Prof. Dr. E. Ábrahám

F. Corzilius, S. Schupp, T. Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2 bis 3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Montag, den 12.05.2014 um 9:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch in Ihrem Tutorium vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- Sofern nicht näher spezifiziert, steht die Notation log ohne Angabe einer Basis für den natürlichen Logarithmus.
- Wenn Sie in einer Aufgabe einen **Algorithmus** erstellen müssen, so soll dieser in **Pseudo-Code** (gerne an `Java` bzw. `C++` orientiert) oder in `Java` bzw. `C++` verfasst werden. Reichen Sie den Algorithmus **nur in Maschinenschrift** ein, entweder Ihrem Tutor vor dem Abgabedatum per Email oder ausgedruckt an die Lösung geheftet. Einfache Syntaxfehler (z. B. ein fehlendes Semikolon) geben keine Punktabzüge. Inwieweit im Pseudo-Code abstrahiert werden darf, wird in der Aufgabenstellung bekannt gegeben. Ansonsten gilt es, sich möglichst nahe an den Möglichkeiten, die `Java` bzw. `C++` bieten, zu orientieren.

Tutoraufgabe 1 (Implementierung eines ADTs):

Wir spezifizieren den ADT Union-Find wie folgt:

Wertebereich: Sei `Element` ein Datentyp, der eine ganze Zahl in seinem Feld `key` enthält, d. h. `e.key` ist vom Typ `int` für `e` vom Typ `Element`.

Der Wertebereich des ADT Union-Find enthält für alle $n \geq 1$ alle Partitionen von $\{\mathtt{e}_1, \ldots, \mathtt{e}_n\}$ mit $\mathtt{e}_i\mathtt{.key} = i$ für alle $i \in \{1, \ldots, n\}$. D. h. der Wertebereich enthält für alle $n \geq 1$ und $1 \leq k \leq n$ alle Mengen $\{T_1, \ldots, T_k\}$ mit $T_i \subseteq \{\mathtt{e}_1, \ldots, \mathtt{e}_n\}$, $\mathtt{e}_i\mathtt{.key} = i$ für alle $1 \leq i \leq n$, $T_i \neq \emptyset$ für alle $1 \leq i \leq k$, $T_i \cap T_j = \emptyset$ für alle $1 \leq i < j \leq k$ und

$$\bigcup_{i=1}^{k} T_i \;=\; \{\mathtt{e}_1, \ldots,\ \mathtt{e}_n\}\,.$$

Eine Menge in einer Partition wird durch eins ihrer Elemente, ihren eindeutigen Repräsentanten, identifiziert.

Operationen:

- Der Konstruktor `UnionFind(int n)` erstellt die Partition $\{\{\mathtt{e}_1\}, \ldots, \{\mathtt{e}_n\}\}$ von $\{\mathtt{e}_1, \ldots, \mathtt{e}_n\}$ mit $\mathtt{e}_i\mathtt{.key} = i$ für alle $1 \leq i \leq n$. Vorbedingung: $\mathtt{n} \geq 1$.
- `Element find(Element e)` gibt den Repräsentanten derjenigen Menge T_j aus der Partition zurück, die `e` enthält. Vorbedingung: $1 \leq \mathtt{e.key} \leq n$.
- `void union(Element e1, Element e2)` vereinigt in der Partition die Mengen mit den Repräsentanten `e1` und `e2`. Vorbedingung: Die Partition enthält Mengen mit Repräsentanten `e1` und `e2`.

(Um diese Datenstruktur sinnvoll verwenden zu können, benötigen wir noch eine Methode, die für ein $1 \leq i \leq n$ das in der Datenstruktur verwendete Element `e` mit $\mathtt{e.key} = i$ zurückgibt; um die Aufgabe nicht zu komplex werden zu lassen, vernachlässigen wir diese Methode in der Schnittstelle.)

Beispiel: Wir initialisieren eine Union-Find Datenstruktur `UnionFind uf = UnionFind(5)` und erhalten

e1	e2	e3	e4	e5
1	2	3	4	5

Führen wir nun `uf.union(e1, e3)` aus, erhalten wir

e1 e3	e2	e4	e5
1 3	2	4	5

wobei `uf.find(e1) == uf.find(e3) == e1`. Führen wir zudem `uf.union(e5, e3)` aus, erhalten wir

e1 e3 e5	e2	e4
1 3 5	2	4

wobei `uf.find(e5) == e1`.

