \textit{falls } m = 1 \\ n-1, & \textit{falls } d = q \textit{ und } m = 2 \\ q + d(q-1)(m-1), & \textit{sonst.} \end{cases}$$

Beweis: Als obere Schranke verwenden wir wieder Korollar 6.2. Um die untere Schranke zu zeigen, führen wir vollständige Induktion über m.

$m = 1$: In diesem Fall ist $d < q$, da nach Voraussetzung $d < n = q$ ist. Daher ist nur der Individualtest möglich. Er braucht $q - 1 = n - 1$ Fragen.

$d < q$ und $m = 2$: Diesen Fall haben wir in Lemma 6.4 gezeigt.

$d = q$ und $m = 2$: Dies ist Lemma 6.6.

$d = q$ und $m = 3$: Es gilt $q + d(q-1)(m-1) = 2q^2 - q$. Damit folgt die Behauptung aus Lemma 6.7.

$m \to m + 1$: Wir nehmen an, daß $D(d,q,m) = q + d(q-1)(m-1)$ ist. Wir haben gesehen, daß dies sowohl für $d < q$ und $m = 2$ als auch für

$d = q$ und $m = 3$ gilt. Wir zeigen jetzt, daß $D(d, q, m+1) = q + d(q-1)m$ ist. Sei also $\mathcal{O}_m$ das für den (q^m, q)-Baum existierende Orakel. Wir definieren das Orakel $\mathcal{O}_{m+1}$. Wird ein Knoten x gefragt, so antwortet $\mathcal{O}_{m+1}$ so:

- x liegt in Tiefe $0, \ldots, m$: $\mathcal{O}_{m+1}$ antwortet wie $\mathcal{O}_m$.
- x liegt in Tiefe $m+1$ und $c(x) \leq q-1$: „Nein".
- x liegt in Tiefe $m+1$ und $c(x) = q$: $\mathcal{O}_{m+1}$ antwortet so, wie $\mathcal{O}_m$ auf den Vater von x antworten würde.

Sei $\mathcal{A}$ ein Suchalgorithmus. Ohne Einschränkung können wir annehmen, daß $\mathcal{A}$ niemals ein Blatt x mit $c(x) = q$ fragt. Tut er dies doch, so wurde bereits auf alle Geschwister von x mit „Nein" geantwortet. Damit kann $\mathcal{A}$ den Vater von x fragen und hat die gleiche Information.

Hat $\mathcal{A}$ die d defekten Blätter gefunden, so wurde bei keinem Blatt „Ja" geantwortet. Der Algorithmus hat daher in Tiefe m genau d kontaminierte Knoten gefunden, von denen jeweils $q-1$ Söhne gefragt worden sind. Es sind also in Tiefe $0, \ldots, m$ nach Induktionsvoraussetzung $q+d(q-1)(m-1)$ Knoten und dazu $d(q-1)$ Blätter gefragt worden. Das sind insgesamt $q + d(q-1)m$ Fragen.

□

6.3.3 $q < d \leq (q-1)q^{m-2}$

Wir haben bei den Suchalgorithmen schon erwähnt, daß es für $d > q$ längere Laufzeiten gibt, wenn in jeder, bis auf der letzten Tiefe des Baumes, immer nur $d-1$ kontaminierte Knoten existieren. Wir brauchen darum in diesem Abschnitt andere Orakel als im Fall $d \leq q$. Sie werden, wie im Fall $q = 2$, ausnutzen, daß der Fragende bis kurz vor Ende der Suche immer nur maximal $d-1$ verschiedene kontaminierte Teilbäume gefunden haben wird. Er muß deshalb bei einem kontaminierten inneren Knoten immer alle Söhne fragen, um alle d defekten Blätter zu finden.

Wir geben, wie im Kapitel 5, zunächst wieder einige Orakel für den Fall an, daß der Fragende die Anzahl der defekten Blätter in einem Baum nicht kennt.

Lemma 6.9 *Seien $q, m \in \mathbb{N}$, $q \geq 3$ und T der vollständige reguläre q-Baum mit $n = q^m$ Blättern. Der Fragende weiß, daß T ein oder zwei defekte Blätter* *enthält. Dann gibt es ein Orakel, so daß nach $mq-1$ Fragen noch $d=1$ oder $d=2$ möglich ist.*

Bis nach der $mq-3$-ten Antwort kann sich das Orakel auf zwei defekte Blätter festlegen und insgesamt $mq-1$ Fragen erzwingen. Nach der $mq-2$-ten Antwort gibt es für das Orakel noch eine Möglichkeit, sich festzulegen, um insgesamt $mq-1$ Fragen zu erzwingen.

Beweis: $m=1$: Das Orakel antwortet $q-2$ mal „Nein" und dann „Ja". Nach $q-1$ Fragen ist damit ein defektes Blatt gefunden worden, und der Status des letzten Blattes ist unklar.

Sind erst maximal $q-3$ Fragen gestellt worden und legt sich das Orakel auf zwei defekte Blätter fest, so antwortet das Orakel schon bei der $q-2$-ten Frage „Ja". Das letzte defekte Blatt ist dann eins der beiden ungefragten Blätter. Es ist also noch eine Frage notwendig.

Sind bereits $q-2$ Fragen gestellt worden, so legt sich das Orakel auf ein defektes Blatt fest. Bisher hat es immer „Nein" geantwortet. Es ist dann noch eine Frage notwendig, um das Blatt zu finden.

$m \to m+1$: Das Orakel antwortet zunächst wieder $q-2$ mal „Nein". Danach sind noch mindestens 2 Teilbäume T_1 und T_2 der Größe q^m vorhanden. Ist die $q-1$-te Frage die Wurzel eines dieser Teilbäume (ohne Einschränkung T_1), so ist die Antwort „Ja". In T_1 antwortet das nach Induktion vorhandene Orakel. Wird die Wurzel oder ein anderer Knoten von T_2 gefragt, so lautet die Antwort „Nein". Wird in T_1 die qm-te Frage gestellt, so legt sich das Orakel auf ein defektes Blatt in T_1 fest.

Ist die $q-1$-te Frage in einem der T_i, $i=1,2$ (ohne Einschränkung T_1), so ist die Antwort „Nein". In T_2 wird das nach Induktionsvoraussetzung vorhandene Orakel angewendet. Jede weitere Frage in T_1 wird dann ebenfalls mit „Nein" beantwortet.

Bis $d=1$ im kontaminierten Teilbaum T_i feststeht, sind nach Induktionsvoraussetzung qm Fragen in T_i notwendig. Außerdem sind in beiden Fällen zunächst $q-2$ Fragen gestellt worden, und weiterhin noch mindestens 2 Fragen, bis feststeht, daß der andere Teilbaum T_{3-i} rein ist. Das sind insgesamt $q(m+1)$ Fragen.

Muß sich das Orakel festlegen, so können wir die Beweisidee mit dem kleinsten kontaminierten Teilbaum aus den Lemmata 5.9 und 5.10 direkt übernehmen.

□

Lemma 6.10 *Sei $q, m \in \mathbb{N}$, $q \geq 3$ und T der vollständige reguläre q-Baum mit $n = q^m$ Blättern. Der Fragende weiß, daß T d oder $d+1$ defekte Blätter enthält mit $1 \leq d \leq q-1$. Dann gibt es ein Orakel, das nach $q+dq(m-1)-1$ Fragen noch beide Möglichkeiten zuläßt. Muß es sich festlegen, wieviele Blätter defekt sind, so kann es insgesamt $q+dq(m-1)-1$ Fragen erzwingen.*

Beweis: $m = 1$: Das Orakel antwortet bei Blättern $d-1$ mal „Ja", dann $q-d-1$ mal „Nein" und bei der $q-1$-ten Frage wieder „Ja". Muß es sich vorher auf zwei defekte Blätter festlegen, antwortet es auch schon bei der $q-2$-ten Frage „Ja".

$m \geq 2$: Das Orakel antwortet im Baum in Tiefen ≥ 2 zunächst immer „Nein". In Tiefe 1 antwortet es normalerweise d mal „Ja", danach $q-d$ mal „Nein". In den kontaminierten Teilbäumen wird das Orakel aus Lemma 6.9 angewendet. Dies funktioniert, wenn bei jeder Frage in Tiefe 1, bei der „Ja" geantwortet werden soll, erst maximal $q-2$ Fragen gestellt worden sind, da das Orakel aus Lemma 6.9 zunächst auch $q-2$ mal „Nein" antwortet.

Wird aber ein Knoten in Tiefe 1 gefragt, von dem schon mindestens $q-1$ Söhne gefragt worden sind, und ist dies eine der ersten d Fragen in Tiefe 1, so ist die Antwort ausnahmsweise „Nein". Da in Tiefe 1 bisher nie „Nein" geantwortet worden ist und $d \leq q-1$ ist, kann also in d Teilbäumen der Größe q^m das Orakel aus Lemma 6.9 angewendet werden.

In beiden Fällen legen sich die Orakel in den Teilbäumen nach der qm-ten Frage auf ein defektes Blatt fest.

Damit der Fragende weiß, daß es nur d defekte Blätter sind, muß er in beiden Fällen pro kontaminiertem Teilbaum qm Fragen stellen. Zusätzlich sind im ersten Fall q Fragen in Tiefe 1 und im zweiten Fall q Fragen im reinen Teilbaum gestellt worden. Das sind insgesamt $q+dqm$ Fragen. Genau das war zu zeigen.

Da ab Tiefe 1 in allen Teilbäumen das Orakel aus Lemma 6.9 angewendet wird, folgt die Laufzeit beim Festlegen direkt aus diesem Lemma.

□

Lemma 6.11 *Seien $q, m \in \mathbb{N}$, $q \geq 3$, $m \geq 2$ und T der vollständige reguläre q-Baum mit $n = q^m$ Blättern. Der Fragende weiß, daß T d oder $d+1$ defekte Blätter enthält mit $q-1 \leq d \leq (q-1)q$. Dann gibt es ein Orakel, das nach $q^2+dq(m-2)-1$ Fragen noch beide Möglichkeiten zuläßt. Muß es sich festlegen, wieviele Blätter defekt sind, so kann es insgesamt $q^2+dq(m-2)-1$ Fragen erzwingen.*

Beweis: Im Fall $d = q - 1$ ist

$$q + dq(m-1) - 1 = q^2 + dq(m-2) - 1.$$

Die Behauptung folgt damit aus Lemma 6.10.

Ist $d \geq q$, so zerlegt das Orakel

$$d = \sum_{i=1}^{q} d_i,$$

wobei $1 \leq d_i \leq q-1$ für alle $i = 1, \ldots, q$ gilt. In jedem Teilbaum T_i der Größe q^{m-1} wendet das Orakel das nach Lemma 6.10 existierende Orakel für d_i defekte Blätter an, wobei es sich bei der letzten Frage im Teilbaum T_i auf d_i defekte Blätter festlegt.

Um zu wissen, daß es nur d defekte Blätter sind, muß der Fragende von jedem Teilbaum wissen, daß er nur d_i defekte Blätter enthält. Er muß daher

$$\sum_{i=1}^{q} (q + d_i q(m-2)) = q^2 + dq(m-2)$$

Fragen stellen.

Muß sich das Orakel festlegen, so sind die Überlegungen wieder die gleichen, wie in den vorangegangenen Beweisen.

□

Wir können damit die Suchkomplexität für $q < d \leq (q-1)q^{m-2}$ beweisen.

Satz 6.12 *Ist $m \geq 2$ und $q < d \leq (q-1)q^{m-2}$, so gilt*

$$D(d,q,m) = q^l + (d-1)q(m-l) - 1,$$

wobei $l = \left\lceil \log_q \frac{d}{q-1} \right\rceil + 1$ ist.

Beweis: Damit es ein solches d geben kann, muß $m \geq 3$ gelten. Formal ist die Aussage damit auch für $m = 2$ richtig. Das erspart uns einige Fallunterscheidungen.

Ist $q = 2$, so haben wir die Aussage schon in Satz 5.13 bewiesen. Sei also $q \geq 3$.

In Korollar 6.2 ist schon die obere Schranke bewiesen worden. Wir geben jetzt ein Orakel an, das so viele Fragen erzwingt.

Es gilt $2 \leq l < m$. Außerdem gilt

$$\begin{aligned} & l = \left\lceil \log_q \frac{d}{q-1} \right\rceil + 1 \\ \iff \quad & \log_q \frac{d}{q-1} + 1 \leq l < \log_q \frac{d}{q-1} + 2 \\ \iff \quad & d \leq (q-1)q^{l-1} < qd. \end{aligned}$$

Insbesondere ist also $d - 1 \geq (q-1)q^{l-2}$ und $d \leq (q-1)qq^{l-2}$. Das Orakel zerlegt daher

$$d - 1 = \sum_{i=1}^{q^{l-2}} d_i,$$

wobei $q - 1 \leq d_i \leq (q-1)q$ für alle $i = 1, \dots, q^{l-2}$ gilt. In jedem der q^{l-2} Teilbäume, deren Wurzel in der Tiefe $l-2$ liegt, wendet das Orakel das nach Lemma 6.11 vorhandene Orakel an. Zunächst legt es sich in jedem Teilbaum T_i bei der $q + d_i q(m-l)$-ten Frage auf d_i defekte Blätter fest. Im vorletzten Teilbaum legt es sich entweder auf d_i oder $d_i + 1$ defekte Blätter fest, so daß im letzten Teilbaum nur eine Frage gespart wird. Damit sind insgesamt

$$\begin{aligned} & \sum_{i=1}^{q^{l-2}} \left(q^2 + d_i q(m-l)\right) - 1 \\ = \quad & q^l + \sum_{i=1}^{q^{l-2}} (d_i)\, q(m-l) - 1 \\ = \quad & q^l + (d-1)q(m-l) - 1 \end{aligned}$$

Fragen notwendig.

□

6.3.4 $(q-1)q^{m-2} < d < n$

Für so große d ist der Individualtest optimal.

Lemma 6.13 *Sei $q \in \mathbb{N}$, $q \geq 3$ und T der vollständige reguläre q-Baum mit q^2 Blättern. Der Fragende weiß, daß die Anzahl der defekten Blätter d von T zwischen $q-1$ und q liegt. Dann gibt es ein Orakel, so daß nach q^2-1 Fragen* *noch $d = q - 1$ und $d = q$ möglich ist. Insbesondere kennt der Fragende nach $q^2 - 1$ Fragen noch nicht den Status aller Blätter.*

Muß sich das Orakel festlegen, so kann es immer $q^2 - 1$ Fragen erzwingen.

Beweis: Dies ist das Lemma 6.10 mit $d = q - 1$ und $m = 2$.

□

Satz 6.14 *Für $m \geq 2$ und $d > (q-1)q^{m-2}$ gilt*

$$D(d, q, m) = q^m - 1 = n - 1.$$

Beweis: Im Fall $q = 2$ haben wir dies in Satz 5.14 gesehen. Sei also $q \geq 3$. Offensichtlich reichen immer $q^m - 1$ Fragen. Nach Lemma 5.12 genügt es, die Behauptung für $d = (q-1)q^{m-2} + 1$ zu beweisen.

Im Fall $m = 2$ ist $d = q$. Damit folgt die Behauptung aus Satz 6.8.

Ist $m \geq 3$, so wendet das Orakel in den q^{m-2} Teilbäumen, deren Wurzel in Tiefe $m - 2$ liegt, das Lemma 6.13 an. Bei der letzten Frage legt es sich auf $q - 1$ defekte Blätter fest. Lediglich im vorletzten Teilbaum legt es sich derart auf $q - 1$ oder q defekte Blätter fest, daß im letzten Teilbaum noch $q^2 - 1$ Fragen notwendig sind.

