Informatik B – Sommersemester 2008

Musterlösung zur Klausur am 22.07.2008

Aufgabe 1: **Graphalgorithmen** $4+4$ **Punkte** $+2$ **Zusatzpunkte**

Berechnen Sie für den abgebildeten Graphen G den MST und verwenden Sie dazu

a) den Algorithmus von Prim mit dem Startknoten d: Geben Sie dazu tabellarisch die Reihenfolge an, in der die Knoten aus der Prioritäts–Warteschlange entfernt werden und jeweils den Zeiger auf den Vorgängerknoten.

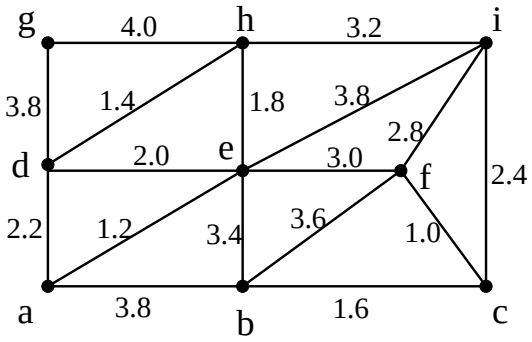
b) dem Algorithmus von Kruskal: Geben Sie dafür die Reihenfolge an, in der die Kanten in den MST aufgenommen werden und dazu jeweils die Größe der neuen Komponente, die durch die zugehörige `UNION`–Operation entsteht.

Hinweis: Die richtig ausgefüllten Tabellen werden in diesem Fall als ausreichende Begründung der Lösung anerkannt. Es ist Ihnen aber (insbesondere wenn Sie etwas unsicher sind) freigestellt, weitere Kommentare zu geben, die mit in die Bewertung einfließen können.

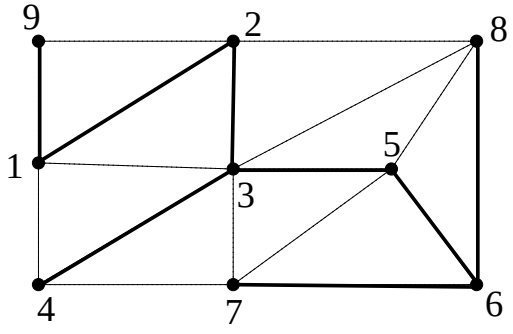
c) Zusatzfrage: Wir betrachten Graphen mit n Knoten, einer Kantengewichtsfunktion w und der zusätlichen Eigenschaft, dass jeder Knoten höchstens $\sqrt{n}$ Nachbarn hat. Welche Laufzeit in Abhängigkeit von n haben in diesem Fall die Algorithmen von Kruskal und von Prim? Geben Sie möglichst gute obere Schranken an.

Lösungen für Teil a und b:

Graph G:

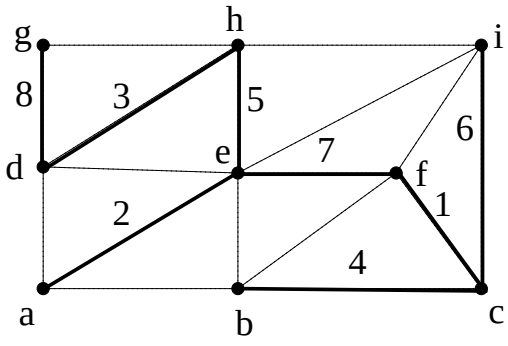


MST − Prim



Nummerierung <==> Reihenfolge

MST − Kruskal



Nummerierung <==> Reihenfolge

a) Prim:

Zeitpunkt	1	2	3	4	5	6	7	8	9
Knoten v	d	h	e	a	f	c	b	i	g
$\Pi(v)$	NIL	d	h	e	e	f	c	c	d

b) Kruskal:

Zeitpunkt	Kante	Komponentengröße nach UNION–Operation
1	$\{c, f\}$	2
2	$\{a, e\}$	2
3	$\{d, h\}$	2
4	$\{b, c\}$	3
5	$\{e, h\}$	4
6	$\{c, i\}$	4
1	$\{e, f\}$	8
2	$\{d, g\}$	9

Lösung Teil c: Nach dem Handschlaglemma ist $2E| = \sum_{v \in V} deg(v) \leq n \cdot \sqrt{n}$, d.h. $|E| \in O(n \cdot \sqrt{n})$. Da sich bei einer Halden–Implementierung der Prioritätswarteschlage alle n Minimum–Extraktionen und alle $|E|$ Schüssel–Updates jeweils in $\log n$ Zeit ausführen lassen, ergibt sich eine $O(n \cdot \sqrt{n} \cdot \log n) = O(n^{\frac{3}{2}} \log n)$ Laufzeit für den Algorithmus von Prim.

