TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

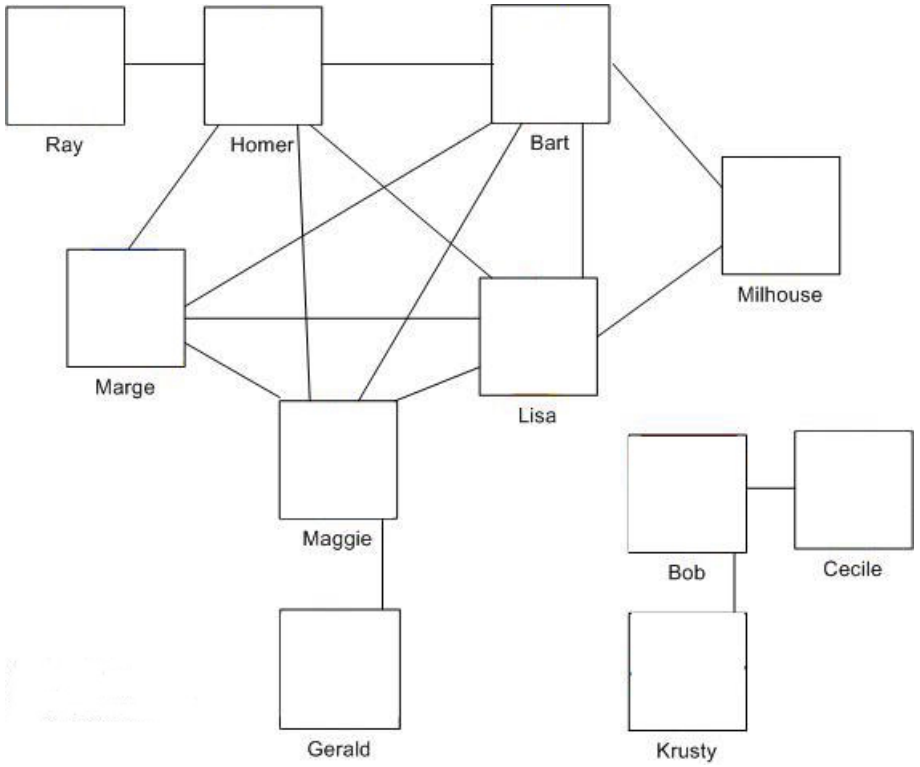
Lehrstuhl für Sprachen und Beschreibungsstrukturen
Grundlagen: Algorithmen und Datenstrukturen
Prof. Dr. Helmut Seidl, J. Kranz, A. Reuss, R. Vogler

SS 2016
Übungsblatt 10
18.06.2016

Abgabe: **26.06.2016** (bis 23:59 Uhr)

Aufgabe 10.1 (P) Netzwerk

Wir stellen uns ein soziales Netzwerk als Graph vor. In diesem Graph bilden die Personen des Netzwerks die Knoten. Sind zwei Personen befreundet, gibt es im Graphen eine ungerichtete Kante zwischen den entsprechenden Knoten. Das folgende Bild zeigt einen Ausschnitt aus einem solchen Graphen:



a) Wie kann man mit Hilfe der Tiefensuche feststellen, ob es eine Verbindung zwischen zwei Personen gibt? In diesem Fall sagen wir, dass sich diese beiden Personen kennen.

b) Überlegen Sie sich einen Algorithmus (basierend auf der Breitensuche), um festzustellen, über wieviele Personen sich zwei Personen minimal kennen. Z.B. kennt Milhouse Gerald minimal über zwei weitere Person (Lisa oder Bart und Maggie).

c) Ergänzen Sie Ihren Algorithmus, so dass er eine kürzeste Verbindung ausgibt. Zum Beispiel für die Verbindung Milhouse - Gerald wird „Milhouse - Bart - Maggie - Gerald“ ausgegeben.

Aufgabe 10.2 (P) Baumtraversierung

Ein nichtleerer Binärbaum kann (unter anderem) in PreOrder, InOrder und PostOrder traversiert werden. Diese Traversierungen sind wie folgt rekursiv definiert:

- PreOrder:
  a) Besuche die Wurzel.
  b) Traversiere den linken Teilbaum in PreOrder, falls dieser nichtleer ist.
  c) Traversiere rechten Teilbaum in PreOrder, falls dieser nichtleer ist.
- InOrder:
  a) Traversiere den linken Teilbaum in InOrder, falls dieser nichtleer ist.
  b) Besuche die Wurzel.
  c) Traversiere den rechten Teilbaum in InOrder, falls dieser nichtleer ist.
- PostOrder:
  a) Traversiere den linken Teilbaum in PostOrder, falls dieser nichtleer ist.
  b) Traversiere den rechten Teilbaum in PostOrder, falls dieser nichtleer ist.
  c) Besuche die Wurzel.

