Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2022

H. Pflug, J. Dietel FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Klausurvorbereitung, Blatt 1 20.06.2022

Aufgabe 1

Welche minimale Laufzeitkomplexität haben die folgenden Codefragmente bezüglich der Größe des Werts *n*? Falls sich Worst- und Best-Case unterscheiden, geben Sie beide Werte an.

a)
```
for (int i=1; i<=n; i++) {
  a[i] = 0;
}
```
O(n)

b)
```
for (int i=1; i<=n; i++) {
  for (int j=1; j<=n; j++) {
    k++;
  }
}
```
$O(n^2)$

c)
```
for (int i=1; i<=n; i++) {
  for (int j=1; j<=i; j++) {
    k++;
  }
}
```
$O(n^2)$

d)
```
for (int i=1; i<=n; i++) {
  a[i]=0;
}
for (int i=1; i<=n; i++) {
  for (int j=1; j<=n; j++) {
    a[i]=a[i]+i+j;
  }
}
```
$O(n^2)$

e)
```
if (x>100) {
  y=x;
} else {
  for (int i=1; i<=n; i++) {
    if (a[i]>y) {
      y=a[i];
    }
  }
}
```
Best case:
O(1)
Worst case:
O(n)

f)
```
for (int i=1; i<=n; i++) {
  if (i>2) {
    return;
  }
}
```
O(1)

g)
```
int i=n;
while(i>=1) {
  x++;
  i/=2;
}
```
O(log n)

h)
```
int innereSchrittweite=1;
for (int i=0; i<n; i++) {
  for (int k=0; k<n; k+=innereSchrittweite) {
    innereSchrittweite*=2;
  }
}
```
O(n)

i)
```
public static long fakultaet(int n) {
  if (n>0) {
    return (n*fakultaet(n-1));
  } else {
    return 1;
  }
}
```
O(n)

j)
```
public static int quersumme(long n) {
  int sum=0;
  while(n>0) {
    sum+=n%10;
    n/=10;
  }
  return sum;
}
```
O(log n)

k)
```
public static boolean hasDoppelte(int[] m) {
  //O in Abhaengigkeit zur Feldlaenge
  Arrays.sort(m);
  for (int i=1; i<m.length; i++) {
    if (m[i-1]==m[i]) {
      return false;
    }
  }
  return true;
}
```
Best case:
O(n log n)

Worst case: $O(n^2)$

Aufgabe 2

Gegeben seien folgende Klassendefinitionen für eine einfach verkettete Liste:

```
public class Node
{
	public Node next;
	public int value;

	public Node (int wert)
	{
		value = wert;
	}
}

public class List
{
	private Node first;

	public List (Node node)
	{
		first = node;
	}

}
```

Ergänzen Sie die Klasse `List` durch folgende Methoden:

`public void addFirst(int wert)`

Fügt `wert` als ersten Knoten in die Liste ein.

`public int get (int pos)`

gibt aus der Liste den Wert des Eintrags an der Stelle `pos` zurück.

Falls die Stelle `pos` nicht existiert, wird eine `RuntimeException` geworfen.

Hinweise:

- Das erste Element hat die Position 0.
- Sie dürfen den Klassen weitere Hilfsmethoden oder –attribute hinzufügen. Machen Sie aber deutlich, zu welcher Klasse die Methode bzw. das Attribut gehört.

```
public void addFirst(int wert) {
    Node neu = new Node(wert);
    Node f = first;
    first = neu;
    neu.next = f;
  }

  public int get(int pos) {
    if (pos<0 || first==null) {
      throw new RuntimeException("Pos. "+pos+" nicht vorhanden");
    }

    Node z = first;

    for (int i = 0; i < pos; i++) {
      if (z.next == null) {
        throw new RuntimeException("Pos. " + pos + " nicht vorhanden");
      }
      z = z.next;
    }
    return z.value;
  }
```

Aufgabe 3

Implementieren Sie die Funktion

```
public static int[] mische(int[] a, int[] b)
```

die zwei bereits aufsteigend sortierte Integer-Felder übergeben bekommt und diese beiden Felder zu einem einzigen sortierten Feld zusammenfasst und dieses zurückgibt. Gehen Sie dabei wie beim Merge-Sort-Algorithmus vor.

Hinweis: Die beiden übergebenen Felder sind nicht unbedingt gleich groß.

```
public static int[] mische(int[] a, int[] b) {
            int ax = 0;
            int bx = 0;
            int[] c = new int[a.length + b.length];
            int cx = 0;
            // Solange noch Elemente in beiden Feldern sind
            while (ax < a.length && bx < b.length) {
                  if (a[ax] <= b[bx]) {
                        c[cx] = a[ax];
                        ax++;
                  } else {
                        c[cx] = b[bx];
                        bx++;
                  }
                  cx++;
            }
            // Ein Feld ist leer. Restliche Elemente
            // aus dem anderen Feld kopieren. Es werden einfach
            // beide Felder kopiert, da es in einem Feld keine
            // restlichen Elemente gibt und der Aufruf nichts
            // bewirkt.
            System.arraycopy(a, ax, c, cx, a.length - ax);
            System.arraycopy(b, bx, c, cx, b.length - bx);
            return c;
      }
```

Aufgabe 4

Schreiben Sie eine Methode
`public double[] sortierteTeilliste(double[] liste, int n).`
Die Methode sucht im Feld *liste* nach einer aufsteigend sortierten Teilliste von *n* Werten und gibt diese Teilliste als Rückgabewert zurück. Bei mehreren möglichen Teillisten wird die zurückgegeben, die am weitesten vorne im Feld steht. Ist keine Teilliste der erforderlichen Größe im Feld vorhanden, wird `null` zurückgegeben. Beispiel: Das Feld *liste* habe das Aussehen:

```
2, 5, 10, 3, 7, 11, 4, 1, 5, 9, 14, 20, 24, 13, 14
```

Für n=3 ergibt sich als Teilliste (2, 5, 10).

