Data och Informationsteknik / Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Erland Holmström

Göteborg 6 mars 2014

Examination in

Datastrukturer och Algoritmer IT TDA416

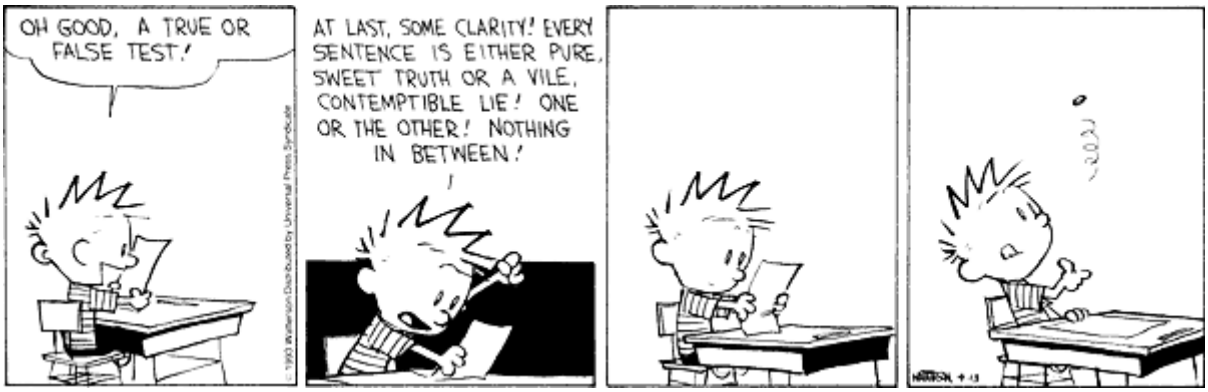
DAY: Tuesday DATE: 2014-03-11 TIME: 14.00-19.00 (5 tim) ROOM: M

Responsible teacher:	Erland Holmström 0708-710 600
Results:	Are sent by mail from Ladok.
Solutions:	Are eventually posted on home page.
Inspection of grading:	The exam can be found in our study expedition after posting of results. Time for inspection of grading are announced on homepage after the result are published or mail Erland and we find a time.
Grade limits:	CTH: 3=28p, 4=38p, 5= 48p. Maxpoäng = 60. *OBS ! A grade of 3 or better also requires you to have at least 10p on both part A and B individually.*
Aids on the exam:	Only a summary of Javas API that is handed out on the exam or that you have printed beforehand, see homepage.

Observe:

- Start by reading through all questions so you can ask questions when I come. I usually will come after appr. 2 hours.
- *All answers must be motivated* when applicable and not otherwise stated.
- Write legible! Draw figures. Solutions that are difficult to read are not evaluated!
- Answer concisely and to the point.
- The advice and directions given during course must be followed.
- Programs should be written in Java, indent properly, use comments and so on.
- Start every new problem on a new sheet of paper.

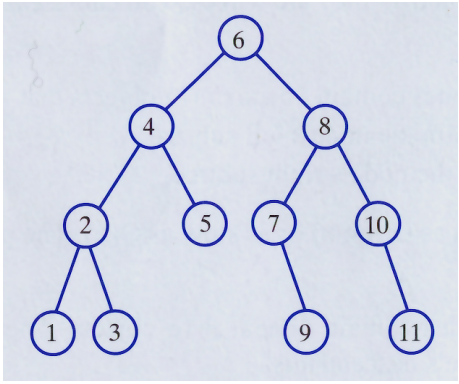
Good Luck!



Del A Teori

(Tänk på att detta är del A, jag vill tex ha idébeskrivning som förklarar hur och varför algoritmer fungerar och eventuellt pseudokod om du vill men inte körbar Java kod här. Beskriv kortfattat" == steg 1. Du måste också motivera dina svar.

Problem 1. *Småfrågor:* Avgör om vart och ett av följande påståenden är sant eller falsk eller svara på frågan. För att få poäng måste du ange en kortfattat *motivation* till ditt svar tex kan du för första frågan ange definitionen på en heap och visa om trädet uppfyller/inte uppfyller den.



a) Är trädet till höger en heap?

b) Är det ett binärt sökträd?

c) Är det 1-balanserat?

d) Lista trädets noder i preorder.

e) Lista trädets noder i nivå-ordning.

Antag att du har ett binärt träd med höjd h. En ensam nod har höjd 1.

f) Vad är det maximala antalet noder i trädet?

g) Vad är det minimala antalet noder i trädet?

h) Vad är det minimala antalet löv?

Analysera komplexiteten på följande metoder? (Bortse från eventuella syntaxfel)

i)

```
boolean MyEquals(String str) {
  return str.contains("hello");
}
```

j)

```
boolean checkString(String[] strs, String str) {
  for (int i=0; i<strs.length; i++) {
    if (strs[i]==str) {
      return true;
    } else {
      return false;
  }
}
```

(10p)

Problem 2. *Testar: Träd*

Med **kortaste träd** avser man ett träd där den längsta vägen från roten till ett löv är så kort som möjligt för givna data.

a) Rita upp det kortaste binära sökträdet från följande strängar:

jan, feb, mar, apr, maj, jun, jul, aug, sept, okt, nov, dec.

Är det unikt?

b) Antag nu att du har 1000 (eller n) strängar. Beskriv algoritmen (steg 1) för att konstruera det kortaste binära sökträdet. Komplexitet?