In allen Teilbäumen sind damit q^2 Fragen notwendig. Nur im letzten Teilbaum reichen $q^2 - 1$ Fragen. Damit werden insgesamt

$$q^{m-2}q^2 - 1 = q^m - 1 = n - 1$$

Fragen benötigt, und es gibt $(q-1)q^{m-2} + 1$ defekte Blätter.

□

6.4 Zusammenfassung

Wir haben in diesem Kapitel die Suchkomplexität in vollständigen regulären q-Bäumen untersucht. Wie im Fall $q = 2$ ergaben sich für Bäume ab einer von q abhängigen Größe wieder vier verschiedene Fälle für die Suchkomplexität. Sie sind $d = 1$, $2 \leq d \leq q$, $q < d \leq (q-1)\frac{n}{4}$ und $d > (q-1)\frac{n}{4}$. Die Such- und Antwortstrategien ähneln denen aus Kapitel 5 bei den binären Bäumen.

Die Suchkomplexität beträgt

$$D(d,q,m) = \begin{cases} (q-1)m, & \text{falls } d = 1 \\ n-1, & \text{falls } 2 \leq d < q \text{ und } m = 1 \\ n-1, & \text{falls } d = q \text{ und } m = 2 \\ q + d(q-1)(m-1), & \text{falls } 2 \leq d \leq q \\ q^l + (d-1)q(m-l) - 1 & \\ \quad \leq n - q - 1, & \text{falls } q < d \leq (q-1)q^{m-2} \\ n-1, & \text{falls } (q-1)q^{m-2} < d, \end{cases}$$

wobei $l = \left\lceil \log_q \frac{d}{q-1} \right\rceil + 1$ ist.

Kapitel 7

Parallele Suche in binären Bäumen

7.1 Einleitung

Bisher haben wir die sequentielle Suche in Bäumen untersucht. Daneben spielt in Anwendungen häufig die parallele Suche eine wichtige Rolle. Ein paralleler Suchalgorithmus $\mathcal{A}$ für einen Suchprozeß $(S, \mathcal{F})$ muß dabei alle Fragen auf einmal formulieren. Anschließend werden alle Fragen beantwortet. Der Algorithmus muß dabei sicherstellen, daß er bei allen möglichen Antworten das gesuchte Element x^* aus der Menge S eindeutig bestimmt hat. Die Suchkomplexität des Suchprozesses ist wieder als die minimale Anzahl von Fragen definiert, für die es einen parallelen Suchalgorithmus gibt. Sie wird mit $L_{pre}(S, \mathcal{F})$ bezeichnet.

Wir werden sehr bald sehen, daß ein paralleler Suchalgorithmus bei Bäumen sehr ineffektiv ist. Darum führen wir mehr Abstufungen zwischen der sequentiellen und der parallelen Suche ein.

Sei $r \in \mathbb{N}$ und $(S, \mathcal{F})$ ein Suchprozeß. Wir betrachten in diesem Kapitel Algorithmen, die die Fragen in r Runden stellen. Alle Fragen, die in der gleichen Runde gestellt werden, werden dabei parallel beantwortet. Die Laufzeit des Algorithmusses ist wieder die Anzahl aller Fragen, die er stellt. Das Minimum der Laufzeiten aller Algorithmen bezeichnen wir mit $L_r(S, \mathcal{F})$.

Für $r = 1$ ist dies eine normale parallele Suche. Auf der anderen Seite ist $L(S, \mathcal{F})$ immer eine untere Schranke für $L_r(S, \mathcal{F})$. Sobald $r \geq L(S, \mathcal{F})$ ist,

ist $L_r(S,\mathcal{F}) = L(S,\mathcal{F})$, da der Algorithmus die notwendigen Fragen einfach sequentiell stellen kann.

Wir können also sagen, daß für alle $r \in \mathbb{N}$ stets

$$L(S,\mathcal{F}) \leq L_r(S,\mathcal{F}) \leq L_{pre}(S,\mathcal{F})$$

gilt, wobei $L_r(S,\mathcal{F})$ monoton fallend in r ist und beide Schranken annimmt.

Wir werden sehen, daß diese beiden Schranken bei Bäumen sehr weit auseinanderliegen. Die Ergebnisse können dann als ein „Trade Off" zwischen der Anzahl der Runden und der Anzahl der notwendigen Fragen verstanden werden.

Als Suchmenge S wählen wir in diesem Kapitel wieder die Blätter eines vollständigen binären Baumes. Erlaubte Fragen sind wieder alle Teilbäume. $L_r(T)$ bezeichnet dann die Suchkomplexität bei r Runden in dem Baum T.

Definition 7.1 *Sei $m \in \mathbb{N}_0$, $r \in \mathbb{N}$ und T der vollständige binäre Baum mit $n = 2^m$ Blättern. Dann sei*

$$R(r,m) = L_r(T).$$

Wir werden in den folgenden Abschnitten einen optimalen Algorithmus und ein Orakel angeben, mit denen sich $R(r,m)$ berechnen läßt.

7.2 Obere Schranke

Satz 7.2 *Seien $r \in \mathbb{N}$ und $m \in \mathbb{N}_0$. Dann gilt*

$$R(r,m) \leq (r + (m \bmod r))2^{\lfloor \frac{m}{r} \rfloor} - r.$$

Beweis: Es gilt $R(r,0) = 0$. Für $m = 0$ stimmt damit die Behauptung. Sei also $m \geq 1$. Wir setzen $a = \lfloor \frac{m}{r} \rfloor$ und $b = m \bmod r$. Dann gilt $m = ar + b$, wobei $0 \leq b < r$ ist. Der Suchalgorithmus $\mathcal{A}_{r,m}$ ist folgendermaßen definiert:

- Ist $r > m$, so wende binäre Suche an.
- Ist $r \leq m$, so frage in der ersten Runde in Tiefe a genau 2^a-1 Knoten. Ist $r > 1$, so wende in dem einzigen kontaminierten Teilbaum der Größe 2^{m-a} den Algorithmus $\mathcal{A}_{r-1,m-a}$ an.

Dieser Algorithmus ist korrekt. Ist $r > m$, so fragt er m Runden lang jeweils einen Knoten und danach r Runden lang keinen Knoten. Anschließend ist das defekte Blatt identifiziert.

Ist $r \leq m$, so fragt der Algorithmus auch nur r Runden lang. Da $r \geq 1$ ist, ist $a \leq m$. Die Tiefe a existiert also im Baum und sie hat 2^a Knoten. Aus den $2^a - 1$ Antworten läßt sich somit der kontaminierte Teilbaum ermitteln. In der letzten Runde gilt $r = 1$ und daher $a = m$. Der Algorithmus fragt demnach alle bis auf ein Blatt.

Wir zeigen, daß

$$L(\mathcal{A}_{r,m}) = (r + (m \bmod r))2^{\lfloor \frac{m}{r} \rfloor} - r \tag{7.1}$$

ist. Für $r > m$ sind beide Seiten der Gleichung gleich m. Sei also $r \leq m$. Wir führen vollständige Induktion über r. Ist $r = 1$, so haben wir weiter oben schon gesehen, daß $2^m - 1$ Fragen gestellt werden, wie in Gleichung (7.1) behauptet.

Ist $r \geq 2$, so ist

$$L(\mathcal{A}_{r,m}) = 2^a - 1 + L(\mathcal{A}_{r-1,m-a}).$$

Ist $b = r - 1$, so gilt $m - a = (a + 1)(r - 1)$. Damit gilt nach Induktionsvoraussetzung

$$L(\mathcal{A}_{r-1,m-a}) = ((r - 1))2^{a+1} - (r - 1) = (2r - 2)2^a - r + 1.$$

Also ist

$$L(\mathcal{A}_{r,m}) = (2r - 1)2^a - r = (r + b)2^a - r.$$

Ist $b < r - 1$, so ist $m - a = a(r - 1) + b$. Es gilt daher

$$L(\mathcal{A}_{r-1,m-a}) = ((r - 1) + b)2^a - (r - 1) = (r + b - 1)2^a - r + 1.$$

Das liefert wiederum

$$L(\mathcal{A}_{r,m}) = (r + b)2^a - r.$$

□

Bei der Definition der Algorithmen $\mathcal{A}_{r,m}$ müßten wir den Fall $r > m$ nicht extra definieren. Solange $r > m$ ist, stellt der Algorithmus keine Frage und verringert r um 1. Ist $r = m$, so führt er die binäre Suche durch.

Eine bemerkenswerte Eigenschaft der Algorithmen $\mathcal{A}_{r,m}$ ist, daß sich die in jeder Runde gefragten Knoten sehr ähnlich sind. Betrachten wir nach jeder Runde den übrigbleibenden Baum, so fragen sie zunächst $m-b$ mal in Tiefe a und danach b mal in Tiefe $a+1$. Sie dürften sich daher sehr effizient implementieren lassen.

7.3 Untere Schranke

Um ein Orakel zu konstruieren, analysieren wir zunächst die erste Runde. Wir betrachten hier nur die Fälle $m \geq 1$, da $R(r,0) = 0$ ist.

Lemma 7.3 *Sei $m \in \mathbb{N}$, T der vollständige binäre Baum mit 2^m Blättern und $l \in \mathbb{N}$ mit $1 \leq l \leq m$. Ein paralleler Algorithmus stelle k Fragen. Ist $k \leq 2^l - 2$, so kann ein Orakel so antworten, daß der übrigbleibende Baum einen vollständigen binären Baum der Größe 2^{m-l+1} enthält. Ist $k = 2^l - 1$, so gibt es ein Orakel, das einen der folgenden Fälle erreicht:*

1. *Der übrigbleibende Baum enthält einen vollständigen binären Baum der Größe 2^{m-l+1}.*

2. *Der übrigbleibende Baum enthält einen vollständigen binären Baum der Größe 2^{m-l}, und das Orakel hat nur „Nein“ geantwortet.*

Beweis: Wir führen vollständige Induktion über m.

$\boldsymbol{m = 1}$**:** Es gilt $l = 1$. Damit ist $2^l - 2 = 0$ und $2^l - 1 = 1$. Bei 0 Fragen bleibt ein Baum der Größe $2 = 2^{m-l+1}$ übrig. Bei einer Frage wird entweder die Wurzel oder ein Blatt gefragt. Bei der Wurzel antwortet das Orakel „Ja“, und Fall 1 ist erfüllt. Bei einem Blatt antwortet es „Nein“. Das erfüllt Fall 2.

$\boldsymbol{m \geq 2}$**:** Ist $l = 1$, so ist die Argumentation die gleiche wie im Fall $m = 1$. Auf die Frage nach der Wurzel wird „Ja“ und sonst „Nein“ geantwortet.

Ist $2 \leq l \leq m$ und $k \leq 2^l - 2$, so betrachten wir die Anzahl k_1 und k_2 der Fragen in den beiden Teilbäumen der Größe 2^{m-1}. Gilt $k_1 \leq 2^{l-1} - 2$, so antwortet das nach Induktionsvoraussetzung für diese Fragen vorhandene Orakel im ersten Teilbaum. Im zweiten Teilbaum wird nur „Nein“ geantwortet. Nach Induktionsvoraussetzung bleibt damit im ersten Teilbaum ein Baum der Größe $2^{(m-1)-(l-1)+1} = 2^{m-l+1}$ übrig. Ist $k_2 \leq 2^{l-1} - 2$, so antwortet das Orakel entsprechend im zweiten Teilbaum. Tritt keiner dieser beiden Fälle ein, so muß $k_1 = k_2 = 2^{l-1} - 1$ sein, da $k_1 + k_2 \leq k \leq 2^l - 2$ gilt.

Wir wenden wieder in beiden Teilbäumen das nach Induktionsvoraussetzung vorhandene Orakel an. Wird in einem Teilbaum der Fall 1 erreicht, so folgt daraus die Behauptung. Tritt in beiden Teilbäumen Fall 2 ein, so kommen beide Bäume der Größe $2^{(m-1)-(l-1)}$ noch als kontaminiert in Frage, da das Orakel in beiden Teilbäumen nur „Nein“ geantwortet hat. Damit bleibt ein Baum der Größe 2^{m-l+1} übrig.

Ist schließlich $2 \leq l \leq m$ und $k = 2^l - 1$, so seien wieder k_1 und k_2 die Anzahl der Fragen in beiden Teilbäumen. Ist $k_1 \leq 2^{l-1} - 2$ oder $k_2 \leq 2^{l-1} - 2$, so kann das Orakel nach Induktionsvoraussetzung Fall 1 erfüllen. Gilt aber $k_1, k_2 \geq 2^{l-1} - 1$, so ist zumindest ein $k_i = 2^{l-1} - 1$, $i = 1, 2$. In diesem Teilbaum kann Fall 1 oder Fall 2 erreicht werden. Im anderen Teilbaum antwortet das Orakel nur „Nein“.

□

Das Lemma deutet schon an, daß es vernünftig ist, $2^l - 1$ Fragen in der ersten Runde zu stellen, wobei l noch näher zu bestimmen ist. Dies ist die Aussage des nächsten Lemmas.

Lemma 7.4 *Seien $r, m \in \mathbb{N}$ und $\mathcal{A}$ ein optimaler Algorithmus für r Runden, also $L(\mathcal{A}) = R(r, m)$. Dann existiert ein $l \in \{1, \ldots, m\}$, so daß $\mathcal{A}$ in der ersten Runde genau $2^l - 1$ Fragen stellt.*

Beweis: Angenommen $\mathcal{A}$ fragt in der ersten Runde k Fragen, wobei es ein $l \in \mathbb{N}$ mit $1 \leq l \leq m$ gibt, so daß $2^{l-1} \leq k \leq 2^l - 2$ ist. Dann muß $l \geq 2$ sein. Nach Lemma 7.3 kann ein Orakel derart antworten, daß ein Baum der Größe 2^{m-l+1} übrigbleibt. Damit gilt

$$L(\mathcal{A}) \geq 2^{l-1} + R(r-1, m-l+1).$$

Wir definieren den Suchalgorithmus $\mathcal{B}$ so, daß er in der ersten Runde in Tiefe $l-1$ genau $2^{l-1} - 1$ Fragen stellt. In dem verbleibenden Baum der Größe 2^{l-l+1} wenden wir einen optimalen Algorithmus an. Dann gilt

$$L(\mathcal{B}) = 2^{l-1} - 1 + R(r-1, m-l+1).$$

Damit ist aber $L(\mathcal{B}) \leq L(\mathcal{A}) - 1$, was im Widerspruch zur Optimalität von $\mathcal{A}$ steht.

□

Als letzte Vorbereitung brauchen wir noch eine Wachstumseigenschaft der in Satz 7.2 angegebenen Schranke.

Lemma 7.5 *Für $r \in \mathbb{N}$ und $m \in \mathbb{N}_0$ sei*

$$B(r,m) = (r + (m \bmod r))2^{\lfloor \frac{m}{r} \rfloor} - r.$$

Dann gilt

$$B(r, m+1) = B(r,m) + 2^{\lfloor \frac{m}{r} \rfloor}.$$

Beweis: Wir bemerken zunächst, daß

$$\left\lfloor \frac{m+1}{r} \right\rfloor = \left\lfloor \frac{m}{r} \right\rfloor$$

äquivalent zu

$$(m \bmod r) < r - 1$$

ist. Ist dies der Fall, so ist $((m+1) \bmod r) = 1 + (m \bmod r)$. Daraus folgt die Behauptung. Ist aber $(m \bmod r) = r - 1$, so gilt

$$\begin{aligned}
B(r, m+1) &= (r + ((m+1) \bmod r))2^{\lfloor \frac{m+1}{r} \rfloor} - r \\
&= (r+0)2^{1+\lfloor \frac{m}{r} \rfloor} - r \\
&= (2r)2^{\lfloor \frac{m}{r} \rfloor} - r \\
&= (r + (m \bmod r))2^{\lfloor \frac{m}{r} \rfloor} - r + 2^{\lfloor \frac{m}{r} \rfloor} \\
&= B(r,m) + 2^{\lfloor \frac{m}{r} \rfloor}.
\end{aligned}$$

□

Wir können damit zeigen, daß die Algorithmen $\mathcal{A}_{r,m}$ optimal sind.

