Lösningsskisser Datastrukturer IT TDA416 2017-03-11

Problem 1. Uppvärmning:

Vilken eller vilka av följande påstående är sann(a) och vilken eller vilka är falsk(a) eller svara på frågan. Alla svar skall motiveras.

a) Västafallskomplexiteten för find i ett splay träd är O(n).

L: Ja så är det, splay trädet kan bli hur skeva som helst.

b) I en hashtabell tar det alltid konstant tid att slå upp värdet till en nyckel.

L: Nej så är det inte. Med en dålig hashfunktion kan det ta hur lång tid som helst eller i varje fall O(n) i värsta fall.

c) Antag att `o1.hashCode()` returnerar samma värde som `o2,hashCode()`. Kommer då `o1.equals(o2)` alltid att returnera true?

L: Nej så behöver det inte vara, olika element kan ge samma hashnyckel, det är ett av problemen man måste lösa med hashtabeller.

Problem 2. *Testar: Träd*

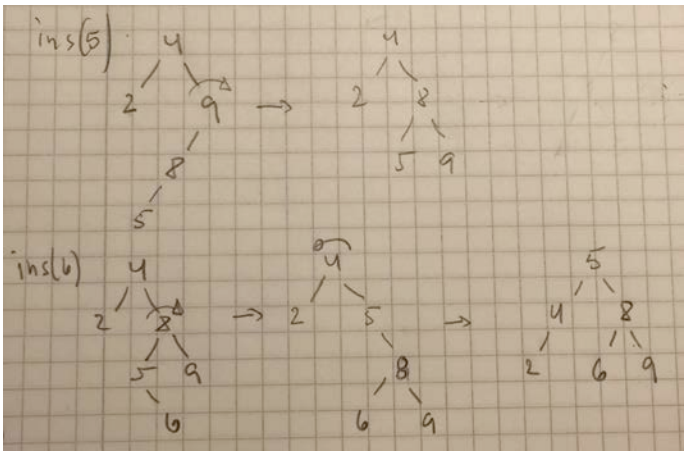
Beskriv kortfattat egenskaperna ...

a) binärt sökträd, AVL-träd, splay-träd, B-träd, samt partiellt ordnat+vänstebalanserat träd (dvs en heap).

Lösning:

BST: roten är större än alla noder i vänster delträd och mindre än alla noder i höger delträd dvs det är ordnat Ej balanserat.
AVL: som BST men också höjd-balanserat.
Splay: som BST men när ett element sätts in i trädet så roteras trädet så det elementet kommer i roten. Ingen balans.
B: som BST men man kan ha mer än ett värde i noden och mer än 2 barn. Ett 2-3 träd som är ett specialfall har tex 1 eller 2 värden i noderna och 0, 2 eller 3 barn. Balanserat.
POV: nodens värde är mindre än dess barns värden och trädet är fullt utom på sista nivån från höger dvs partiellt ordnat. Balanserat.

b) Sätt in talen 4, 9, 2, 8, 5, 6 i given ordning i ett AVL-träd.. ...



Lösning:
Efter insättning av 5:

har nod 4 balans 3-1=+2 och nod 9 har -2=> rotera höger kring 9 (balanseringen sker nerifrån och upp i trädet)

Efter insättning av 6:

har nod 4 balans 3-1=+2 och nod 8 har -1 dvs trädet är höger-vänster tungt => rotera höger runt 8, sedan vänster runt 4.

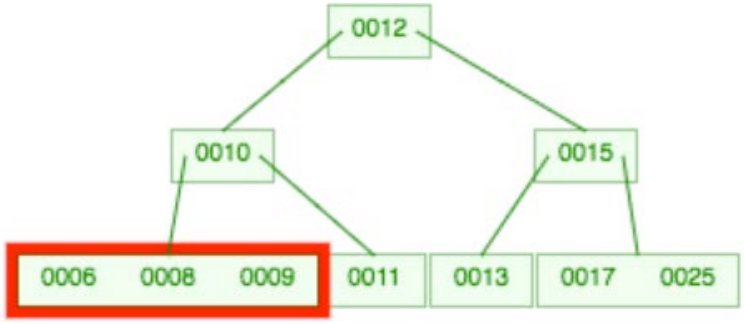
Motiveringen måste vara med dvs du måste säga något om hur du gör valen av rotation.