a) Implementieren Sie die Union-Find Datenstruktur unter Nutzung *genau einer Instanz* einer wie in der Vorlesung vorgestellten **doppelt verketteten Liste** (`List`). Es dürfen jedoch beliebig viele primitive (elementare) Datentypen verwendet werden. *Tipp: Benutzen Sie zum Trennen der Mengen in einer Partition ein Element mit dem Wert 0.*

b) Implementieren Sie die Union-Find Datenstruktur unter Nutzung *genau einer Instanz* `t` eines (nicht reduzierten) **doppelt verketteten Baumes** (`Tree`), welcher wie der in der Vorlesung vorgestellte Baum definiert ist, nur dass jedes `Element` (Knoten) `e` zusätzlich einen Vorgänger `e.father` hat (mit `t.root.father == null` und `e.father != null` für `e != t.root`).

Sie können die folgenden Operationen für `Tree` verwenden:

- `void addChild(Element e, int key)` fügt dem gegebenen Element (Knoten) `e` ein Kind mit Wert `key` hinzu. Vorbedingung: `e` ist ein Knoten im Baum.
- `void removeChild(Element e1, Element e2)` entfernt das Kind `e2` von `e1`. Vorbedingung: `e1` ist ein Knoten im Baum, `e2` ist ein Kind von `e1`.
- `void appendChild(Element e1, Element e2)` schneidet `e2` von seinem Vater ab und fügt es anschließend `e1` als Kind hinzu. Vorbedingung: `e1` und `e2` befinden sich im Baum und `e2` ist kein Vorgänger von `e1` (befindet sich nicht auf dem Pfad von der Wurzel zu `e1`).

Es dürfen jedoch beliebig viele primitive (elementare) Datentypen verwendet werden. *Tipp: Benutzen Sie zum Trennen der Mengen in einer Partition ein Element mit dem Wert 0.*

Aufgabe 2 (Implementierung eines ADTs): (3+3=6 Punkte)

Betrachten Sie einen Ringspeicher einer festen Größe k, welcher durch folgende Spezifikation definiert ist:

Wertebereich: Speicher der Kapazität k, d. h. er kann maximal k Elemente mit einem Schlüssel (im Folgenden Wert genannt) vom Typ `int` enthalten.

Operationen:

- Der Konstruktor `CircularBuffer(int k)` erstellt einen leeren Ringspeicher der Kapazität k. Vorbedingung: $k > 1$.
- `bool isEmpty()` prüft ob der Ringspeicher leer ist.
- `bool isFull()` prüft ob der Ringspeicher voll ist.
- `int read()` gibt den Wert des ältesten Elementes aus dem Ringspeicher zurück und löscht das Element anschließend. Die anderen Elemente werden nicht verändert. Vorbedingung: der Ringspeicher ist nicht leer.

- `void write(int key)` schreibt ein Element mit dem gegebenen Wert in den Ringspeicher. Ist der Ringspeicher voll, so wird das älteste Element überschrieben. Die vorigen Elemente im Speicher (bis auf das älteste falls der Speicher voll ist) bleiben unverändert.

Beispiel: Wir initialisieren mit `CircularBuffer cb = CircularBuffer(5)` einen leerer Ringspeicher der Kapazität 5. Führt man `cb.write(2)`, `cb.write(6)`, `cb.write(4)` und `cb.write(3)` in dieser Reihenfolge aus, erhält man die Belegung (2, 6, 4, 3), wobei das linke Element mit dem Wert 2 das älteste ist. Dann ist `cb.read() = 2` und führt zum Speicher (6, 4, 3). Führt man des Weiteren `cb.write(7)` und `cb.write(1)` aus, so erhält man den vollen Speicher (6, 4, 3, 7, 1). Ein weiteres Ausführen von `cb.write(8)` überschreibt die Speicherzelle mit `6` und führt also zu (4, 3, 7, 1, 8).

a) Implementieren Sie den Ringspeicher unter Nutzung genau einer Instanz des in der Vorlesung vorgestellten **Arrays** `int[]`. Es dürfen jedoch beliebig viele primitive (elementare) Datentypen verwendet werden.

b) Implementieren Sie den Ringspeicher unter Nutzung genau einer Instanz der in der Vorlesung vorgestellten **einfach verketteten Liste** (`List`) mit den folgenden Operationen:

- `Element insert(Key k)` fügt ein Element mit Schlüssel `k` am Anfang der Liste ein.
- `void remove(Element x)` entfernt das Element `x` aus der Liste. Vorbedingung: `x` ist in der Liste.
- `Element successor(Element x)` gibt das Nachfolgerelement von Element `x` in der Liste zurück. Vorbedingung: `x` ist in der Liste.
- `Element predecessor(Element x)` gibt das Vorgängerelement von Element `x` in der Liste zurück. Vorbedingung: x ist in der Liste.
- `int length()` gibt die Länge der Liste zurück.

