Testfragen zur Vorlesung "Theoretische Informatik I"

Bäume, Traversierung

- T = binärer Baum mit N Blättern $\Rightarrow$ Tiefe(T) $\geqslant$?

  ▷ $N \leqslant 2^{\text{Tiefe}(T)}$ $\Rightarrow$ Tiefe(T) $\geqslant \log_2 N$.

- r = Wurzel, a = linkes Kind, b = rechtes Kind. Wie sehen Post-, Pre- und Inoder Transversierungen aus?

  ▷ Postoder $= T_a \to T_b \to r$, Preoder $= r \to T_a \to T_b$, Inoder $= T_a \to r \to T_b$

Prioritätswarteschlangen $\Rightarrow$ Heaps

- Baum T besitzt Heap-*Struktur*, wenn ...?
- Baum T mit Prioritäten $p : T \to \mathbb{R}$ besitzt Heap-*Ordnung*, wenn ...?

  ▷ ... Kinder kleiner als ihr Vater sind.

- Wie schnell läufen delete-max, repair und insert?

  ▷ $= O(\text{Tiefe}(T))$ $\Rightarrow$ $= O(\log n)$. Warum? Da T Heap $\Rightarrow$ Tiefe(T) $= \Theta(\log n)$.

  delete-max $\Rightarrow$ das *letzte* Heap-element x wird als Wurzel gesetzt $\Rightarrow$ vertausche (wenn nötig) den Wurzel x mit dem grösten Sonn usw. $\Rightarrow$ x rutscht in $O(\log n)$ Schritten in richtigen Platz

  repair $\Rightarrow$ funktioniert genauso

  insert(x) $\Rightarrow$ setze x an der ersten freien Position (als einen neuen Blatt) $\Rightarrow$ rufe repair auf.

- Ist ein Heap für die Suche geeignet?

  ▷ Nein! Prioritäten von Kindern sind nicht sortiert.

- Ist ein Heap für Sortierung geeignet?

  ▷ Ja! Zuerst delete-max (entferne den Vater), dann repair das ältere Kind ist nun der Vater), usw. $\Rightarrow$ Laufzeit $O(n \log n)$.

Sortieralgorithmen

- **Quicksort**
  - Wie läuft es?

    ▷ Rekursiv $Q(i,j)$ wählt ein Pivot $p \in [i,j]$, zerteillt das Array in $A[i,p-1], A[p], A[p+1,j]$ mit $\max A[i,p-1] < A[p] < A[p+1,j]$ und ruft die Prozeduren $Q(i,p-1)$ und $Q(p+1,j)$ rekursiv auf.
  - Ist Quicksort ein Algorithmus oder eine Klasse von Algorithmen?

    ▷ Da die Auswahl von Pivotelementen nicht spezifiziert ist $\Rightarrow$ eine Klasse von Algorithmen.
  - Worts-case Laufzeit?

    ▷ Sehr schlecht = quadratisch. Z.B. wenn Pivot = immer der linkeste $\Rightarrow$ auf $(1,2,\ldots,n)$ werden $Q(1,n) \to Q(2,n) \to \cdots \to Q(n,n)$ aufgerufen $\Rightarrow$ Rekursionstiefe n und Laufzeit $\sum_{i=1}^{n}(n-i) = \Omega(n^2)$.

- Erwartete Laufzeit?

  ▷ Sehr gut $= O(n \log n)$. Wähle Pivot rein zufällig. Sei $V_{i,j}$ Indikatorvariable für das Ereignis "es kommt zum Vergleich der Werte von i und j" $\Rightarrow$ Erwartete Laufzeit[1]

$$= \mathrm{E}\left[\sum_{i<j} V_{ij}\right] = \sum_{i<j} \mathrm{E}\left[V_{ij}\right] = \sum_{i<j} \frac{2}{j-i+1} = O(n \cdot H_n) = O(n \log n)$$

- Rekursion kann man mit dem Stack beseitigen. Welches der Teilprobleme – größeres oder kleineres – muß auf den Stack gelegt werden?

  ▷ Sei m_k die Fröße des *kleineren* Teilproblems im Schritt k $\Rightarrow$ $m_1 \leqslant n/2$, $m_2 \leqslant m_1/2 \leqslant n/2^2$, $\ldots$, $m_k \leqslant n/2^k$ $\Rightarrow$ Stackhöche ist $k \leqslant \log n$ $\Rightarrow$ $k \leqslant \log n$ Aufrufe von Quicksort genügen $\Rightarrow$ Gesaamtlaufzeit $= O(n \log^2 n)$.

  Wenn man aber das *kleinere* Teilproblem auf den Stack legen würde $\Rightarrow$ könnte man bis zu $\Omega(n)$ Schritte mit jeweils Zeitaufwand $\Omega(n \log n)$ brauchen $\Rightarrow$ Gesaamtlaufzeit $= O(n^2 \log n)$.

- **Mergesort**. Wie fuktioniert, welche Laufzeit?

  ▷ Ein rekursives Sortierverfahren (Postoder Reihenfolge): teile das Array in zwei Teilen, sortiere diese teilen und schließlich mische die zwei (bereits sortierten!) Teilarrays.

  Laufzeit $\Rightarrow$ $M(n) \leqslant 2 \cdot M(n/2) + O(n)$ $\Rightarrow$ $M(n) = O(n \log n)$

  $\Rightarrow$ sehr gut $\Rightarrow$ sogar worst-case ist klein $\Rightarrow$ gut für *externen* Sortieren (auf der Festplatte)

- **Distribution Counting** für Zahlen $x_1, \ldots, x_n \leqslant M$. Wie funktioniert es? Laufzeit?

  ▷ Berechne $Z[i]$ = wie oft die Zahl i in Array A vorkommt.

  Laufzeit $= O(n + M)$ $\Rightarrow$ linear für $M \leqslant n$.

- **Radixsort** für Zahlen $x_1, \ldots, x_n \leqslant M$. Wie funktioniert es? Laufzeit? Wie sollte man die Basis b wählen, um Laufzeit $= O(n)$ zu erreichen?

  ▷ Man wählt eine Basis b und stellt die Zahlen in d-ären Darstellung dar $\Rightarrow$ jede Zahl hat $\log_b M$ Ziffern.

  Im i-ten Schritt $i = 1, \ldots, \log_b M$ verteile die Zahlen nach ihrem i-ten Bit in b Queus $0, 1, \ldots, b-1$

  $\Rightarrow$ die Zahlen sind bereits nach der ersten i (kleinsten) Bits sortiert.

  Laufzeit? Es gibt $\log_b M$ Schritte und jeder Schritt braucht Zeit $O(n + b)$

  $\Rightarrow$ Laufzeit $= O(n + b) \log_b M$.

  Ist $M \leqslant n^c$ $\Rightarrow$ wähle $b = n$ $\Rightarrow$ $(n + b) \log_b M = O(n)$.

- **Untere Schranke** für Sortieren? Beweisidee.

  ▷ Ein Sortierbaum T hat Vergleiche von der Form $x_i < x_j$ als Knoten. Eingaben sind Vektoren $a = (a_1, a_2, \ldots, a_n)$ von verschiedenen Zahlen aus kann man $\{1, 2, \ldots, n\}$. Ein Ordnungstyp von a ist die (einzige) Permutation π mit $a_{\pi(1)} < a_{\pi(2)} < \ldots < a_{\pi(n)}$.
  Jede Eingabe wird den einzigen Blatt von T erreichen und alle Eingaben, die denselben Blatt erreichen, denselben Ordnungstyp haben müßen
  Für jeden Ordnungstyp gibt es eine Eingabe, die diesen Typ hat
  $\Rightarrow$ T muß für jeden Ordnungstyp ein Baltt haben
  $\Rightarrow$ T mußmindestens $n!$ Blätter $\Rightarrow$

$$\text{Tiefe}(T) \geqslant \log_2(n!) \geqslant \log_2(n/2)^{n/2} = \frac{n}{2} \log_2 \frac{n}{2} = \Omega(n \log n).$$

[1] H_n = harmonische Reihe.

Das Wortebuchproblem

- **AVL-Bäume**. Wie sehen sie aus? Wie tief sind sie? Wie realisiert man remove und insert Operationen?

  ▷ T ist ein AVL-Baum, wenn *alle* Unterb"aume balanziert: ist v ein Knoten mit dem linken Kind a und dem rechten Kind b $\Rightarrow$ $|\text{Tiefe}(T_a) - \text{Tiefe}(T_b)| \leqslant 1$.
  Wie tief T sein kann? $\Rightarrow$ $\leqslant 2\log n$.
  remove und insert realisiert man mit *Rotationen* (Zick-Zick, Zack-Zick, usw.)

- **Splay-Bäume**. Was ist das? Wie realisiert man remove und insert Operationen $\Rightarrow$ Unterschied von AVL Rotationen? Wo bleiben die Schlüssel, die oft gefragt sind?

  ▷ Was ist das? $\Rightarrow$ Suchbäume mit Splay-Operationen.
  Unterschied von AVL $\Rightarrow$ Erst rotiere den *Großvater*, nur dann den Vater.
  Oft gefragte Schlüssel werden an der Spitze des Baumes bewegt.

- (a, b)**-Bäume** (oder B-Bäume). Was ist das? Wie sind die Schlüssel auf den Knoten verteilt? Wie tief diese Bäume sind?

  ▷ Alle Blätter haben gleiche Tiefe und $a \leqslant \#$ Kinder(v) $\leqslant b$
  Knoten v speichert Schlüsselmenge X_v mit $|X_v| = \#$ Kinder -1.
  Ist $X_v = \{x_1 < x_2 < \cdots < x_k\}$ $\Rightarrow$ $X_{v_1} \subseteq (-\infty, x_1)$, $X_{v_k} \subseteq (x_k, \infty)$ und $X_{v_i} \subseteq (x_{i-1}, x_i)$.
  Wie tief? $\Rightarrow$ $\log_a |T| \leqslant \text{Tiefe}(T) \leqslant \log_b |T|$.

Hashing

- Wie sieht diese Datenstruktur aus?

  ▷ Schlüsselmenge U (sehr groß) und eine Tabelle $T = \{0, 1, \ldots, m-1\}$.
  Hashfunktion = Abbildung $h : U \to T$ $\Rightarrow$ Schlüssel $x \in U$ sitzt in der Zelle $h(x)$.

- Hashing mit Verkettung $\Rightarrow$ Auslatungsfaktor? $\Rightarrow$ erwartete Länge einer Liste?

  ▷ Hashing mit Verkettung $\Rightarrow$ jede Zelle hat einen Zeiger auf eine eigene Liste.
  Auslastunsfaktor $\lambda = n/m$ ($m = \#$ Zellen und $n = \#$ besetzten Zellen).
  Erwartete Länge L_i einer Liste $\Rightarrow$ $p = \Pr\{h(x) = i\} = 1/m$ $\Rightarrow$ $\mathrm{E}[L_i] = n/m = \lambda$.

- Hashing mit offener Adressierung $\Rightarrow$ erwartete Laufzeit einer erfolglosen einer freien Zelle?

  ▷ Hashing mit offener Adressierung $\Rightarrow$ mehrere Hashfunktionen $(h_1, \ldots, h_m)$ anstatt Listen $\Rightarrow$ für jedes $x \in U$ muß $(h_1(x), \ldots, h_m(x))$ eine Permutation der Zellen sein.
  Ist Zelle $h_1(x)$ besetzt $\Rightarrow$ probiere die Zelle $h_2(x)$ usw.
  Erwartete Laufzeit einer erfolglosen Suche $\Rightarrow$

  $$\Pr\{\text{ersten } i \text{ Zellen besetzt}\} = \binom{m-i}{n-i} \Big/ \binom{m}{n} \leqslant \left(\frac{n}{m}\right)^i = \lambda^i$$

  $\Rightarrow$

  $$\mathrm{E}[\text{Laufzeit}] \leqslant \sum_{i=0}^{\infty} \lambda^i = \frac{1}{1-\lambda}.$$

  Interpretation: λ is die W'keit, auf einer besezten Zelle zu stoßen $\Rightarrow$ geometrische Verteilung mit Erfolgswahrscheinlichkeit $1 - \lambda$.

Graphalgorithmen

- **Tiefenfuche, Breitensuche**. Welcher Unterschied? Wer benutzt Stack und wer Queue?

  ▷ Tiefensuche = Nachbarn ins Queue
  Breitensuche = Nachbarn in Stack $\Rightarrow$ die Knoten werden in der Reihenfolge ihres *Abstands* von s besucht $\Rightarrow$ liefert auch kürzeste Wege.

- **Kürzeste Wege** $\Rightarrow$ **Dikstra**. Allgemeiner Schritt $S := S \cup \{w\}$:
  Was ist w?
  Was muss man danach aktualizieren?
  Distanzwert $d[w]$:= Länge eines kürzesten S-Weges zu w? Warum gilt die Gleichheit

  $$d[w] = \text{Länge eines kürzesten Weges zu } w?$$

  ▷ w = Knoten mit kürzestem S-Weg zu s.
  Zu aktualizieren sind Distanzwerte $d[u]$ von Nachbarn u von v.

  Sei $p = (s, \ldots, u, v, \ldots, w)$ ein beliebiger Weg zu w
  $\Rightarrow$ $d[w] \leqslant d[v]$ nach der Auswahl von w
  $\Rightarrow$ $d[v] \leqslant$ Länge(p), denn $(s, \ldots, u, v)$ ein S-Weg zu v ist
  $\Rightarrow$ $d[w] \leqslant d[v] \leqslant$ Länge(p) □

- **Minimale Spannbäume**. *Prim* und *Kruskal*. Welcher Unterschied?

  ▷ Prim = kürzeste *kreuzende* Kante $\Rightarrow$ kein Zyklus möglich
  Kruskal = kürzeste Kante, die keinen Zyklus erzeugt.