Tentamen TI1310 Datastructuren en Algoritmen, 15 april 2011, 9.00-12.00
TU Delft, Faculteit EWI, Basiseenheid Software Engineering
Bij het tentamen mag alleen de boeken van Goodrich en Tamassia worden gebruikt

Opgave 1 (30 = 10 + 17 + 3 punten)

Bijlage 1 bevat de implementatie van klasse BirthDay. De klasse bevat de gegevens van de verjaardag van iemand. De datum wordt gerepresenteerd met een dagnummer dat loopt van 1 t/m 366 (vanwege schrikkeljaren). Bestudeer de implementatie grondig voordat je verder gaat.

Bijlage 2 bevat een gedeeltelijke implementatie van klasse BirthDayList die BirthDay-objecten bevat. Aan deze klasse moeten enkele methoden (equalDates(), iterator()) worden toegevoegd. Bestudeer ook deze implementatie grondig voordat je begint met het beantwoorden van de vraag.

a. Gevraagd om de implementatie van een methode **`equalDates()`** die bepaalt of er twee BirthDay-objecten zijn met hetzelfde dagnummer. De implementatie moet van de orde O(n) zijn. Oplossingen die niet van deze orde zijn worden niet goedgekeurd!

```
public boolean equalDates()
post: returns true iff there if the list contains at least two
BirthDay objects with the same dayNumber
```

b. Gevraagd wordt om de implementatie van een klasse BirthDayIterator zoals hieronder is gespecificeerd. De iterator bezoekt (en retourneert) de elementen van een BirthDayList in oplopende volgorde van dagnummers. Vb: bevat de lijst objecten o1 = (“a”,45), o2 = (“b”,12) en o3 = (“c”,30) dan wordt eerst o2, dan o3 en tenslotte o1 geretourneerd.

```
BirthDayIterator implements Iterator
//invariant: the objects are visited in ascending order of dayNumbers

+BirthDayIterator(BirthDayList bList)
//post: had created a BirthDayIterator object

+reset()
//post: has reset the BirthDayIterator

+hasNext() : boolean
//post: returns true iff there are objects to be visited

+next() : Object
//pre: hasNext()
//post: returns next element to be visited
```

c. Gevraagd wordt om de implementatie van methode **`iterator()`** die een BirthDayIterator retourneert.

Opgave 2 (20 = 4 + 3 + 2 + 5 + 1 + 5 punten)

Gegeven een array gevuld met n gehele getallen. De getallen zijn in oplopende volgorde geordend. In het array kan een getal één of meer keer voorkomen. Een voorbeeld van zo'n array met (16) elementen is: [0,0,1,1,1,2,4,4,4,4,5,5,8,9,9,9]

Gevraagd wordt een methode MRand() die de **grootste** index geeft van een array-element dat kleiner is dan of gelijk is aan een gegeven getal M. Is M kleiner dan het eerste array-element dan is de waarde van MRand() = -1.

Toelichting: bij het gegeven array is MRand(1, array) = 4, MRand(2, array) = 5, MRand(3, array) = 5, MRand(20, array) = 15 en MRand(-1, array) = -1.

De implementatie van de methode dient er als volgt uit te zien:

```
public static int MRand (int M, int[] array){
//pre:  de elementen van array zijn oplopend geordend
//post: retourneert de grootste index van een getal dat kleiner is dan
//      of gelijk aan M

      int dak;
      int vloer;
      int huidige;
      //initialisatie van dak en vloer
            …

      while (<voorwaarde>){
            …
      }
      return <expressie>;
}
```

Het algoritme dient gebaseerd te zijn op de volgende regels:

1. Alle elementen in array[0], ..., array[dak-1] zijn kleiner dan of gelijk aan M,
2. Alle elementen in array[vloer+1], ..., array[n-1] zijn groter dan M.
3. Het algoritme maakt gebruik van binair zoeken (O(log(n)).

Een implementatie die niet is gebaseerd op de gegeven regels of die gebruik maakt van andere dan de gegeven variabelen, wordt niet goedgekeurd.