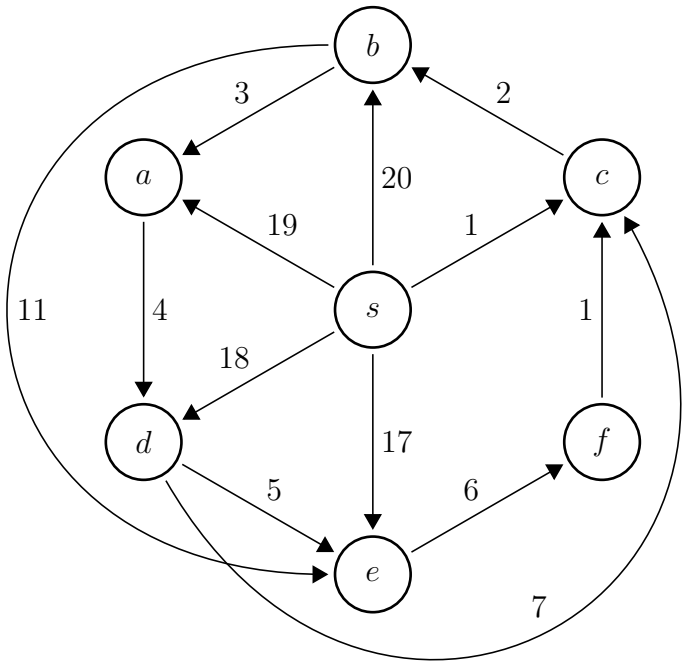
Wir bemerken, dass die Anordnung der Knoten nach PreOrder genau der Anordnung nach der dfs-Nummer entspricht, wenn die Tiefensuche einen linken Teilbaum stets vor dem korrespondierenden rechten Teilbaum besucht. Analog dazu entspricht die Anordnung nach PostOrder genau der Anordnung nach der (dfs-)finish-Nummer.

Geben Sie Algorithmen `preNext(v)` und `postNext(v)` an, die zu einem Knoten v in einem Binärbaum den in der PreOrder bzw. PostOrder folgenden Knoten w berechnet. Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Pseudocodes.

Berechnen Sie außerdem die asymptotische Laufzeit, wenn mittels der Operationen `preNext(v)` und `postNext(v)` die vollständige PreOrder bzw. PostOrder berechnet wird (also n-maliges Anwenden der Funktion).

Aufgabe 10.3 (P) Dijkstra

Führen Sie den Algorithmus von Dijkstra auf dem folgenden Graphen durch, um jeweils einen kürzesten Weg von s zu jedem anderen Knoten zu finden. Protokollieren Sie nachvollziehbar Ihre Vorgehensweise, und markieren Sie zum Schluss alle Kanten, die zum gefundenen Kürzeste-Wege-Baum gehören.



Aufgabe 10.4 [6 Punkte] **Suchen in Bäumen (alte Klausuraufgabe)**

a) [1] Beweisen Sie per Induktion, dass es in einem Baum mit $n \geq 1$ Knoten genau $n - 1$ Kanten gibt:

b) [0.5] Die Laufzeit einer Tiefensuche auf einem Graphen mit n Knoten und m Kanten ist $\Theta(n + m)$. Für die Laufzeit in Abhängigkeit von n ergibt sich $\mathcal{O}(n^2)$, da es bis zu $n^2 - n$ Kanten geben kann. Geben Sie eine möglichst kleine Abschätzung der Laufzeit einer Tiefensuche **auf einem Baum** in Abhängigkeit nur von der Zahl n der Knoten an, wenn die Suche bei der Wurzel startet:

$\mathcal{O}($ $)$

c) [1] Gegeben seien nun zwei Zahlen a und b mit $a < b$ und ein binärer Suchbaum B mit **n** Knoten als Suchstruktur über einer Liste von paarweise verschiedenen, aufsteigend

sortierten Zahlen. In der Liste gebe es $\mathbf{m} > 1$ Zahlen z aus dem Bereich zwischen a und b (d.h. $a \leq z \leq b$).

Betrachten Sie das folgende Verfahren, um alle Elemente zwischen a und b auszugeben:

> Lokalisiere (mittels Suche in B) das Listenelement, dessen Eintrag z die kleinste Zahl ist, für die $z \geq a$ gilt. Gehe die Liste durch bis zum ersten Auftreten eines Eintrags größer als b (oder bis zum Ende der Liste).

Geben Sie für dieses Verfahren den asymptotischen Worst-Case-Aufwand in $\mathcal{O}$-Notation in Abhängigkeit von m und n an und begründen Sie Ihre Antwort kurz.

