Lehrstuhl für Informatik I
Effiziente Algorithmen und
wissensbasierte Systeme
Universität Würzburg

Würzburg, den 8. Februar 2016

Prof. Dr. Alexander Wolff
M.Sc. Krzysztof Fleszar

Vorlesung Algorithmen und Datenstrukturen WS 2015/16

Klausur - Gruppe A = B

Muster-Lösung

Vorname: _____ Name: _____

Matrikelnummer: _____

Aufgabe	1	2	3	4	5	6	7	8	**Gesamt**
mögliche Punkte	3	4	4	5	3	4	4	6	33
erreichte Punkte									

Zusatzaufgabe: Aufgabe 8(d)–(e) mit insgesamt 8 Zusatzpunkten.

Bearbeitungszeit: 90 Minuten. Mit 16 der 33 (+8) Punkte haben Sie bestanden.

Aufgabe 1

/ 3 Punkte

Zeigen Sie mittels vollständiger Induktion, dass für alle $k \in \mathbb{N}_{\geq 0}$ gilt:

$$2\sum_{i=0}^{k} 3^i = 3^{k+1} - 1$$

IA: [1 P]
Verwendung der IV: [1 P]
IS korrekt: [+1 P]

IA: $(k=0)$

$$2\sum_{i=0}^{0} 3^i = 2 \cdot 3^0 = 2 = 3 - 1 = 3^{0+1} - 1 \quad \checkmark$$

IS: $(k \to k+1)$

$$2\sum_{i=0}^{k+1} 3^i = 2 \cdot 3^{k+1} + 2 \cdot \sum_{i=0}^{k} 3^i$$

$$\overset{IV}{=} 2 \cdot 3^{k+1} + 3^{k+1} - 1$$

$$= 3^{k+2} - 1 \quad \checkmark$$

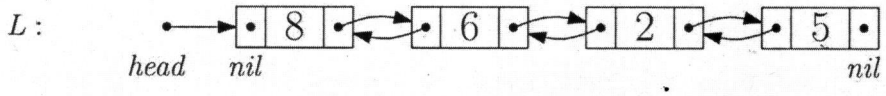
Aufgabe 2

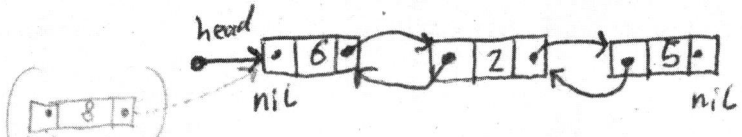
/ 4 Punkte

Gegeben sei folgende Methode:

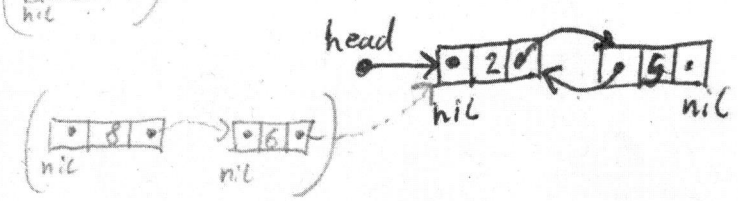
```
SomeMethod(List L, int k)
    x = L.head
    while x ≠ nil and k > 0 do
        x = x.next
        if x ≠ nil then
            x.prev = L.head.prev
        L.head = x
        k = k − 1
```

SomeMethod wird auf die unten abgebildete Liste L mit dem Parameter $k = 2$ angewandt. Zeichnen Sie die Liste nach jedem Schleifendurchlauf nochmal hin. (Wir nehmen an, dass die Methode Zugriff auf die Attribute der Listen und der Listenelemente hat.)





[1 P]



[1 P]

Was macht die Methode mit L in Abhängigkeit von k?

Sie entfernt die ersten $\min\{k, L.length\}$-Elemente vom Listenanfang.