Satz 7.6 *Seien $r \in \mathbb{N}$ und $m \in \mathbb{N}_0$. Dann gilt*

$$R(r,m) = (r + (m \bmod r))2^{\lfloor \frac{m}{r} \rfloor} - r.$$

Insbesondere ist für $m \geq 1$ der Algorithmus $\mathcal{A}_{r,m}$ optimal.

Beweis: In Satz 7.2 haben wir schon „$\leq$“ gezeigt. Im Fall $m = 0$ gilt „$=$“, so daß wir $m \geq 1$ annehmen.

Für $r > m$ gilt $L(\mathcal{A}_{r,m}) = m$. Da $\lfloor \frac{m}{r} \rfloor = 0$ und $m \bmod r = m$ ist, ist dies genau die Behauptung.

Im Fall $r = 1$ folgt der Satz durch $m = l$ direkt aus Lemma 7.3. Jeder Algorithmus muß mindestens $2^m - 1$ Fragen stellen, und genau das macht $\mathcal{A}_{1,m}$.

Wir haben die Behauptung also noch für $2 \leq r \leq m$ zu zeigen. Dazu führen wir vollständige Induktion über m.

$m = 1$: Es gibt kein r mehr, für das die Behauptung noch zu zeigen wäre.

$m \geq 2$: Sei $\mathcal{B}$ ein optimaler Algorithmus. Nach Lemma 7.4 gibt es ein $l \in \mathbb{N}$, $1 \leq l \leq m$, so daß $\mathcal{B}$ in der ersten Runde $2^l - 1$ Fragen stellt. Nach Lemma 7.3 kann ein Orakel so antworten, daß noch ein Baum der Größe 2^{m-l} vorhanden ist. Damit gilt

$$L(\mathcal{B}) \geq 2^l - 1 + R(r-1, m-l) =: \beta(l).$$

Wir setzen wieder $a, b \in \mathbb{N}_0$ so, daß $m = ar + b$ und $0 \leq b < r$ ist. Damit zeigen wir, daß $\beta(l)$ minimal für $l = a$ wird. Daraus folgt, daß

$$L(\mathcal{B}) = 2^a - 1 + R(r-1, m-a) = L(\mathcal{A}_{r,m})$$

ist und damit die Behauptung.

Ist $a \leq l < m$, so ist

$$\begin{aligned} & \beta(l+1) - \beta(l) \\ = \; & 2^{l+1} - 1 + R(r-1, m-l-1) - \big(2^l - 1 + R(r-1, m-l)\big) \\ = \; & 2^l + R(r-1, m-l-1) - R(r-1, m-l) \\ = \; & 2^l - 2^{\left\lfloor \frac{m-l-1}{r-1} \right\rfloor} \end{aligned}$$

nach Lemma 7.5. Wir beweisen jetzt, daß dies ≥ 0 ist, indem wir zeigen, daß $l \geq \left\lfloor \frac{m-l-1}{r-1} \right\rfloor$ ist. Aus $l \geq a$ folgt

$$\begin{aligned} & l - \left\lfloor \frac{m-l-1}{r-1} \right\rfloor \\ \geq \; & a - \left\lfloor \frac{m-a-1}{r-1} \right\rfloor \\ = \; & a - \left\lfloor \frac{ar+b-a-1}{r-1} \right\rfloor \\ = \; & a - \left\lfloor a + \frac{b-1}{r-1} \right\rfloor \end{aligned}$$

$$\begin{aligned} &= \left\lfloor \frac{b-1}{r-1} \right\rfloor \\ &= 0, \end{aligned}$$

da $b < r$ und $r \geq 2$ ist.

Ist andererseits $1 < l \leq a$, so ist

$$\begin{aligned} & \beta(l-1) - \beta(l) \\ = \; & 2^{l-1} - 1 + R(r-1, m-l+1) - \left(2^l - 1 + R(r-1, m-l)\right) \\ = \; & -2^{l-1} + R(r-1, m-l+1) - R(r-1, m-l) \\ = \; & -2^{l-1} + 2^{\left\lfloor \frac{m-l}{r-1} \right\rfloor}. \end{aligned}$$

Für die Exponenten gilt

$$\begin{aligned} & \left\lfloor \frac{m-l}{r-1} \right\rfloor - (l-1) \\ \geq \; & \left\lfloor \frac{m-a}{r-1} \right\rfloor - (a-1) \\ = \; & \left\lfloor \frac{ar+b-a}{r-1} \right\rfloor - a + 1 \\ = \; & \left\lfloor a + \frac{b}{r-1} \right\rfloor - a + 1 \\ = \; & \left\lfloor \frac{b}{r-1} \right\rfloor + 1 \\ \geq \; & 0. \end{aligned}$$

Damit ist gezeigt, daß $\beta(l)$ das Minimum bei $l = a$ annimmt.

□

Es stellt sich noch die Frage, wann $R(r,m)$ die Suchkomplexität des sequentiellen oder des parallelen Falles annimmt.

Satz 7.7 *Seien $r, m \in \mathbb{N}$. Ist $r = 1$, so ist $R(r,m) = 2^m - 1$. Ist $r \geq m$, so ist $R(r,m) = m$. Für alle $2 \leq r \leq m-1$ gilt*

$$m < R(r,m) < 2^m - 1.$$

Beweis: Die Fälle $r = 1$ und $r \geq m$ sind klar. Da $R(r, m)$ definitionsbedingt monoton fallend in r ist, berechnen wir für $m \geq 3$ die Werte $R(2, m)$ und $R(m-1, m)$. Es gilt

$$R(2,m) = (2 + (m \bmod 2))2^{\lfloor \frac{m}{2} \rfloor} - 2 = \begin{cases} 2^{1+\frac{m}{2}} - 2, & \text{falls } m \text{ gerade} \\ 3 \times 2^{\frac{m-1}{2}} - 2, & \text{falls } m \text{ ungerade.} \end{cases}$$

Dies ist immer kleiner als $2^m - 1$, da $m \geq 3$ ist.

$$\begin{aligned} R(m-1,m) &= ((m-1) + (m \bmod (m-1)))2^{\lfloor \frac{m}{m-1} \rfloor} - (m-1) \\ &= ((m-1)+1)2^1 - m + 1 \\ &= m + 1. \end{aligned}$$

□

7.4 Zusammenfassung

Wir haben in diesem Kapitel die Suchkomplexität von parallelen Algorithmen bei Bäumen untersucht. Wie im Kapitel 5 sind die ersten Ergebnisse ernüchternd. Der Individualtest ist bei vollständigen binären Bäumen optimal. Eine parallele Suche kann also aus der Baumstruktur keinen Nutzen ziehen.

Um diesen riesigen Unterschied zwischen der sequentiellen und der parallelen Suche zu überbrücken, haben wir die Suche auf mehrere Runden aufgeteilt. Zwischen den beiden Extremen haben wir damit eine Abstufung gefunden. Es zeigte sich dabei, daß die Suchkomplexität des sequentiellen Falles erst erreicht wird, wenn so viele Runden zur Verfügung stehen, daß pro Runde nur eine Frage gestellt werden muß.

Die vorgestellten optimalen Algorithmen für r Runden haben die angenehme Eigenschaft, daß sie maximal einmal während der Suche die Tiefe, in der sie Knoten im noch vorhandenen Baum fragen, ändern. Sie dürften sich darum effektiv implementieren lassen.

Kapitel 8

Parallele Suche in q-Bäumen

8.1 Einleitung

Wir verallgemeinern jetzt die Ergebnisse aus Kapitel 7 auf vollständige reguläre q-Bäume. Wie bei allen Suchprozessen haben die regulären q-Bäume nicht die von den binären Bäumen bekannte Eigenschaft, daß nach jeder Frage wieder ein regulärer Baum übrigbleibt. Wir betrachten daher etwas verallgemeinerte Bäume. Sie sind ab einer gewissen Tiefe regulär. Die Anzahl der Knoten, an denen sie nicht regulär sind, ist außerdem klein. Es wird sich zeigen, daß es immer einen optimalen Algorithmus gibt, der nach jeder Runde wieder einen solchen Baum erzeugt.

Definition 8.1 *Seien $m \in \mathbb{N}_0$, $q, d \in \mathbb{N}$, $q \geq 2$ und S ein Baum mit d Blättern, die alle die gleiche Tiefe in S haben. Dann sei $T(S, q, m)$ der Baum, der dadurch entsteht, daß in S jedes Blatt durch den vollständigen regulären (q^m, q)-Baum ersetzt wird.*

Die Eigenschaft, daß alle Blätter von S die gleiche Tiefe haben, läßt sich immer erreichen, indem an ein zu hoch liegendes Blatt von S ein Blatt angehängt wird. Der Baum ist dann nicht mehr irreduzibel. Die Suchkomplexität ändert sich jedoch nicht. Durch diese Forderung erreichen wir aber folgendes: Wenn zwei Knoten x und y in $T(S, q, m)$ die gleiche Tiefe t haben und diese Tiefe mindestens die Tiefe von S ist, so sind die zugehörigen Teilbäume T_x und T_y gleich groß. Bei einem Knoten in T, dessen Tiefe mindestens die Tiefe von S ist, ist also für die Suchkomplexität nur der Abstand zu den Blättern von S von Bedeutung. Hat der Baum S die Tiefe t_1 und ein Knoten x die Tiefe $t \geq t_1$, so sagen wir, daß x in der Ebene $t - t_1$ liegt. Die Blätter von S liegen damit in Ebene 0. Die Blätter von $T(S, q, m)$ liegen in Ebene m. Für alle $t \in \{0, \ldots, m\}$ hat $T(S, q, m)$ in Ebene t genau dq^t Knoten. Abbildung 8.1 zeigt einen Baum $T(S, q, m)$.

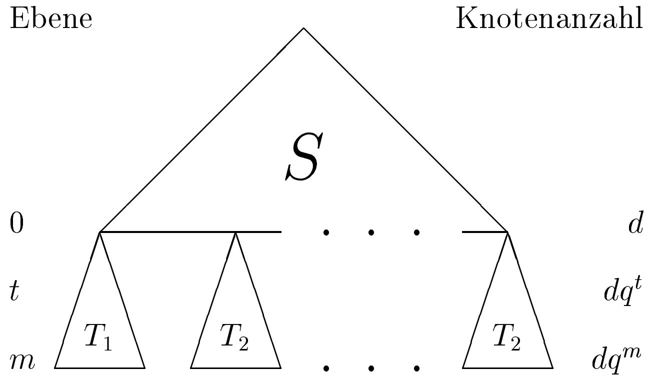

Abbildung 8.1: Ein Baum $T(S, q, m)$

Wir untersuchen für $r \geq 1$ die Suchkomplexität $L_r(T(S, q, m))$. Besonders interessiert sind wir dabei an dem Fall, daß S nur aus einem Blatt besteht. In diesem Fall ist $T(S, q, m)$ der vollständige q-reguläre Baum mit q^m Blättern. Wir geben im nächsten Abschnitt einen Suchalgorithmus an und berechnen seine Laufzeit. Danach zeigen wir, in welchen Fällen dieser Algorithmus optimal ist.

8.2 Obere Schranke

Wir definieren einen Suchalgorithmus $\mathcal{A}$ induktiv über r. Dazu verwenden wir ein Ergebnis aus Kapitel 6. Für einen (q^m, q)-Baum T gilt im sequentiellen Fall bei einem defekten Blatt nach Satz 6.3 $L(T) = (q - 1)m$. Diese Suchkomplexität läßt sich schon in m Runden erreichen, indem in jeder Runde $q - 1$ Knoten der Tiefe 1 in dem restlichen Baum gefragt werden. Sind also d, m, q und S wie in Definition 8.1 und $r \in \mathbb{N}$ gegeben, so arbeitet $\mathcal{A}$ wie folgt:

1. Ist $r \leq m$, so frage in Ebene $a = \lfloor \frac{m}{r} \rfloor$ alle bis auf einen Knoten. Ist $a < m$, so wende den nach Induktion für $r-1$ vorhandenen Algorithmus im restlichen Baum an.

2. Ist $r > m$, so wende einen optimalen Algorithmus für $r - m$ Runden in S an. Führe dann in dem übrigbleibenden (q^m, q)-Baum den oben beschriebenen Algorithmus für r Runden durch.

Zusammen mit der Laufzeitanalyse zeigen wir, daß dieser Algorithmus das defekte Blatt nach r Runden gefunden hat und nach jeder Runde der noch in Frage kommende Teilbaum ein Baum von der Form $T(S', q, m')$ ist.

Satz 8.2 *Der Algorithmus $\mathcal{A}$ ist wohldefiniert, und es gilt*

$$L(\mathcal{A}) = \begin{cases} (d - 1 + r + (q-1)(m \bmod r))\, q^{\lfloor \frac{m}{r} \rfloor} - r, & \textit{falls } r \leq m \\ L_{r-m}(S) + (q-1)m, & \textit{falls } r > m. \end{cases}$$

Beweis: Für $r > m$ findet $\mathcal{A}$ nach $r - m$ Runden das defekte Blatt von S. Dafür braucht er $L_{r-m}(S)$ Fragen. In T ist dies damit der einzige kontaminierte Knoten in Ebene 0. Für den damit verbleibenden regulären (q^m, q)-Baum benötigt der Algorithmus in den verbleibenden m Runden $(q-1)m$ Fragen. Danach ist das defekte Blatt gefunden.

Sei also $r \leq m$. Wir führen vollständige Induktion über r.

$\boldsymbol{r = 1}$**:** Es gilt $a = m$. Der Algorithmus fragt somit $dq^m - 1$ Blätter. Der Baum hat insgesamt dq^m Blätter. Das defekte Blatt ist damit nach einer Runde eindeutig identifiziert. Der Algorithmus ist also korrekt. Außerdem gilt für $r = 1$ und $m \geq r$

$$\begin{aligned} & (d - 1 + r + (q-1)(m \bmod r))\, q^{\lfloor \frac{m}{r} \rfloor} - r \\ = \; & (d + (q-1)0)\, q^m - 1 \\ = \; & dq^m - 1. \end{aligned}$$

$\boldsymbol{r \geq 2}$**:** Es gilt $1 \leq a \leq m/2 < m$. Der Algorithmus fragt in der ersten Runde in Ebene a genau $dq^a - 1$ Knoten. Es bleibt demzufolge nach der ersten Runde genau ein kontaminierter Knoten in Tiefe a übrig. Der dazugehörige Teilbaum hat die Größe q^{m-a}. Für ihn ist also nach Induktionsvoraussetzung ein Algorithmus vorhanden. Dieser braucht noch

$$((r-1) + (q-1)((m-a) \bmod (r-1)))\, q^{\lfloor \frac{m-a}{r-1} \rfloor} - (r-1)$$

Fragen[1]. Wir setzen $b := m - ar = (m \bmod r)$.