Es dürfen jedoch beliebig viele primitive Datentypen verwendet werden.

Tutoraufgabe 3 (ADT einer Implementierung):

Gegeben sei die folgende Klasse in der Programmiersprache Java:

```
public class A {

    private int[] pos;
    private int[] neg;

    public A() {
        this.pos = new int[67108864];
        this.neg = new int[67108864];
    }

    public boolean contains(int i) {
        if (i < 0) {
            final int j = -i - 1;
            return (this.neg[j / 32] & (1 << (j % 32))) != 0;
        } else {
            return (this.pos[i / 32] & (1 << (i % 32))) != 0;
        }
    }

    public void add(int i) {
        if (i < 0) {
            final int j = -i - 1;
```

```
            this.neg[j / 32] |= (1 << (j % 32));
        } else {
            this.pos[i / 32] |= (1 << (i % 32));
        }
    }

    public void remove(int i) {
        if (i < 0) {
            final int j = -i - 1;
            this.neg[j / 32] &= (-(1 << (j % 32)) - 1);
        } else {
            this.pos[i / 32] &= (-(1 << (i % 32)) - 1);
        }
    }

}
```

Geben Sie eine möglichst allgemeine Spezifikation (Wertebereich und Signatur) für den abstrakten Datentyp an, der durch diese Klasse implementiert wird. Dabei ist der Wertebereich des Datentyps `int` die Menge $I = \{-2^{31}, -2^{31}+1, \ldots, 2^{31}-1\}$.

Aufgabe 4 (ADT einer Implementierung): (4 Punkte)

Gegeben seien folgende zwei Klassen in der Programmiersprache Java:

```
public class B {

  private Node head;

  public B() {
    this.head = null;
  }

  public Object get(int i) {
    Node c = this.head;
    while (c != null) {
      int j = c.getKey();
      if (j == i) {
        return c.getValue();
      } else if (j < i) {
        c = c.getRight();
      } else {
        c = c.getLeft();
      }
    }
    return null;
  }

  public void put(int i, Object o) {
    if (this.head == null) {
      this.head = new Node(i, o);
    } else {
      Node c = this.head;
      while (true) {
        int j = c.getKey();
        if (j == i) {
          c.setValue(o);
          return;
        } else if (j < i) {
          if (c.getRight() == null) {
            c.setRight(new Node(i, o));
            return;
          }
          c = c.getRight();
        } else {
          if (c.getLeft() == null) {
            c.setLeft(new Node(i, o));
            return;
          }
          c = c.getLeft();
        }
      }
    }
  }

}
```

```
public class Node {

    private final int key;
    private Node left;
    private Node right;
    private Object value;

    public Node(int i, Object o) {
        this.left = null;
        this.right = null;
        this.key = i;
        this.value = o;
    }

    public int getKey() {
        return this.key;
    }

    public Node getLeft() {
        return this.left;
    }

    public Node getRight() {
        return this.right;
    }

    public Object getValue() {
        return this.value;
    }

    public void setLeft(Node c) {
        this.left = c;
    }

    public void setRight(Node c) {
        this.right = c;
    }

    public void setValue(Object o) {
        this.value = o;
    }

}
```

Geben Sie eine möglichst allgemeine Spezifikation (Wertebereich und Signatur) für den abstrakten Datentyp an, der durch die Klasse `B` implementiert wird. Die Menge aller Objekte (inklusive `null`) sei hierbei mit *Obj* bezeichnet (d. h. der Wertebereich des Datentyps `Object` ist *Obj*).

Tutoraufgabe 5 (ADT verwenden):

Gegeben sei die folgende Spezifikation des abstrakten Datentyps `BinaryTree` für binäre Bäume:

Wertebereich: Spezifiziert durch alle möglichen Belegungen der Variablen
`int value`, `BinaryTree leftChild` und `BinaryTree rightChild`

Operationen:

- `int value()` gibt den Wert von `value` zurück.
- `BinaryTree left()` gibt den Wert von `leftChild` zurück.
- `BinaryTree right()` gibt den Wert von `rightChild` zurück.

