Algorithmen und Datenstrukturen

SoSe 2008 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Algorithmen und Datenstrukturen

Gesamtübersicht

- Organisatorisches / Einführung
- Grundlagen: RAM, $\mathcal{O}$-Notation, Rekursion, Datenstrukturen
- Sortieren
- Wörterbücher und Mengen
- Graphen und Graphalgorithmen

Geordnete Mengen als Wörterbücher

Wir betrachten (wie bisher) Mengen M, allerdings mit einer totalen Ordnungsrelation $\leq$ auf dieser Menge. versehen.
Beispiel: Betrachte $M = \mathbb{Z}$ mit der üblichen Ordnung.

Die Elemente von M können wir uns (zunächst) in einem Feld $A[1..|M|]$ abgespeichert denken, und zwar *ordnungserhaltend*, d.h.: $i < j$ gdw. $A[i] < A[j]$.
(Wir können die Forderung auch noch abschwächen, falls wir zulassen wollen, dass zwei Elemente des Feldes gleich sein dürfen. Versuchen Sie, für diesen Fall “ordnungserhaltend” zu definieren.)

Wörterbücher: Mengen mit eingeschränkten Operationen

Nachschlagen (Lookup), also die Elementanfrage
Einfügen (Insert)
Löschen (Delete) eines einzelnen Elementes (musste gemäß bisheriger Spezifikation durch andere Mengenoperationen nachgebildet werden (wie genau?))
Initialisieren (init/clear)
Wir wissen bereits: Das Suchen in geordneten Feldern geht schnell (logarithmischer Zeitaufwand VL6).
Allerdings ist das Einfügen teuer (Linearzeit), da im Schnitt die Hälfte der Feldelemente verschoben werden muss.
Für nicht (oder nur sehr selten) zu verändernde geordnete Mengen ist die Darstellung durch ordnungserhaltende Felder aber nicht schlecht.

Geordnete Wörterbücher

Algorithm 1 Allgemeine Suche

Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

$\alpha \leftarrow 1$; $\omega \leftarrow n$;
$i \leftarrow$ some number in $[\alpha..\omega]$
while $(x \neq A[i]) \wedge (\omega > \alpha)$ **do**
  if $x < A[i]$ **then**
    $\omega \leftarrow i - 1$
  else
    $\alpha \leftarrow i + 1$
  $i \leftarrow$ some number in $[\alpha..\omega]$
end while
return i (if $A[i] = x$); NO otherwise

Spezialfälle
Diese unterscheiden sich durch genauere Angabe von "some number".
(*Teilungspunkt*)
Lineare Suche: $i \leftarrow \alpha$
Binäre Suche:
$i \leftarrow \left\lceil \frac{\alpha+\omega}{2} \right\rceil$
Interpolationssuche:
$i \leftarrow \alpha + \left\lfloor \frac{x-A[\alpha]}{A[\omega]-A[\alpha]}(\omega - \alpha + 1) \right\rfloor$

Geordnete Wörterbücher: Rekursion versus Induktion

Algorithm 2 Allgemeine Suche

Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

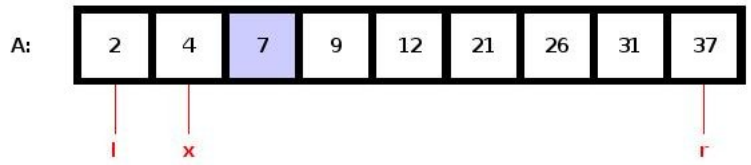
$\alpha \leftarrow 1$; $\omega \leftarrow n$;
$i \leftarrow$ some number in $[\alpha..\omega]$
while $(x \neq A[i]) \wedge (\omega > \alpha)$ **do**
  if $x < A[i]$ **then**
    $\omega \leftarrow i - 1$
  else
    $\alpha \leftarrow i + 1$
  $i \leftarrow$ some number in $[\alpha..\omega]$
end while
return i (if $A[i] = x$); NO otherwise