Ist $b < r-1$, so gilt $m-a = ar+b-a = a(r-1)+b$. Damit gilt $\lfloor \frac{m-a}{r-1} \rfloor = a$ und $((m-a) \bmod (r-1)) = b$. Insbesondere ist daher $m-a \geq r-1$. Folglich ist

$$\begin{aligned}
& L(\mathcal{A}) \\
= \; & dq^a - 1 + ((r-1) + (q-1)b)\, q^a - (r-1) \\
= \; & (d + (r-1) + (q-1)b)\, q^a - r.
\end{aligned}$$

Ist $b = r-1$, so gilt $m - a = ar + (r-1) - a = (a+1)(r-1)$. Damit gilt $\lfloor \frac{m-a}{r-1} \rfloor = a+1$ und $((m-a) \bmod (r-1)) = 0$. Wieder ist $m - a \geq r-1$ und damit

$$\begin{aligned}
& L(\mathcal{A}) \\
= \; & dq^a - 1 + (r-1)\, q^{a+1} - (r-1) \\
= \; & dq^a + (qr - q)\, q^a - r \\
= \; & (d + (r-1) + (q-1)(r-1))\, q^a - r \\
= \; & (d - 1 + r + (q-1)b)\, q^a - r.
\end{aligned}$$

□

Wir analysieren nun, wie der Ausdruck

$$a(m) = (d - 1 + r + (q-1)(m \bmod r))\, q^{\lfloor \frac{m}{r} \rfloor} - r$$

wächst. Ist $m \bmod r < r-1$, so gilt

$$a(m+1) - a(m) = (q-1) q^{\lfloor \frac{m}{r} \rfloor}.$$

Ist $m \bmod r = r-1$, so haben wir

$$\begin{aligned}
& a(m+1) - a(m) \\
= \; & (d-1+r) q^{1+\lfloor \frac{m}{r} \rfloor} - (d-1+r+(q-1)(r-1)) q^{\lfloor \frac{m}{r} \rfloor} \\
= \; & (qd - q + qr - d + 1 - r - qr + q + r - 1) q^{\lfloor \frac{m}{r} \rfloor} \\
= \; & d(q-1) q^{\lfloor \frac{m}{r} \rfloor}.
\end{aligned}$$

[1] Für diesen Baum ist der Parameter $d = 1$.

Insbesondere gilt also immer

$$a(m+1) \geq a(m) + (q-1)q^{\lfloor \frac{m}{r} \rfloor}. \tag{8.1}$$

Zu gegebenem Baum T kann es verschiedene Algorithmen geben. Dies liegt daran, daß T nicht eindeutig die Parameter S und m bestimmt. Wir zeigen in einem wichtigen Fall, daß die Laufzeit davon unabhängig ist. Es sei $d = 1$ und $m \geq 1$. Der (q^m, q)-Baum ergibt sich auch, wenn S der vollständige q-reguläre Baum der Tiefe 1 ist und m durch $m-1$ ersetzt wird.

Satz 8.3 *Für $d = 1$ und $m \geq 1$ ist die Laufzeit der beiden Algorithmen mit $|S| \leq q$ gleich.*

Beweis: Wir bezeichnen mit $\mathcal{A}$ den Algorithmus zu den Parametern $d = 1$ und m. $\mathcal{A}'$ sei der zu $d = q$ und $m-1$. S ist entsprechend nur ein Blatt, während S' der (q,q)-Baum ist. Es gilt dann $L(S) = 0$ und $L(S') = q-1$.

Ist $r > m$, so ist also $L(\mathcal{A}) = L(S) + (q-1)m = (q-1)m$. Da $r > m-1$ ist, gilt $L(\mathcal{A}') = L(S') + (q-1)(m-1) = (q-1) + (q-1)(m-1)$.

Ist $r = m$, so gilt wieder $L(\mathcal{A}) = (q-1)m = L(\mathcal{A}')$.

Ist $r < m$, so ist

$$L(\mathcal{A}) = (r + (q-1)(m \bmod r))\, q^{\lfloor \frac{m}{r} \rfloor} - r$$

und

$$L(\mathcal{A}') = ((q-1) + r + (q-1)((m-1) \bmod r))\, q^{\lfloor \frac{m-1}{r} \rfloor} - r.$$

Eine Fallunterscheidung $(m \bmod r) = 0$ und $(m \bmod r) > 0$ zeigt, daß diese Ausdrücke in beiden Fällen übereinstimmen.

□

8.3 Untere Schranke

Um die untere Schranke zu beweisen, analysieren wir wieder die erste Runde der Suche. Die betrachteten parallelen Algorithmen müssen das defekte Blatt nicht finden. Sie stellen lediglich die erste Runde eine Suche dar. Wir verwenden dafür die folgende Schreibweise:

Definition 8.4 *Sei $\mathcal{B}$ ein paralleler Suchalgorithmus für einen Baum T. Sei $\mathcal{O}$ ein Orakel. Dann ist $\mathcal{B}_{\mathcal{O}}(T)$ der Baum, der aus den noch als kontaminiert in Frage kommenden Knoten besteht. Insbesondere ist $\mathcal{B}_{Nein}(T)$ der übrigbleibende Baum, falls das Orakel nur „Nein“ antwortet.*

Bei einigen Suchalgorithmen $\mathcal{B}$ wird $\mathcal{B}_{\text{Nein}}(T) = \emptyset$ sein. Dies ist dann für den ganzen Baum kein zulässiges Orakel. Wir werden aber später wieder Orakel für den ganzen Baum aus Teilorakel für Teilbäume zusammensetzen. In manchen Teilbäumen ist es dann durchaus möglich, daß dort kein Blatt mehr übrigbleibt. Wir werden daher den Fall $\mathcal{B}_{\text{Nein}}(T) = \emptyset$ nicht ausschließen.

Mit dieser Notation definieren wir die minimale Länge von parallelen Algorithmen, die das Blatt bis auf eine gewünschte Genauigkeit finden. Wir erinnern daran, daß $L(\mathcal{B})$ die Anzahl der Fragen des Suchalgorithmusses $\mathcal{B}$ und $|T|$ die Anzahl der Blätter des Baumes T bezeichnet.

Definition 8.5 *Sei T ein Baum, $a, b \in \mathbb{N}_0$, $a \geq 1$ und $b \leq a$. Dann sei*

$$\begin{aligned} k(a,b,T) &= \min\{n \in \mathbb{N} : \exists \textit{ paralleler Algorithmus } \mathcal{B}\textit{, so daß} \\ &\qquad \textit{(i) } L(\mathcal{B}) \leq n, \\ &\qquad \textit{(ii) für alle Orakel } \mathcal{O} \textit{ gilt } |\mathcal{B}_{\mathcal{O}}(T)| \leq a \textit{ und} \\ &\qquad \textit{(iii) } |\mathcal{B}_{Nein}(T)| \leq b\}. \end{aligned}$$

Für einen Baum T und $a \in \mathbb{N}$ ist $k(a,a,T)$ damit die minimale Anzahl von Fragen, die ein paralleler Algorithmus stellen muß, damit danach bei allen möglichen Antworten nicht mehr als a Blätter vom Baum als defekt in Frage kommen. Insbesondere ist $k(1,1,T)$ die Suchkomplexität $L_{pre}(T)$.

Häufig ist es für den Fragenden unangenehmer, wenn das Orakel nur „Nein“ antwortet. Um dies zu kontrollieren, haben wir den Parameter b eingeführt.

Wir zeigen eine Rekursionsformel zur Berechnung von $k(a,b,T)$.

Satz 8.6 *Sei T ein Baum, $a, b \in \mathbb{N}_0$, $a \geq 1$ und $b \leq a$. Ist T ein Blatt, so ist*

$$k(a,b,T) = \begin{cases} 1, & \textit{falls } b = 0 \\ 0, & \textit{falls } b \geq 1. \end{cases}$$

Ist $T = (T_1, \ldots, T_q)$, so gilt

$$\begin{aligned} k(a,b,T) &= \min\left\{ k(a,b_1,T_1) + \ldots + k(a,b_q,T_q) \mid \sum_{i=1}^{q} b_i \leq b, \right. \\ &\qquad \left. 1 + k(a,b_1,T_1) + \ldots + k(a,b_q,T_q) \mid \sum_{i=1}^{q} b_i \leq a \right\}, \end{aligned}$$

wobei bei der Minimumsbildung für alle $i \in \{1, \ldots, q\}$ immer $b_i \geq 0$ sein muß.

Beweis: Ist T ein Blatt, so muß im Fall $b = 0$ eben dieses Blatt gefragt werden. Wird vom Orakel nur „Nein" geantwortet, dann bleibt kein Blatt übrig. Ist $b \geq 1$, so muß keine Frage gestellt werden.

Sei also $T = (T_1, \ldots, T_q)$.

„$\leq$": Wir nehmen zunächst an, daß das Minimum in Satz 8.6 bei

$$k(a, b_1, T_1) + \ldots + k(a, b_q, T_q)$$

mit $\sum_{i=1}^{q} b_i \leq b$ angenommen wird. Für jedes $i \in \{1, \ldots, q\}$ existiert zum Teilbaum T_i ein Algorithmus $\mathcal{B}_i$, so daß $L(\mathcal{B}_i) = k(a, b_i, T_i)$ ist. Für alle Orakel $\mathcal{O}$ gilt $|\mathcal{B}_{i\mathcal{O}}(T)| \leq a$ und $|\mathcal{B}_{i\mathrm{Nein}}(T)| \leq b$. Wir definieren somit den Algorithmus $\mathcal{B}$ als den, der parallel in allen T_i jeweils $\mathcal{B}_i$ fragt. Damit ist

$$L(\mathcal{B}) = \sum_{i=1}^{q} L(\mathcal{B}_i) = \sum_{i=1}^{q} k(a, b_i, T_i).$$

Ist $\mathcal{O}$ ein Orakel, das in einem Teilbaum T_i „Ja" antwortet, so gilt

$$|\mathcal{B}_{\mathcal{O}}(T)| = |\mathcal{B}_{i\mathcal{O}}(T_i)| \leq a.$$

Außerdem ist

$$\begin{aligned} |\mathcal{B}_{\mathrm{Nein}}(T)| &= |\mathcal{B}_{1\mathrm{Nein}}(T_1) \mathbin{\mathring{\cup}} \ldots \mathbin{\mathring{\cup}} \mathcal{B}_{q\mathrm{Nein}}(T_q)| \\ &= |\mathcal{B}_{1\mathrm{Nein}}(T_1)| + \ldots + |\mathcal{B}_{q\mathrm{Nein}}(T_q)| \\ &\leq b_1 + \ldots + b_q \\ &\leq b. \end{aligned}$$

$\mathcal{B}$ erfüllt damit die Voraussetzungen (ii) und (iii) aus Definition 8.5. Also ist

$$k(a, b, T) \leq L(\mathcal{B}) = \sum_{i=1}^{q} k(a, b_i, T_i).$$

Die andere Möglichkeit ist, daß das Minimum in Satz 8.6 bei

$$1 + k(a, b_1, T_1) + \ldots + k(a, b_q, T_q)$$

mit $\sum_{i=1}^{q} b_i \leq a$ angenommen wird.

Wie im letzten Fall existiert wieder zu jedem T_i ein Suchalgorithmus $\mathcal{B}_i$ mit allen oben erwähnten Eigenschaften. $\mathcal{B}$ fragt wiederum alle $\mathcal{B}_i$ parallel und zusätzlich die Wurzel des gesamten Baumes T. Damit ist

$$L(\mathcal{B}) = 1 + \sum_{i=1}^{q} L(\mathcal{B}_i) = 1 + \sum_{i=1}^{q} k(a, b_i, T_i).$$

Antwortet ein Orakel $\mathcal{O}$ in einem T_i „Ja“, so ist

$$|\mathcal{B}_{\mathcal{O}}(T)| = |\mathcal{B}_{i\mathcal{O}}(T_i)| \leq a.$$

Antwortet ein Orakel $\mathcal{O}$ in allen T_i nur „Nein“ und an der Wurzel „Ja“, so gilt

$$\begin{aligned} |\mathcal{B}_{\mathcal{O}}(T)| &= |\mathcal{B}_{1\text{Nein}}(T_1) \mathbin{\mathring{\cup}} \ldots \mathbin{\mathring{\cup}} \mathcal{B}_{q\text{Nein}}(T_q)| \\ &= |\mathcal{B}_{1\text{Nein}}(T_1)| + \ldots + |\mathcal{B}_{q\text{Nein}}(T_q)| \\ &\leq b_1 + \ldots + b_q \\ &\leq a. \end{aligned}$$

Außerdem ist

$$|\mathcal{B}_{\text{Nein}}(T)| = |\emptyset| = 0,$$

da die Wurzel von T gefragt wurde. Also erfüllt $\mathcal{B}$ wieder alle Voraussetzungen, und es gilt damit

$$k(a, b, T) \leq L(\mathcal{B}) = 1 + \sum_{i=1}^{q} k(a, b_i, T_i).$$

„$\geq$“: Sei $\mathcal{B}$ ein beliebiger optimaler Algorithmus, der alle drei Bedingungen aus der Definition 8.5 erfüllt. Wir bezeichnen für jedes $i \in \{1, \ldots, q\}$ mit $\mathcal{B}_i$ die Fragen, die in T_i gefragt werden und setzen $b_i = |\mathcal{B}_{i\text{Nein}}(T_i)|$. Folglich gilt für alle i

$$L(\mathcal{B}_i) \geq k(a, b_i, T_i).$$

Fragt der Algorithmus $\mathcal{B}$ die Wurzel des Baumes T, so wählen wir als Orakel $\mathcal{O}$ jenes, welches auf die Wurzel „Ja“ und sonst nur „Nein“ antwortet. Da $\mathcal{B}$ die Bedingung (ii) erfüllt, gilt $|\mathcal{B}_{\mathcal{O}}(T)| \leq a$. Also haben wir

$$a \geq |\mathcal{B}_{\mathcal{O}}(T)| = |\mathcal{B}_{1\text{Nein}}(T_1) \mathbin{\mathring{\cup}} \ldots \mathbin{\mathring{\cup}} \mathcal{B}_{q\text{Nein}}(T_q)| = b_1 + \ldots b_q.$$

Damit kommt

$$L(\mathcal{B}) = 1 + \sum_{i=1}^{q} L(\mathcal{B}_i) = 1 + \sum_{i=1}^{q} k(a, b_i, T_i)$$

in der Minimumsbildung vor.

Fragt $\mathcal{B}$ die Wurzel des Baumes T nicht, so gilt nach Voraussetzung an den Algorithmus $\mathcal{B}$

$$b \geq |\mathcal{B}_{\text{Nein}}(T)| \geq b_1 + \ldots b_q.$$

In diesem Fall kommt somit

$$L(\mathcal{B}) = \sum_{i=1}^{q} L(\mathcal{B}_i) = \sum_{i=1}^{q} k(a, b_i, T_i)$$

in der Minimumsbildung vor.

□

Zunächst haben wir eine einfache Tatsache über den Einfluß des Parameters b.

Korollar 8.7 *Sei T ein Baum, $a, b \in \mathbb{N}_0$, $a \geq 1$ und $b \leq a$. Dann ist*

$$k(a, a, T) \leq k(a, b, T) \leq k(a, a, T) + 1.$$

Beweis: Ist T ein Blatt, so gilt die Behauptung nach Satz 8.6. Ist T ein Baum $(T_1, \ldots, T_q)$, so gilt $k(a, a, T) \leq k(a, b, T)$, da in der Rekursionsformel jede mögliche Summe für $k(a, b, T)$ auch für $k(a, a, T)$ zugelassen ist. Ist andererseits $k(a, a, T) = \sum_{i=1}^{q} k(a, b_i, T)$ mit $\sum_{i=1}^{q} b_i \leq a$, so ist

$$k(a, b, T) \leq 1 + \sum_{i=1}^{q} k(a, b_i, T).$$

□

Wir zeigen im folgenden die schon am Ende von Kapitel 7 angedeutete Tatsache, daß ein paralleler Algorithmus, der unter n Blättern ein defektes Blatt finden soll, mindestens $n-1$ Fragen stellen muß. Dazu erinnern wir daran, daß $L_{pre}(T) = k(1, 1, T)$ ist.

