Algorithmen und Datenstrukturen Zusammenfassung

Maximilian Ortwein

3. Februar 2012

Inhaltsverzeichnis

1 Sortieren

1.1 selection sort

Minimum suchen und an anfang der Liste, dann Liste verkleinern um 1.

1.2 Quicksort

Wähle Pivot element, alle elemente die Größer als pivot sind rechts von Pivot, alle die kleiner als Pivot links von pivot.
Wähle in den Teillisten ein Pivot und mache das von oben.

1.3 Merge Sort

Teile liste in k teile z.b. k = 2 in zwei teile
Teile teillisten wieder in k teile
bis jedes element einzeln, dann füge die teile wieder zusammen und sortiere dabei die teillisten

1.4 Heapsort

Array das Heapeigenschaft erfüllt kann direkt in einen Heap umgewandelt werden. Heapeigenschaf: Die Kinder eines Knotens sind kleiner als der Knoten selbst. In einem Teilbaum eines Knotens kommt kein größerer Wert vor.
Kommt eine neuer Knoten hinzu wird er solange mit seinem parentknoten vertauscht, bis die Heapeigenschaft hergestellt ist.
Laufzeit:$O(\log n)$
wird die wurzel entfernt, wird der unterste rechte knoten als neue wurzel genommen und solange versickert bis die heapeigenschaft wiederhergestellt ist. Laufzeit:$O(n \log n)$
Jedes Sortierverfahren benötigt im schlechtesten fall $\Omega(n \log n)$ Vergleiche

1.5 Bucketsort

Array wird in intervalle (buckets aufgeteilt), und dann die zahlen in die Buckets geworfen. Jedes Bucket wird dann mit irgendeinem algorithmus sortiert und dann werden die Buckets hintereinander geschalten.
Mittlere Laufzeit:$O(n)$

1.6 Countingsort

Für jeden möglichen Wert wird ein Bucket bereitbgestellt. Die Werte werden dann in die entsprechenden Buckets geworfen. Damit ergibt sich eine automatische sortierung.

Laufzeit:$O(n+k)$

1.7 Radix-Sort

Die Zahlen werden nach ihren stellen sortiert. z.B. im 10er System: zuerst werden die 1er stellen Sortiert, dann die 10ner Stellen, dann die 100er Stellen und so weiter. Dabei werden die Zahlen nach den jeweiligen stellen in Fächer eingeordnet und nach jedem sorterschritt wieder von vorne nach hinten entnommen.
Laufzeit: $\Theta(s \cdot (n+d))$

1.8 Radix-Exchange-Sort

Geht nur für Binärzahlen, sortiert alle 0en an den anfang und alle 1en an das ende, ist nicht stabil
Laufzeit:$\Theta(s \cdot n)$

1.9 gegenüberstellung

Algorithmus	Lauf-zeit-klasse worst	-zeit-klasse average	-klasse best	Speicher	stabil	Einschränkung
SelectionSort	n^2	n^2	n^2	$\Theta(1)$	✘	keine
QuickSort	n^2	$n \log n$	$n \log n$	$\mathcal{O}(n)$	✘	keine
MergeSort	$n \log n$	$n \log n$	$n \log n$	$\mathcal{O}(n)$	✔	keine
HeapSort	$n \log n$	$n \log n$	n	$\Theta(1)$	✘	keine
untere Schranke	$n \log n$	$n \log n$	n	n.a.	n.a.	keine
BucketSort	$n \log n$	n	n	$\Theta(1)$	✔	reelle Zahlen aus $(0,1]$
CountingSort	$n+k$	$n+k$	$n+k$	$\Theta(n+k)$	✔	ganze Zahlen aus $(0,k-1)$
RadixSort	$s \cdot (n+d)$	$s \cdot (n+d)$	$s \cdot (n+d)$	$\Theta(n+d)$	✔	d-äre Zahlen, Wortlänge s
RadixExchangeSort	$s \cdot n$	$s \cdot n$	$s \cdot n$	$\mathcal{O}(n)$	✘	Binärzahlen, Bitlänge s

2 Suchen

2.1 Folgen

2.1.1 Lineares Suchen

Laufzeit:$\Theta(n)$
Suchen in einem Unsortiertem Array bzw Liste. Durchlaufen aller Elemente