Gevraagd

a. Geef de initialisatie van dak en vloer. Laat zien dat na de initialisatie regel 1 en regel 2 gelden.
b. Geef aan wanneer het zoeken naar de hoogste index van een element kleiner dan of gelijk aan M kan worden gestopt.
c. Geef de voorwaarde **<voorwaarde>** voor de herhalingsopdracht.
d. Geef de opdrachten die binnen de herhalingsopdracht worden uitgevoerd. Laat zien dat na uitvoering van de opdrachten regels 1 en 2 geldig zijn
e. Geef de expressie **<expressie>**.
f. Laat zien dat het algoritme van de orde log(n) is.

Opgave 3 (30 = 8 + 8 + 8 + 3 + 3 punten)
Bijlage 3 bevat de gedeeltelijke inplementaties van klasse IntNode en klasse BinaryTree. Klasse BinaryTree is een binaire zoekboom opgebouwd uit IntNode objecten. Alleen de interne knopen bevatten gehele getallen, de externe knopen bevatten de constante **`MIN_INFINITY`**.
Bij de zoekboom worden values die kleiner zijn dan de value van de wortel aan de linkerkant toegevoegd en values gelijk aan of groter dan de value van de wortel aan de rechterkant.

Gevraagd om de implementatie van een drietal methoden:

a. Geef een implementatie van methode **`add()`** die een geheel getal toevoegt aan de binaire zoekboom.

```
public boolean add(int i, IntNode n)
//post: er is een IntNode met value = i aan de zoekboom toegevoegd
```

b. Geef een implementatie van methode **`contains()`** die nagaat of een geheel getal i voorkomt in een binaire boom met wortel n.

.
```
public boolean contains(int i, IntNode n)
//post: retourneert true dsd de boom met wortel n het getal i bevat
```

c. Geef een implementatie van methode contains() die nagaat of de boom met wortel n alle getallen bevat die in een andere boom met wortel m zijn opgeslagen.

.
```
public boolean contains(IntNode m, IntNode n)
//post: retourneert true dsd n alle getallen bevat die in m zijn
//       opgeslagen
```

d. Wat is de (worst) tijd-complexiteit van de implementatie van contains van opgave b? Licht je antwoord toe.

e. Wat is de (worst) tijd-complexiteit van de implementatie van contains van opgave c? Licht je antwoord toe.

Opgave 4 (20 = 7 + 10 + 3 punten)

a. Gegeven de volgende differentievergelijking voor T(n):

$T(n) = 1$, als $n = 1$
$T(n) = 2T(n-1)$, voor $n > 1$

Geef een oplossing voor deze differentievergelijking. Laat zien hoe je deze oplossing hebt afgeleid.

b. In boek is het algoritme van Dijkstra besproken om het **kortste** pad tussen twee knopen in een ongerichte graaf te bepalen. In Bijlage 4 is een kopie van het algoritme gegeven.

Gegeven een gerichte, acyclische graaf waarbij aan elke edge een gewicht is toegekend. Gevraagd wordt een aanpassing van het algoritme van Dijkstra om het **langste** pad te bepalen tussen twee knopen. Geef in Bijlage 4 aan welke veranderingen er nodig zijn. Leg uit wat de rol is van de veranderingen en waarom jouw oplossing correct is.
Lever Bijlage 4 in.

c. Kan jouw algoritme ook worden toegepast bij een gerichte graaf die cykels kan bevatten. Licht je antwoord toe.

Bijlage 1

```
//klasse met gegevens van de datum van de verjaardag van iemand
//i.p.v. een "echte" datum wordt het dagnummer (1 t/m 366) gebruikt
public class Birthday implements Comparable{

	private String name;    //name of the person
	private int dayNumber;  //jan 1 -> 1, dec 31 -> 366

	public Birthday(String nm, int dn){
		setName(nm);
		setDayNumber(dn);
	}

	public void setName(String nm){     name = nm;         }
	public void setDayNumber(int nr){   dayNumber = nr;    }
	public String getName(){            return name;       }
	public int getDayNumber(){          return dayNumber; }

	public int compareTo(Object other){
		Birthday that = (Birthday)other;
		int dNummer1 = this.getDayNumber();
		int dNummer2 = that.getDayNumber();
		return (dNummer1 - dNummer2);
	}

	public boolean equals(Object other){
		if (other instanceof Birthday){
			Birthday that = (Birthday)other;
			return this.name.equals(that.name) &&
				this.dayNumber == that.dayNumber;
		}
		return false;
	}
}
```

