Informatik B – Sommersemester 2007

Klausur 20.07.2007

Name: .....

Matrikelnummer: .....

Aufgabe	1	2	3	4	Gesamt
Punkte					

Tutor: ☐ David Karcher ☐ Bogumil Mikolajczyk ☐ Max Neumann ☐ alte Zulassung

Ich bin eiverstanden, dass mein Klausurergebnis mit Matrikelnummer auf einer FU-internen Webseite einzusehen ist:

JA ○ NEIN ○

Wichtige Hinweise:

1) Wenn die Lösung einer Aufgabe nicht auf den entsprechenden Zettel bzw. auf seine Rückseite passt, bitte einen Hinweis auf das Zusatzblatt geben.

2) Bitte nur mit Kugelschreiber oder Tinte schreiben und keine rote Farbe verwenden.

3) Alle Lösungen sind kurz (stichpunktartig), aber inhaltlich ausreichend zu kommentieren!

4) Einziges erlaubtes Hilfsmittel ist ein **einseitig, handschriftlich** gefülltes A4-Blatt mit Fakten und Formeln eigener Wahl.

Aufgabe 1: Graphen und Graphalgorithmen $2+3+2+(3)$ **Punkte**

Für eine beliebige positive, ganze Zahl n definieren wir einen Graphen $G_n = (V_n, E_n)$ auf der Knotenmenge $V_n = \{1, 2, \ldots, n\}$, wobei zwei Knoten genau dann durch eine Kante verbunden sind, wenn der Betrag ihrer Differenz beim Teilen durch 3 den Rest 2 lässt:

$$E_n = \{\, \{i, j\} \mid |i - j| \bmod 3 = 2\}$$

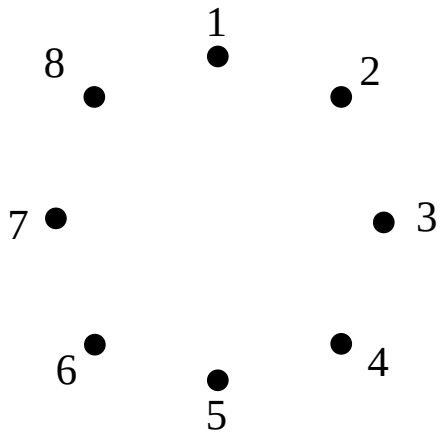
a) Ergänzen Sie das linke Bild zu einer vollständigen Zeichnung des Graphen G_8 und ordnen Sie den Kanten die folgenden Gewichte zu:

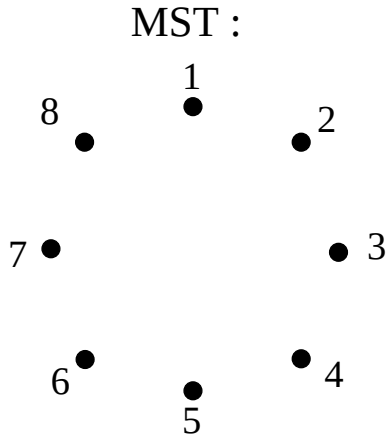
$$w(\{i, j\}) = |i - j| + \frac{\min(i, j)}{10}$$

b) Konstruieren Sie mit dem Algorithmus von **Prim** einen MST von G_8 mit der gegebenen Gewichtsfunktion und Startknoten 1. Zeichnen sie in das rechte Bild den MST ein und geben Sie die Reihenfolge an, in der die Knoten in die Startkomponente aufgenommen werden.

c) Welche der Graphen G_n sind zusammenhängend und welche nicht? Geben Sie dafür eine kurze Begründung.

d) Zusatzaufgabe: Zeigen Sie, dass $|E_n| = \Omega(n^2)$.





Reihenfolge der Knoten bei MST nach Prim:

Aufgabe 2: **Java** $4+3$ **Punkte**

a) Welche Bildschirmausgabe erhält man bei Auruf der folgenden Methode `test`?

```
public static void test(){
    int[] a = {4,3,2,1};
    int[] b = {0,3,6,9};
    int[] c = a;
    int[] d = (int[]) a.clone();
    int[][] A = {a,b,c,{1,2,3,4}};
    int[][] B = {a,b,c,d};

    A[2][1] *= 2;
    for(int i=0; i<4; i++) B[i][1] += 1;

    System.out.println( "A[1][1] = " + A[1][1]);
    System.out.println( "a[1] = " + a[1]);
    System.out.println( "(a == d) = " + (a==d));
    System.out.println( "a.equals(c) = " + (a.equals(c)));
    System.out.println( "a.equals(d) = " + (a.equals(d)));
    System.out.println( "A[2] == B[2] = " + (A[2]== B[2]));
}
```