Korollar 8.8 *Sei T ein beliebiger Baum, so gilt $k(1, 1, T) = |T| - 1$ und $k(1, 0, T) = |T|$.*

Beweis: Ist $|T| = 1$, also T ein Blatt, so gilt nach Satz 8.6 $k(1,1,T) = 0$ und $k(1,0,T) = 1$.

Ist $T = (T_1, \ldots, T_q)$, so ist in der Rekursionsformel für $k(1,1,T)$ genau ein $b_j = 1$, $j \in \{1, \ldots, q\}$. Alle anderen b_i sind 0. Bei der Berechnung von $k(1,0,T)$ sind bei der ersten Summe alle $b_i = 0$. Bei der zweiten Summe ist wieder nur genau ein $b_j = 1$. Nach Induktionsvoraussetzung ist aber $k(1,0,T_i) = |T_i|$ und $k(1,1,T_i) = |T_i| - 1$ für alle $i \in \{1, \ldots, q\}$.

□

Wir berechnen die Funktion k für die Bäume $T(S,q,m)$. Zunächst betrachten wir den Fall, daß S nur ein Blatt ist.

Korollar 8.9 *Seien $a, b, q, m \in \mathbb{N}_0$, $q \geq 2$, $m \geq 1$, $1 \leq a \leq q$ und $b < a$. Weiterhin sei T der vollständige reguläre (q^m, q)-Baum. Dann gilt*

$$k(a,a,T) = q^m - (a-1)q^{m-1} - 1$$

und

$$k(a,b,T) = k(a,a,T) + 1 = q^m - (a-1)q^{m-1}.$$

Beweis: Wir führen vollständige Induktion über m.

***m* = 1:** T besteht aus q Blättern. Es gilt somit $T = (T_1, \ldots, T_q)$ mit $|T_i| = 1$ für alle $i \in \{1, \ldots, q\}$. Es gilt damit für alle i: $k(a,0,T_i) = 1$ und $k(a,1,T_i) = 0$. Daraus folgt nach Satz 8.6 $k(a,a,T) = q - a$ und $k(a,b,T) = q - a + 1$.

***m* → *m* + 1:** Es gilt wieder $T = (T_1, \ldots, T_q)$. Nach Induktionsvoraussetzung ist für alle $i \in \{1, \ldots, q\}$

$$k(a,a,T_i) = q^{m-1} - (a-1)q^{m-2} - 1.$$

Für jedes $0 \leq b' < a$ gilt

$$k(a,b',T_i) = q^{m-1} - (a-1)q^{m-2}.$$

In Satz 8.6 kann bei der Minimumsbildung maximal ein $b_i = a$ sein. Also gilt

$$\begin{aligned} & k(a,a,T) \\ \geq\ & (q-1)(q^{m-1} - (a-1)q^{m-2}) + q^{m-1} - (a-1)q^{m-2} - 1 \\ =\ & q^m - (a-1)q^{m-1} - 1. \end{aligned}$$

Andererseits wird dieser Wert erreicht, indem $b_1 = a$ und alle anderen $b_i = 0$ für $i \in \{2, \ldots, q\}$ gesetzt wird.

Bei der Minimumsbildung für $k(a, b, T)$ sind in der ersten Summe alle $b_i < a$. Die zweite Summe ist nach dem schon gezeigten genau $k(a, a, T) + 1$. Damit folgt, daß $k(a, b, T) = k(a, a, T) + 1$ ist.

□

Wir erweitern dieses Ergebnis nun auf die Bäume $T(S, q, m)$.

Korollar 8.10 *Seien $a, b, d, q, m \in \mathbb{N}_0$, $q \geq 2$, $d, m \geq 1$, $1 \leq a \leq q$, $b < a$ und S ein Baum mit $|S| = d$. Weiterhin sei $T = T(S, q, m)$. Dann gilt*

$$k(a, a, T) = dq^m - d(a-1)q^{m-1} - 1$$

und

$$k(a, b, T) = k(a, a, T) + 1 = dq^m - d(a-1)q^{m-1}.$$

Insbesondere hängt $k(a, b, T)$ nicht von der Struktur von S, sondern nur von $d = |S|$ ab.

Beweis: Ist S nur ein Blatt, so ist $d = 1$, und die Aussage ist eben die aus Korollar 8.9.

Ist $S = (S_1, \ldots, S_l)$, so setzen wir für jedes $j \in \{1, \ldots, l\}$: $d_j = |S_j|$ und T_j als den an S_j hängenden Baum. Nach Induktionsvoraussetzung gilt dann für alle j

$$k(a, a, T_j) = d_j q^m - d_j(a-1)q^{m-1} - 1,$$

und für alle $0 \leq b' < a$ gilt

$$k(a, b', T_j) = d_j q^m - d_j(a-1)q^{m-1}.$$

Die Argumentation ist damit die gleiche wie im letzten Beweis: Bei der Rekursionsformel von $k(a, a, T)$ aus Satz 8.6 kann maximal ein $b_i = a$ sein. Bei $k(a, b, T)$ sind wieder entweder alle $b_i < a$, oder es wird die zweite Summe genommen. In beiden Fällen ist $k(a, b, T) = k(a, a, T) + 1$.

□

Dieses Korollar ermöglicht damit die folgende Definition.

Definition 8.11 *Sei $a, d, q, m \in \mathbb{N}$, $q \geq 2$, $a, d \leq q$ und S ein Baum mit $|S| = d$. Dann sei*

$$k(a, d, q, m) = k(a, a, T(S, q, m)) = dq^m - d(a-1)q^{m-1} - 1.$$

Wir weisen darauf hin, daß diese Definition nicht $m = 0$ zuläßt, obwohl der Baum $T(S, q, 0)$ definiert ist. Ihn werden wir hinterher gesondert betrachten.

Damit wissen wir genügend über die erste Runde, um beliebige Algorithmen für r Runden zu betrachten. Zunächst kommt dazu ein sehr technisches Lemma. Es zeigt, daß optimale Algorithmen in der ersten Runde nicht zu tief im Baum fragen.

Lemma 8.12 *Seien $q, m \in \mathbb{N}_0$, S ein Baum, $T = T(S, q, m)$ und $\mathcal{B}$ ein optimaler Algorithmus für r Runden für die Suche in T. Sei $t \in \{0, \ldots, m\}$ minimal, so daß $\mathcal{B}$ in der ersten Runde maximal $dq^t - 1$ Fragen stellt. $\mathcal{B}$ frage in der ersten Runde einen Knoten x, der in einer Ebene größer als t liegt. Der Vorfahre von x in Ebene t sei y. Dann existiert ein optimaler Algorithmus $\mathcal{C}$ für r Runden, der sich in der ersten Runde von $\mathcal{B}$ nur dadurch unterscheidet, daß er y anstatt x fragt.*

Beweis: Zunächst ist t wohldefiniert, da $\mathcal{B}$ maximal $|T| - 1 = dq^m - 1$ Fragen stellt. Damit $\mathcal{B}$ einen solchen Knoten x fragen kann, muß $t < m$ gelten.

Der Suchalgorithmus $\mathcal{C}$ fragt in der ersten Runde den Knoten y und bis auf x alle Knoten, die $\mathcal{B}$ in der ersten Runde fragt. Ab der zweiten Runde sei $\mathcal{C}$ ein beliebiger optimaler Algorithmus für $r - 1$ Runden.

Sei $\mathcal{O}$ ein optimales Orakel für $\mathcal{C}$. Wir konstruieren ein Orakel $\mathcal{P}$ für $\mathcal{B}$, das mindestens die gleiche Laufzeit erzwingt wie $\mathcal{O}$ bei $\mathcal{C}$. Da $\mathcal{O}$ optimal ist, folgt damit $L(\mathcal{C}) \leq L(\mathcal{B})$.

Für das Orakel $\mathcal{O}$ gibt es zwei Fälle:

Fall 1: $\mathcal{O}$ antwortet auf y mit „Nein“. $\mathcal{P}$ antwortet auf x „Nein“ und übernimmt alle anderen Antworten von $\mathcal{O}$ aus der ersten Runde. Der für $\mathcal{B}$ nach der ersten Runde verbleibende Baum hat dann den für $\mathcal{C}$ verbleibenden Baum als Teilbaum. Damit gilt $L(\mathcal{C}) \leq L(\mathcal{B})$.

Fall 2 : $\mathcal{O}$ antwortet auf y mit „Ja“. Nach der ersten Runde bleibt somit ein (q^{m-t}, q)-Baum übrig. Ein Orakel $\mathcal{P}$ für den Algorithmus $\mathcal{B}$ muß also nur einen zu einem Knoten in Ebene t gehörenden Teilbaum übriglassen. Nach Voraussetzung hat $\mathcal{B}$ erst maximal $dq^t - 1$ Fragen gestellt. Die Ebene t hat jedoch dq^t Knoten. Es gibt demnach einen Teilbaum, in dem noch keine Frage gestellt worden ist. $\mathcal{P}$ antwortet so, daß dieser Teilbaum noch kontaminiert sein kann. Damit gilt wieder $L(\mathcal{C}) \leq L(\mathcal{B})$.

□

Dieses Lemma läßt sich solange anwenden, bis der optimale Algorithmus in der ersten Runde nur noch Fragen in Ebenen kleiner-gleich t stellt. Von jetzt an nehmen wir das bei optimalen Algorithmen immer an. Der in Satz 8.2 analysierte Algorithmus $\mathcal{A}$ erfüllt dieses Kriterium.

Wir wollen nicht unerwähnt lassen, daß das Lemma 8.12 eventuell über die leere Menge spricht, es also keinen optimalen Algorithmus $\mathcal{B}$ gibt, der derart tief liegende Knoten fragt.

Wir zeigen noch eine für die Induktion wichtige Eigenschaft eines optimalen Algorithmusses.

Lemma 8.13 *Seien $m, q \in \mathbb{N}_0$, $q \geq 2$, S ein Baum, $T = T(S, q, m)$ und $r \in \mathbb{N}$. Ist $r > m$, so existiert ein optimaler Algorithmus für r Runden, der in der ersten Runde nur in S fragt.*

Beweis: Sei wieder $d = |S|$. Ein möglicher Algorithmus $\mathcal{A}$ fragt $r-m$ Runden lang nur in S, um dort den kontaminierten Knoten zu finden. In den restlichen m Runden findet er dann mit $(q-1)m$ Fragen das defekte Blatt. Es gilt also

$$L_r(T) \leq L_{r-s}(S) + (q-1)m \leq (d-1) + (q-1)m.$$

Sei $\mathcal{B}$ ein beliebiger optimaler Algorithmus. $\mathcal{B}$ stelle in der ersten Runde κ Fragen. Ist $\kappa < d$, so folgt durch wiederholte Anwendung von Lemma 8.12 auf $\mathcal{B}$, daß der gesuchte Algorithmus existiert.

Ist $\kappa \geq d$, so zeigen wir, daß $\mathcal{B}$ mindestens $d - 1 + (q-1)m$ Fragen stellt. Damit ist $\mathcal{B}$ nicht besser als $\mathcal{A}$. Also ist in diesem Fall $\mathcal{A}$ optimal und fragt in der ersten Runde nur in S.

Jedes der d Blätter $s_1, \ldots, s_d$ von S ist Wurzel des Teilbaumes $T_j := T_{s_j}$. Wir setzen für alle $i = 1, \ldots, d$ e_i als die Anzahl der Fragen, die im Teilbaum von T_i gefragt werden. Falls die Wurzel des Teilbaums, also das Blatt von S gefragt wird, so zählen wir das bei e_i nicht mit. Weiterhin sei

$$e = \min\{e_i : i = 1, \ldots, d\}.$$

Fall 1: $\boldsymbol{e = 0}$. Es existiert ein $j \in \{1, \ldots, d\}$ mit $e_j = 0$. Das Orakel antwortet so, daß nach den κ Fragen der komplette Teilbaum T_j übrig bleibt. Somit gilt

$$L(\mathcal{B}) \geq \kappa + (q-1)m > d - 1 + (q-1)m.$$

Fall 2: $\boldsymbol{e > 0}$. Das Orakel wählt ein $j \in \{1, \ldots, d\}$ mit $e_j = e$. Es antwortet so, daß die Wurzel von T_j kontaminiert ist. Die Fragen in T_j werden wie bei der üblichen sequentiellen Suche nach einem Blatt beantwortet. In den folgenden Runden sind also in T_j noch mindestens $(q-1)m-e$ Fragen notwendig. In jedem der d Teilbäume sind in der ersten Runde mindestens e Fragen gestellt worden. Damit gilt

$$L(\mathcal{B}) \geq de + (q-1)m - e = (d-1)e + (q-1)m \geq d - 1 + (q-1)m.$$

□

Wir können jetzt die Suchkomplexität für $1 \leq |S| \leq q$ berechnen.

Satz 8.14 *Der in Satz 8.2 analysierte Algorithmus ist für* $1 \leq d \leq q$ *optimal.*

Beweis: Ist $m = 0$, so ist $T = S$ und $L(\mathcal{A}) = L_r(S)$. Dies ist per Definition optimal. Sei also $m \geq 1$. Wir zeigen mit vollständiger Induktion über r, daß $\mathcal{A}$ optimal ist.

Ist $r = 1$, so ist $L(\mathcal{A}) = dq^m - 1 = |T| - 1$. Dies ist nach Korollar 8.8 optimal.

Sei also $r \geq 2$ und $\mathcal{B}$ ein beliebiger optimaler Algorithmus für r Runden. Zunächst nehmen wir ohne Einschränkung an, daß $\mathcal{B}$ gemäß der Lemmata 8.12 und 8.13 ausgewählt wurde.

Wir setzen wieder κ als die Anzahl der Fragen, die $\mathcal{B}$ in der ersten Runde fragt. $t \in \mathbb{N}_0$ sei minimal, so daß $\kappa \leq dq^t - 1$ ist. Es gilt $0 \leq t \leq m$ (siehe Abbildung 8.1).

Wir setzen R als den Baum $T(S, q, t)$. Damit gilt $T = T(R, q, m-t)$. Nach Lemma 8.12 stellt $\mathcal{B}$ nur Fragen in R. Nach den Antworten des Orakels bleibt von R der Baum R' übrig. Der insgesamt übrigbleibende Baum T' ist dann $T(R', q, m-t)$. Da der Algorithmus in der ersten Runde nur in R gefragt hat, haben wir für alle $\varrho \in \mathbb{N}$, $\varrho \geq 2$ die Ungleichung

$$L_\varrho(R) \leq \kappa + L_{\varrho-1}(R'). \tag{8.2}$$

Wir wählen ein Orakel, das die Größe von $|R'|$ maximiert. Wir setzen diese Größe als $\delta = |R'|$. Ist $t > 0$, so muß $\mathcal{B}$ in der ersten Runde nach Definition 8.11 mindestens $k(\delta, d, q, t)$ Fragen stellen.

Damit haben wir die Ungleichung

$$\kappa \geq k(\delta, d, q, t). \tag{8.3}$$

Es gilt dann

$$L(\mathcal{B}) \geq \kappa + L_{r-1}(T') = \kappa + L_{r-1}(T(R', q, m-t)). \tag{8.4}$$

Wir zeigen, daß dies mindestens $L(\mathcal{A})$ ist.

