Institutionen för
Datavetenskap
Chalmers, Göteborgs universitet

VT09
DIT960
17/8-09

T E N T A M E N för

Datastrukturer DV, 7.5p, DIT960

DAG : 20 augusti 2009 Tid : 8.30-13.30 SAL : V

Ansvarig	:	Bror Bjerner, tel 772 10 29, 55 54 40
Resultat	:	senast den 28/8 -09
Hjälpmedel	:	Häftet med Fakta-rutor, papper och penna.
Betygsgränser GU	:	Godkänt = 28 p, Väl godkänt = 48 p

OBS ! För att få betyg Godkänt eller bättre krävs minst 10 poäng på både A- och B-delen

(Om du endast tentar en del måste du lämna in senast **11.30**)

OBSERVERA NEDANSTÅENDE PUNKTER

- Börja varje ny uppgift på nytt blad.
- Skriv personlig kod på varje blad.
- Använd bara ena sidan på varje blad.
- Klumpiga, komplicerade och/eller oläsliga delar ger poängavdrag.
- **L y c k a t i l l ! ! !**

Del A

Datastrukturer på abstrakt nivå.

Uppg 1: **a)** Stoppa in 'värdena' 1, 2, 3, 6, 4 och 5 i ett binärt sökträd. Du skall stoppa in dem i **exakt** denna ordning (dvs först 1, sedan 2, sedan 3, sedan 6 osv) och utför en sökning med splay-balansering efter 5. Visa de balanseringar som utförs vid sökningen genom att visa trädet du har före, med noderna som ingår i balanseringen markerade, och efter varje rotation.

Obs! inga balanseringar vid insättningen, bara vid sökning.

(3 p)

b) Stoppa in samma noder i samma ordning i ett AVL-träd och redovisa när obalanserna uppstår och hur balanseringarna sker.

(3 p)

c) Stoppa in samma noder i samma ordning i ett rödsvart träd, där insättning sker enligt 'top-down med flip'. Markera svarta noder med en fyrkant och röda noder med en cirkel.Visa varje 'flip' och rita om trädet efter varje rotation.

(4 p)

Uppg 2: Vilken eller vilka av följande påstående är sann(a) och vilken eller vilka är falsk(a). Svara bara med Sant eller Falskt. För varje delfråga ger rätt svar 1 p och fel svar ger -0.6 p. Poängen för hela uppgiften kan däremot inte bli negativ.

a) Insättning i binärt sökträd respektive ett AVL-träd har samma komplexitet.

b) Dijkstra's algoritm fungerar på alla oriktade grafer.

c) Köer kan implementeras effektivt med en enkellänkad datastruktur. (Dvs alla operationer av $\mathbf{O(1)}$).

d) Sammansmältningssortering (merge sort) har alltid lägsta värstafalls-komplexiteten av sorteringsmetoderna, om du inte vet hur sorteringsnycklarna ser ut.

e) Binär sökning i ett sorterat fält är inte alltid snabbare än linjär sökning.

f) Insättningssorterings (insertion sort) komplexitet beror inte på hur elementen är ordnade i fältet.

(6 p)

Uppg 3:

80	90	110	40	30	60	100	50	120	10	20	70

Visa hur 'quick sort' fungerar genom att sortera fältet ovan enligt den förenklade strategin, som angavs på en föreläsning. Pivot-elementet skall vara **elementet längst till höger** i delsegmentet som skall sorteras. Markera med bågar vilka element som byter plats och med en liten pil pivotelementen. Rita ett nytt fält efter varje pivottering. Det är givetvis tillåtet (och lämpligt) att utföra pivotteringarna av delfälten 'samtidigt' på varje nivå. Notera att du inte skall byta sorteringsmetod utan göra 'quick sort' ända ner i botten. Vidare skall sorteringen göras i fältet.

(8 p)

Uppg 4: **a)** Låt $G = (V, E)$ vara en graf med $|V|$ noder och $|E|$ bågar. Antag att vi vill färga varje nod i grafen G antingen svart eller vit på ett sådant sätt att två grannoder inte får samma färg. Vi har en startnod som har färgen svart. Skriv en algoritm i pseudokod som antingen returnerar den färgade grafen om det går eller annars svarar att det är omöjligt att färga den på rätt sätt. Ett exempel på en graf som går att färga på detta sätt är en rektangel och ett exempel på en graf som inte går att färga är en triangel.

(10 p)

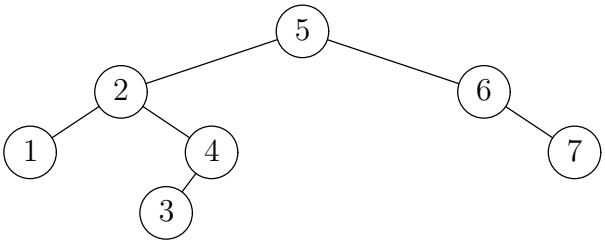
b) Vad är värstafalls-komplexiteten för metoden med avseende på antalet bågar $\mathbf{0}(|E|)$ och/eller antalet noder $\mathbf{0}(|V|)$, motivera.