Problem 3. *Testar: 2-3 träd*

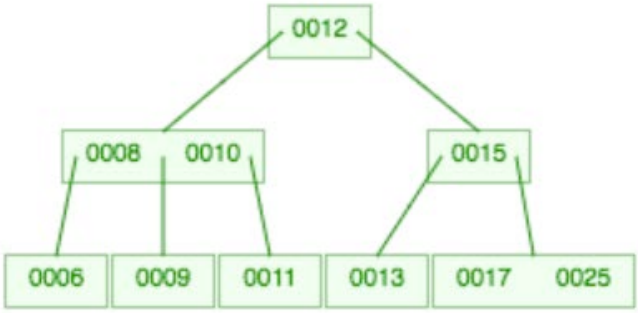
Låt T vara (2, 3) trädet nedan i vilket vi har heltalsnycklar. ...

Lösning:

insert(8):
steg 1: sätt in i ett löv med BST algoritm.

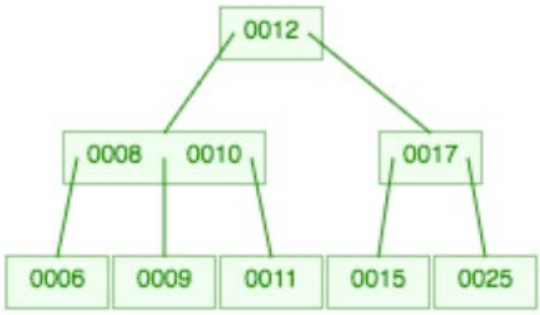


Steg 2: eftersom det blir 3 element i lövet måste vi göra en split, dvs flytta upp 8an (mittenvärdet) till föräldernoden och dela på lövet. Detta görs vid behov rekursivt uppåt i trädet:



remove(13):

hitta noden med 13, tag bort den, 15 har då ett barn med 2 element gör en merge till 15,17,25 och sedan en split med mittennoden som förälder eftersom 3 data inte är ok (och dessutom skall alla löv ligga på samma nivå)



Man kan också säga =>dela noden till 2 barn => ej BST=> vänster rotera runt 15:

Problem 4. *Testar: sortering*

Varför är det viktigt hur man väljer pivotelement i quicksortalgoritmen? ...

Lösning: Quicksortalgoritmen kan vara $O(n^2)$ i värsta fall om man väljer pivotelementet olyckligt tex första eller sista elementet. T.ex. ger en sekvens sorterad i stigande ordning att när vi gör en partition så kommer uppdelningen med element som är mindre än pivot-elementet alltid att bestå av ett element.

Det är alltså kombinationen av speciellt utseende på indata och val av pivotelement som ställer till det.

Bästa sättet att välja pivotelement är att välja medianen men för att hitta den måste man sortera. Om man istället väljer pivot elementet slumpmässigt så har vi en god chans att inte hela tiden välja ett dåligt pivotelement. Förväntad komplexitet är då O(nlog n).

Problem 5. *Testar: Grafer*

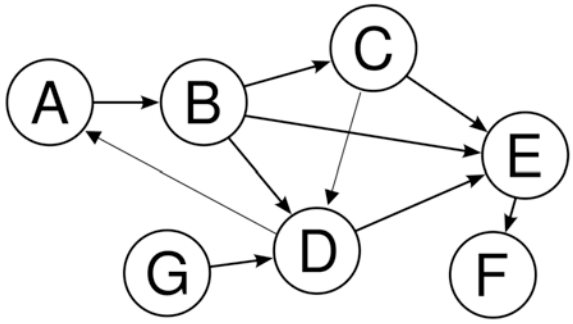
a) Beskriv (dvs rita) hur grafen nedan representeras som en grannlista.

b) Som en grannmatris.

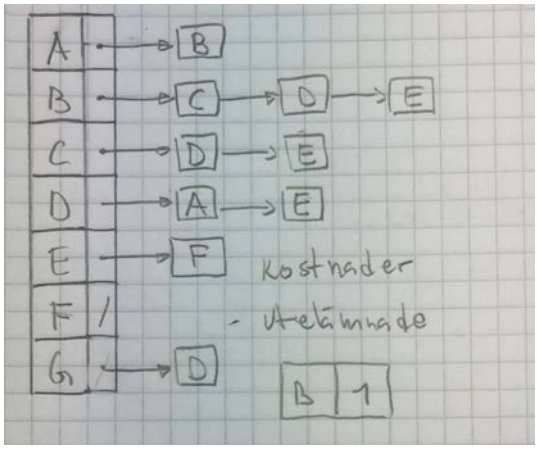
c) Ge en kort (steg 1) beskrivning av djupet först i grafer samt beskriv i vilken ordning noderna i grafen besöks om vi gör en djupet-först genomgång av grafen vid start i nod A. Välj i bokstavsordning om ordningen inte ges av algoritmen tex vid val av en av två möjliga grannar.