2.1.2 Selbstanordnende Folgen

3 Strategien:
- MF (move to Front) Der besuchte Schlüssel wird an den Anfang der Liste gesetzt.

- T (transponse) Der besuchte Schlüssel wird mit seinem Vorgänger vertauscht
- FC (Frequency Count) Die Liste wird nach einem Zähler sortiert, der die Zugriffshäufigkeiten angibt
Es gilt für einen Allgemeinen Algorithmus A:
$C_{FM}(S) \leq 2 \cdot C_A(S) + X_A(S) - F_A(S) - |S|$

2.1.3 Sortierte Arrays

Da das Array Sortiert ist muss immer nur noch die Hälfte des Arrays betrachtet werden, da das gesuchte Element größer oder Kleiner als das gefundene Element sein muss.
Laufzeit:$\Theta(\log n)$

2.2 geordnete Wörterbücher

Ein geordnetes Wörterbuch bedeutet, das zu jeder Zeit mit linearem Aufwand jedes paar aus Element und Schlüssel gefunden werden kann.

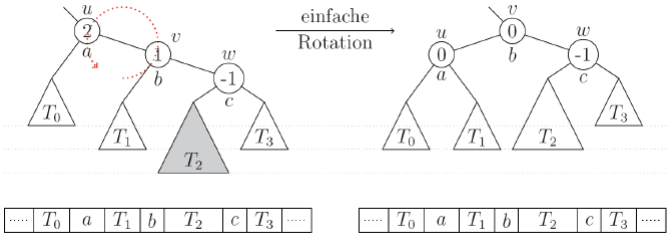
2.2.1 Binäresuchbäume

In einem Knoten werden parent, rechtes und linkes Kind und das item gespeichert. Der Baum wird immer durch inorder durchlaufen (Links Mitte Rechts).
Suchbaumeigenschaft: Die Knoten im Linken Teilbaum eines Knotens sind immer Kleiner als der Knoten selbst und die Knoten im Rechten Teilbaum eines Knotens sind immer größer als der Knoten selbst.
Höhe: min: $\lfloor \log n \rfloor$ max: $n - 1$
ADT-Dictionary:
find(k) Element mit Schlüssel k finden
search(T,k) Element mit Schlüssel k in Baum T suchen
insert(a,k) Element a mit Schlüssel k einfügen
remove(k) Element mit Schlüssel k entfernen

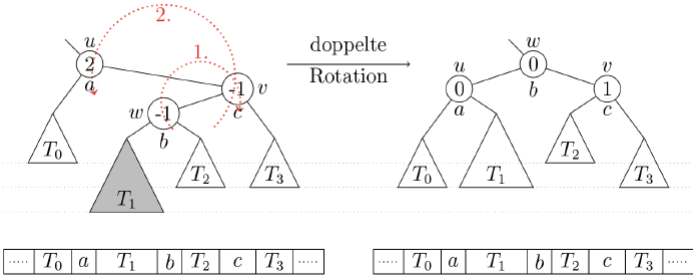
2.2.2 AVL-Bäume

AVL-Bäume sorgen dafür das die Höhe eines Baumes möglichst minimal bleibt.
Für Jeden Knoten eines Binärbaumes gilt, das sich die höhe seiner Kinder um höchstens 1 unterscheidet. Ein AVL-Baum mit n Knoten hat die höhe $\Theta(\log n)$
Bei insert und bei remove muss beim AVL-Baum auf die Balanciertheit geachtet werden. Deshalb werden in jedem Knoten noch die höhenunterschiede der Teilbäume gespeichert. Wenn der Baum unbalanciert ist, muss er rotiert werden.

