



DAP2 Klausur

Datum: 9. Oktober 2006

Zu beachten:

- Leserlich schreiben!
- Nur dokumentenechte Schreibgeräte verwenden!
- Nur Papier verwenden, das von uns ausgeteilt wurde!
- Es sind keine Hilfsmittel erlaubt!

Die Klausur dauert 90 Minuten. Es gibt 60 Punkte zu erreichen. Zum Bestehen der Klausur sind 30 Punkte notwendig.

Wir als Klausurveranstalter sind organisatorisch nicht dazu in der Lage, vor bzw. während der Klausur zu überprüfen, ob die Teilnehmer/-innen dazu berechtigt sind, die Klausur mitzuschreiben bzw. ob sie ordnungsgemäß bei der jeweils zuständigen Stelle angemeldet sind. Daher gilt: Durch Ihre **Unterschrift** erkennen Sie an, dass diese Klausur **unter Vorbehalt** stattfindet, d.h. die Teilnahmeberechtigung und Anerkennung der Klausur wird erst nach der Klausur von der jeweils zuständigen Stelle überprüft und ist nicht automatisch durch die Teilnahme an der Klausur gegeben. Darüber hinaus bestätigen Sie durch Ihre Unterschrift, dass Sie **prüfungsfähig** sind.

Matrikelnummer:

Nachname:

Vorname:

Studienrichtung:

Unterschrift:

Lösungen (blau)
Anmerkungen (grün)

Punkte:

*(Nur vom
Klausurveranstalter
auszufüllen)*

Th1: 10	Th2: 14	Th3: 10	Gesamt:
Th4: 7	Th5: 13	Th6: 6	60

Thema 1. (insg. 10 Punkte) *O-Notation*

Aufgabe 1.a. (4 Punkte)

Ergänze den angegebenen Pseudocode, so dass er eine Worst-Case Laufzeit von $\Theta(n^3)$ und eine Best-Case Laufzeit von $\Theta(n)$ besitzt. Die Variable A bezeichnet ein Array mit den Indizes $1 \dots n$, in dem während dem Ablauf des Algorithmus nur ganze Zahlen zwischen $-n$ und n^2 gespeichert werden dürfen.

Gib eine Belegung des Arrays an, in dem der Worst- bzw. Best-Case auftritt.

```
for  $j := n$  to 1 do
  if  $A[j] = \mathbf{n^2}$  then
    for  $i := \mathbf{1}$  to  $\mathbf{n^2}$  do
       $A[j] := A[j] - 1$ 
    end for
  end if
end for
```

Worst-Case Belegung von A : $A[1] = \mathbf{n^2}$, $A[2] = \mathbf{n^2}$, $A[3] = \mathbf{n^2}$, \dots , $A[n] = \mathbf{n^2}$

Best-Case Belegung von A : $A[1] = \mathbf{1}$, $A[2] = \mathbf{1}$, $A[3] = \mathbf{1}$, \dots , $A[n] = \mathbf{1}$

Anmerkungen siehe Seite 14

Aufgabe 1.b. (3 Punkte)

Gib die Θ -Notation für die Laufzeitfunktion $f(n) := n^3 \log(n^3) + (n \log(n))^3$ an und beweise sie formal.

$$f(n) = \Theta((n \log n)^3)$$

Beweis:

$$\Theta(f(n)) = \{ g(n) > 0 \mid \exists c_1, c_2, n_0 > 0: \forall n > n_0: c_1 f(n) \leq g(n) \leq c_2 f(n) \}$$

$$c_1 (n \log n)^3 \leq n^3 \log n^3 + (n \log n)^3 \leq c_2 (n \log n)^3$$

$$c_1 (n \log n)^3 \leq n^3 3 \log n + (n \log n)^3 \leq c_2 (n \log n)^3 \quad | : (n \log n)^3$$

$$c_1 \leq 3 / (\log n)^2 + 1 \leq c_2$$

$$\lim_{n \rightarrow \infty} 1 / (\log n)^2 = 0 \quad \leftarrow \text{monoton fallende Nullfolge}$$

Obige Gleichung gilt für $c_1=1$ (da der Bruch >0), und z.B. $c_2=4$ (da $(\log n)^2 \geq 1$), für alle $n > n_0=2$.