[1,5 P] **[0,5 P]**

Aufgabe 3

/ 4 Punkte

Gegeben sei ein Feld $A = \langle s, s, s, \ldots, s, w, w, w, \ldots, w \rangle$ der Länge n. Geben Sie in gut kommentiertem Pseudocode einen Algorithmus an, der die Länge der s-Sequenz in Zeit $O(\log n)$ bestimmt.

```
int SLength(int[] A)
n = A.length
if n == 0 then return 0
if A[n] == "s" then return n
l = 1, r = n
while (l < r)
    m = |(l+r)\2|
    if A[m] == "w" then r = m
    else               l = m+1
return l-1
```

insg. [0 P] wenn alle s -Elemente der Reihe nach gezählt
[1 P] richtige Idee (Binäre-Suche) -> Laufzeit
[1 P] (gute) Kommentare
[1 P] weitgehendst korrekt
[+1 P] ganz korrekt (Sonderfälle aufgefangen)

(Die Studis dürfen annehmen, dass mindestens ein s und ein w in A ist.)

Aufgabe 4

/ 5 Punkte

Sie möchten eine Wüste möglichst schnell überqueren. Sie haben eine anfangs volle Wasserflasche dabei, die für k Kilometer ausreicht. Auf Ihrem Weg passieren Sie n Oasen, wo Sie Ihre Flasche wieder auffüllen können. Da das Auffüllen jedoch lange dauert (Sie müssen erst das Wasser aus einem tiefen Brunnen holen), möchten Sie möglichst wenige Zwischenstopps vornehmen.

Geben Sie in gut kommentiertem Pseudocode einen Algorithmus CountMinStops an, der die minimale Anzahl der Zwischenstopps zurückgibt. Als Eingabe erhält der Algorithmus den Wert k sowie ein Feld A der Länge n, wobei $A[i]$ für jedes $i = 1, \ldots, n$ den Abstand (in km) der i-ten Oase zum Startpunkt enthält. Die n-te Oase ist das Ziel. Falls das Ziel nicht erreicht werden kann, geben Sie ∞ zurück.

int CountMinStops(int[] A, int k)

```
n = A.length
if A[1] > k then return ∞
s = 0
W = k - A[1]
for i = 2 to n do
    if A[i] - A[i-1] > W then
        if A[i] - A[i-1] > k then return ∞
        W = k
        s = s + 1
    W = W - (A[i] - A[i-1])
return s
```

[1 P] richtige Idee
[+0,5 P] (gute) Kommentare
[+0,5 P] Korrektheit

Um was für einen Typ von Algorithmus handelt es sich? **[1 P]**

Greedy

Analysieren Sie die Worst-Case-Laufzeit Ihres Algorithmus in Abhängigkeit von der Anzahl n der Oasen.

n Schleifendurchläufe mit jeweils O(1) Laufzeit **[1 P]**

→ O(n)

Beweisen Sie die Korrektheit Ihres Algorithmus: Warum liefert er die kleinstmögliche Anzahl von Zwischenstopps?

[0,5 P] : guter Ansatz
[+0,5 P] : Beweis korrekt/vollständig

(Austauschargument)

Optimale Lösung kann in unsere Lösung verwandelt werden ohne Stoppanzahl zu erhöhen:

Solange OPT ≠ unsere Lösung, betrachte ersten Stopp i von OPT ≠ Stopp i unserer Lösung

Da unser Algorithmus spät-möglichst stoppt:

Stopp i von OPT < Stopp i unserer Lösung

⇒ Setze Stopp i von OPT = Stopp i unserer Lösung (zulässig, da auch in unserer Lösung zulässig.)

Aufgabe 5

/ 3 Punkte

Beim doppelten Hashing ist es wichtig, dass die Hashfunktionen zueinander passen. Argumentieren Sie bei den folgenden Hashfunktionen, warum sie als Funktion $h_1(k)$ für das doppelte Hashing in einer Tabelle $T[0..31]$ mit

$$h(k) = (h_0(k) + ih_1(k)) \bmod 32$$

und

$$h_0(k) = (3k + 5) \bmod 32$$

nicht geeignet sind.

