Universität Karlsruhe (TH)
Fakultät für Informatik
Lehrveranstaltung Informatik II

Sommersemester 2003

Prof. Dr. Peter C. Lockemann
(lockemann@ira.uka.de)
Gábor M. Surányi (suranyi@ira.uka.de)
Daniel Pfeifer (pfeifer@ira.uka.de)
Philipp Bender (bender@ira.uka.de)

Übungsblatt 10

Sortieren, Bäume, Graphen und Suchen

Abgabe: bis 09.07.2003, 13:30 Uhr in den Einwurfkästen im Untergeschoß des neuen Infobaus

Erreichbare Punkte: 33 Theoriepunkte (33 T), 20 Praxispunkte (20 P)

Aufgabe 1: *Sortieren (6 T)*

Zeigen Sie, daß der maximale Aufwand von Sortieren durch Einfügen (Insert-Sort) linear ist, d.h. O(n), falls die gegebene Reihung $A[0{:}n\text{-}1]$ derart in Teilreihungen $A[j_0{:}j_1\text{-}1]$, $A[j_1{:}j_2\text{-}1]$,..., $A[j_{p-1}{:}j_p]$ (mit j_0=0, j_p=n-1) zerlegt werden kann (mit $k=\max\{(j_{i+1}-j_i)|i\in\{0,...,p-1\}\}$, und $k \ll n$, k fest), so daß stets gilt: $\forall i\in\{0,\ldots p-2\}\forall l\in\{j_i,\ldots,j_{i+1}-1\}\forall m\in\{j_{i+1},\ldots,j_{i+2}-1\}: A[l]\le A[m]$.

Aufgabe 2: *B-Bäume (6 T)*

Zeichnen Sie die B-Bäume des Typs τ(1, h), die durch die nachfolgende Sequenz von Einfüge- und Löschoperationen in einer entsprechenden B-Baum-Datenstruktur entstehen. Dabei sollen Zahlen als Schlüssel in einen aufsteigend sortierten Baum eingefügt bzw. aus ihm gelöscht werden. Die Nutzdaten werden hier der Einfachheit halber vernachlässigt. E(a) bezeichne das Einfügen einer Zahl a, L(b) das Löschen der entsprechenden Zahl b:
E(10), E(20), E(30), E(40), E(50), E(60), E(70), E(25), L(10), L(30), L(60)
Falls es bei den Operationen zum Überlauf, Unterlauf, Splitten oder Verschmelzen von Knoten kommt, dann kennzeichnen Sie dies bitte an den entsprechenden Knoten jeweils durch die Buchstaben Ü, U, S bzw. V. Zeichnen Sie in derartigen Fällen auch alle Zwischenschritte für die Transformation des B-Baums bis dieser wieder einen konsistenten Zustand erreicht hat.

Aufgabe 3: *Implementierung eines Binärbaums (10 P)*

In der Vorlesung wurde in Kapitel 11 die Implementierung einer Menge als Binärbaum mittels der Klasse **`TreeSet`** vorgestellt. Der Java-Code für **`TreeSet`** ist dem Übungsblatt beigefügt.

a) Erweitern Sie die Klasse **`TreeSet`** und implementieren Sie die Methode **`public int getHeight()`**! **`getHeight()`** soll die Höhe des entsprechenden Baums bestimmen. Testen Sie die Methode durch den Aufbau (mindestens 2) verschiedener Binärbäume (mit mindestens 5 Elementen). (5 P)
b) Implementieren Sie in der Klasse **`TreeSet`** die Methode **`public java.util.List getPostOrderList()`**, die die Elemente des durch ein **`TreeSet`**-Objekt repräsentierten Binärbaums in **umgekehrter Sortierreihenfolge** als Liste (des Typs **`java.util.List`**) zurückliefert. Der (Zeit-) Aufwand des entspr. Algorithmus soll O(n) sein. Testen Sie ihre Methode für die Binärbäume aus Teilaufgabe a). (5 P)

Hinweis: Die Klasse **`TreeSetIterator`** ist dem Quellcode nur der Vollständigkeit halber beigefügt und braucht nicht weiter beachtet zu werden.