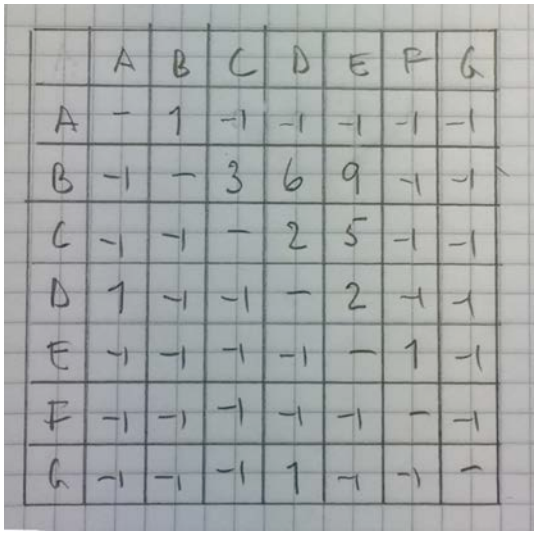
d) Ge en kort beskrivning av Dijkstras algoritm. (kort = steg 1; ide och steg 2; pseoudokod). Applicera sedan Dijkstras algoritm på grafen nedan med start i nod B. Gör en tabell över algoritmens fortskridande och skriv ner vad som händer.

Avstånd: (A,B,1), (B,C,3), (B,D,6), (B,E,9), (C,D,2), (C,E,5), (D,A,1), (D,E,2), (E,F,1), (G,D,1)



Lösning: a+b)