Die gleiche Laufzeit wird auch vom Algorithmus von Kruskal erreicht, wobei hier die dominierende Größe vom Sortieren der $|E|$ Kantengewichte kommt, die Kosten für die UNION– und FIND–Operationen lassen sich durch $O(n \log n)$ (es geht sogar noch besser) abschätzen.

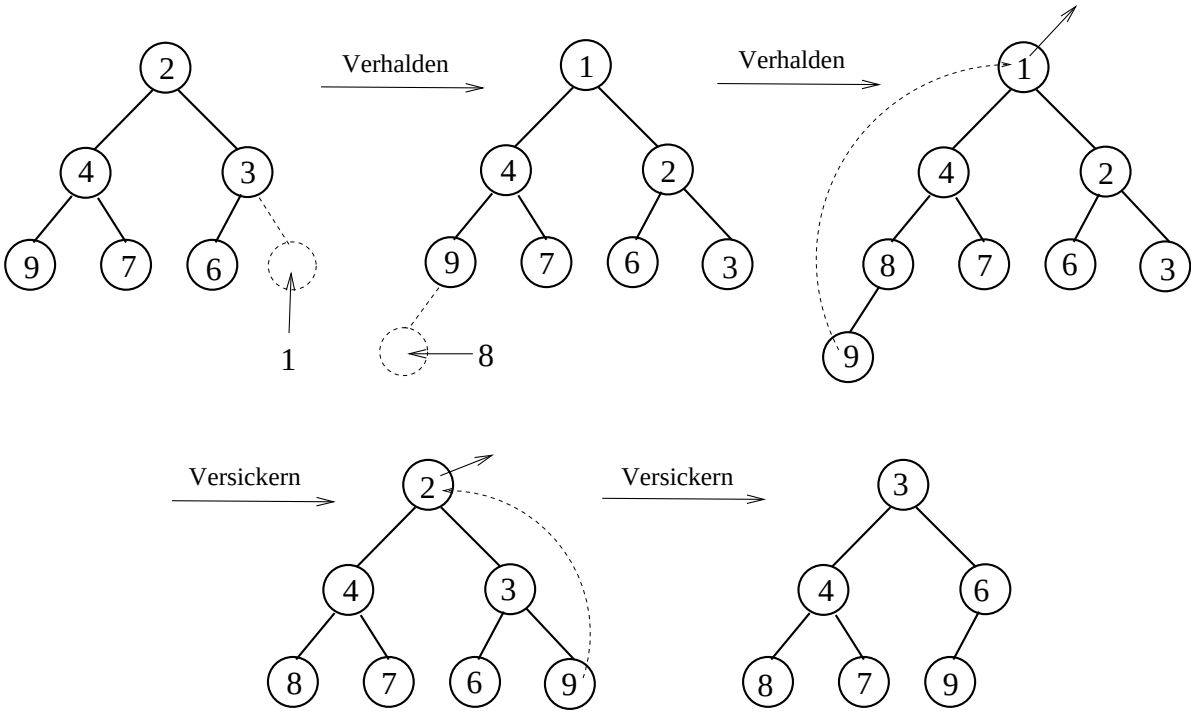
Aufgabe 2: **Halden** $5 + 2$ **Punkte**

a) Eine Halde ist durch die folgende Array–Darstellung gegeben:

2	4	3	9	7	6

Zeichen Sie die Halde als Binärbaum, fügen Sie dann die Elemente 1 und 8 nacheinander ein und führen Sie danach zwei Löschoperationen aus. Zeichnen Sie den Zustand der Halde nach jeder Einfüge– und Löschoperation.
Zur Vereinfachung der Zeichnungen können Sie alle Blätter ignorieren.

Lösung: Die gestrichelten Linien zeigen jeweils die Vorbereitung auf den folgenden Schritt.