Bijlage 2

```
public class BirthDayList{

	private ArrayList<BirthDay> birthDays;

	public BirthDayList(){
		birthDays = new ArrayList<BirthDay>(366);
	}

	public void add(BirthDay bd){ birthDays.add(bd);            }
	public BirthDay get(int i){   return birthDays.get(i);      }
	public int size(){            return birthDays.size();      }

	//public boolean equalDates()

	//public Iterator iterator()
}
```

Bijlage 3

```
public class IntNode{

      private int value;
      private IntNode left, right;

      public IntNode(int v, IntNode l, IntNode r){
            setValue(v);
            setLeft(l);
            setRight(r);
      }

      public int getValue(){              return value;     }
      public IntNode getLeft(){           return left;      }
      public IntNode getRight(){          return right      }
      public voud setValue(int v){        value = v ;       }
      public void setLeft(IntNode l){     left  = l;        }
      public void setRight(IntNode r){    right = r;        }
}


public class BinaryTree{

      public static final int MIN_INFINITY = Integer.MIN_VALUE;
      private IntNode root;
      private int size;

      public BinaryTree(){
            root = new IntNode(MIN_INFINITY,null,null);
            size = 0;
      }

      public boolean isExternal(IntNode n){
            return n.getValue() == MIN_INFINITY;
      }

      public int size(){                  return size;            }

      public void add(int i){             add(i,root)};           }
      public boolean contains(int i){     return contains(i,root);}

      public boolean contains(BinaryTree m){
            return contains(root,m.root);
      }

      //public void add(int i, IntNode n)

      //public boolean contains(int i, IntNode n)

      //public boolean contains(IntNode n, IntNode m)
}
```

Bijlage 4 **Lever deze bijlage in!**

Naam:

Studienummer:

Algorithm ShortestPath(G, v):

Input: A simple undirected weighted graph G with nonnegative edge weights, and a distinguished vertex v of G.

Output: A label $D[u]$, for each vertex u of G, such that $D[u]$ is the length of a shortest path from v to u in G

Initialize $D[v] \leftarrow 0$ and $D[u] \leftarrow +\infty$ for each vertex $u \neq v$.
Let a priority queue Q contain all the vertices of G using the D labels as keys.
while Q is not empty **do**
  {pull a new vertex u into the cloud}
  $u \leftarrow Q$.removeMin()
  for each vertex z adjacent to u such that z is in Q **do**
    {perform the ***relaxation*** procedure on edge (u,z)}
    if $D[u] + w((u,z)) < D[z]$ **then**
      $D[z] \leftarrow D[u] + w((u,z))$
      Change to $D[z]$ the key of vertex z in Q.
return the label $D[u]$ of each vertex u

Code Fragment 13.14: Dijkstra's algorithm for the single-source shortest path problem.

Korte uitwerking:

Opgave 1a.
Als je de arrayList lineair zou doorlopen, en per BirthDay zoekt naar een ander BirthDay met hetzelfde dagnummer, krijg je een algoritme van de orde $O(n^2)$.
Om snelheid te winnen, dien je te "betalen" met geheugen. Creeer in de methode equalDates() een array (van BirthDays) met 366 plaatsen. Plaats alle BirthDay-objecten uit de ArrayList in het array op de index die gelijk is aan het dagnummer. Op het moment dat je een BirthDay object op een niet lege plaats gaat plaatsen, weet je dat er twee BirthDay-objecten zijn met hetzelfde dagnummer.

1b.
Je moet de BirthDays-objecten in oplopende volgorde volgorde opleveren. Je kunt dat doen door de ArrayList te sorteren, of door een datastructuur te gebruiken die het kleinste of het grooste element geeft, m.a.w een PriorityQueue.