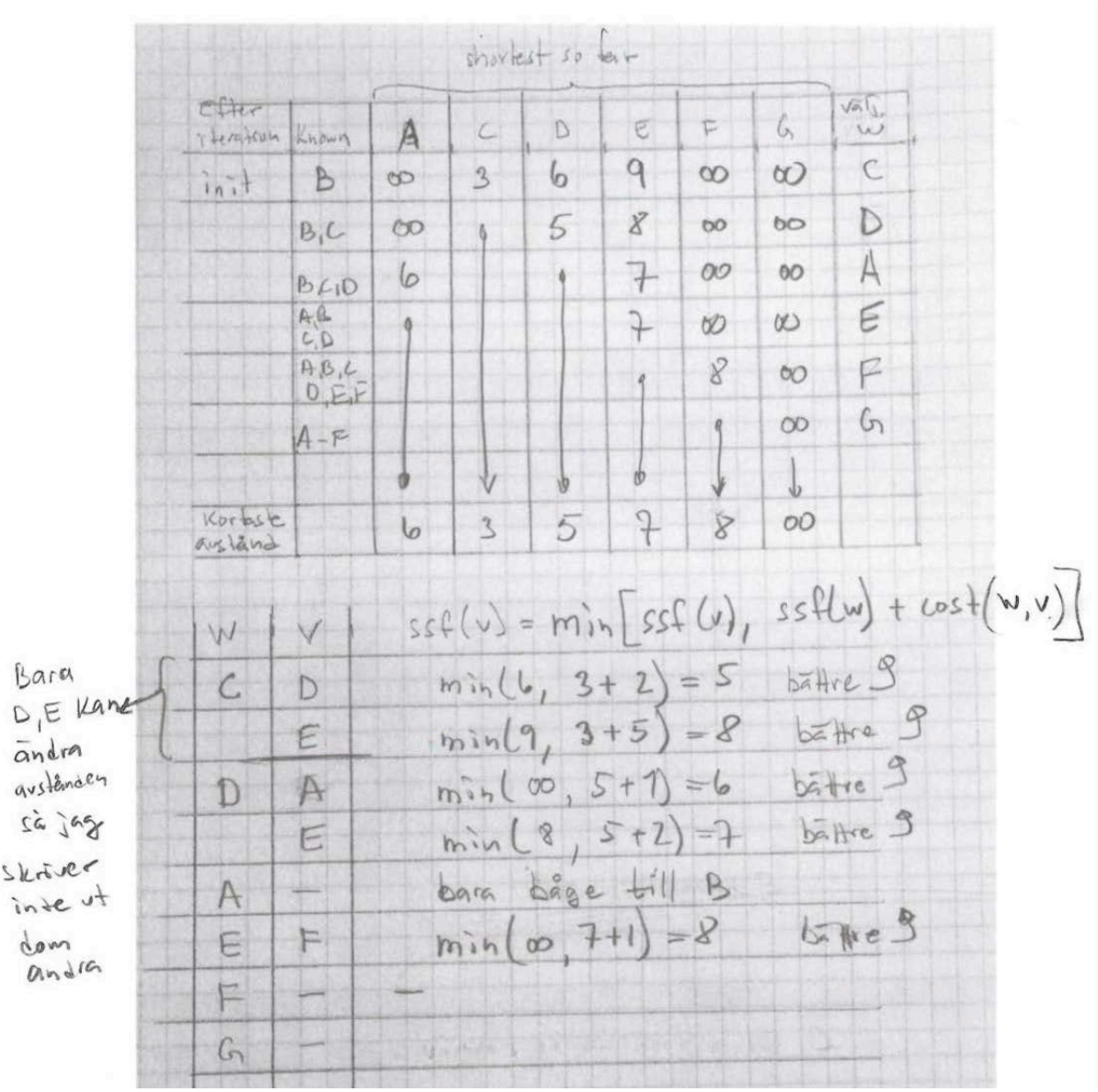


	A	B	C	D	E	F	G
A	-	1	-1	-1	-1	-1	-1
B	-1	-	3	6	9	-1	-1
C	-1	-1	-	2	5	-1	-1
D	1	-1	-1	-	2	-1	-1
E	-1	-1	-1	-1	-	1	-1
F	-1	-1	-1	-1	-1	-	-1
G	-1	-1	-1	1	-1	-1	-

c) För dfs se bok/OH bilder

dfs(A) = A, B, C, D, E, F, G

d) För beskrivning av Dijkstra se Bok/OH bilder



Problem 6. ...implementera en samling som en dubbellänkad cirkulär struktur....

Angående iteratorn:

I metoden size kan man använda iteratorn om man vill

I add har man ingen nytta av den för man har direkt access till slutet.

I remove fungerar inte iteratorn. Iteratorn ger ju innehållet i en nod och vi behöver en pekare till noden för att kunna ta bort den.

Kod se nästa sida

```
public int size() {                          a)
  if(last==null) {
    return 0;
  } else {
    Node p = last.next;
    int size = 1;
    while(p!=last) {
      size++;
      p = p.next;
    }
    return size;
  }
}
public boolean add(E elem) {                 b)
  if(last==null) { // tom lista
    last = new Node(elem);
    last.next = last.prev = last;
  } else {
    last = last.next = new Node(last.next, last, elem);
    last.next.prev = last;
  }
  return true;
}
public boolean remove(Object o) {            c)
  if(last==null) { return true;}
  Node p = last.next; // start of list
  while (!p.elem.equals(o) // while not found
          && p != last) { // and not end of list
    p = p.next;
  } // now o is found or we are at end of list
  if(p.elem.equals(o)) {
    removeThisNode(p);
    return true;
  }
  return false;
} // end remove

private void removeThisNode(Node p) {
  if( p==p.next) { // only one element
    last = null;
  } else {
    p.next.prev = p.prev;
    p.prev.next = p.next;
    if(p==last) {
      last = p.prev;
    }
  }
} // end removeThisNode
```

d) Kompl. O(n), O(1), O(n)

Problem 7. Angående iteratorn: I toString är det tveksamt om man bör använda iteratorn. Man skall ju skriva en ”effektiv metod” och vi vet inte hur iteratorn är implementerad. Den kan vara korrekt (och effektiv) eller den kan tex använda sig av get (och vara ineffektiv). I equals är kanske det mest korrekta att använda iteratorn (här fanns inget speciellt krav på effektivitet) men det fungerar också med andra sätt, se lösningen.

```
public String toString() {
  StringBuilder sb = new StringBuilder();
  sb.append("[ ");
  preOrderTraverse(root, sb);
  sb.append("]");
  return sb.toString();
}
private void preOrderTraverse(Entry entry, StringBuilder sb){
  if (entry != null){
    sb.append(entry.element.toString() + " ");
    preOrderTraverse(entry.left, sb);
    preOrderTraverse(entry.right, sb);
  }
} // ========= ========= ========= =========
public boolean equals(Object rhs) {
  if (this == rhs) return true; // first test only for speed
    // same class?
    if( rhs == null || getClass() != rhs.getClass()){
      return false;
  } else {   // compare parts
    BinarySearchTree he = (BinarySearchTree)rhs;
    if(this.size() != he.size()) {return false;}
    // lösnings alternativ 1 med iteratorn, helt oberoende av
    // toString och det är kanske det bästa alternativet
    // formatet från toString kan ju ändras
    Iterator itme = this.iterator();
    Iterator ithe = he.iterator();
    while(itme.hasNext()) {
      if ( itme.next() != ithe.next() ) {
        return false;
      }
    }
    return true;
    // lösnings alternativ 2 med preOrderTraverse
    StringBuilder sb1 = new StringBuilder();
    preOrderTraverseSimple(this.root, sb1);
    StringBuilder sb2 = new StringBuilder();
    preOrderTraverseSimple(he.root, sb2);
    return sb1.toString().equals(sb2.toString());
    // lösnings alternativ 3 med toString från ovan
    return this.toString().equals(he.toString());
  }
}
```