Fall 1: $r > m+1$. $\mathcal{B}$ fragt nach Lemma 8.13 nur in S. Es gilt also $t = 0$. Damit ist $R = S$. Außerdem ist $r - 1 > m = m - t$. Wir haben infolgedessen nach Induktionsvoraussetzung und Ungleichung (8.2):

$$\begin{aligned} & L(\mathcal{B}) \\ = \; & \kappa + L_{r-1}(T(R', q, m)) \\ = \; & \kappa + L_{r-1-m}(R') + (q-1)m \\ \geq \; & L_{r-m}(R) + (q-1)m \\ = \; & L(\mathcal{A}). \end{aligned}$$

Fall 2: $r = m + 1$. Wie im Fall 1 ist $t = 0$. Es gilt nach Induktionsvoraussetzung

$$L_m(T') \geq (\delta - 1)q + (q-1)m.$$

Außerdem ist $L_1(S) = d - 1 < q$. Für $\delta \geq 2$ ist $L(\mathcal{B}) > L(\mathcal{A})$. Also ist $\mathcal{B}$ nicht optimal. Für $\delta = 1$ ist nach Korollar 8.8 $\kappa = d - 1$ und damit $L(\mathcal{B}) = L(\mathcal{A})$.

Fall 3: $r \leq m$ und $1 \leq t \leq \lfloor \frac{m}{r} \rfloor$. Zunächst gilt

$$\frac{r-1}{m-t} \leq \frac{r-1}{m - \frac{m}{r}} = \frac{r-1}{m - \frac{m}{r}} = \frac{r}{m} \leq 1. \tag{8.5}$$

Also ist $r - 1 \leq m - t$. Wir setzen $\delta' = \min\{\delta, q\}$ und behaupten

$$\kappa \geq dq^t - d(\delta' - 1)q^{t-1} - 1 =: A, \tag{8.6}$$

und für $B := L_{r-1}(T')$ gilt

$$B \geq (\delta' - 1 + (r-1) + (q-1)((m-t) \bmod (r-1)))q^{\lfloor \frac{m-t}{r-1} \rfloor} - (r-1). \tag{8.7}$$

Für $\delta' = \delta$, also $\delta \leq q$, folgt Gleichung (8.6) aus Gleichung (8.3) und Gleichung (8.7) aus der Induktionsvoraussetzung für $r-1$. Ist $\delta > q$, so behauptet Gleichung (8.6) $\kappa \geq dq^{t-1} - 1$. Nach Definition von t gilt aber sogar $\delta \geq dq^{t-1}$. Für Gleichung (8.7) wählen wir aus R' einen beliebigen Teilbaum R'' mit q Blättern aus. Dann ist $T'' = T(R'', q, m - t)$ ein Teilbaum von T', und die Gleichung (8.7) gilt nach Induktionsvoraussetzung sogar für T''.

Aus Ungleichung (8.4) folgt, daß $L(\mathcal{B}) \geq A + B$ ist.

Wir zeigen zunächst, daß der Ausdruck $A + B$ minimal für $\delta' = 1$ ist. Wir betrachten dazu die Veränderung, wenn δ' zu $\delta' + 1$ wird. A wird um

$dq^{t-1} \leq q^t$ kleiner. B wächst um $q^{\lfloor \frac{m-t}{r-1} \rfloor} \geq q^t$. Ersetzen wir in $A + B$ δ' durch 1, so folgt

$$L(\mathcal{B}) \geq \underbrace{dq^t - 1}_{(I)} + \underbrace{(r-1+(q-1)((m-t) \bmod (r-1)))q^{\lfloor \frac{m-t}{r-1} \rfloor} - (r-1)}_{(II)}. \tag{8.8}$$

Wir zeigen, daß der Ausdruck $(I) + (II)$ minimal für $t = \lfloor \frac{m}{r} \rfloor$ ist, wobei wir alle $t \in \{0, \ldots, \lfloor \frac{m}{r} \rfloor\}$ zulassen. $t = 0$ ist dabei schon eine Vorarbeit für Fall 4.

Ist $t \geq 1$, und ersetzen wir t durch $t-1$, so wächst (II) um $(q-1)q^{\lfloor \frac{m-t}{r-1} \rfloor}$. Dies zeigt wieder eine Fallunterscheidung $(m-1) \bmod (r-1) < r-2$ und $(m-1) \bmod (r-1) = r-2$. Andererseits verringert sich (I) um $(q-1)dq^{t-1}$. Dies ist maximal $(q-1)q^t$. Mit Gleichung (8.5) folgt also, daß sich der Ausdruck $(I) + (II)$ nicht verringert. Der minimale Wert von $(I) + (II)$ ergibt sich somit, wenn wir für $t = \lfloor \frac{m}{r} \rfloor$ einsetzen. Dies haben wir im Beweis zu Satz 8.2 bereits berechnet. Es ist genau $L(\mathcal{A})$.

Fall 4: $r \leq m$ und $t = 0$. Aus der Definition von t folgt $\kappa < d$. Nach Induktionsvoraussetzung folgt

$$L_{r-1}(T') = (\delta - 1 + (r-1) + (q-1)(m \bmod (r-1)))q^{\lfloor \frac{m}{r-1} \rfloor} - (r-1).$$

Die Summe $\kappa + L_{r-1}(T')$ wird wieder für $\delta = 1$ minimal, da ein größeres δ den Wert von $L_{r-1}(T')$ um mindestens q erhöht und $\kappa < d \leq q$ ist. Genau hier versagt die Induktion übrigens, wenn $d > q$ wäre.

Ist $\delta = 1$, so ist wieder wie oben nach Korollar 8.8 $\kappa = d - 1$. Es gilt also

$$L(\mathcal{B}) \geq d - 1 + ((r-1) + (q-1)(m \bmod (r-1)))q^{\lfloor \frac{m}{r-1} \rfloor} - (r-1).$$

Dies ist der Ausdruck aus Gleichung (8.8) für $t = 0$. Damit ist auch in diesem Fall $L(\mathcal{B}) \geq L(\mathcal{A})$.

Fall 5: $r \leq m$ und $\lfloor \frac{m}{r} \rfloor < t \leq m$. Wir wenden zunächst Lemma 8.13 an. Dazu setzen wir $S' = T(S, q, m-r+1)$. Dann ist $T = T(S', q, r-1)$. Es gibt also einen optimalen Algorithmus für r Runden, der in der ersten Runde nur in S' fragt. In unserem Fall bedeutet dies, daß $t \leq m - r + 1$ ist. Insbesondere ist demnach $t \leq m - 1$, da $r \geq 2$ ist. Wir wissen also $\lfloor \frac{m}{r} \rfloor < t \leq m - r + 1$. Falls es ein solches t nicht geben kann, haben wir Fall 5 mit Lemma 8.13 bereits auf einen anderen Fall zurückgeführt.

Existieren solche t, so gilt

$$\frac{r-1}{m-t} > \frac{r-1}{m - \frac{m}{r}} = \frac{r-1}{m - \frac{m}{r}} = \frac{r}{m}. \tag{8.9}$$

Wir setzen wieder $\delta' = \min\{\delta, q\}$. Es gelten damit die Gleichungen (8.6) und (8.7). Für (8.7) ist lediglich zu beachten, daß $t \leq m - r + 1$, also $r - 1 \leq m - t$ ist. Wir minimieren wieder die Summe $A + B$. Ersetzen wir δ' durch $\delta' + 1$, so verringert sich A um dq^{t-1}. Dies ist mindestens $dq^{\lfloor \frac{m}{r} \rfloor}$. B wächst um $q^{\lfloor \frac{m-t}{r-1} \rfloor}$. Dies ist nach Gleichung (8.9) maximal $q^{\lfloor \frac{m}{r} \rfloor}$. Ist t gegeben, so wird die Summe $A + B$ für $\delta' = q$ minimal.

Ist aber $\delta' = q$, so ist

$$\kappa \geq dq^{t-1} \tag{8.10}$$

und

$$L_{r-1}(T') \geq (q-1+(r-1)+(q-1)((m-t) \bmod (r-1)))q^{\lfloor \frac{m-t}{r-1} \rfloor} - (r-1). \tag{8.11}$$

Setzen wir stattdessen in die Gleichungen (8.6) und (8.7) $\delta' = 1$ und statt t den Wert $t - 1$ ein und benutzen die Gleichung (8.1), so erhalten wir

$$\kappa \geq dq^{t-1} - 1$$

und

$$\begin{aligned} & L_{r-1}(T') \\ \geq\ & ((r-1) + (q-1)((m-t+1) \bmod (r-1)))q^{\lfloor \frac{m-t+1}{r-1} \rfloor} - (r-1) \\ \geq\ & ((r-1) + (q-1)((m-t) \bmod (r-1)))q^{\lfloor \frac{m-t}{r-1} \rfloor} + (q-1)q^{\lfloor \frac{m-t}{r-1} \rfloor} - r + 1 \\ =\ & (q - 1 + (r-1) + (q-1)((m-t) \bmod (r-1)))q^{\lfloor \frac{m-t}{r-1} \rfloor} - (r-1). \end{aligned}$$

Diese beiden Schranken sind nicht größer als die aus den Gleichungen (8.10) und (8.11).

Ist also $t > \lfloor \frac{m}{r} \rfloor$, so ist der minimale Wert von $A + B$ nicht kleiner als der minimale Wert von $A + B$ für $t - 1$. Betrachten wir also alle in Fall 5 vorkommenden t und zusätzlich $t = \lfloor \frac{m}{r} \rfloor$, so wird der minimale Wert von $A + B$ wieder bei $t = \lfloor \frac{m}{r} \rfloor$ angenommen. In Fall 3 haben wir schon gesehen, daß der Wert dann $L(\mathcal{A})$ ist.

□

Mit diesem Satz können wir insbesondere die Suchkomplexität bei r Runden für vollständige q-reguläre Bäume errechnen. Wir setzen dazu den Baum S als Blatt, woraus $d = 1$ folgt. Setzen wir $R(r, q, m)$ als die Suchkomplexität, so erhalten wir

Korollar 8.15 *Ist $m, q, r \in \mathbb{N}_0$, $q \geq 2$ und $r \geq 1$, dann ist*

$$R(r,q,m) = \begin{cases} (r + (q-1)(m \bmod r))\, q^{\lfloor \frac{m}{r} \rfloor} - r, & \textit{falls } r \leq m \\ (q-1)m, & \textit{falls } r > m. \end{cases}$$

Um den Induktionsbeweis zu führen, haben wir auch andere Bäume S zugelassen. Es zeigte sich, daß der Algorithmus $\mathcal{A}$ optimal ist, falls $|S| \leq q$ ist. Ist aber $r \geq m+2$, so hat $\mathcal{A}$ in S mindestens 2 Runden Zeit zu fragen. Wie die Fragen der ersten $m-r$ Runden bei $\mathcal{A}$ aussehen, haben wir nicht konstruktiv angegeben. Dies würde eine genau Kenntnis der Baumstruktur von S erfordern.

Ist schließlich $|S| > q$, so muß der Algorithmus $\mathcal{A}$ nicht mehr optimal sein. $\mathcal{A}$ fragt solange in S, bis von S nur noch genau ein Blatt übrigbleibt. Dies ist aber für $|S| > q$ nicht mehr unbedingt optimal.

Nebenbei beantwortet der Satz im Fall $|S| \leq q$ auch die Frage, nach wievielen Runden die sequentielle Suchkomplexität erreicht wird. Diese liegt bei $L(S) + (q-1)m$. Dafür sind maximal $L(S) + m$ Runden notwendig. Für $q \geq 3$ stimmen diese beiden Zahlen also, anders als im binären Fall, nicht mehr überein.

8.4 Zusammenfassung

Wir haben in diesem Kapitel die Fragestellung des letzten Kapitels auf vollständige reguläre q-Bäume verallgemeinert. Wir konnten wieder die Suchkomplexität bestimmen und einen optimalen Algorithmus angeben. Dazu haben wir nicht nur reguläre Bäume betrachtet. Die von uns betrachteten Bäume können an der Wurzel zunächst einen kleinen nichtregulären Baum besitzen. Dieser Baum darf maximal q Blätter haben. An jedem Blatt „hängt“ dann ein vollständiger regulärer q-Baum mit q^m Blättern. Diese Menge ermöglicht eine rekursive Berechnung der Suchkomplexität für r Runden, da nach jeder Runde entweder wieder ein Baum von dieser Form entsteht oder der Algorithmus als nicht optimal nachgewiesen werden kann.

Mit den erweiterten Mitteln dieses Kapitels konnten wir außerdem zeigen, daß bei jedem beliebigem Baum mit n Blättern jeder parallele Algorithmus $n-1$ Fragen stellen muß. Wir haben dazu für die allgemeinere Frage „Wieviele parallele Fragen sind notwendig, damit nur noch $a \in \mathbb{N}$ Blätter als defekt in Frage kommen?“ eine Rekursionsformel hergeleitet. Dabei zeigte sich, daß das Orakel, welches nur „Nein“ antwortet, eine eigene Analyse erfordert. Dies zeigte sich bereits bei binären Bäumen in Lemma 7.3.

Kapitel 9

Suchen mit Lügen

9.1 Einleitung

Eine interessante Modifikation der kombinatorischen Suche ergibt sich, wenn das Orakel lügen darf. Wenn es dies uneingeschränkt darf, ist offensichtlich keine erfolgreiche Suche möglich. Die Anzahl der erlaubten Lügen wird daher beschränkt. Wir werden in diesem Kapitel eine Lüge bei der Suche in binären Bäumen zulassen.

Stanislaw Ulam hat in [ULA–76] das erste Mal ein Problem erwähnt, bei dem der antwortende Spieler lügen darf. Die Problemstellung war: „Wieviele Fragen der Form 'Ist $x \leq a$?' mit $a \in \{1, \ldots, 1000000\}$ sind notwendig, um eine Zahl zwischen 1 und 1000000 zu finden, wenn der Antwortende einmal lügen darf?".

Andrzej Pelc hat in [PEL–87] gezeigt, daß 25 Fragen ausreichend und notwendig sind, wenn der Fragende beliebige Teilmengen von $\{1, \ldots, 1000000\}$ fragen darf. Um die untere Schranke zu beweisen, werden nach jeder Antwort die Elemente der Menge $\{1, \ldots, 1000000\}$ auf die Mengen X_0, X_1 und X_2 aufgeteilt. X_0 enthält dabei die Elemente, die bisher mit allen Antworten verträglich sind. X_1 enthält diejenigen, bei denen schon einmal gelogen worden ist, wenn sie das gesuchte Element sind. X_2 beinhaltet schließlich die Elemente, die selbst bei einer Lüge nicht mehr in Frage kommen. Wir werden diese Aufteilung später ebenfalls anwenden.

Als nächstes verwendet Pelc die gewichtete Größe der Mengen X_0 und X_1, um die Anzahl der noch notwendigen Fragen abzuschätzen. Die daraus gewonnenen unteren Schranken für die Suchkomplexität sind bei der Suche in Bäume jedoch schlecht. Wir haben darum diese Beweisidee nicht übernommen.

Dieses Problem hat viele weitere Fragestellungen motiviert.

In [AIG–95] hat Aigner gezeigt, daß sogar 25 Fragen der Form „Ist $x \leq a$?" ausreichen, um das Element zu finden.

In [AIG–96] hat Aigner die kombinatorische Suche mit Lügen untersucht, wenn die Fragen mehr als zwei mögliche Antworten haben. Durch eine Erweiterung der Gewichtsfunktion konnte Aigner für beliebige Mengen $\{1, \ldots, n\}$ und Antworten aus der Menge $\{1, \ldots, q\}$ die Suchkomplexität bei einer erlaubten Lüge bestimmen. Für die parallele Suche konnte Aigner im Fall, daß q eine Primzahlpotenz ist, die Suchkomplexität bis auf eine Frage genau bestimmen. Sie liegt interessanter Weise maximal eins über der Suchkomplexität im sequentiellen Fall.

Das ursprüngliche Problem von Ulam ist mittlerweile für eine beliebige Anzahl von erlaubten Lügen gelöst. Eine Übersicht über die Suchkomplexitäten steht in [HIL–95].

Wir untersuchen in diesem Kapitel die Suchkomplexität in vollständigen binären Bäumen, wenn das Orakel einmal lügen darf. Hat der Baum 2^m Blätter, $m \in \mathbb{N}$, so bezeichnen wir die Suchkomplexität mit $E(m)$.