(2 p)

Del B

Datastrukturer på implementeringsnivå.

Uppg 5: Givet ett binärt träd vill du traversera trädet i enligt **bredden först**. Till exempel, för trädet:



skall elementen ges i följande ordning: 5, 2, 6, 1, 4, 7 och 3

Givet är en modul `BinTree`:

```
module BinTree (
  BT,        -- type of binary trees
  emptyTree, -- BT a
  isEmpty,   -- BT a -> Bool
  leftSub,   -- BT a -> BT a
  rightSub,  -- BT a -> BT a
  rootVal,   -- BT a -> a
  insert,    -- Eq a => a -> BT a -> BT a
  remove,    -- Eq a => a -> BT a -> BT a
  inorder,   -- BT a -> [a]
  contains   -- Eq a => a -> BT -> Bool
)
```

Du skall nu definiera funktionen

```
breadthFirst :: BT a -> [a]
```

som ger elementen i trädet i preorder. För full poäng krävs att funktionen är av $\mathbf{O}(n)$

Notera: du skall **inte** göra denna funktion inuti modulen !!

(Du har även tillgång till köer och/eller stackar, se nästa sida.)

(7 p)

```
module Stack (
 Stack,      -- type of stacks
 emptyStack, -- Stack a
 isEmpty,    -- Stack a -> Bool
 push,       -- a -> Stack a -> Stack a
 pop,        -- Stack a -> Stack a
 top         -- Stack a -> a
)

module Queue (
  Queue,      -- type of queues
  isEmpty,    -- Queue a -> Bool
  emptyQueue, -- Queue a
  enqueue,    -- a -> Queue a -> Queue a
  dequeue,    -- Queue a -> Queue a
  front       -- Queue a -> a
)
```

Uppg 6: Antag att vi vill implementera en samling (collection) som en enkellänkad cikulär struktur, där variabeln `last` refererar till det sista elementet i den länkade strukturen. Vi startar med följande deklarationer:

```
public class  SingleCircularCollection<E>
       extends AbstractCollection<E>{

   protected Entry  last;

   protected class Entry {
      Entry next;
      E     elem;
      public Entry(  E e, Entry n ) {
        next = n; elem = e;
      }
}
```

Din uppgift är att implementera nedanstående 3 metoder. Du får inte lägga till någon instansvariabel. Du får heller inte använda dig av iteratorn.

a) Skriv en metod `public boolean add(E elem)`, som lägger till ett element **först** i samlingen.

(2 p)

b) Skriv en metod `public int size()`, som ger antalet element i samlingen.

(2 p)

c) Skriv en metod `public boolean removeLast()`, som tar bort det sista elementet i samlingen.

(3 p)

d) Vad är komplexiteten för var och en av dessa 3 metoder, det krävs att alla 3 är rätt för att få poäng.

(1 p)

Uppg 7: **a)** Du skall implementera Dijkstra's algoritm, som ger den kortaste vägen (shortest path) mellan 2 noder i en oriktad viktad graf. Grafen har implementerats med en matris (se nästa sida). Vikterna är alla positiva heltal och noderna är numrerade från 0 till antalet noder-1. (Jag har förenklat grafen en del för att det inte skall bli för många oväsentliga detaljer.)

Metoden ges en startnod och en slutnod att starta med. Resultatet skall vara en iterator av bågar, som deklareras i grafen, för den kortaste vägen mellan noderna i grafen. Om ingen väg kan hittas returneras den tomma iteratorn.

Du får själv välja om du vill använda en priotitetskö eller inte.

(14 p)

b) Vad är komplexiteten för metoden med avseende på antalet noder $\mathbf{0}(|V|)$ och/eller antalet bågar $\mathbf{0}(|E|)$, motivera.

(1 p)

```
public class UndirectedGraph {

 public static class Edge {

   private int node1,
               node2,
               weight;

   public Edge( int n1, int n2, int w ) {
     node1  = n1;
     node2  = n2;
     weight = w;
   }

   public String toString() {
     return "(" + node1 + ","
                + node2 + ","
                + weight + ")";
   }
 } // Edge

 private int[][] graph;

 public UndirectedGraph( int noOfNodes ) {
   graph = new int[noOfNodes][noOfNodes];
   for( int i = 0; i < noOfNodes; i++ )
      Arrays.fill( graph[i], -1 );
 } // UndirectedGraph

 public void addEdge( int n1, int n2, int w ) {
   assert n1 >= 0 && n1 < graph.length &&
          n2 >= 0 && n2 < graph.length &&
          w  >  0;
   graph[n1][n2] = w;
   graph[n2][n1] = w;
 } // addEdge

 public Iterator<Edge>
        shortestPath( int startNode, int endNode ) {
    ******     Här skriver du din kod      *****
 } // shortestPath
```