Für n=4 ergibt sich als Teilliste (1, 5, 9, 14). Beachten Sie, dass dies der Anfang der größeren sortierten Teilliste (1, 5, 9, 14, 20, 24) ist. Dies ist erlaubt.

```
public static double[] sortierteTeilliste(double[] liste, int n) {
            //Naive Textsuche
            //Es geht noch deutlich effizienter.
            //Diese Loesung hat aber den Vorteil, dass
            //man das Denken weitgehend einstellen kann.
            outer:
            for (int i=0; i<=liste.length-n; i++) {
                  for (int j=1; j<n; j++) {
                        if (liste[i+j-1]>liste[i+j]) {
                              continue outer;
                        }
                  }
                  //gefunden
                  double[] ret = Arrays.copyOfRange(liste, i, i+n);
                  return ret;
            }
            return null;
}
```

Aufgabe 5

Gegeben sei die folgende Klasse *Node*:

```
public class Node {
   public double data;
   public Node left;
   public Node right
}
```

Schreiben Sie eine Methode

```
public static boolean isHeap(Node root);
```

die *true* zurückgibt, falls der Baum mit der Wurzel *root* ein Heap ist (mit dem größten Element an der Spitze) und *false* sonst.

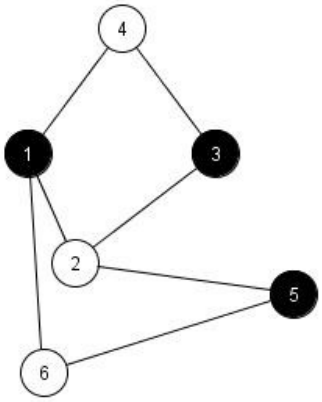
```
public static boolean isHeap(Node root) {
            return aufsteigend(root) &&
                   linksvollstaendig(root);
}
private static boolean aufsteigend(Node n) {
            //Prueft, ob fuer jeden Knoten gilt, dass die
            //Kinder des Knotens kleiner sind als der Knoten
            //selbst
            if (n.left != null) {
                   if (n.left.data > n.data) {
                          return false;
                   }
                   if (! aufsteigend(n.left)) {
                          return false;
                   }
            }
```

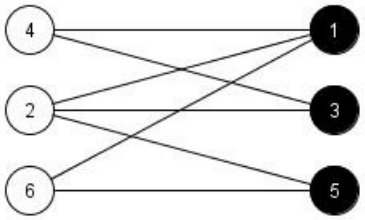
```
            if (n.right != null) {
                if (n.right.data > n.data) {
                    return false;
                }
                if (! aufsteigend(n.right)) {
                    return false;
                }
            }
            return true;
}

public static boolean linksvollstaendig(Node r) {
            //Knoten in Levelorder durchgehen.
            //In einem linksvollstaendigen Baum duerfen keine
            //Luecken entstehen. Hinter einem null-Eintrag duerfen
            //also nur noch weitere null-Eintraege stehen.

            LinkedList<Node> queue = new LinkedList<Node>();
            queue.addLast(r);          //Wurzel in Queue stecken
            boolean firstNull=false;   //noch keine Luecke gefunden
            while(!queue.isEmpty()) {
                Node n = queue.removeFirst();
                if (n==null) {
                    firstNull=true;
                } else {   //Element gefunden
                    if (firstNull) {
                        //Element hinter null-Eintrag gefunden
                        //-> nicht linksvollstaendig
                        return false;
                    }
                    //Kindelemente in Queue stecken
                    queue.addLast(n.left);
                    queue.addLast(n.right);
                }
        }
        return true;
}
```

Aufgabe 6

a) Jeder Schüler wird durch einen Knoten repräsentiert. Zwischen zwei Schülern gibt es eine Kante, wenn Sie nicht gemeinsam in eine Klasse kommen sollen.

b) Die Bedingung ist: Man kann die Knoten in zwei Farben so einfärben, dass verbundene Knoten immer unterschiedliche Farben haben. Jede Farbe steht dabei für eine Schulklasse.
Oder anders formuliert: Man kann den Graphen in zwei Knotengruppen trennen. Die Knoten innerhalb einer Knotengruppe sind untereinander nicht verbunden. Verbindungen gibt es nur zwischen Knoten aus unterschiedlichen Knotengruppen. Solche Graphen nennt man *bipartit.* Im Bild ist der gleiche Graph einmal eingefärbt und einmal zusätzlich nach Gruppen geordnet zu sehen.





c) Man geht mit Tiefensuche durch den Graphen. Man startet bei einem Knoten und färbt ihn (z.B Knoten 1 schwarz). Während der Tiefensuche färbt man alle neu erreichten Knoten in der zum Ausgangsknoten gegenteiligen Farbe. Außerdem überprüft man für jeden neu erreichten Knoten alle Verbindungen zu bereits eingefärbten Knoten. Diese müssen alle die gegenteilige Farbe besitzen.

d) Es kann passieren, dass der Graph in mehrere Teilgraphen zerfällt, die überhaupt nicht miteinander verbunden sind. Beispiel: Bei zwei Teilgraphen T1 und T2 gibt es vier Knotengruppen T1/schwarz, T1/weiß, T2/schwarz und T2/weiß. Man könnte jetzt sowohl T1/schwarz und T2/schwarz als auch T1/schwarz und T2/weiß in einer Schulklasse zusammenlegen, je nachdem, welche Kombination eine gleichmäßigere Verteilung ergibt.