9.2 Notation

Wir verwenden in diesem Kapitel die für diese Suchprozesse oben bereits erwähnte Notation. Ist $(S, \mathcal{F})$ ein Suchprozeß, bei dem das Orakel maximal $e \in \mathbb{N}_0$ lügen darf, so wird die Menge S nach jeder Antwort in die Mengen $X_0, \ldots, X_e, X_{e+1}$ partitioniert. Für jedes $i \in \{0, \ldots, e+1\}$ besteht dabei die Menge X_i aus den Elementen x, bei denen bisher genau i mal gelogen worden ist, falls x das gesuchte Element ist. Da maximal e mal gelogen werden darf, besteht die Menge X_{e+1} aus den Elementen, die nicht mehr defekt sein können. Der Suchprozeß kann erst dann enden, wenn die Mengen $X_0, \ldots, X_e$ zusammen nur noch ein Element beinhalten. Dies ist dann das gesuchte Element.

Sind die Mengen $X_0, \ldots, X_{e+1}$ gegeben, so sei $C(X_0, \ldots, X_{e+1})$ die Anzahl der noch notwendigen Fragen bei Kenntnis dieser Partition.

9.3 Obere Schranke

Für die obere Schranke geben wir wieder von m abhängige Suchalgorithmen an. Um sie formulieren zu können, brauchen wir zunächst eine Partition der natürlichen Zahlen.

Definition 9.1 *Sei* $B_1 = \{1\}$*. Ist für* $i \geq 1$ $B_i = \{a, \ldots, b\}$ *mit* $a, b \in \mathbb{N}$*, so sei* $B_{i+1} = \{b+1, \ldots, 2b-a+2\}$*.*

Wir zeigen, daß dies wirklich eine Partition ist, zu jedem $m \in \mathbb{N}$ also genau ein $i \in \mathbb{N}$ existiert, so daß $m \in B_i$ gilt.

Satz 9.2 *Es gilt für alle* $i \in \mathbb{N}$

$$B_i = \{1 + \sum_{j=1}^{i-1} j, \ldots, \sum_{j=1}^{i} j\} = \{1 + \frac{(i-1)i}{2}, \ldots, \frac{i(i+1)}{2}\}.$$

Insbesondere gilt $|B_i| = i$*, und die* B_i *bilden eine Partition von* $\mathbb{N}$*, wobei für alle* $i, j, x, y \in \mathbb{N}$ *mit* $x \leq y$*,* $x \in B_i$ *und* $y \in B_j$ *immer* $i \leq j$ *gilt. Die Partition ist in diesem Sinne also monoton.*

Beweis: Die Gleichung für B_i wird durch vollständige Induktion bewiesen. Die anderen Behauptungen folgen dann direkt aus dieser Gleichung.

$\square$

Wir definieren im folgenden ein Menge von Suchalgorithmen, unter denen sich auch ein optimaler befindet. Dazu benötigen wir zunächst einige Überlegungen. Das Orakel darf maximal einmal lügen. Die noch in Frage kommenden Blätter vom Baum T liegen also entweder in X_0 oder in X_1. Wird ein Teilbaum U gefragt, so ändern sich die Mengen, je nach Antwort, folgendermaßen:

Ist die Antwort „Ja“, so behalten alle Blätter, die in U liegen, ihre Zugehörigkeit bei. Blätter, die nicht in U liegen, wandern von X_0 nach X_1 und von X_1 nach X_2.

Ist die Antwort „Nein“, so behalten Blätter außerhalb von U ihre Zugehörigkeit, während die Blätter von U verschoben werden.

Definition 9.3 *Seien* $k, m \in \mathbb{N}$ *mit* $k \leq m$*. Dann ist* $\mathcal{A}_{k,m}$ *der folgende Algorithmus:*

1. *Frage k mal die Hälfte vom Baum X_0.*

2. *Frage den ganzen Baum X_0.*

3. *Ist $X_0 = \emptyset$, so wende in X_1 binäre Suche an.*

4. *Ist $X_0 \neq \emptyset$, so wende in dem Baum X_0 von allen Algorithmen $\mathcal{A}_{k',m-k}$ mit $1 \leq k' \leq m-k$ einen mit minimaler Laufzeit an.*

Wir zeigen jetzt, daß dies ein korrekter Algorithmus ist.

Lemma 9.4 *Seien $k, m \in \mathbb{N}$ mit $k \leq m$. Dann ist nach jeder der ersten k Antworten die Menge X_0 ein vollständiger binärer Teilbaum. Nach der $k+1$-ten Antwort gilt $X_0 = \emptyset$ oder ($X_1 = \emptyset$ und $|X_0| = 2^{m-k}$). Der Algorithmus ist also wohldefiniert.*

Beweis: In den ersten k Fragen wird jeweils ein Knoten x gefragt, wobei x ein Sohn der Wurzel von X_0 ist. Sei y der Bruder von x. Nach jeder Antwort gilt dann entweder $X_0 = T_x$ oder $X_0 = T_y$. Der Baum X_0 hat also nach k Fragen die Größe 2^{m-k}.

Die $k+1$-te Frage ist der komplette Baum X_0. Ist die Antwort „Ja", so verändert sich X_0 nicht. Es gilt aber $X_1 = \emptyset$. Das Blatt kann daher nur noch in X_0 liegen.

Ist die Antwort „Nein", so ist $X_0 = \emptyset$. Das Orakel kann folglich nicht mehr lügen. In X_1 kann demnach eine binäre Suche durchgeführt werden.

□

Die Laufzeit von jedem $\mathcal{A}_{k,m}$ ist eine obere Schranke für $E(m)$. Wir setzen aus formalen Gründen $F(0) = 0$, und für $m \in \mathbb{N}$ definieren wir

$$F(m) = \min_{k=1,\ldots,m} L(\mathcal{A}_{k,m}).$$

Damit ist $F(m)$ eine obere Schranke für $E(m)$. Um $F(m)$ zu berechnen, benötigen wir zunächst eine Gleichung für $L(\mathcal{A}_{k,m})$.

Lemma 9.5 *Sei $k, m \in \mathbb{N}$ mit $k \leq m$. Dann ist*

$$L(\mathcal{A}_{k,m}) = k + 1 + \max\{F(m-k), m\}.$$

Beweis: Nach den ersten $k+1$ Fragen gibt es nach Lemma 9.4 zwei Möglichkeiten:

$\boldsymbol{X_0 = \emptyset}$**:** Die folgende binäre Suche braucht m Fragen, da X_1 der Baum T ist. Insgesamt sind also $k+1+m$ Fragen gestellt worden.

$\boldsymbol{X_1 = \emptyset}$**:** Der Baum X_0 hat die Größe 2^{m-k} und muß das defekte Blatt beinhalten. Ist $m-k=0=F(0)$, so sind keine weiteren Fragen notwendig. Ist $m-k>0$, so werden noch $F(m-k)$ Fragen gestellt. Das sind zusammen $k+1+F(m-k)$ Fragen.

Das Orakel wählt die Antwort so, daß das Maximum dieser beiden Werte erreicht wird.

□

Wir berechnen damit $F(m)$.

Satz 9.6 *Sei $m \in \mathbb{N}$. Dann existiert genau ein $i \in \mathbb{N}$ mit $m \in B_i$. Setzen wir*

$$j = m - \frac{(i-1)i}{2},$$

so gilt genau dann $L(\mathcal{A}_{k,m}) = F(m)$, wenn $k \in \{j, \ldots, i\}$ gilt. Insbesondere ist

$$F(m) = m + i + 1.$$

Beweis: Die Eindeutigkeit von i folgt aus Satz 9.2. Wir führen vollständige Induktion über m.

$\boldsymbol{m = 1}$**:** Es ist $i=1$ und damit $j=1$. Nach Definition von F gilt $L(\mathcal{A}_{1,1}) = F(1)$. Der Algorithmus $\mathcal{A}_{1,1}$ fragt immer nur Blätter. Ein Orakel, welches sich bei den ersten beiden Antworten widerspricht, erfordert eine dritte Frage. Nach Lemma 9.5 ist damit

$$L(\mathcal{A}_{1,1}) = 3 = m + i + 1.$$

$\boldsymbol{m = 2}$**:** Es ist $i=2$ und $j=1$. Es gilt

$$L(\mathcal{A}_{1,2}) = L(\mathcal{A}_{2,2}) = 5 = m + i + 1.$$

$\boldsymbol{m = 3}$**:** Es ist $i=2$ und $j=2$. Daneben gilt

$$L(\mathcal{A}_{2,3}) = 6 = m + i + 1$$

und

$$L(\mathcal{A}_{1,3}) = L(\mathcal{A}_{3,3}) = 7.$$

$\boldsymbol{m \geq 4}$**:** Es ist $i \geq 3$. Außerdem gilt nach Satz 9.2 $1 \leq j \leq i$.

Betrachten wir zuerst den Fall $j \leq k \leq i$. Antwortet das Orakel auf die $k+1$-te Frage „Nein“, so reichen m weitere Fragen. Die Laufzeit beträgt also $k+1+m \leq i+1+m$. Antwortet das Orakel „Ja“, so ist die Laufzeit gleich $k+1+F(m-k)$. Da $j \leq k \leq i$ ist, gilt nach Satz 9.2 $m-k \in B_{i-1} \cup B_{i-2}$. Ist $m-k \in B_{i-1}$, so gilt nach Induktionsvoraussetzung für $m-k$

$$k+1+F(m-k) = k+1+(m-k)+(i-1)+1 = m+i+1.$$

Der Fall $m-k \in B_{i-2}$ ist nur für $k=i$ und $j=1$ möglich. Wäre $j>1$, so wäre $m \geq 2+\frac{(i-1)i}{2}$ und damit $m-k \geq m-i = 1+\frac{(i-2)(i-1)}{2}$. Dies widerspricht Satz 9.2. Analog würde $k<i$ einen Widerspruch erzeugen. Damit gilt in diesem Fall

$$k+1+F(m-k) = k+1+(m-k)+(i-2)+1 = m+i.$$

Das Maximum ist also sowohl für $k<i$ als auch für $k=i$ genau $m+i+1$.

Ist $k<j$, so ist $m-k \in B_i$. Das Orakel antwortet auf die $k+1$-te Frage „Ja“. Die Laufzeit ist damit mindestens

$$k+1+F(m-k) = k+1+(m-k)+i+1 = m+i+2 > m+i+1.$$

Ist schließlich $k>i$, so antwortet das Orakel auf die $k+1$-te Frage „Nein“. Die Laufzeit ist folglich mindestens

$$k+1+m > m+i+1.$$

□

Betrachten wir den Beweis, so können wir folgern, daß das Orakel im Fall $j \leq k < i$ auf $k+1$-te Frage „Ja“ sagen muß, um die angegebene Suchkomplexität zu erzwingen. Ist $k=i$ und $j=1$, so muß das Orakel „Nein“ antworten. Ist $k=i$ und $j>1$, so sind beide Antworten möglich.

Für $j<i$ ist $m+1 \in B_i$. Für $j=i$ ist $m+1 \in B_{i+1}$. Damit gilt

Korollar 9.7 *Seien m, i und j wie in Satz 9.6. Ist $j<i$, so ist $F(m+1) = F(m)+1$. Ist $j=i$, so ist $F(m+1)=F(m)+2$.*

9.4 Untere Schranke

Wir werden im folgenden beweisen, daß die $\mathcal{A}_{k,m}$ optimal sind. Dafür zeigen wir zunächst ein für alle Suchprozesse geltendes Lemma.

Lemma 9.8 *Sei $(S, \mathcal{F})$ ein Suchprozeß, bei dem $e \in \mathbb{N}_0$ Lügen erlaubt sind und $(X_0, \ldots, X_{e+1})$ und $(Y_0, \ldots, Y_{e+1})$ zwei Zustände. Es gelte für alle $x \in S$:*

$$(x \in X_i \text{ und } x \in Y_j) \Longrightarrow (i \geq j)\,.$$

Dann gilt

$$C(X_0, \ldots, X_{e+1}) \leq C(Y_0, \ldots, Y_{e+1}).$$

Beweis: Sei $r = C(Y_0, \ldots, Y_{e+1})$ und $\mathcal{A}$ ein optimaler Algorithmus für diesen Zustand. Wir führen vollständige Induktion über r.

***r* = 0:** Sei x^* das gesuchte Element. Dann gilt $S - \{x^*\} \subseteq Y_{e+1}$. Nach Voraussetzung gilt damit auch $S - \{x^*\} \subseteq X_{e+1}$. Also ist

$$C(X_0, \ldots, X_{e+1}) = 0.$$

$r \to r+1$: Der Suchalgorithmus $\mathcal{B}$ fragt beim Zustand $(X_0, \ldots, X_{e+1})$ die gleiche Frage, die $\mathcal{A}$ bei $(Y_0, \ldots, Y_{e+1})$ stellt. Bei jedem Suchalgorithmus können sich zwei Zustände ergeben. Wir bezeichnen mit $(X'_0, \ldots, X'_{e+1})$ und $(Y'_0, \ldots, Y'_{e+1})$ die sich bei „Ja" ergebenden Zustände. Entsprechend bezeichnen $(X''_0, \ldots, X''_{e+1})$ und $(Y''_0, \ldots, Y''_{e+1})$ die auf „Nein" folgenden Zustände. Die Tupel $(X'_0, \ldots, X'_{e+1})$ und $(Y'_0, \ldots, Y'_{e+1})$ beziehungsweise $(X''_0, \ldots, X''_{e+1})$ und $(Y''_0, \ldots, Y''_{e+1})$ erfüllen wieder die Voraussetzung des Satzes. Also gilt nach Induktionsvoraussetzung

$$C(X'_0, \ldots, X'_{e+1}) \leq C(Y'_0, \ldots, Y'_{e+1})$$

und

$$C(X''_0, \ldots, X''_{e+1}) \leq C(Y''_0, \ldots, Y''_{e+1}).$$

Da $C(X_0, \ldots, X_{e+1})$ und $C(Y_0, \ldots, Y_{e+1})$ jeweils das Maximum der Laufzeit von beiden möglichen Folgezuständen ist, folgt die Behauptung.

□

Mit diesem Lemma können wir zeigen, daß unter den Algorithmen $\mathcal{A}_{k,m}$ immer ein optimaler Algorithmus ist.

Satz 9.9 *Für alle $m \in \mathbb{N}$ gilt $E(m) = F(m)$.*

Beweis: Es gilt für alle m $E(m) \leq F(m)$. Angenommen der Satz ist falsch. Dann sei m minimal mit $E(m) < F(m)$. Da $E(1) = 3 = F(1)$ ist, ist $m \geq 2$. Wir definieren die Parameter i und j zu m, wie in Satz 9.6. Da $m \geq 2$ ist, ist $i \geq 2$. Sei $\mathcal{B}$ ein optimaler Algorithmus, also $L(\mathcal{B}) = E(m) < F(m)$. Es gilt somit für alle $k = 1, \ldots, m$

$$L(\mathcal{B}) < L(\mathcal{A}_{k,m}).$$

Sei $\kappa + 1$ die Nummer der ersten Frage, bei der $\mathcal{B}$ von allen Algorithmen $\mathcal{A}_{k,m}$ mit $j \leq k \leq i$ abweicht. Gibt es zu diesem m mehrere optimale Algorithmen, so wählen wir einen mit maximalem κ.