Algorithm 3 Allgemeine Suche rekursiv SR

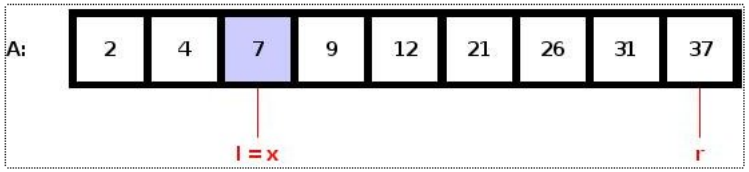
Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

$i \leftarrow$ some number in $[1..n]$
if $(x < A[i]) \wedge (i > 1)$ **then**
  return rescale SR$(A[1..i-1], x)$
else if $(x > A[i]) \wedge (i < n)$ **then**
  return rescale SR$(A[i+1..n], x)$
else
  return i (if $A[i] = x$); NO otherwise

Interpolationssuche am Beispiel: Wir suchen $y = 7$.



$x = 1 + \frac{7-2}{37-2} * (9-1) \approx 2,15 \rightarrow 2$ ist neuer *Teilungspunkt*



Satz: Sind die Zahlen im Feld $A[1], \ldots, A[n]$ gleichverteilt unabhängig aus dem Intervall (l, r) gezogen, so beträgt die Zugriffszeit für Interpolationssuche im mittleren Fall $\mathcal{O}(\log \log n)$.

Geordnete Wörterbücher: Zur Korrektheit

Algorithm 4 Allgemeine Suche

Input(s): a sorted array $A : \mathbb{Z}[1..n]$, an element x
Output(s): if found: position i such that $x = A[i]$; NO otherwise

$\alpha \leftarrow 1$; $\omega \leftarrow n$;
$i \leftarrow$ some number in $[\alpha..\omega]$
while $(x \neq A[i]) \wedge (\omega > \alpha)$ **do**
  if $x < A[i]$ **then**
    $\omega \leftarrow i - 1$
  else
    $\alpha \leftarrow i + 1$
  $i \leftarrow$ some number in $[\alpha..\omega]$
end while
return i (if $A[i] = x$); NO otherwise

Schleifeninvariante

(A) $(x \in A \implies x \in \{A[\alpha], \ldots, A[\omega]\}) \wedge$
(B) $(\alpha \leq \omega \implies \alpha \leq i \leq \omega)$

Beim ersten Schleifeneintritt: ✓

Wird Schleifenrumpf betreten, so gilt entweder $x < A[i]$ oder $x > A[i]$.

Da A sortiert, folgt aus $x < A[i]$ (und der Gültigkeit der Invarianten vor Schleifeneintritt): $(x \in A \implies x \in \{A[\alpha], \ldots, A[i-1]\})$.

Analog: $x > A[i]$.

Der zweite Invariantenteil ist klar.

Bei Schleifenaustritt gilt die Invariante sowie $(x = A[i] \vee \omega \leq \alpha)$. Gilt $x \neq A[i]$, muss $\omega \leq \alpha$ sein. Wäre $x \in A$, so $x = A[\alpha] = A[i] = A[\omega]$, Widerspruch zu $x \neq A[i]$.

Geordnete Wörterbücher: Suchbäume

Dem allgemeinen Suchschema kann man leicht einen Baum zuordnen.
In den Knoten werden dabei die unterschiedlichen Werte des Teilungspunktes eingetragen.
Beachte, dass bei linearer und binärer Suche der Teilungspunkt nicht von den konkreten Feldinhalten abhängt,
Genauer hat bei linearer Suche der zugehörige Suchbaum lineare Gestalt, während er bei binärer Suche ein Binärbaum ist, bei dem sich die Blätter nur auf zwei Ebenen befinden.
Das Nachschlagen kostet bei binärer Suche daher logarithmische Zeit: im schlimmsten Fall (der erfolglosen Suche) wird bis zu den Blättern ein Pfad verfolgt.
Das Einfügen ist jedoch sehr kostspielig, solange wir intern auf Feldern arbeiten und der Suchbaum nur konzeptuell existiert; eine deutliche Verbesserung bietet ein *(expliziter) binärer Suchbaum*.