Geben Sie hierzu jeweils auch ein k an, das die Problematik aufzeigt.

jeweils:
[0,5 P]: Erklärung
[0,5 P]: k

(a) $h_1(k) = 30 - 2 \cdot (k \bmod 15)$

$h_1(1) = 28$ nicht teilerfremd zu 32 $(k=1)$

($h_1(0) = 30$ ✓)

(b) $h_1(k) = (k + 3) \bmod 10$

$h_1(2) = 5$ nicht teilerfremd zu 32. $(k=2)$

($h_1(0) = 3$ ✓)

(c) $h_1(k) = (29 - 2k) \bmod 19$

$h_1(1) = 27 \bmod 19 = 8$ nicht —"— $(k=1)$

($h_1(0) = 10$ ✓)

Aufgabe 6

/ 4 Punkte

Wählen Sie aus der folgenden Menge von Funktionen eine maximale Teilmenge aus, deren Summe in $O(n^{3/2} \log n)$ liegt. Kreisen Sie die gewählten Funktionen ein!

$\{ 7n^2,$ $\frac{1}{2}n^3,$ $n^2 - n^{1/2},$ $n^{1.5} \log_2(100n),$

$(n+1)^{3/2} \log_2 n,$ $n^{1.1},$ $2^n,$ $100000^{\log_2 n},$

$1000^{100^{10}} n \cdot (\log_2 n)^{100^{10}},$ $n\sqrt{n},$ $n^{\log_2 n},$ $2^{\log_2 n},$

$(\sqrt{8})^{\log_2 n},$ $2^{3 \log_2 \sqrt{n}} \cdot \sqrt{n},$ $(\sqrt{2})^{\log_2(n^3)} \log(n^3),$ $3^{3/2 \cdot \log_2 n}$ $\}$

[-0,5 P] für jede Funktion, die fälschlicherweise (nicht) markiert wurde.

Aufgabe 7

/ 4 Punkte

Gesucht ist ein Algorithmus, der ermittelt, wieviele Züge man mindestens benötigt, um auf einem Schachbrett mit $n \times n$ Feldern einen Springer von Feld (x_1, y_1) zu Feld (x_2, y_2) zu bewegen.

Die Schachregeln legen für jedes Feld (x, y) eine Menge $Z_{(x,y)}$ von bis zu acht Feldern fest. In einem Zug muss der Springer von seinem Feld (x, y) auf eines der Felder in $Z_{(x,y)}$ bewegt werden.

Geben Sie *in Worten* einen effizienten Algorithmus an, der die erforderliche Anzahl an Zügen in $O(n^2)$ Zeit berechnet. Ist das Zielfeld nicht erreichbar, so soll ∞ zurückgegeben werden.

Sie dürfen Methoden aus der Vorlesung verwenden.

Definiere Graph $G = (V, E)$ mit

$V = \{ (x, y) \mid 1 \leq x, y \leq n \}$

$E = \{ \{(x, y), (x', y')\} \mid (x, y) \in V \text{ und } (x', y') \in Z_{(x,y)} \}$

Führe Breitensuche in G ausgehend von (x_1, y_1) aus und speichere Distanzen.
→ Liefert Distanzen (Anzahl Züge) von (x_1, y_1) zu jedem anderen Feld.

Gebe Distanz in Feld (x_2, y_2) zurück.

[1 P]: Idee: Modellierung als Graph
[+1 P]: Graph sinnvoll definiert
[+1 P]: BFS
[+1 P]: Laufzeit

Bei anderen Ansätzen:
insg. [0 P] wenn Ansatz nicht polynomiell
[1 P]: gute Idee mit polyn. Laufzeit (effizient)
[+1 P]: Laufzeit in O(n^2)
[+2 P]: Korrektheit

Aufgabe 8

Sei A ein Feld der Länge n. Eine *Inversion* ist ein Paar (i, j) mit der Eigenschaft, dass $1 \leq i < j \leq n$ und $A[i] > A[j]$.