Für den Algorithmus $\mathcal{B}$ gibt es drei Möglichkeiten, sich im κ+1-ten Schritt von allen $\mathcal{A}_{k,m}$ zu unterscheiden:

Fall 1: $\boldsymbol{\mathcal{B}}$ fragt einen Teilbaum $\boldsymbol{T}$ von $\boldsymbol{X_0}$, der kleiner als die Hälfte von $\boldsymbol{X_0}$ ist. Wir setzen (Y_0, Y_1, Y_2) als den sich ergebenden Zustand, wenn das Orakel „Nein“ antwortet. Hätte $\mathcal{B}$ statt dessen wieder X_0 halbiert, so würde das neue Tupel (X_0, X_1, X_2) die Voraussetzungen von Lemma 9.8 erfüllen. Damit existiert ein optimaler Algorithmus, der später von allen $\mathcal{A}_{k,m}$ abweicht. κ war also nicht maximal.

Fall 2: $\boldsymbol{\mathcal{B}}$ fragt einen Vorfahren der Wurzel von $\boldsymbol{X_0}$. Ist $\kappa < i$ oder $\kappa = i$ und $j > 1$, so antwortet das Orakel „Ja“. X_0 ist damit ein Baum der Größe $2^{m-\kappa}$. Demnach ist

$$L(\mathcal{B}) = \kappa + 1 + E(m - \kappa) = \kappa + 1 + F(m - \kappa) = L(\mathcal{A}_{i,m}).$$

Wir verwenden hierbei die Bemerkungen nach Satz 9.6.

Ist $\kappa = i$ und $j = 1$, so antwortet das Orakel auch „Ja“. Die Menge X_0 ist wieder ein Baum der Größe $2^{m-\kappa}$. Die Menge X_1 enthält auch einen Baum der Größe $2^{m-\kappa}$, und zwar den Bruder von der Wurzel von X_0. Es ist $m - \kappa + 1 < m$, da $\kappa = i \geq 2$ ist. Nach Induktionsvoraussetzung würde also bei einem Baum der Größe $2^{m-\kappa+1}$ ein optimaler Algorithmus als erstes den halben Baum fragen. Für den Nachfolgezustand (X_0, X_1) gilt also $|X_0| = |X_1| = 2^{m-\kappa}$. Damit ist $C(X_0, X_1) = E(m - \kappa + 1) - 1$. Folglich ist

$$L(\mathcal{B}) \geq i + E(m - i + 1).$$

Andererseits gilt

$$L(\mathcal{A}_{i-1,m}) = i + F(m - i + 1) = i + E(m - i + 1).$$

Dies widerspricht $L(\mathcal{B}) < F(m)$.

Ist $\kappa > i$, so hat $\mathcal{B}$ einmal den ganzen Baum X_0 gefragt. Sei $k \in \mathbb{N}$ so gesetzt, daß der Algorithmus dies in der $k+1$-ten Frage getan hat. Damit $\mathcal{B}$ optimal sein kann, muß nach Satz 9.6 und der Induktionsvoraussetzung für den verbleibenden Baum $j \leq k \leq i$ sein. Antwortet das Orakel auf die $k+1$-te Frage „Nein", so ist $L(\mathcal{B}) = 1 + k + m$. Lautet die Antwort des Orakels „Ja", so ist $L(\mathcal{B}) = 1 + k + E(m-k) = 1 + k + F(m-k)$. Das Maximum davon ist aber genau $L(\mathcal{A}_{k,m})$.

Fall 3: B fragt einen Knoten $\boldsymbol{x}$, der nicht in $\boldsymbol{X_0}$ liegt und kein Vorfahre von der Wurzel von $\boldsymbol{X_0}$ ist. Hat $\mathcal{B}$ schon vorher einmal nach der k-ten Frage den zu diesem Zeitpunkt vorhandenen Baum X_0 gefragt, so ist wie im Fall 2 $L(\mathcal{B}) \geq L(\mathcal{A}_{k-1,m})$. Hat $\mathcal{B}$ nie den ganzen Baum X_0 gefragt, so hat es in den Fragen $1, \ldots, \kappa$ immer den Baum X_0 halbiert, da es sonst schon vor der $\kappa+1$-ten Frage mit keinem $\mathcal{A}_{k,m}$ mehr übereingestimmt hätte. Es gilt also $X_0 \cup X_1 = T$ und $|X_0| = 2^{m-\kappa}$. Das Orakel kann „Nein" antworten, womit der Baum X_0 immer noch die Größe $2^{m-\kappa}$ hat. Es wäre dann

$$L(\mathcal{B}) \geq \kappa + 1 + E(m-\kappa) = \kappa + 1 + F(m-\kappa).$$

Es könnte auch „Ja" antworten. Damit ist $X_1 = T$, und es gilt

$$L(\mathcal{B}) \geq \kappa + 1 + m.$$

Das Maximum ist aber wieder $L(\mathcal{A}_{\kappa,m})$.

□

Wir zeigen noch eine explizite Formel für $E(m)$.

Korollar 9.10 *Sei $m \in \mathbb{N}$. Dann ist*

$$E(m) = m + \left\lceil \sqrt{2m + \frac{1}{4}} + \frac{1}{2} \right\rceil.$$

Beweis: Es gilt $E(m) = F(m)$. Nach Satz 9.6 ist $F(m) = m + i + 1$, wobei $i \in \mathbb{N}$ so gewählt ist, daß $m \in B_i$ liegt. Dieses i ist das minimale i', so daß

$$m \leq \frac{i'(i'+1)}{2}$$

ist. Diese Ungleichung ist äquivalent zu

$$i' \geq \sqrt{2m + \frac{1}{4}} - \frac{1}{2}.$$

Also gilt

$$i = \left\lceil \sqrt{2m + \frac{1}{4}} - \frac{1}{2} \right\rceil .$$

Somit ist

$$\begin{aligned} E(m) &= F(m) \\ &= m + i + 1 \\ &= m + \left\lceil \sqrt{2m + \frac{1}{4}} - \frac{1}{2} \right\rceil + 1 \\ &= m + \left\lceil \sqrt{2m + \frac{1}{4}} + \frac{1}{2} \right\rceil . \end{aligned}$$

□

Korollar 9.11 *Für $m \in \mathbb{N}$ gilt*

$$m + \sqrt{2m} + \frac{1}{2} < E(m) < m + \sqrt{2m} + \frac{1}{8\sqrt{2m}} + \frac{3}{2}.$$

Beweis: Die untere Schranke folgt direkt aus Korollar 9.10.

Nach dem Mittelwertsatz existiert ein $\xi \in]2m, 2m + \frac{1}{4}[$, so daß

$$\frac{\sqrt{2m + \frac{1}{4}} - \sqrt{2m}}{\frac{1}{4}} = \frac{1}{2\sqrt{\xi}}$$

gilt. Da $\xi > 2m \geq 1$ ist, gilt $\sqrt{\xi} > \sqrt{2m}$. Also ist

$$\frac{\sqrt{2m + \frac{1}{4}} - \sqrt{2m}}{\frac{1}{4}} < \frac{1}{2\sqrt{2m}}.$$

Dies ergibt

$$\sqrt{2m + \frac{1}{4}} < \sqrt{2m} + \frac{1}{8\sqrt{2m}}.$$

Also gilt

$$\left\lceil \sqrt{2m + \frac{1}{4}} + \frac{1}{2} \right\rceil < \sqrt{2m + \frac{1}{4}} + \frac{3}{2} < \sqrt{2m} + \frac{1}{8\sqrt{2m}} + \frac{3}{2}.$$

□

9.5 Zusammenfassung

Wir haben in diesem Kapitel die Suchkomplexität bei einer erlaubten Lüge in vollständigen binären Bäumen untersucht. Die Analyse zeigte, daß die optimalen Algorithmen k-mal eine binäre Suche durchführen und dann den Baum X_0 als ganzes überprüfen. Entweder liegt das Blatt dann in diesem Baum, oder es wurde schon einmal gelogen. Die Laufzeit dieser Algorithmen läßt sich bestimmen. Das k ist dabei nicht immer eindeutig. Wir konnten zu gegebenem m die k bestimmen, welche eine minimale Laufzeit garantieren. Schließlich haben wir für die Suchkomplexität eine geschlossene Formel angegeben. Sie beträgt bei einem vollständigen binären Baum mit 2^m Blättern

$$E(m) = m + \left\lceil \sqrt{2m + \frac{1}{4}} + \frac{1}{2} \right\rceil .$$

Die ersten Werte zeigt Tabelle 9.1.

m	1	2	3	4	5	6	7	8	9	10
$E(m)$	3	5	6	8	9	10	12	13	14	15

Tabelle 9.1: Die Suchkomplexität bei einer erlaubten Lüge

Wir haben hier nur den Fall einer Lüge behandelt. Bei mehreren erlaubten Lügen wird die Situation wesentlich komplizierter. Wird der komplette Baum X_0 gefragt und antwortet das Orakel „Nein“, so ist $X_0 = \emptyset$. Der Fall e-Lügen ist dann auf den Fall $e-1$-Lügen reduziert worden. Antwortet das Orakel aber „Ja“, so kann der Fragende sich nicht sicher sein, ob das gesuchte Blatt in X_0 liegt. Der Suchalgorithmus kann daher nicht den restlichen Baum ignorieren, wie dies bei einer erlaubten Lüge möglich ist.

Bezeichnungen

Wir führen hier die wichtigsten in der Arbeit verwendeten Bezeichnungen auf.

$\mathbb{N}$	1,2,3...
$\mathbb{N}_0$	0,1,2,3...
$\lvert T\rvert$	Die Anzahl der Blätter vom Baum T.
T_x	Der zu x gehörige Teilbaum. Er enthält x und alle Nachfolger von x.
$(T_1,\ldots,T_q)$	Der Baum, bei dem die Wurzel q-Söhne hat und für $i=1,\ldots,q$ der i-te Sohn die Wurzel des Baumes T_i ist.
q-Baum	Ein Baum, bei dem jeder Knoten maximal q Söhne hat.
$\mathcal{T}_q$	Die Menge aller q-Bäume.
(n,q)-Baum	Ein q-Baum mit n Blättern.
$\mathcal{T}_{n,q}$	Die Menge aller (n,q)-Bäume.
$(S,\mathcal{F})$	Ein Suchprozeß. Es gilt $\mathcal{F}\subseteq 2^S$.
$L(\mathcal{A})$	Die Laufzeit des Algorithmus $\mathcal{A}$.
$L(T)$	Die Suchkomplexität beim Baum T.
$L(S,\mathcal{F})$	Die Suchkomplexität des Suchprozesses.
$L_r(S,\mathcal{F})$	Die Suchkomplexität des Suchprozesses bei r Runden.
$L_{pre}(S,\mathcal{F})$	$L_1(S,\mathcal{F})$.

Die Wurzel eines Baumes hat die Tiefe 0. Die Tiefe eines Baumes ist die maximale Tiefe seiner Blätter.

Die vom Algorithmus gefundenen Blätter eines Baumes bezeichnen wir als **defekt**. Sobald von einem Teilbaum bekannt ist, daß er mindestens ein defektes Blatt beinhaltet, heißt er **kontaminiert**. Ein Knoten x heißt kontaminiert, wenn T_x kontaminiert ist.

Literaturverzeichnis

[AIG–88] Martin Aigner, *Combinatorial Search*, Teubner-Wiley, Leipzig/New York, 1988.

[AIG–95] Martin Aigner, *Ulams Millionenspiel*, Mathematische Semesterberichte **42** (1995), 71–80.

[AIG–96] Martin Aigner, *Searching with Lies*, Journal of Combinatorial Theory, Series **A 73** (1996).

[ALT–93] Ingo Althöfer, Eberhard Triesch, *Edge search in graphs and hypergraphs of bounded rank*, Discrete Mathematics **115** (1993), 1–9.

[ALT–96] Ingo Althöfer, Matthias Löwe, *Edge Search in Hypergraphs*, Discrete Mathematics **162** (1996), 267–272.

[DAM–95] Peter Damaschke, *Searching for faulty leaves in binary trees*, 21st International Workshop on Graph-Theoretic Concepts in Computer Science WG'95, Aachen 1995, Lecture Notes in Computer Science **1017**, 265–274.

[DUH–93] Du, Hwang, *Combinatorial group testing and its applications*, World Scientific Publ. Co., Singapore 1993.

[HIL–95] Ray Hill, *Searching with Lies*, London Mathematical Society Lecture Note Series **218** (1995), 41–70.

[PEL–87] Andrzej Pelc, *Solution of Ulam's Problem on Searching with a Lie*, Journal of Combinatorial Theorie, Series **A 44** (1987), 129–140.

[PUL–76] N.J. Pullman, *Matrix Theory and its Applications*, Marcel Dekker, INC., 1976.

[RIV–77] Ronald L. Rivest, *The Game of "N" Questions on a Tree*, Discrete Mathematics **17** (1977), 181–186.

[TRI–94] Eberhard Triesch, *A probabilistic upper bound for the edge identification complexity of graphs*, Discrete Mathematics **125** (1994), 371–376.

[TRI–96] Eberhard Triesch, *A Group Testing Problem for Hypergraphs of Bounded Rank*, Discrete Applied Mathematics **66** (1996), 185–188.

[TRI–97] Jens Schlaghoff, Eberhard Triesch, *Improved results for competitive group testing*, eingereicht bei Discrete Applied Mathematics.

[ULA–76] Stanislaw Ulam, *Adventures of a Mathematician*, Scribner, New York 1976.

Zusammenfassung

Die kombinatorische Suche beschäftigt sich mit der Anzahl der notwendigen Fragen, um in einer Menge S ein oder mehrere sogenannte defekte Elemente zu finden. Das Thema dieser Arbeit ist die Fragestellung:

> Wie verändert sich die Suchkomplexität, wenn die Menge S aus den Blättern eines Wurzelbaumes T besteht und Testmengen nur komplette Teilbäume sein dürfen?

Die Anzahl der Fragen in einem Baum T, die ausreichen ein Blatt zu finden, hängt von der Struktur des Baumes T ab. Unter allen q-Bäumen, die eine vorgegebene Anzahl von Fragen erfordern, läßt sich ein Baum mit minimaler Blattanzahl rekursiv berechnen. Kapitel 4 beschreibt die Rekursionsvorschrift.

Sind mehrere Blätter defekt, so existiert zu jedem $n \in \mathbb{N}$ ein binärer Baum mit n Blättern, bei dem $n-1$ Fragen notwendig sind. Zum Suchproblem von mehreren defekten Blättern in vollständigen q-regulären Bäumen steht in Kapitel 5 und 6 die exakte Formel für die Suchkomplexität.

Ein paralleler Suchalgorithmus, der ein defektes Blatt sucht, muß bei n Blättern $n-1$ Fragen stellen. In den Kapiteln 7 und 8 steht die exakte Suchkomplexität, wenn die Fragen in mehreren Runden gestellt werden, wobei die Fragen jeder einzelnen Runde parallel beantwortet werden.

Wird in einem vollständigen binären Baum der Tiefe m ein Blatt gesucht, so beträgt die Suchkomplexität m. Kann aber eine der Antworten falsch sein, so wächst die Suchkomplexität auf

$$m + \left\lceil \sqrt{2m + \frac{1}{4}} + \frac{1}{2} \right\rceil .$$

Lebenslauf

06.04.1968	geboren in Detmold
1974–1978	Besuch der Grundschule in Blomberg
1978–1987	Besuch des Gymnasiums in Blomberg
Mai 1987	Abitur
1987–1994	Mathematikstudium mit Nebenfach Informatik an der Universität Bielefeld
04.01.1988–31.03.1989	Unterbrechung des Studiums wegen Einberufung zum Wehr- und Zivildienst
1990–1992	Tutor für verschiedene Vorlesungen in Mathematik und Informatik
April 1994	Diplom in Mathematik
01.10.1994–31.03.1997	Promotionsstipendiat im Graduiertenkolleg „Algorithmische Diskrete Mathematik“ an der Freien Universität Berlin mit Stipendium der Deutschen Forschungsgemeinschaft