Hinweis: erster "Algorithmus der Woche" `http://www-i1.informatik.rwth-aachen.de/~algorithmus/Algorithmen/algo1/Folien.pdf`

Binäre Suchbäume

Ein binärer Suchbaum für die n-elementige geordnete Menge $M = \{x_1 < x_2 < \cdots < x_n\}$ ist ein Binärbaum mit n Knoten $V = \{v_1, \ldots, v_n\}$. Es gibt eine Injektion $INH : V \to M$, *Beschriftung* genannt.
Die Beschriftung *bewahrt die Ordnung*, d.h., sind v_i bzw. v_j Knoten aus dem linken bzw. rechten Unterbaum des Teilbaumes mit Wurzel v_k, so gilt $INH(v_i) < INH(v_k) < INH(v_j)$.
$\rightsquigarrow$ Wir können den "Kindern" der Blätter des Suchbaums konzeptuell offene Intervalle (x_ℓ, x_{ell+1}) zuordnen. (siehe Tafelbild / Beispiel)
Ein Inorder-Durchlauf des Suchbaums listet daher die Menge aufsteigend auf.

Hinweis: Tatsächlich verweisen die Pointer in den Blättern auf NIL.

Praktischer orientierte Erklärungen finden Sie unter `http://www.matheprisma.de/Module/BinSuch/index.htm`.

Binäre Suchbäume: Der Suchlauf

Algorithm 5 Baumbasierte Suche

Input(s): a binary search tree $T = (T_1, r, T_2)$, an element x

Output(s): if found: node v such that $x = \mathrm{INH}(v)$; NO otherwise

$v \leftarrow r$
while $((v \neq \mathrm{NIL}) \wedge (x \neq \mathrm{INH}(v))$ **do**
  if $x < \mathrm{INH}(v)$ **then**
    $v \leftarrow \mathrm{LEFT}(v)$
  else
    $v \leftarrow \mathrm{RIGHT}(v)$
end while
return v (if $v \neq \mathrm{NIL}$); NO otherwise

Korrektheit

Die Überlegungen bei der Feld-Implementierung lassen sich fast wörtlich übertragen.

Insbesondere gilt: Wenn das Programm in einem "richtigen" Knoten terminiert, so ist $x = \mathrm{INH}(v)$.

Terminiert es hingegen in einem "Blattkind" (also NIL), so stellt dieses Blattkind das Intervall $I = (x_\ell, x_{\ell+1})$ dar mit $x \in I$.

Beachte: Einfügen von x ist jetzt leicht zu implementieren:
Nach erfolgloser Suche hat man bereits die richtige Stelle gefunden.

Löschen im binären Suchbaum entspricht Streichen von Knoten.
Problem: Wir wollen weiterhin einen Binärbaum erhalten, der die Ordnung respektiert.
Streichen von x mit $x = \mathrm{INH}(v)$.
1. Fall: Hat v nicht zwei Kinder, so gilt z.B. $\mathrm{LEFT}(v) = \mathrm{NIL}$. Dann ersetze v durch $\mathrm{RIGHT}(v)$.
2. Fall: v hat zwei Kinder. Sei w der rechteste Knoten im linken Unterbaum von v. Ersetze $\mathrm{INH}(v) \leftarrow \mathrm{INH}(w)$ und streiche w wie im ersten Fall.

Beachte: Die Mengenoperationen (implementiert auf dem Binärbaum) hängen im Wesentlichen von der Höhe des Baumes ab. Diese wiederum richtet sich nach der “Geschichte” des Auf- und Abbaus des Binärbaumes. Man kann zeigen, dass “im Mittel” so logarithmische Zeitkomplexitäten erzielt werden.

Durchschnittsanalyse für binäre Suchbäume

Frage: Wie viele Knoten befinden sich im Durchschnitt auf einem Pfad in einem "zufälligen Binärbaum" ?
(Das entspricht bis auf eine additive Konstante von Eins der *mittleren Pfadlänge*.)
Zur Beantwortung dieser Frage müssen wir uns ein geeignetes *Zufallsmodell* bereitstellen. Ein *durchschnittlicher Suchbaum* sei gemäß folgender Annahmen definiert:
1. Er ist nur durch Einfügungen entstanden.
2. Mit Bezug auf die Einfügereihenfolge seien alle Permutationen der Menge der gespeicherten Schlüssel $S = \{x_1, \ldots, x_n\}$ gleich wahrscheinlich.
$P(n)$: mittlere Pfadlänge in einem durchschnittlichen (binären) Suchbaum mit n Schlüsseln.

