Praktikum 3 **Algorithmik SS 2008** **Abgabe: Vierter Praktikumstermin**

Aufgabe 9: - ADT TREE
- TREE-Collection

Aufgabe 10: ADT UNION FIND

Aufgabe 11: Experimentelle Laufzeitanalyse

Name: ..... Matr-Nr: .....

Datum: …… Unterschrift des Dozenten (wenn bestanden): ……..……

Für alle Aufgaben ist eine ausführliche Dokumentation zu erstellen!

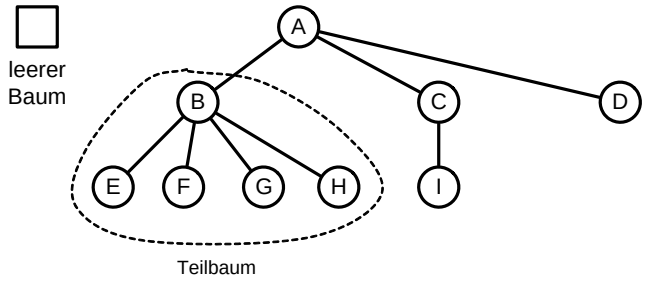
Literatur: Kapitel 3 und 4 der Vorlesung, insbesondere das Kapitel über die Union-Find-Strukturen.

Aufgabe 9 (TREE)

Ein Baum (tree) ist eine endliche Menge von Elementen oder Knoten. Ist die Menge nicht leer, so ist einer der Knoten die Wurzel, während die restliche Knotenmenge in Untermengen gruppiert werden, die jeweils selbst wieder Bäume sind. Zwischen den Knoten ist eine hierarchische Beziehung definiert:

- Jeder Teilbaum ist Kind einer Wurzel
- Jede Wurzel ist Vater eines Teilbaumes

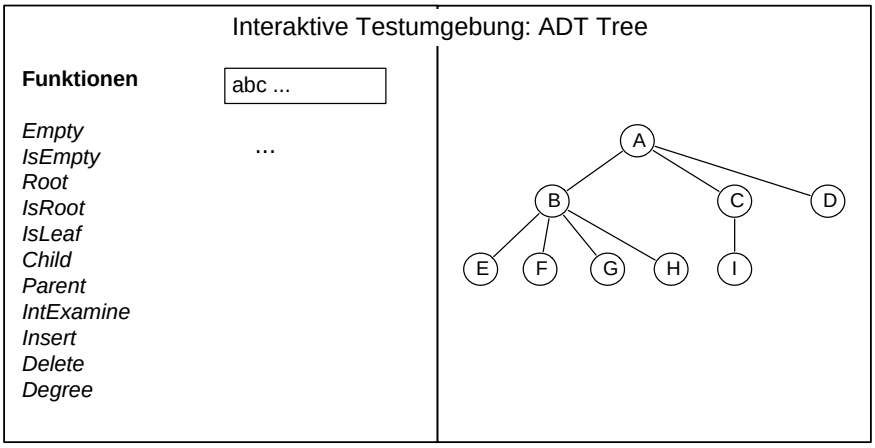
Ein Baum mit nur einem einzigen Knoten heißt Blatt und wird auch als leerer Baum bezeichnet.



a) Definieren Sie den ADT TREE analog zu dem ADT BINTREE aus Kapitel 3. Ein Baum T vom Typ TREE ist im Gegensatz zu einem Baum BT vom TypBINTREE ein Mehrwegebaum. Definieren Sie für TREE eine zusätzliche Funktion *Degree*(T). Dokumentation!

b) Implementieren Sie den ADT TREE mit Sentinel-Technik als Java-Klassen. Schreiben Sie eine ausführliche Java-Doc.

c) Entwickeln Sie eine interaktive Testumgebung, mit der Sie die ADT-TREE-Funktionen manuell testen können. Geben Sie das Ergebnis einer ADT-Operation aus. Falls der Baum T verändert wird, stellen Sie den aktuellen Baum in einem eigenen Fenster grafisch dar.