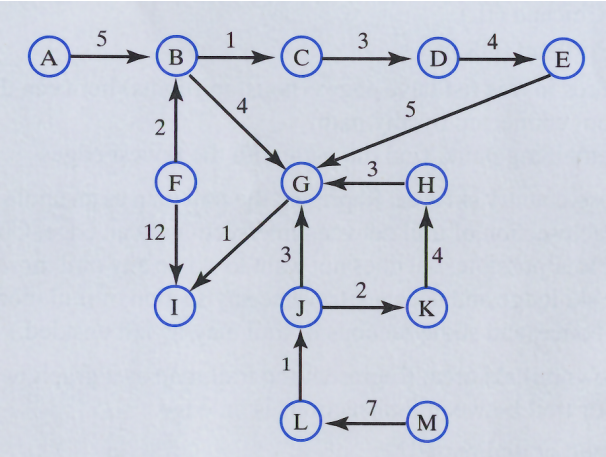
(8p)

Problem 3. *Testar: Grafer*

Vid alla uppgifterna nedan så *görs val i bokstavsordning* tex startar du i A i uppg a).

a) Beskriv en algoritm för topologisk sortering och ge en topologisk sortering av grafen till vänster.

b) Beskriv en algoritm för djupet först sökning och ge en djupet först genomlöpning med start i M av grafen till vänster. Lista noderna i upptäcksordning (discovery order).



(12p)

Del B Implementeringar

Problem 4. *Testar: länkade strukturer, sortering, rekursion*

En **aktivitetslista** består av en *sorterad* lista med aktiviteter. En aktivitet består av en beskrivning samt en start- och en sluttid tex enligt {"*", 2, 5}. Beskrivningen är inte så intressant här så jag använde bara en "*" men det kan tex vara en sträng med ansvarig för aktiviteten och talen kan tex representera klockslag (även om jag använder heltal för enkelhets skull). Aktiviteten startar här på 2 och slutar på 5.

Man kan lägga till aktiviteter till listan men dom måste vara kompatibla med de aktiviteter som redan finns i listan. Två aktiviteter är kompatibla om deras intervall inte överlappar. 1,3 och 3,5 är kompatibla men 1,3 och 2,5 är det inte.

Du skall skriva en klass, `ActivityList`, som hanterar en sådan lista. Två inre hjälpklasser finns definierade redan, se nedan.

- Klassen skall implementeras med en *enkellänkad* lista sorterad efter starttiden.
- Du skall skriva metoderna add, remove och getSize.

add är en wrapper som anropar en ***rekursiv*** metod som sätter in aktiviteten på rätt plats i listan om den inte överlappar med någon annan aktivitet i listan. (Tips: Använd samma teknik som för insättning i binära sökträd med retur av pekaren till "next")

remove tar bort och returnerar första elementet i listan

getSize returnerar hur många aktiviteter som finns i listan.

- Inga operationer får vara onödigt ineffektiva tex gå igenom hela listan om det inte behövs.
- Du måste tänka på "säkerhet", det får inte gå att manipulera listan utifrån annat än med de publika metoderna.
- Tänk på felhanteringen

Klassen Activity har publika instansvariabler för enkelhets skull.

Förutom till klassen Node så behöver du inte använda typparametrar.

På vilken sorteringsmetods-idé har du baserat din implementation? Motivera.

(15p)

Problem 5. *Testar: Iteratorer*

a) Vilka förändringar måste göras med din lösning i förra uppgiften om du vill implementera en iterator (förutom att lägga till en inre iterator klass enl. nedan)? Förklara också vad dina förändringar används till och hur det sker.

b) Implementera iteratorn enligt nedan.
Med kunskapen att listan är implementerad som en enkellänkad lista så kan den här klassen skrivas utan att man löst uppg 4.

```
private class ActivityListIt implements Iterator<Activity> {
    ...
    public ActivityListIt() {...}
    /*****************************
    * Test if next is going to be able to return an object
    * @return true if next is going to be able to
    * return an object
    */
    public boolean hasNext( ) {...}
    /*****************************
    * Return the next item and move the iterator forward.
    * @return the stored item
    * @throws NoSuchElementException if there is no such object
    */
    public Activity next( ) {...}
    // ****************************
    public void remove( ) {
      throw new UnsupportedOperationException();
    } // ****************************
  } // end class ActivityListIterator
```

(15)

Ett skal för klassen ActivityList

```
public class ActivityList {

  public void add(Activity fresh) {...}
  public Activity remove(){...}
  public int getSize(){...}

  // ***********************************************************
  public static class Activity {
    /** The data values. */
    public String data = null;
    public int start = 0;
    public int end = 0;

    /** Construct an activity with the given data values.
    * @param data The data value
    * @param start The start time
    * @param end The end time
    */
    public Activity(String data, int start, int end) {
      this.data = data;
      this.start = start;
      this.end = end;
    }
    public String toString() {
      return "{" + data + "," + start + "," + end + "}";
    }
  } //end class activity
  // ***********************************************************
  private static class Node<E> {
    /** The data values. */
    private E data = null;
    private Node<E> next = null;

    /** Construct a node with the given data values.
    * @param data The data value
    * @param next The next node
    */
    private Node(E data, Node next) {
      this.data = data;
      this.next = next;
    }
  } //end class Node
} // end class ActivityList
```