Anmerkungen siehe Seite 15

Aufgabe 1.c. (3 Punkte)

Es gelte $g(n) = O(h(n))$. Vereinfache den folgenden Ausdruck mit Hilfe der O -Notationen (O , Ω , etc.).

$$O(g(n)) \setminus o(h(n)) = \begin{cases} \Theta(g(n)) & \text{falls } g(n) = \Theta(h(n)) \\ \text{leere Menge} & \text{sonst.} \end{cases}$$

Thema 2. (insg. 14 Punkte) *Sortieren*

Betrachte den folgenden Algorithmus, der ein Feld $A[1..n]$ sortiert.

```
1: procedure GNOMESORT(ref A)
2:    $i := 2$ 
3:   while  $i \leq n$  do
4:     if  $A[i - 1] \leq A[i]$  then
5:        $i := i + 1$ 
6:     else
7:       Vertausche  $A[i - 1]$  mit  $A[i]$ 
8:        $i := \max\{i - 1, 2\}$ 
9:     end if
10:  end while
11: end procedure
```

Aufgabe 2.a. (7 Punkte)

Überlege, wie GnomeSort funktioniert, und beantworte folgende Fragen:

Ist der Algorithmus *adaptiv*?

Ja Nein

Ist der Algorithmus *stabil*?

Ja Nein

Ist der Algorithmus *in-situ*?

Ja Nein

Sortiert der Algorithmus auch richtig, wenn gleiche Schlüssel vorkommen?

Ja Nein

Funktioniert der Algorithmus nur mit ganzen Zahlen?

Ja Nein

Was ist die Best-Case Anzahl der Vertauschungen?

0

Was ist die Worst-Case Anzahl der Vertauschungen?

$\Theta(n^2)$

Was ist die Best-Case Laufzeit?

$\Theta(n)$

Was ist die Worst-Case Laufzeit?

$\Theta(n^2)$

Aufgabe 2.b. (4 Punkte)

Sortiere die folgende Zahlenfolge mittels GnomeSort, und gib den Inhalt von A nach den ersten vier Vertauschungen an.

	1	2	3	4	5	6	7	8
A :	4	1	2	5	7	3	6	0

1 4

2 4

3 7

3 5

Zwischenschritte
(waren nicht nötig) →

	1	2	3	4	5	6	7	8
A :	1	2	4	3	5	7	6	0

Aufgabe 2.c. (3 Punkte)

Wodurch wird bei Quick-Sort der zusätzliche Speicherbedarf verursacht?

Rekursionsstack

Quick-Sort kann mit logarithmischem Speicheraufwand realisiert werden. Dafür wird eine Partition rekursiv, die andere iterativ berechnet. Welche wird rekursiv berechnet, und warum führt das zu logarithmischem Speicheraufwand?

die kleinere Partition die größere Partition

Begründung:

der rekursiv zu bearbeitende Teil wird jedesmal mindestens halbiert
--> logarithmisch viele rekursive Aufrufe --> log. viel zus. Speicher

Was ist die schlechtest mögliche Laufzeit von randomisiertem Quick-Sort?

$O(n^2)$

Thema 3. (insg. 10 Punkte) *Suchen*

Betrachte Suchstrukturen (Bäume, Skiplisten) mit jeweils n Schlüsseln. B-Bäume haben die Ordnung m . Beantworte die folgenden Fragen.

Aufgabe 3.a. (3 Punkte) AVL-Baum

Die Differenz der Tiefe zweier Blätter ist (abhängig von n) maximal:

linear
 logarithmisch
 konstant

Was ist die Laufzeit einer Rechts-Rotation?

$O(\quad 1 \quad)$

Was ist die Laufzeit einer Rechts-Links-Rotation?

$O(\quad 1 \quad)$

Wie viele Rotationen (in O -Notation) sind beim Löschen aus einem AVL-Baum nötig?

$O(\quad \log n \quad)$

Beim Einfügen in einen AVL-Baum kann eine Links-Rechts-Rotationen nötig sein. Kann eine solche Links-Rechts-Rotation auch beim Löschen notwendig sein?

Ja Nein

Aufgabe 3.b. (3 Punkte) B-Baum

Was ist die kleinste Anzahl von Zeigern in einem inneren Knoten (\neq Wurzel)?

$\lceil m/2 \rceil$

Was ist die größte Anzahl von Zeigern eines Knoten?

m

Betrachte zwei anfangs leere B-Bäume in die jeweils n Zahlen eingefügt werden: in den ersten werden die Zahlen in absteigend sortierter Reihenfolge eingefügt, in den zweiten Baum werden die Zahlen in zufälliger Reihenfolge eingefügt. Wie groß ist die erwartete Differenz der Tiefe der resultierenden Bäume (abhängig von n)?

linear
 logarithmisch
 konstant

Aufgabe 3.c. (4 Punkte) Skipliste

\mathcal{L} ist eine Skipliste mit n Elementen. Beim Erzeugen von \mathcal{L} haben wir eine „unfaire Münze“ für die Höhenbestimmung benutzt: der Münzwurf ergab in einem Viertel der Fälle „Kopf“ (bei „Kopf“ wird die Höhe des Elementes erhöht).