Aufwand:

Begründung:

d) [3.5] Fritzchen, ein unerfahrener Programmierer, hat vergessen, die Liste zu verketten. Sie sollen ihm nun helfen, trotzdem alle Elemente zwischen a und b zu finden.

Implementieren Sie eine entsprechende Suche auf B nach den entsprechenden zum Bereich zwischen a und b gehörenden Blättern, wobei nicht in Unterbäumen gesucht werden soll, deren Einträge nicht im Bereich zwischen a und b liegen können.

Benutzen Sie bei Ihrer Implementierung die folgenden Methoden und Attribute:

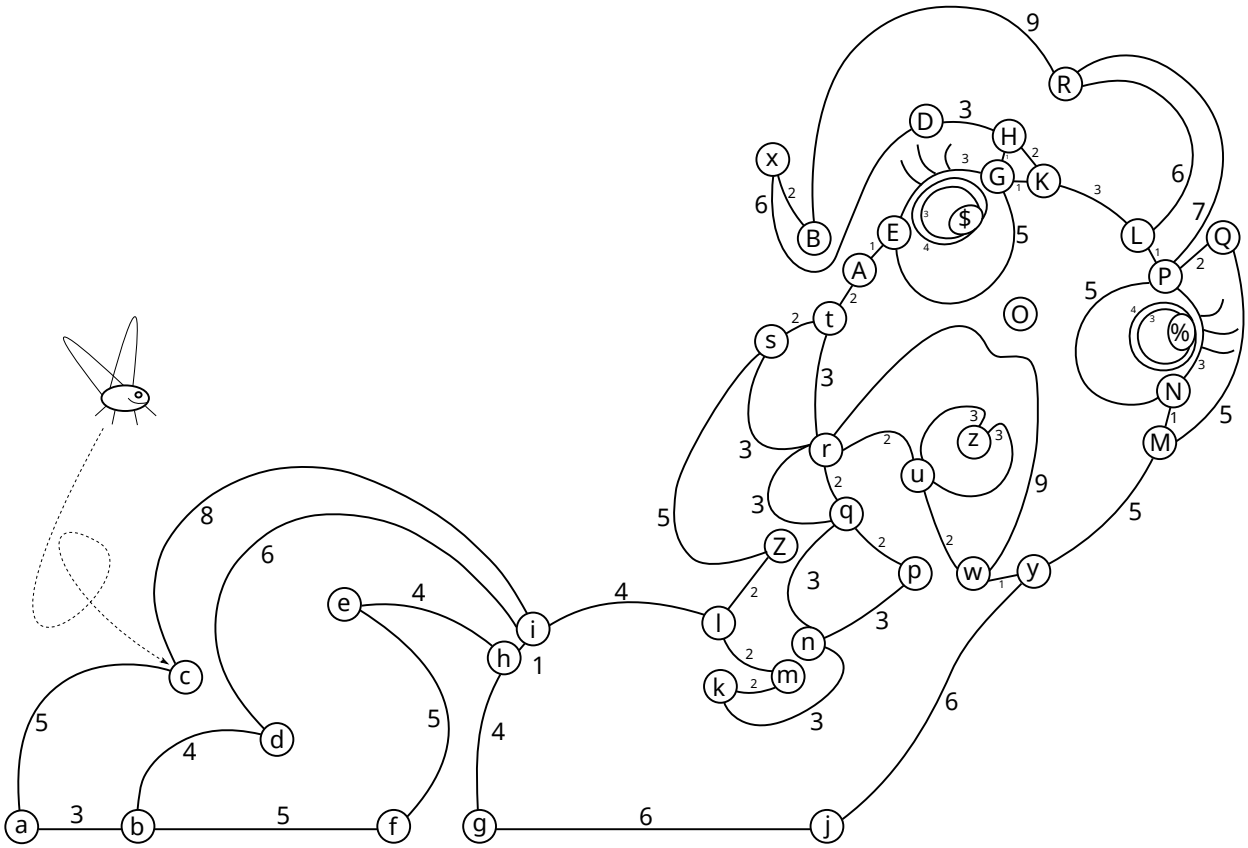
- `isLeaf(t)` – liefert true genau dann, wenn `t` ein Blatt ist
- `t.number` – in `t` gespeicherter Schlüssel; wenn `t` ein Blatt ist, entspricht dieser einem Eintrag in der (gedachten, nicht verketteten) Liste
- `t.leftChild` – gibt das linke Kind von `t` zurück
- `t.rightChild` – gibt das rechte Kind von `t` zurück

Die gefundenen Zahlen z aus dem gesuchten Bereich sollen aufsteigend durch den Aufruf `out(z);` ausgegeben werden.

```
public static void find(Tree t, int a, int b) {









}
```

Aufgabe 10.5 [4 Punkte] **(H) Dijana und Stratos**

Machen Sie sich mit der folgend dargestellen Sachlage vertraut:



Die männliche Fliege Stratos landet auf Knoten 'c' des buschigen Schwanzes von Pony Dijana. Vom Fliegen müde, möchte Stratos auf dem kürzesten Weg zu Knoten 'Q' des rechten Ohres von Dijana krabbeln. Stratos entscheidet sich, zunächst zu rasten und mithilfe des Algorithmus von Dijkstra die kürzesten Wege zu allen Knoten von Dijana zu berechnen, bevor er seine Reise beginnt. Helfen Sie Stratos, indem Sie den Inhalt der Prioritätswarteschlange nach jedem Schritt des Algorithmus notieren. Geben Sie außerdem jeweils an, ob der Knoten einen neuen Vaterknoten im Baum der kürzesten Wege ab Knoten 'c' zugewiesen bekommt und – falls ja – welchen. Markieren Sie schließlich den kürzesten Weg im Graphen und geben Sie dessen Länge an.

Aufgabe 10.6 [5 Punkte] **(H) Programmieraufgabe: Dijomilia**

In dieser Aufgabe werden Sie die Implementierung des binomialen Haufens der Aufgabe „Binomilia“ von Blatt 7 derart erweitern, dass sie sich für eine Implementierung des Algorithmus von Dijkstra eignet. Sie können für diese Aufgabe entweder Ihre eigene Lösung als Grundlage verwenden oder Sie auf der Musterlösung der von Binomilia[1] aufbauen. Benutzen Sie außerdem die Vorgaben aus dem Ordner `bijomilia-angabe/`. Gehen Sie vor wie folgt:

a) Ändern Sie den Typen der Schlüssel im Baum von `int` auf einen generischen Typen `T`, der das Interface `Comparable<T>` implementiert. Passen Sie die Implementierung entsprechend an.

[1] `https://www.moodle.tum.de/mod/resource/view.php?id=445020`

b) Wir möchten Referenzen auf ein Objekt vom Typ `BinomialHeapHandle` als *Handle* für die `decreaseKey(...)`-Operation verwenden. Passen Sie die Methode `void insert(...)` entsprechend an. **Hinweis**: Sie dürfen die Signatur und den Rückgabewert der Methode ändern.

c) Implementieren Sie die Methode `void siftUp()` der Klasse `BinomialTreeNode`, die die Heap-Eigenschaft nach einer Verkleinerung eines Schlüssels repariert. **Hinweis:** Sie müssen die Klasse `BinomialTreeNode` noch an anderen Stellen ändern.

d) Implementieren Sie schließlich die Methode `void decreaseKey(T key)` der Java-Klasse `BinomialTreeNode`, sowie außerdem die Methode `void decreaseKey(Object handle, T eNew )` der Klasse `BinomialTree`. **Hinweis:** Es muss nicht darauf geachtet werden, dass die Methode `void decreaseKey(...)` in `BinomialTree` ein definiertes Verhalten aufweist, wenn sie ein ungültiges Handle erhält (z.B. bezüglich eines Knotens, der sich nicht im Haufen befindet).

Aufgabe 10.7 [5 Punkte] (H) Programmieraufgabe: Triangulina

Betrachten Sie folgendes Zahlendreieck:

3
7 4
2 **4** 6
8 5 **9** 3

Wir suchen in einem solchen Dreieck die maximale Summe entlang eines Pfades, der folgenden Regeln folgt:

- Der Pfad beginnt an der Spitze des Dreiecks.
- Der Pfad endet in der Grundlinie des Dreiecks.
- In jedem Schritt bewegen wir uns ausgehend von der aktuellen Position eine Zeile nach unten und links bzw. rechts. In obigem Beispiel können wir von der 7 aus also entweder zur 2 oder zur 4 gehen.

Im Beispieldreieck ist der gesuchte Pfad rot markiert. Er hat die Länge $3 + 7 + 4 + 9 = 23$.

Implementieren Sie die Methode `TriangulinaResult findMaxSumPath(int[] triangle)`, die zu einem gegebenen Dreieck bestehend aus positiven Zahlen den maximalen Pfad und dessen Länge ermittelt. Das Dreieck wird dabei als Feld dargestellt, indem die Zeilen aneinander gehängt werden. Die Klasse `TriangulinaResult` enthält die Länge des Pfades und ein Feld von Indices, die den Pfad repräsentieren. Im gegebenen Beispiel würde der Pfad durch das Feld $[0, 1, 4, 8]$ repräsentiert werden (Index 0 ist hier links).

Hinweis: Alle mitgelieferten Tests müssen in annehmbarer Zeit[2] auf einem aktuellen Rechner ausgeführt werden können.

[2] 100 Millionen Jahre sind z.B. nicht annehmbar.