Aufgabe 4: *Graphen (8 T)*

a) Geben Sie für den Graph G = ({1, 2, 3, 4, 5, 6}, {(1, 2), (1, 6), (2, 5), (3, 1), (3, 5), (3, 6), (4, 1), (5, 1), (5, 6), (6, 4) }) eine Darstellung in Diagramm, Listen- und Matrixform an! (3 T)

b) Sei G ein ungerichteter Graph, bei dem jeder Knoten mindestens 2 Kanten besitzt. Beweisen oder widerlegen Sie die Aussage: G hat einen Zyklus. (3 T)

c) Sei G wie in Teilaufgabe b). Beweisen oder widerlegen Sie die Aussage: Jeder Knoten von G liegt auf einem Zyklus. (2 T)

Aufgabe 5: *Tiefensuchbäume (3 T)*

Entwickeln Sie einen Tiefensuchbaum für den Graph in Abbildung 2, und legen Sie dabei die Besuchsreihenfolge der Knoten durch Numerierung fest. Die Tiefensuche soll in Knoten 1 starten und jeweils die Kanten in der Reihenfolge ihrer angegebenen Numerierung behandeln. Geben Sie auch die bei der Suche auftretenden Baum-, Vorwärts-, Rückwärts- und Querkanten an und markieren Sie sie entsprechend mit B, V, R bzw. Q.

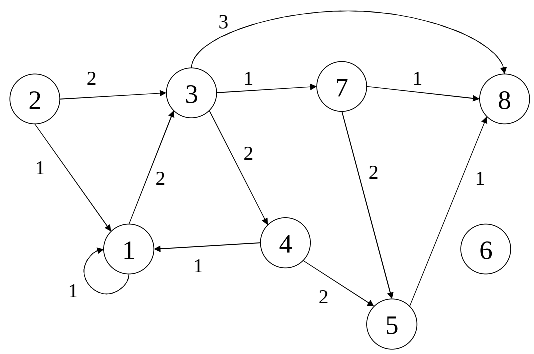

Abbildung 2: Gerichteter Graph

Aufgabe 6: *Labyrinthe und Graphen (10 T)*

a) Wie können Sie ein Labyrinth wie in Aufgabe 1 Übungsblatt 7 als Graph interpretieren? Um was für eine Art von Graph handelt es sich dabei? (2,5 T)

b) Was schließen Sie daraus für die Anzahl der Innenwände eines Labyrinths der Größe n*m (also ohne die Randbegrenzung)? Begründen Sie Ihre Antwort. (2 T)

c) Entwerfen Sie einen Algorithmus zur Labyrinthgenerierung mit Aufwand O(m*n) der auf der Tiefensuche basiert! Formulieren Sie den Algorithmus mit Hilfe von Java-Pseudocode. (5,5 T)

Aufgabe 7: *Breitensuche für Labyrinthlösungspfad (10 P)*
Implementieren Sie einen Algorithmus zur Suche eine Lösungspfads für ein Labyrinth mittels Breitensuche. Verwenden Sie dabei wieder das Labyrinth-Framework, das Sie schon aus Übungsblatt 7 kennen, und vervollständigen Sie in der Klasse `StudentMazeSolver` die Methode `private MazeSolution doBreadthSearch()`. Eine entsprechende Datei für `StudentMazeSolver` ist dem Übungsblatt beigefügt. Zur Vereinfachung der Aufgabe sind ist bereits folgende Klassen bzw. Methoden für Sie darin implementiert:

- Die Klasse `Node` dient zur Speicherung von Labyrinthräumen (mit x,y-Koordinaten), die während der Breitensuche betrachtet werden sollen. Knoten können (über ein entsprechendes Argument im Konstruktor von `Node`) verkettet werden und auf diese Weise Pfade im Labyrinth repräsentieren.
- Eine Klasse `NodeQueue` ermöglicht die Verwaltung von Schlangen für Knoten (vom Typ `Node`) .
- Die Methode `private MazeSolution getCurrentSolution(Node node)` erlaubt es Ihnen, einen Labyrinthpfad (der durch eine verkettete Knotenliste repräsentiert ist) in ein Objekt der Klasse `MazeSolution` zu transformieren, wie es vom Labyrinth-Framework als Lösung erwartet wird.

Als weitere Hilfestellung ist nachfolgend noch der Algorithmus der *allgemeinen* Breitensuche mit Java-Pseudocode dargestellt.
Hinweis: Die oben beschriebenen Klassen sollen zur Vereinfachung der Aufgabe dienen – Sie dürfen aber die Klasse `StudentMazeSolver` auch komplett selbst implementieren, sofern Ihre Lösung die Breitensuche anwendet.

```
public void breitenSuche(Knoten s)
// s ist der Startknoten für die Suche.
{
  Menge b = leere Menge; // Menge besuchter Knoten.
  Schlange q = leere Schlange;
  Füge s in q ein (enqueue).

  while (q nicht leer) {
    Knoten n =
      gib und entferne erstes Element aus q (front, dequeue);
    Füge n in b ein.
    Menge anf = alle Nachfolgenoten von n;
    for (alle nf in anf) {
      if (nf nicht in b) {
        Füge nf in q ein (enqueue).
      }
    }
  }
}
```