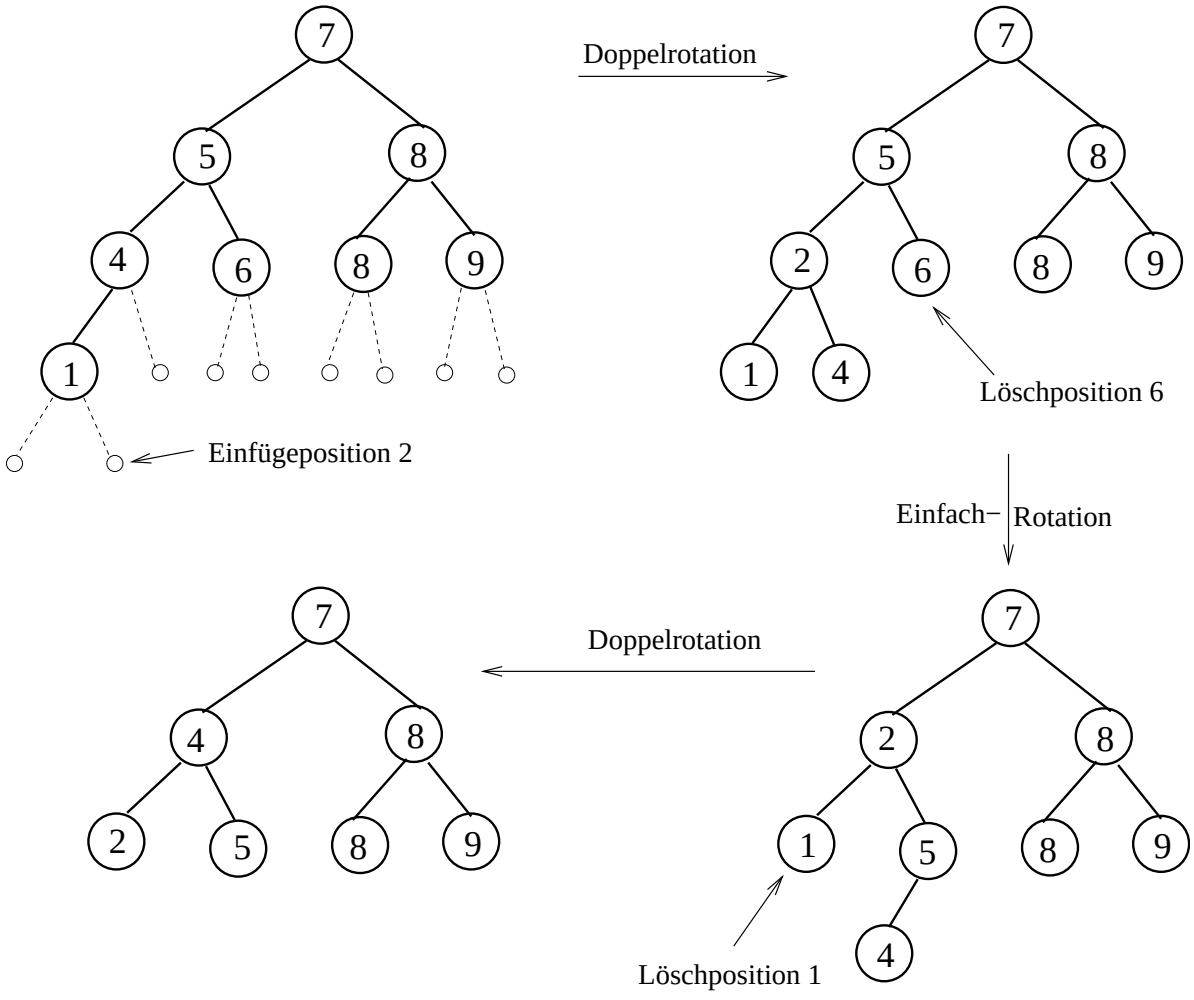
b) Gegeben seien zwei Halden H_1 und H_2 mit jeweils $m = 2^n$ Einträgen. Wie kann man in $O(\log m) = O(n)$ Zeit eine Halde H aufbauen, die alle 2^{n+1} Einträge aus den Halden H_1 und H_2 hält.

Lösung: Da beide Halden 2^n Einträge haben, gibt es jeweils einen Eintrag in der untersten Schicht. Wir löschen zuerst aus H_2 einen Eintrag, setzen dann das gelöschte Element als neue Wurzel über H_1 auf der linken und H_2 auf der rechten Seite und lassen es versickern. Der so entstandene Binärbaum hat die Vollständigkeitseigenschaft, denn durch das Löschen in H_2 gibt es im untersten Level nur ein Element (aus H_1) und das steht ganz links. Die Ordnungseigenschaft ist auch erfüllt und die Kosten für die Löschoperation und das Versickern waren jeweils $O(n)$.

Aufgabe 3: **AVL-Bäume** 6 **Punkte**

Zeichen Sie einen Binärbaum mit 8 inneren Knoten, der die Vollständigkeitseigenschaft hat, und tragen Sie die Schlüsselfolge $1, 4, 5, 6, 7, 8, 8, 9$ so in die inneren Knoten ein, dass ein AVL–Baum entsteht. Fügen Sie in diesen AVL–Baum den Schlüssel 2 ein und löschen Sie dann zuerst den Schlüssel 6 und danach den Schüssel 1. Führen Sie (falls notwendig) die entsprechenden Rotationen aus.
Zur Vereinfachung der Zeichnungen können Sie nach dem ersten Schritt wieder alle Blätter ignorieren.