[-1 P] wenn statt Indexe Werte verwendet insg. [0 P] bei sonst. Fehlern

(a) Geben Sie alle Inversionen im Feld $\langle 2, 3, 8, 6, 1 \rangle$ an. / 2 P.

1 2 3 4 5

(1,5), (2,5), (3,4), (3,5), (4,5)

(Als Werte: (2,1), (3,1), (8,6), (8,1), (6,1) ← Falsch, da Werte

(b) Geben Sie ein Feld der Länge n mit Wertebereich $\{1, \ldots, n\}$ an, bei dem die Anzahl der Inversionen maximal ist (über alle Felder der Länge n mit gleichem Wertebereich). / 2 P.

$\langle n, n-1, n-2, \ldots, 2, 1 \rangle$ **[1 P]**

Wie viele Inversionen sind es? $\frac{n(n-1)}{2}$ **[1 P]**

(c) Erklären Sie, wie die Laufzeit $T(A)$ von InsertionSort von der Anzahl $I(A)$ der Inversionen im Feld A abhängt, wobei $n = A.length$. / 2 P.

$T(A) = \Theta(n + I(A))$ **[1 P]: T(A) (Kein Punktabzug wenn Theta/O fehlt.)**
[1 P]: Erklärung

Pro Inversion muss ein Swap (= while-Schleifen durchlauf) stattfinden.

(d) / 4 Zusatzpunkte

Die Datenstruktur Rot-Schwarz-Baum soll um eine Methode CountSmaller(k) augmentiert werden, die die *Anzahl* der Elemente im Baum bestimmt, die kleiner als die Eingabe k sind. Die Laufzeit von CountSmaller soll in $O(\log m)$ sein, wobei m die aktuelle Anzahl der Elemente im Baum ist.

Hierzu wird für jeden Knoten v das Attribut $v.size$ eingeführt, das die Anzahl der Elemente im Teilbaum mit Wurzel v speichert.

Begründen Sie, warum der Wert von *size* in allen Knoten aufrechterhalten werden kann, ohne die asymptotische Laufzeit der übrigen Rot-Schwarz-Baum-Methoden zu ändern:

v.size lässt sich aus v.left.size und v.right.size berechnen, **[1 P]**
Nach der Vorlesung kann man somit size in allen Knoten aufrechterhalten ohne die asympt. Laufzeit der R-S-Methoden zu ändern.

Vervollständigen Sie die angegebene rekursive Methode CountSmaller.

```
int CountSmaller(int k, Node x = root)
    if x == nil then
        return 0
    if k < x.key then
        return CountSmaller(k, x.left)
    else if k > x.key then
        return x.left.size + 1 + CountSmaller(k, x.right)
    else
        if x.left == nil then
            return 0
        else return x.left.size
```

[1 P]: "return 0 "-Lücken korrekt
[1 P]: "return x.left.size "-Lücken korrekt
[1 P]: rekursive Aufrufe korrekt

(e) / 4 Zusatzpunkte

Geben Sie in gut kommentiertem Pseudocode eine Methode CountInversions an, die die Anzahl der Inversionen eines Feldes A der Länge n in Zeit $O(n \log n)$ bestimmt. Verwenden Sie hierzu CountSmaller. Langsamere Lösungen geben hier weniger Punkte.

```
int CountInversions(int[] A)
k=0 // #inversions
T = new augmented R-S-Tree // initially empty
for i=n downto 1 do
    T.Insert(A[i])
    k=k+ T.CountSmaller(A[i])   // T contains A[i..n],
                                // thus we count #j
                                // with i<j<=n and
                                // A[j]<A[i].
return k
```

[1 P]: (gute) Idee
[+0,5 P]: Korrektheit
[+0,5 P]: (gute) Kommentare
[+2 P]: Laufzeit in nlog n