Wieviele Vergleiche erwarten wir beim Suchen in einer korrekten randomisierten Skipliste pro Ebene?

1

Wieviele Vergleiche erwarten wir beim Suchen in \mathcal{L} pro Ebene (in Abhängigkeit von n)?

- linear
- logarithmisch
- konstant

Was ist die erwartete Höhe von \mathcal{L} in O -Notation?

$O(\log n)$

Welche Laufzeit können wir beim Suchen in \mathcal{L} erwarten?

$O(\log n)$

Thema 4. (insg. 7 Punkte) *Hashing*

Aufgabe 4.a. (4 Punkte) Betrachte ein Hasharray \mathcal{H} der Größe n mit Double Hashing. Die Hashfunktionen sind

$$h_1(k) := (k \bmod m_1) + d_1 \text{ und } h_2(k) := (k \bmod m_2) + d_2$$

für geeignete Konstanten $m_1, m_2, d_1, d_2 \in \mathbb{N}$. Sind folgende Aussagen korrekt (K) oder nicht korrekt (NK)? Bitte ankreuzen.

$n = 2^r - 1$ mit $r \in \mathbb{N}$ ist eine gute Wahl, da Arraygrößen die nicht durch 2 teilbar sind von Vorteil für das Hashverfahren sind.

K NK

$n = 2^r$ mit $r \in \mathbb{N}$ ist eine gute Wahl, da Computer durch die Binärdarstellung mit Zweierpotenzen effizient rechnen können.

K NK

$n \in \mathbb{P}$ (\mathbb{P} ist die Menge der Primzahlen) ist eine schlechte Wahl, da der Hashwert und n dann immer teilerfremd sind.

K NK

$m_2 = d_2 = 1$ ist equivalent zu linearem Sondieren.

K NK

$m_1 = n$ ist eine gute Wahl, aber dann muss $d_1 = 0$ gelten.

K NK

$d_1 = d_2 = 0$ ist eine schlechte Wahl, da dadurch nicht immer alle Array-Indizes erreicht werden können.

K NK

Um sicherzustellen, dass alle Array-Indizes gleichmässig sondiert werden können, muss gelten: $m_2 < m_1$ und $d_1 = d_2$.

K NK

Um sicherzustellen, dass alle Array-Indizes gleichmässig sondiert werden können, muss gelten: $m_2 \leq m_1$ und $d_1 + d_2 > 0$.

K NK

Aufgabe 4.b. (3 Punkte)

Betrachte folgendes teilgefülltes Hasharray der Größe n mit der folgenden multiplikativen Hashfunktion mit Sondieren:

$$h(k, i) = (\lfloor n(kA \bmod 1) \rfloor + 4i) \bmod n$$

wobei $A = \frac{1}{20}$ und $n = 10$.

Füge den Schlüssel 34 in das folgende Hasharray ein.

Hasharray vor Einfügen:

0	1	2	3	4	5	6	7	8	9
	22	64		28		52	34 ¹		

Welche Positionen werden sondiert?

7, 1, 5

Andere Gruppe: 4, 7, 0

Hasharray nach Einfügen des Schlüssels 34:

0	1	2	3	4	5	6	7	8	9
	22	64		28	<u>34</u>	52	34 ¹		

¹Fehler in Angabe. Vor/Während der Klausur erfolgte der Hinweis, dass hier eine (beliebige) andere Zahl hingehört (z.B. 54).

Thema 5. (insg. 13 Punkte) *Graphen*

Aufgabe 5.a. (5 Punkte) *Definitionen*

Eine Gruppe T von trinkfesten Informatikstudenten fliegt gemeinsam auf eine griechische Insel um zu urlaubeu. Dabei besuchen sie die Lokale L . Sie trinken ausschliesslich große Biere, für die jedes Lokal i ($1 \leq i \leq |L|$) den Preis π_i verlangt. Um die Reise interessanter zu machen, zahlt niemand für sein eigenes Bier, sondern wird immer von einem seiner Freunde eingeladen. Diese Einladungen E kann man graphentheoretisch als gerichtete und gewichtete Kanten vom Zahlenden zum Trinkenden auffassen.