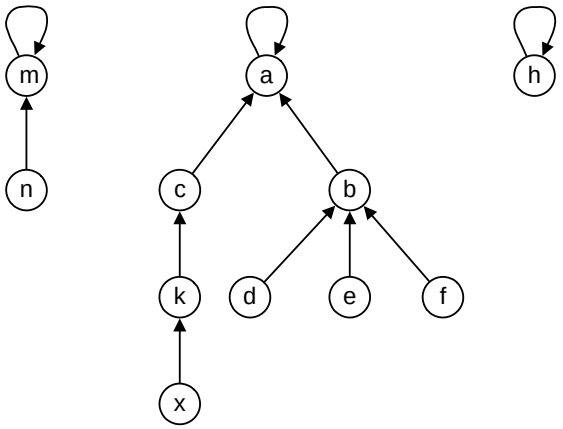


Dies ist nur ein Schema-Beispiel für eine Testumgebung. In Ihrer endgültigen Version wird es weitere interaktive Objekte geben.

Aufgabe 10 (ADT UNIONFIND)

Entwickeln Sie eine Java-Klassenbibliothek für den ADT UNIONFIND aus Kapitel 4 der Vorlesung. Verwenden Sie dazu den ADT TREE aus Aufgabe 9. Das Bild zeigt die Kollektion K der drei Mengen m, a und h:

K = {{m, n}, {a, c, b, k, d, e, f, x}, {h}}



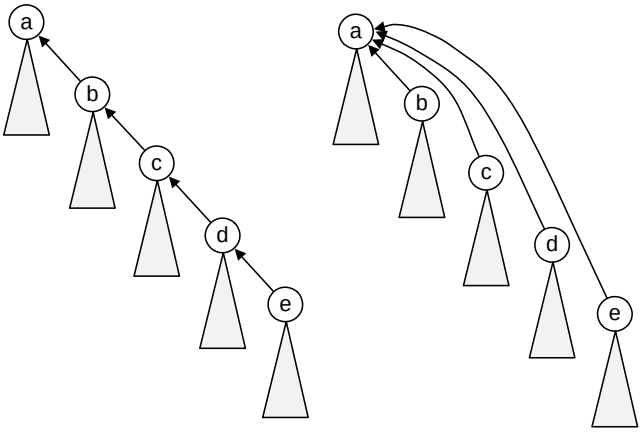
a) Definieren Sie den ADT UNIONFIND und dokumentieren Sie die Strukturen.

b) Implementieren Sie den ADT UNIONFIND in Java und schreiben Sie eine ausführliche Java-Doc. Entwickeln Sie außerdem eine Klasse, mit der sich **Kollektionen** beliebiger Mengen bilden lassen, wie in dem obigen Beispiel dargestellt.

c) Entwickeln Sie eine interaktive Testumgebung, mit der Sie die ADT-UNIONFIND-Funktionen manuell testen können. Geben Sie jeweils das Ergebnis einer ADT-Operation aus. Stellen Sie die Kollektion K in der Mengenschreibweise dar.

d) Implementieren Sie für die Operationen *Union* und *Find* die **Heuristiken**

- **Vereinigung nach Größe** bei *Union(i, j)*. Bei der *Vereinigung nach Größe* zweier Bäume mit Wurzeln *i* und *j* wird die Wurzel des Baumes mit der kleineren Größe zum direkten weiteren Sohn des Baumes mit der größeren Größe. Kanonisches Element der Vereinigungsmenge wird die Wurzel des neuen Baumes.
- **Pfadverkürzung** bei *Find(x)*



Heuristik *Pfadverkürzung*
bei Find(x)

Aufgabe 11 (experimentelle Performance-Analyse)

Programmieren Sie eine Testumgebung, um die Heuristiken *Vereinigung nach Größe* und *Pfadverkürzung* experimentell zu testen. Erzeugen Sie dazu möglichst große Kollektionen von großen Mengen, die mit der Funktion *Union* und der Heuristik *Vereinigung nach Größe* vereinigt werden. Wenden Sie auf die so erzeugte große Menge viele *Find*-Operationen mit der Heuristik *Pfadverkürzung* an.

Um aussagekräftige Ergebnisse zu erzielen, müssen sehr große Kollektionen und Mengen erzeugt werden, vielleicht einige 100.000 oder Millionen. Steuern Sie die jeweilige Größe der Kollektion und die Größe der Mengen interaktiv über die Testumgebung. Stellen Sie die Testergebnisse tabellarisch und grafisch dar. Überlegen Sie sich genau, wie Darstellungsformen für die Testergebnisse geeignet sind.

Was sagt die **Theorie**, wenn **m = f + n** Operationen ausgeführt werden, wobei **f** = # *Find*-Operationen und **n** = # *Make-Set*-Operationen?

Satz (Vereinigung nach Größe und Pfadverkürzung)

Durch die Verwendung der Strategien *Vereinigung nach Größe* und *Pfadverkürzung* benötigen **m** Union-Find-Operation über **n** Elementen $\Theta(m * \alpha(m, n))$ Schritte. $\alpha(m, n)$ = Inverse der Ackermann-Funktion.