De klasse heeft 2 attributen: 1. Een referentie naar de parameter bList, en 2. een priority-queue.

In de constructor wordt de waarde van bList opgeslagen in het 1e attribuut en wordt reset() aangeroepen.

In reset() worden de elementen van de ArrayList opgeslagen in de priorityqueue.

Methode hasNext() onderzoekt of de priority-queue leeg is, method next() geeft het volgende element uit de priority-queue.

1c.
Spreekt voor zichzelf.

Opgave 2.
Deze opgave is een variant op het algoritme van binair zoeken. Deel de rij op in 3 delen, het eerste deel met getallen kleiner dan of gelijk aan het getal M, het derde deel met getallen groter dan M, en het middendeel waarvan we nog niets weten, dus [k,k,k,?,?,?,?,?,g,g,g,g,g,g].

Bij de aanvang van het algoritme zijn het 1^e en 3^e deel leeg en is de situatie [?,?,?,?,?,?,?,?,?,?,?,?,?,?]. Na afronding is de situatie [k,k,k,k,k,k,k,g,g,g,g,g,g,g,g].

De indices van de uiterste elementen van het 1^e deel zijn **0** en **dak-1**, van het 3^e deel **vloer+1** en **n-1**, van het middendeel **dak** en **vloer**.

Op het moment dat dak > vloer is het middendeel leeg en eindigt het algoritme.

Nadere details over de oplossing zijn te vinden op de college slides.

Opgave 3.
Bij de doorlopen van de boom ga je naar links als de waarde die je zoekt kleiner is dan de waarde in een knoop en naar rechts als de waarde groter of gelijk is dan/aan de waarde in de knoop.

3a. zoek via de procedure die boven is genoemd naar een externe knoop. Als je deze gevonden hebt vervang je deze door een interne knoop met het getal i, en twee externe knopen

3b. zoek via de procedure die boven is genoemd naar een knoop die het getal i bevat. Als je zo'n knoop kunt vinden is het antwoord true, anders false.

3c. doorloop de boom met wortel m. Bepaal voor elk getal in deze boom of dit getal in de boom met wortel n voorkomt. Is dit voor alle getallen uit m waar, dan is het antwoord true, anders false.

3d. Als de boom is gedegenereerd tot een lineaire lijst, moet je in het "ergste" geval k stappen doen, dus O(k), waarbij k het aantal knopen in n is.

3e. Als boom m l knopen telt is het "ergste" geval O(k*l).

4a. zie slides

4.b.

Algorithm ShortestPath(G, v):
Input: A simple undirected weighted graph G with nonnegative edge weights, and a distinguished vertex v of G.
Output: A label $D[u]$, for each vertex u of G, such that $D[u]$ is the length of a shortest path from v to u in G

```
Initialize D[v] ← 0 and D[u] ← +∞ for each vertex u ≠ v.
Let a priority queue Q contain all the vertices of G using the D labels as keys.
while Q is not empty do
  {pull a new vertex u into the cloud}
  u ← Q.removeMin()
  for each vertex z adjacent to u such that z is in Q do
    {perform the relaxation procedure on edge (u,z)}
    if D[u] + w((u,z)) < D[z] then
      D[z] ← D[u] + w((u,z))
      Change to D[z] the key of vertex z in Q.
return the label D[u] of each vertex u
```

Code Fragment 13.14: Dijkstra's algorithm for the single-source shortest path problem.

Aanpassing:
1.Initialiseer alle afstanden D[u] als $-\infty$, als v <> u

2. Neem een priorityqueue die het grootste element oplevert => Q.removeMax

Verander de voorwaarde voor de relaxatie in:
If D[u] + w((u,z)) > D[z] then
    D[z] <- D[u] + w((u,z))

c.
Als er cykels zijn, zal het algoritme niet termineren, aangezien door het toevoegen van een cykel er een langer pad zal ontstaan.

Bij het gegeven algoritme wordt een knoop met de grootste afstand verwijderd. De grootste afstand kan dan nog wel via een andere knoop worden veranderd, maar de knoop kan niet meer worden gebruikt om de afstanden in de cykel aan te passen.