Einfache Rotation:



Doppelte Rotation:



2.2.3 Rot-Schwarz-Bäume

5 Bedingnungen:
1. Jeder Knoten ist entweder Rot oder Schwarz
2. Root ist immer Schwarz 3. Alle Blattknoten (NIL) sind schwarz 4. ist ein Knoten rot so sind seine Kinder schwarz 5. Jeder Pfad von einem Knoten zu einem Blatt enthält die gleiche Anzahl schwarzer Knoten Einfügen:
Jeder neu eingefügte Knoten wird rot gefärbt.
Wenn der neu eingefügte Knoten die Wurzel ist, muss diese nach Schwarz umgefärbt werden.
Wenn der Vater des neuen Knotens schwarz ist, bleiben alle Eigenschaften erhalten und es ist nichts zu tun.
Sind sowohl Onkel als auch Vater des neuen Knotens rot, werden diese einfach schwarz gefärbt und der Großvater rot das Problem wird damit um eins nach oben verschoben, das wird solange rekursiv durchgeführt, bis alle Eigenschaften wiederhergestellt sind.
Baum ausgleichen durch Roation.

2.2.4 B-Bäume

1. Alle Blätter liegen auf der selben Schicht
2. Alle Knoten mit Ausnahme der Wurzel besitzen mindestens $m = \lceil \frac{t}{2} \rceil - 1$Werte. (d.h. sind min. bis zur hälfte gefüllt) t ist die anzahl der Kinder.

3. Alle inneren Knoten mit m Werten besitzen m+1 nachfolger.
4. Die maximale Zahl an Werten (m) in einem Knoten ist t-1 wobei t die der maximale Knotengrad ist.
5. Suchbaum eigenschaft (Alle werte im linken Teilbaum sind kleiner als die Werte im rechten Teilbaum)

Einfügen:
Ein neuer Knoten kann nur als Blatt eingefügt werden
Wenn ein überlauf auftritt (also die tiefe t oder größer wird) so muss entweder ein Ausgleich mit einem nicht vollen Knoten stattfinden oder ein zweites Blatt angelegt werden und die werte werden auf die beiden blätter verteilt.
Das neu Anlegen von Knoten wird rekursiv bis zur Wurzel durchgeführt, läuft die Wurzel über, so wird diese geteil. Das ist die einzige möglichkeit in der der Baum nach oben wachsen kann.

Löschen:
Das Löschen eines Inneren Knoten wird auf das Löschen eines Blattes zurückgeführt. Dabei ist zu beachten, dass die Suchbaumeigenschaften hergestellt werden.

Ein Baum der Ordnung d mit n Knoten hat eine höhe von $\Theta(\log_d n)$

3 Streuen

Durch eine Hashfunktion werden einem Wert ein Platz in der Hashtabelle zugewiesen.

3.1 Kollisionen

Zwei oder mehr Werte würden das gleiche Feld belegen
Belegungsfaktor = $\frac{n}{m}$

3.2 Kollisionsbehandlung

3.2.1 Verkettung

Die Werte werden in Verketteten Listen abgespeichert. Wenn mehrere zugriffe auf Zeiger in Listen auftreten, nennt man das Sondieren. Es sind $n = m \ln m$ einfügungen nötig um alle Felder zu füllen. der Belegungsfactor ist damit $\beta = \frac{m \ln m}{m}$

3.2.2 Open-Hashing

Wenn eine Kollision auftritt suche freien Platz in der Tabelle, dazu gibt es drei Möglichkeiten:

neuer Platz = $h(k) + d_i$
1. Lieares sondieren: $d_i = i$, beginne bei 1 für i und dann i++
2. Quadratisches sondieren: $d_i = i^2$ wieder für $i = 1 \dots n$
3. Doppelhashing: $d_i = i \cdot h_2(k)$ wobei h_2 zweite hashfunktion ist, und i ist gleich wie oben.

3.3 Kollisionsvermeidung

Wähle aus einer Menge von Hashfunktion zufällig eine Hashfunktion aus.

3.4 Bloomfilter

Von Wörtern wir ein Binärer Hashwert erstellt und dieser in ein Array das nur mit 0en gefüllt ist geschrieben. Wenn überprüft werden soll ob ein Bestimmer String im Array vorhanden ist, wird von diesem String der Hashwert erstellt und geschaut ob dieser eine eins hat wo im array eine 0 ist oder umgekehrt. Ist das der fall so kann der String nicht im array enthalten sein.