b) Von einen `int`–Array `A` der Länge n sei bekannt, dass alle Einträge aufsteigend geordnet sind. Ausserdem setzen wir voraus, dass `A[0]` mit dem Wert 1 und `A[n-1]` mit dem Wert $n-1$ belegt sind. Definieren Sie eine Methode
`public int findDoubleOccur(int[] A)`,
die für ein Feld mit den beschriebenen Eigenschaften mit Laufzeit $O(\log n)$ eine Zahl findet, die in dem Feld (mindestens) doppelt auftritt. Um sich Hilfsfunktionen zu ersparen, können Sie auch die Parameterliste der Funktion erweitern.

Aufgabe 3: **Binäre Bäume** $4 + 4 + 2$ **Punkte**

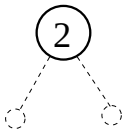
a) Bauen Sie einen AVL–Baum auf, in den nacheinander die Schlüssel $2, 4, 8, 5, 6, 1, 9$ eingefügt werden. Löschen Sie zum Schluss den Schlüssel 4. **Nutzen Sie dazu die Vorlage auf der nächsten Seite** und zeichnen Sie – wie für die ersten drei Operationen gezeigt – die Bäume nach den jeweiligen Suchbaum–Einfüge– bzw. Löschoperationen und gegebenfalls ein zweites Bild nach den notwendigen Rotationen. Zur Vereinfachung der Zeichnungen können Sie alle Blätter ignorieren.

b) Bauen Sie jeweils mit der Schlüsselfolge $2, 4, 8, 5, 6, 1, 9$ zwei Halden auf, die erste durch schrittweises Einfügen und die zweite mit der Bottom–Up–Konstruktion. In beiden Fällen reicht das Endergebnis ohne Zwischenschritte. Entfernen Sie danach in beiden Fällen das minimale Element.

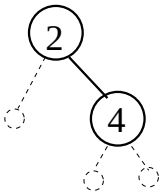
c) Angenommen, Sie haben einen AVL–Baum T_1 der Höhe h, der Schüssel aus dem Bereich $[105, 200]$ hält und einen AVL–Baum T_2 der Höhe $h + 1$, der Schüssel aus dem Bereich $[1, 95]$ hält (wieviele das sind spielt keine Rolle). Beschreiben Sie eine schnelle Methode zur Konstruktion eines AVL–Baums, dessen Schüsselmenge die Vereinigung der Schüsselmengen von T_1 und T_2 ist und analysieren Sie die Laufzeit.

Lösung für Aufgabe 3.a:

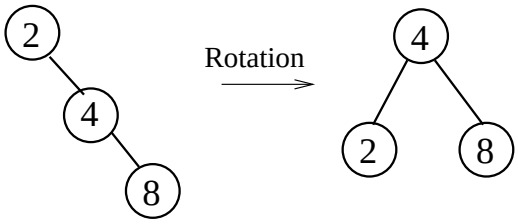
2 einfügen:



4 einfügen:



8 einfügen:



5 einfügen:

6 einfügen:

1 einfügen:

9 einfügen:

4 löschen:

Aufgabe 4: **Sortieren** $2 + 6$ **Punkte**

Sei K_m der vollständige Graph auf der Knotenmege $V = \{1, 2, \ldots, m\}$. Einer Kante $e = \{u, v\}$ wird der folgende Schlüssel $k(e)$ zugeordnet:

$$k(e) = (\max(u, v) - 1) \cdot m + \min(u, v)$$

a) Gegeben sei eine Folge von n Kanten des K_m, die bezüglich ihrer Schlüssel sortiert werden soll.
Skizzieren Sie, wie man die Verfahren Counting–Sort und Radix–Sort in diesem Fall anwenden kann. Es geht nicht darum, die Verfahren im Detail zu erklären, sondern zu erläutern, mit welchen Parametern (welches Feld bzw. wieviele Stufen und was passiert dort) man im konkreten Fall arbeiten sollte, um gute Laufzeiten zu erzielen.

b) Analysieren Sie die Laufzeit beider Verfahren **in Abhängigkeit von der Eingabegröße** n unter den folgenden Voraussetzungen:

$b1)$ $n = \Theta(\sqrt{m})$ $b2)$ $n = \Theta(m \cdot \sqrt{m})$ $b3)$ $n = \Theta(m^2)$

Vergleichen Sie die analysierten Laufzeiten in den drei Fällen mit der Laufzeit von Heap–Sort. Welches oder welche Verfahren sind in dem jeweiligen Fall am besten?