Durchschnittsanalyse für binäre Suchbäume (Forts.)

Gemäß unserem Einfügealgorithmus wird das zuerst gewählte Element in der Wurzel des Binärbaumes abgespeichert.
Nach unserem Modell ist es für jedes Element aus S gleichwahrscheinlich, die Wurzel zu bilden.
Ist x_{i+1} das Element an der Wurzel, so entsteht ein Binärbaum, dessen linker Teilbaum i Elemente enthält, und der rechte enthält $n - i + 1$ viele.
Beide Teilbäume sind zufällig in unserem Sinne, genauer gilt:
Der linke Suchbaum ist ein durchschnittlicher Suchbaum mit den Schlüsseln $S_\ell = \{x_1, \ldots, x_i\}$;
der rechte Suchbaum ist ein durchschnittlicher Suchbaum mit den Schlüsseln $S_r = \{x_{i+2}, \ldots, x_n\}$.

$\rightsquigarrow$ Die mittlere Pfadlänge in einem durchschnittlichen Suchbaum mit der Wurzel x_i beträgt:

$$\frac{i}{n} \cdot (P(i) + 1) + \frac{n - i - 1}{n} \cdot (P(n - i - 1) + 1) + \frac{1}{n}$$

Der erste Term beschreibt die mittlere Pfadlänge zu Schlüsseln aus S_ℓ (plus Eins, da ja von der "neuen Wurzel" x_i aus gerechnet wird).
Der zweite Term beschreibt entsprechend die mittlere Pfadlänge zu Schlüsseln aus S_r.
Der letzte Term berücksichtigt den Beitrag von x_i.
Gemittelt über alle n möglichen Wahlen der Wurzel folgt hieraus:

$$P(n) = \frac{1}{n^2} \cdot \sum_{i=0}^{n-1} [i \cdot (P(i) + 1) + (n - i - 1) \cdot (P(n - i - 1) + 1)]$$

Beachte: $\sum_{i=0}^{n-1}(n-i-1)\cdot(P(n-i-1)+1) = \sum_{i=0}^{n-1} i\cdot(P(i)+1)$. $\rightsquigarrow$

$$\begin{aligned}
P(n) &= \frac{1}{n^2}\left(2\cdot\sum_{i=0}^{n-1} i\cdot(P(i)+1) + \sum_{i=0}^{n-1} 1\right)\\
&= \frac{1}{n^2}\left(2\cdot\sum_{i=0}^{n-1} i\cdot P(i) + 2\cdot\sum_{i=0}^{n-1} i + n\right)\\
&= \frac{1}{n^2}\left(2\cdot\sum_{i=0}^{n-1} i\cdot P(i) + (n-1)\cdot n + n\right)\\
&= 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} i\cdot P(i)
\end{aligned}$$

Lösen der Rekursion $P(n) = 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} i \cdot P(i)$

Lösungsidee: Summe stört $\rightsquigarrow$ Führe Hilfsgröße $S_n = \sum_{i=0}^{n} i \cdot P(i)$ ein:

1. $P(n) = 1 + \frac{2}{n^2}S_{n-1}$ (aus der alten Rekursion)
2. $S_n = n \cdot P(n) + S_{n-1}$ (aus der Summendefinition)

1. und 2. liefern zusammen:

$$S_n = n + \frac{2}{n}S_{n-1} + S_{n-1} = \frac{n+2}{n}S_{n-1} + n$$

mit den Anfangsbedingungen $S_0 = 0$ (und $S_1 = 1$).