4 Ausrichten

Suche nach ähnlichen Zeichenketten in Wörtern
3 Operationen: Substituion, Einfügung, Löschung
Laufzeit: $\Omega(3^{min(m,n)}$
Eine Optimale Ausrichtung kann in $O(nm)$ mit einem Platzverbrauch von $O(min(m,n))$ durchgeführt werden

5 Graphen

Graph: $G = (V, E)$ V sind die Knoten und E sind die Kanten.
Der Kürzerste Weg zwischen zwei Knoten heißt Abstand, Die Anzahl an Knoten mit ungeraden Grad ist gerade in jedem Graphen. Es gibt immer zwei Knoten mit gleichem Grad.

5.1 Adjazent

Adjazent werden zwei Knoten genannt die benachbart sind. In der Adjazentenmatrix steht überall dort eine 1 wenn eine Kante zwischen den beiden Knoten existiert.

5.2 inzident

Eine Beziehung zwischen Knoten und Kante.

5.3 Bäume und Wälder

Ein Zusammenhängender, Kreisfreier Graph heisst Baum. Ein graph der nur aus Bäumen besteht heißt Wald.
-Zwischen zwei Knoten existiert genau ein Weg
- Ein Baum hat n-1 Kanten
- ist minimal zusammenhängend
- ist maximal Kreisfrei

Die Anzahl an Kanten die man aus einem Graphen entfernen muss um seinen Stammbaum zu erhalten heißt zyklomatische zahl = $m - n + k(G)$.

5.4 Durchläufe

5.4.1 Eulertour

Eine Eulertour ist ein Weg der genau alle Kanten einmal enthält. Wenn alle Knoten geraden Grad haben enthält der Graph eine Eulertour

5.5 Tiefensuche

Noch zu besuchende Knoten werden in einem Stack gespeichert. Die suche geht über die Nachbarn in die Tiefe, wenn keine Nachbarn mehr vorhanden sind gehe zum letzen besuchten Knoten und besuche dessen Nachbarn. Wenn gesuchtes Element gefunden wurde breche Suche ab
Laufzeit: $O(n + m)$

5.6 Breitensuche

Alle nicht Besuchten Knoten werden in einer Warteschlange Abgelegt, dann wird von einem Knoten aus jeder Nachbar besucht. Und das solange bis kein Knoten mehr in der Warteschlange ist.
Laufzeit: $O(n + m)$

5.7 Kürzeste Wege

Suche des kürzesten Weges von einem Knoten zu einem Bestimmten oder allen anderen Knoten in einem Gerichteten Graphen.
SSSP = Single-Source Shortest-Path Problem.

5.7.1 Algorithmus von Dijkstra

Der jeweils aktuell Kürzeste Weg von einem Knoten einem anderen wird gespeichert. Ein Knoten der noch nicht besuchte inzidente Kanten hat wird in einer Warteschlange

gespeichert. Für den Jeweils ersten Knoten in der Warteschlange wird nun überprüft welche der inzidenten Kanten den Kürzesten Weg zum nächsten Knoten bildet. Dieser wird mit dem aktuell kürzesten Weg zum gerade betrachteten Knoten addiert und in den neuen Knoten geschrieben. Wenn der neue Knoten bereits einen Weg hat wird der kürzere von beiden genommen.
Funktioniert nur mit nicht negativen Kantengewichten

Laufzeit: $O(m + n \log n)$ oder $O(n + m \log n)$

5.7.2 Algorithmus von Bellman/Ford

Vorteil: Kann auch mit negativen Kantengewichten umgehen.
Nachteil: Benötigt O(nm)
Nicht möglich bei: enthaltene Zykel negativer Länge sonst würde er unendlich diesen Zykel nehmen...
Führt Relaxationen in schematisch fester Reihenfolge durch.

6 amortisierte Analyse

Bei der Account-Methode werden die Gesamt Kosten betrachtet und nicht die Kosten einzelner Operationen.
z.B. Binärzähler:
wechsel von 0 auf 1 Kostet real 1 in unserem Fall 2 dabei wird 1 auf das konto eingezahlt. Wechsel von 1 auf 10 kostet 2 deshalb werden 2 verbraucht, der Kontostand bleibt auf 1. Wechsel von 10 auf 11 kostet 1 deshalb wird wieder 1 auf das Konto gebucht. Wechsel von 11 auf 100 kostet 3, das heisst wir verwenden die 2 und nehmen eins vom Konto. Damit is die laufzeit für einen Binärzähler in $O(n)$