Ackermann-Funktion

$A(1, j) = 2^j$ für $j \geq 1$

$A(i, 1) = A(i - 1, 2)$ für $i \geq 2$

$A(i, j) = A(i - 1, A(i, j - 1))$ für $i, j \geq 2$

Inverse der Ackermann-Funktion

$$\alpha(m,n) = \min\left\{i \geq 1 \middle| A\left(i, \lfloor m/n \rfloor\right) > \log n\right\}$$

Folgende Parameter sollen interaktiv für die Performance-Experimente eingegeben werden

f = Anzahl der Find(x)-Operationen

n = Anzahl der Make-Set-Operationen

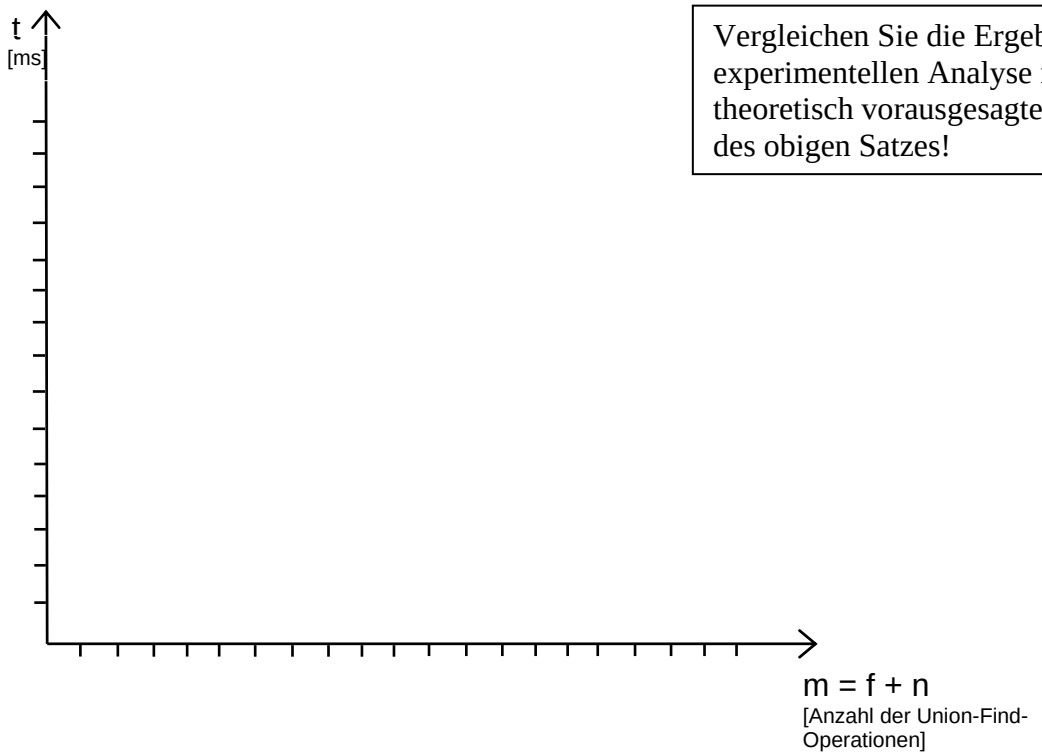
m = f + n

Durch Anwendung der *Union*-Operation wird aus der Kollektion von n Mengen eine n-elementige Menge erzeugt. Bei jeder ***Union(i,j)***-Operation ist die **Heuristik *Vereinigung nach Größe*** anzuwenden. Der Parameter x für die *Find(x)*-Operation soll zufällig aus der n-elementigen Menge gewählt werden. Überlegen Sie sich, wie man das am besten macht.

Bei jeder Ausführung einer ***Find(x)***-Operation soll die Heuristik ***Pfadverkürzung*** angewendet werden.

Die Ergebnisse sollen als **Tabelle** und als **Kurvengrafik** dargestellt werden. Überlegen Sie selbst, welche **Werte für f und n** sinnvoll sind, damit aussagekräftige Resultate erzielt werden können.

f	n	m = f + n	Laufzeit in [ms]
100	10.000	10.100	
200	20.000	20.200	
...	...	...	
1.000	100.000	101.000	
...	...	...	



Vergleichen Sie die Ergebnisse der experimentellen Analyse mit den theoretisch vorausgesagten Ergebnissen des obigen Satzes!