Versuchen wir die **Einsetzungsmethode**: (genauer: Güting)

$$\begin{aligned}
S_n &= n + \frac{n+2}{n}S_{n-1} = n + \frac{n+2}{n}\left((n-1) + \frac{n+1}{n-1}S_{n-2}\right) \\
&= n + \frac{(n+2)(n-1)}{n} + \frac{n+2}{n}\cdot\frac{n+1}{n-1}\left((n-2) + \frac{n}{n-2}S_{n-3}\right) \\
&= n + \frac{(n+2)(n-1)}{n} + \frac{n+2}{n}\cdot\frac{n+1}{n-1}\cdot(n-2) + \frac{n+2}{n}\cdot\frac{n+1}{n-1}\cdot\frac{n}{n-2}S_{n-3}
\end{aligned}$$

$$
\begin{aligned}
S_n &= n + \frac{(n+2)(n-1)}{n} + \frac{n+2}{n} \cdot \frac{n+1}{n-1} \cdot (n-2) + \frac{n+2}{n} \cdot \frac{n+1}{n-1} \cdot \frac{n}{n-2} S_{n-3} \\
&= n + \frac{(n+2)(n-1)}{n} + \frac{n+2}{n} \cdot \frac{n+1}{n-1} \cdot (n-2) + \frac{n+2}{n-1} \cdot \frac{n+1}{n-2} S_{n-3} \\
&= n + \frac{(n+2)(n-1)}{n} + \frac{n+2}{n} \cdot \frac{n+1}{n-1} \cdot (n-2) + \frac{n+2}{n-1} \cdot \frac{n+1}{n-2} \left((n-3) + \frac{n-1}{n-3} S_{n-4} \right) \\
&= (n+2)(n+1) \cdot \left[\frac{n}{(n+1)(n+2)} + \frac{n-1}{n(n+1)} + \frac{n-2}{(n-1)n} + \frac{n-3}{(n-2)(n-1)} + \dots \right] \\
&\leq (n+2)(n+1) \cdot H_{n+2} \quad \text{Erinnerung: Harmonische Zahl}
\end{aligned}
$$

$\rightsquigarrow P(n) = 1 + \frac{2}{n^2} S_{n-1} \leq 1 + \frac{2n(n+1)}{n^2} H_{n+1} \leq 1 + 2H_{n+1} \approx 1 + 2\ln(n+1)$

Üblich sind in der Informatik Logarithmen zur Basis zwei.
Dann können wir noch genauer die Proportionalkonstante angeben:

$$\boxed{P(n) \leq (2 \cdot \ln(2)) \log_2(n) \approx 1,386 \log_2(n)}$$

Hinweis: Löschoperationen verschlechtern den mittleren Fall: Wird eine Folge von mindestens n^2 Update- (kombinierte Lösch- und Einfügeoperationen) ausgeführt, so hat der entstehende Baum eine erwartete Pfadlänge von $\Theta(\sqrt{n})$.

Stimmt die Abschätzung?

Lemma: Ein Baum mit n Knoten hat minimale mittlere Pfadlänge, wenn es 2^ℓ Knoten der Tiefe ℓ gibt mit $0 \leq \ell \leq \log_2(n+1) - 1$.
Genauer gilt:
Es sind $n - 2^{k+1} + 1$ Knoten auf Tiefe $k+1$, mit $k = \lfloor \log_2(n+1) \rfloor - 1$.

$$\begin{aligned} P(n) &\geq \frac{1}{n}\left(\sum_{i=0}^{k}(i+1)2^i + (k+2)(n-2^{k+1}+1)\right) \\ &= \left[k\cdot 2^{k+1} + 1 + (k+2)(n-2^{k+1}+1)\right] \\ &\geq \lfloor (n+1) \rfloor - 1 \end{aligned}$$

Die verwendete Beziehung $\sum_{i=1}^{k} i\cdot 2^i = (k-1)2^{k+1} + 2$ sieht man leicht per Induktion.

Noch ein Differentiationstrick

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{k} i \cdot x^i = x \cdot \sum_{i=1}^{k} i \cdot x^{i-1} = x \cdot \sum_{i=1}^{k} \frac{d}{dx} x^i \\
&= x \frac{d}{dx} \left(\sum_{i=1}^{k} x^i \right) = x \frac{d}{dx} \left(\sum_{i=0}^{k} x^i - 1 \right) = x \frac{d}{dx} \left(\frac{x^{k+1} - 1}{x - 1} - 1 \right) \\
&= \frac{x + x^{k+1} \cdot (kx - k - 1)}{(x-1)^2}
\end{aligned}
$$

Der benötigte Spezialfall folgt durch Betrachtung von $x = 2$.