Lösung: Einige haben den ersten Teil der Aufgabenstellung falsch verstanden und einen AVL–Baum durch schrittweises Einfügen der Schlüsselfolge erzeugt. Das erfordert unnötigen Aufwand und ergibt dazu eine falsche Anfangskonfiguration.
Dabei ist es viel einfacher – durch die Vollständigkeitseigenschaft ist die äußere Form (wie bei einer Halde) vorgegeben und man muss die Werte nur noch von links nach rechts eintragen:



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

Ein Polynom der Form $p(x) = \sum_{k=0}^{n} a_k x^k = a_0 + a_1 x + a_2 x^2 + a_n x^n$ kann durch das Array seiner Koeffizienten $[a_0, a_1, \ldots, a_n]$ repräsentiert werden. Der höchste Index i von einem Nicht–Null–Koeffizienten wird der Grad des Polynoms gennant.
Beispiel: Die Arrays $[0,3,0,2,0,0]$ und $[0,3,0,2]$ repräsentieren beide das Polynom $p(x) = 3x + 2x^3$ vom Grad 3. Die folgende unvollständige Klassendefinition dient zur Beschreibung von Polynomen mit ganzzahligen Koeffizienten:

```
public class Polynom{
int[] A; // Koeffizienten
public Polynom(int[] koeff) {A=koeff;} // Konstruktor
int degree(){.. }              // Grad des Polynoms bestimmen
boolean equals(Polynom p){.. } // Test auf Gleichheit
Polynom sum(Polynom p){.. }    // Summe aus Polynom this und Polynom p
int evaluateAt(int j){.. }     // Wert des Polynoms an der Stelle x=j
}
```

Implementieren Sie die vier Methoden aus der Klasse Polynom, wobei der Gleichheitstest nicht die Arrays sondern die repräsentierten Polynome auf Gleichheit testen soll.

```
int degree(){
     int deg = 0;
     int  i = 0;
     while(i < A.length){
          if(A[i] != 0) deg = i;
          i++;
    }
    return deg;
}

int a(int i){ // Hilfsfunktion fuer i-ten Koeffizienten
    if(i < A.length) return A[i]; else return 0;
}

boolean equals(Polynom p){
    boolean b = true;
    for(int i=0; i<A.length || i<(p.A).length  ; i++){
        if(a(i) != p.a(i)) b = false;
    }
    return b;
}
```

```
Polynom sum(Polynom p){
    int deg  = degree();  // Grad des Summenpolynoms
    int degp = p.degree();
    if(degp > deg) deg = degp;
    int[ ] Asum = new int[deg+1];
    for(int i=0; i <= deg; i++){
           Asum[i] = a(i) + p.a(i);
    }
    Polynom s = new Polynom(Asum);
    return s;
}

int evaluateAt(int j){ // Horner Schema
    int val = 0;
    for(int i =  degree(); i >=0; i--){
       val = A[i] + val * j;
    }
    return val;
}
```

Aufgabe 5: **Vererbung** 4 **Punkte**

a) Betrachten Sie die folgenden Klassendefinitionen. Welche Bildschirmausgabe erhält man bei Auruf der `main`–Methode der Klasse `Tiger`?

```
public class Katze{
    public String ruf;
    public int anzahl;
    public Katze() {ruf = "Miau";  anzahl = 1;} // Konstruktor
    public void rufen() {
          for(int i=0; i<anzahl; i++)
                {System.out.println(ruf);}
    }
    public int f() {return anzahl;}
}

public class Tiger extends Katze{
    public String ruf;
    public int anzahl;
    public Tiger() {          // Konstruktor
         super();
         ruf = "Uaah";
         anzahl = 2;
    }
    public void rufen() {
         for(int i=0; i<anzahl; i++)
                {System.out.println(ruf);}
    }
    public int f() {return anzahl;}
    public void g() {super.rufen();}

    public static void main(String[] args){
         Tiger Tina = new Tiger();
         Katze Kati = Tina;
         System.out.println("Kati.anzahl = " +  Kati.anzahl);
         System.out.println("Kati.f() = " +  Kati.f());
         Kati.rufen();
         Tina.g();
    }
}
```

Lösung: Man muss nur wissen, dass beim Verdecken des Attributs `anzahl` der deklarierte Typ der Referenz `Kati`, also `Katze`, zählt, wogegen bei Überschreiben der Methoden `f()` und `rufen()` der tatsächliche Typ der Instanz, also `Tiger` entscheidet. Deshalb

greifen diese Methoden auch auf die Attribute `anzahl` und `ruf` von `Tiger` zu. Dagegen wird beim Aufruf von `g()` die Methode `rufen()` der Superklasse `Katze` mit den entsprechenden Attributen verwendet. Damit ergibt sich die folgende Ausgabe:

```
Kati.anzahl = 1
Kati.f() =  2
Uaah
Uaah
Miau
```