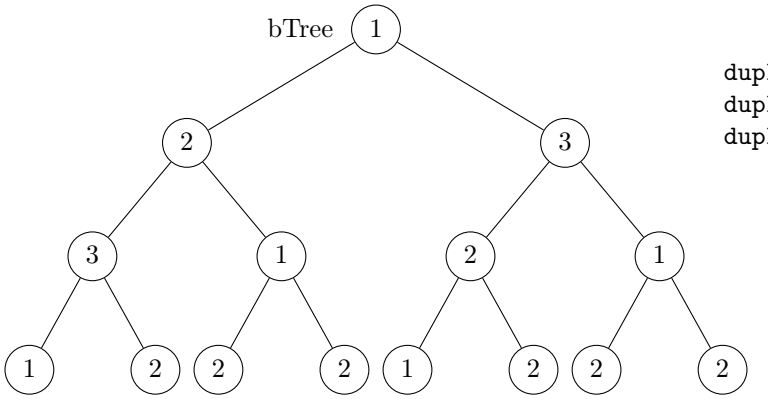
Ein Pfad in einem Binärbaum `b` $\neq$ `null` vom Typ `BinaryTree` ist eine Sequenz $\mathtt{e}_1, \ldots, \mathtt{e}_n$ von `BinaryTree` Objekten mit $n \geq 1$, $\mathtt{e}_1 = \mathtt{b}$, $\mathtt{e}_{i+1} \in \{\mathtt{left(e}_i\mathtt{)}, \mathtt{right(e}_i\mathtt{)}\}$ für alle $1 \leq i \leq n-1$ und $\mathtt{left(e}_n\mathtt{)} = \mathtt{right(e}_n\mathtt{)} = \mathtt{null}$.
Entwerfen Sie

a) einen rekursiven Algorithmus `bool duplicate(BinaryTree b, int v, bool found)`,

b) einen iterativen Algorithmus `bool duplicate(BinaryTree b, int v)`,

der überprüft, ob es im Binärbaum `b` einen Pfad gibt, auf dem mindestens zwei Objekte den Wert `v` haben. Verwenden Sie Stapel(`Stacks`) für den iterativen Algorithmus.

Beispiel:




```
duplicate(bTree,1) = true
duplicate(bTree,2) = true
duplicate(bTree,3) = false
```

Aufgabe 6 (ADT verwenden): (6 Punkte)

Gegeben sei ein Datentyp `Graph`, um ungerichtete Graphen zu repräsentieren. Dieser bietet die folgende Operation:

- `List<Node> getNeighbors(Node node)` gibt eine Liste, die alle Knoten des Graphen, die mit `node` durch eine Kante verbunden sind, enthält. Vorbedingung: `node` ist ein Knoten im Graph.

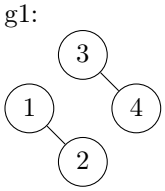
Dabei ist `List<Node>` der Datentyp für (unbegrenzte) Listen von Knoten, auf die mit den folgenden Methoden zugegriffen werden kann:

- `Element insert(Node n)` fügt ein Element mit Schlüssel (Inhalt) `n` am Ende der Liste ein.
- `void remove(Element x)` entfernt das Element `x` aus der Liste. Vorbedingung: `x` ist in der Liste.
- `Element successor(Element x)` gibt das Nachfolgerelement von `x` in der Liste zurück. Vorbedingung: `x` ist in der Liste.
- `int length()` gibt die Länge der Liste zurück.
- `bool contains(Node n)` gibt zurück, ob ein Element mit Schlüssel `n` in der Liste enthalten ist.
- `Element head()` gibt das erste Element in der Liste zurück, wenn die Liste nicht leer ist, und `null` sonst.

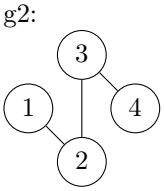
Im Graph `g` ist ein Knoten `n2` aus einem Knoten `n1` erreichbar, wenn es eine Sequenz $o_1, \ldots, o_k$ von Knoten im Graph `g` gibt, sodass $k \geq 1$, $o_1 =$ `n1`, $o_k =$ `n2`, und o_i und o_{i+1} durch eine Kante verbunden sind für alle $1 \leq i < k$.

Beispiel:

g1:



g2:




```
reach(g1,4,1} = false
reach(g1,4,3} = true
reach(g2,4,1} = true
reach(g2,4,4} = true
```

Entwerfen Sie einen Algorithmus `bool reach(Graph g, Node n1, Node n2)`, welcher überprüft, ob `n2` von `n1` in `g` erreichbar ist. Verwenden Sie dabei den DatenTyp `List<Node>`, um Knotenmengen abzuspeichern. Erklären Sie kurz den von Ihnen entworfenen Algorithmus textuell.