Wie ist der Graph G aufgebaut?

$$G = (\mathbf{T}, E), \text{ mit Gewichtsfunktion } w : \mathbf{E} \longrightarrow \{ \pi_i \mid 1 \leq i \leq |L| \}$$

Gib die „Fairness“-Bedingung an, die erfüllt sein muss, damit kein Student mehr bezahlt als er ausgegeben hätte, wenn er für sich selbst bezahlt hätte.

$$\forall t \in T : \sum_{e \in A^-(t)} w(e) \leq \sum_{e \in A^+(t)} w(e)$$

Kann G Mehrfachkanten enthalten?

Ja Nein

Kann G Schleifen enthalten?

Ja Nein

Kann G Kreise enthalten?

Ja Nein

Können die Studenten für jeweils unterschiedliche Geldsummen Bier trinken, ohne damit die Fairness-Bedingung zu verletzen?

Ja Nein

Aufgabe 5.b. (2 Punkte) *Kürzeste Wege (Theorie)*

Benutze den Algorithmus von Dijkstra um den kürzesten Weg von einem Knoten s nach t in einem Graphen G zu berechnen. Nimm an, dass alle Knoten von s aus erreichbar sind.

Welcher ADT (außer dem Graphen selbst) wird bei Dijkstra benötigt?

Priority Queue

Anhand welchem der unten genannten Kriterien kann man feststellen, dass der Algorithmus den kürzesten Weg von s nach t berechnet hat, so dass man den Algorithmus frühzeitig abbrechen kann?

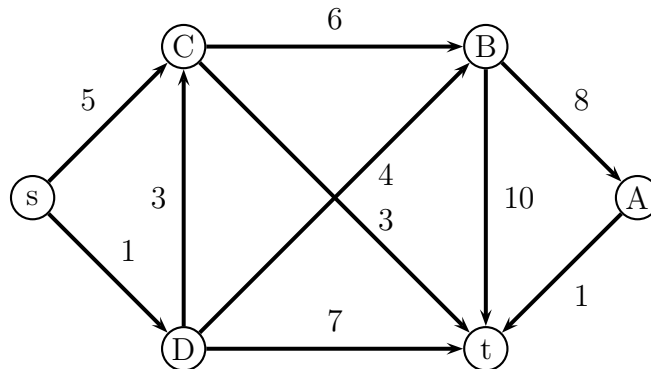
- Sobald alle Kanten betrachtet wurden.
- Sobald alle Knoten einmal erreicht wurden.
- Sobald alle aus t ausgehenden Kanten betrachtet wurden.
- Sobald die erste aus t ausgehende Kante betrachtet wird.
- Sobald alle in t eingehenden Kanten betrachtet wurden.
- Sobald die erste in t eingehende Kante betrachtet wird.
- Sobald alle von s ausgehenden Kanten betrachtet wurden.

Anmerkung zur Angabe:

- *) "Anhand welchem..." (nicht "welcher") => Eine Antwort ankreuzen**
- *) "frühzeitig abbrechen" => Wenn alle Kanten/Knoten betrachtet wurden, ist das kein frühzeitiger Abbruch**

Aufgabe 5.c. (6 Punkte) Kürzeste Wege (Praxis)

Berechne für das angegebene Beispiel den kürzesten Weg von s nach t mittels dem Algorithmus von Dijkstra. Der Algorithmus stoppt sobald als möglich.



Gib die Reihenfolge der Kanten an, in der diese zum Shortest-Path-Tree (SPT) hinzugefügt werden.

(s,D) , (D,C) , (D,B) , (C,t)

Für welche Knoten hat der Algorithmus den kürzesten Weg beim Abbruch schon fertig berechnet? Zähle diese auf:

s, D, C, B, t

Kantengewichte sind keine Kanten(bezeichner)! Das hätte auffallen können, da zB. die "3" doppelt vorkommt, und daher nicht klar ist, welche Kante mit "3" gemeint ist...

Wenn Kanten gesucht sind, sollte die Antwort nicht aus Knoten bestehen.
Wenn Knoten gesucht sind, sollte die Antwort nicht aus Kanten bestehen.

Thema 6. (insg. 6 Punkte) *Sweepline*

Betrachte den in der Vorlesung vorgestellten Sweepline-Algorithmus zum Zählen der Schnitte von isoorientierten Liniensegmenten. Nehmen wir an, jedes Segment ist in einer von drei Farben (rot, grün, blau) eingefärbt; uns interessiert nur die jeweilige Anzahl der Kreuzungen von Segmenten gleicher Farbe.

Sei n_r, n_g, n_b die Anzahl der roten, grünen bzw. blauen Segmente, und $n := n_r + n_g + n_b$. Wir suchen die Anzahl k_r, k_g, k_b der einfarbigen Kreuzungen roter, grüner bzw. blauer Segmente miteinander. Sei k' die Anzahl aller Kreuzungen, und $k^* := k_r + k_g + k_b$.

Wir betrachten zwei verschiedene Algorithmen. Gib die jeweilige Laufzeit möglichst genau und gekürzt in O -Notation an.

- A1** Der Algorithmus aus der Vorlesung. Beim Erkennen einer Kreuzung werden die Farben der beteiligten Segmente betrachtet: sind sie gleich, so wird die der Farbe entsprechende Kreuzungszählvariable hochgesetzt. Ansonsten wird die Kreuzung ignoriert.

$$O(n \log n + k')$$

- A2** Der Algorithmus erzeugt zunächst für jede Farbe eine eigene Liste der entsprechenden Segmente, und wendet den Algorithmus aus der Vorlesung dann auf jede dieser Listen unabhängig an.

$$O(n \log n + k^*)$$

Alternativ gilt auch: $O(n_r \log n_r + n_g \log n_g + n_b \log n_b + k^*)$

Begründung: ausgehend von der langen (ungekürzten, und daher nicht komplett gültigen) Lösung

$$n_r \log n_r + k_r + n_g \log n_g + k_g + n_b \log n_b + k_b$$

kann man die k_x zu k^* zusammenfassen. => lange gültige Lösung

Darüber hinaus: Da $n = n_r + n_g + n_b$ muss mindestens eines der n_x in $O(n)$ sein. => kurze gültige Lösung

In der Lösung sollte kein "R" als Variable vorkommen, wenn in der Angabe nichts dazu steht.

Für diese Aufgabe musste man vom eigentlichen Sweepline-Algorithmus außer der Laufzeit nichts wissen! Dennoch haben fast 70% hier nichts oder nichts auch nur irgendwie halbwegs verwertbares hingeschrieben... (zB. schon die "Lösung" $n \log n$ (einfach so) hat zu Teilpunkten geführt)

Raum für Notizen:

Anmerkungen zu Aufgabe 1.a)

Um Best-&WorstCase zu erreichen, muss die äußere Schleife n -mal durchlaufen werden, die innere im BestCase nie, im WorstCase n^2 -mal. Man muss darauf achten, dass die Bedingung in Zeile 2 so aussieht, dass nur Arrayeinträge n^2 -mal verändert werden, bei denen die resultierenden Werte im erlaubten Zahlenbereich bleiben.

Häufige Fehler:

- *) Die Variable der äußeren Schleife (j) wird stets als Index im Array benutzt. Daher kann ich die Schleife nicht von n bis $2n$ laufen lassen!
- *) Die im Array erlaubten Werte wurden nur allzu gerne ignoriert...

Raum für Notizen:

Anmerkungen zu Aufgabe 1.b)

Logarithmen haben Rechenregeln!

$\log n^3 = 3 \log n$ ist etwas anderes als $(\log n)^3 = \log n * \log n * \log n$

Daher (für $n \geq 4$): $\log n^3 < (\log n)^3$

Wenn man einen Beweis führt, kann man nicht einfach "irgendwann" zu einem Term sagen, dass er eine Nullfolge ist. Um diese Aussage zu treffen, gilt es zunächst einmal den Term zu kürzen. Wenn beim Bruch oben und unten ein wachsender Term steht, kann man i.A. noch keine Konvergenzaussagen treffen.

Über ein Viertel aller Teilnehmer wollten $\Theta(n^3 \log n^3)$ (bzw. das Analogon bei der anderen Gruppe) zeigen: mittels genau dem reziproken ungekürzten Bruch, mit dem $\Theta(n^3 \log^3 n)$ dank Nullfolgenargument bewiesen wurde... Das zeigt, wie wichtig das Kürzen ist: dadurch erkennt man erst, welcher "Beweis" richtig ist...

Lösungen vom Typus $\Theta(n^3)$ (21% der Teilnehmer) lassen uns erschauern. Dass Logarithmen in der O -Notation eine Bedeutung haben, sollte jedem, der eine DAP2 Klausur schreibt, spätestens seit den Sortierverfahren klar sein...

Raum für Notizen: