Algoritmer og datastrukturer
Løsningsforslag

Eksamen – 24. februar 2010

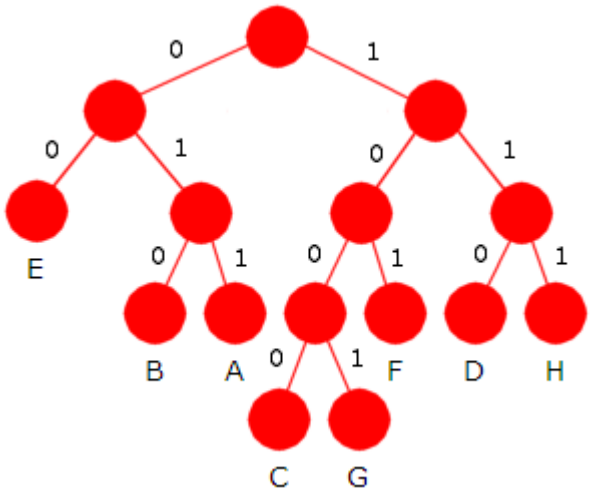
Oppgave 1A

1. Komparatoren sammenligner først lengdene til de to strengene. Variablen *k* vil dermed få verdien −1 siden "Per" er «kortere» enn "Kari".

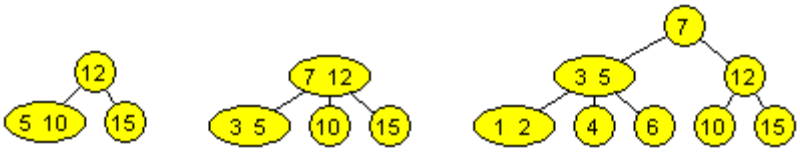
2. Strengene sammenlignes først med hensyn på lengde (kortest er minst) og alfabetisk hvis de har samme lengde. Dermed blir utskriften slik:

Ali Liv Per Kari Nils Jasmin

Oppgave 1B



Oppgave 1C



Oppgave 1D

Det er gitt at parametertabellen *c* kun inneholder bokstavene 'U', 'L' eller 'F'. Oppgaven kan løses på mange forskjellige måter. Her er noen forslag:

1. Gå gjennom tabellen og tell opp hvor mange det er av hver av de tre bokstavene. Skriv så ut i tabellen *c* så mange 'U'-er som antallet sier, deretter så mange 'L'-er som antallet sier og til slutt 'F'-ene.

2. Sorter tabellen ved hjelp av en ferdig sorteringsmetode (da kommer bokstavene i motsatt rekkefølge av det vi ønsker) og snu deretter tabellen. Dette vil fungere, men er egentlig ineffektivt:

```
public static void bytt(char[] a, int i, int j)
{
  char temp = a[i]; a[i] = a[j]; a[j] = temp;
}

public static void snu(char[] c)
{
  for (int v = 0, h = c.length -1; v < h; v++, h--)
    bytt(c, v, h);
}

public static void omorganiser(char[] c)
{
  Arrays.sort(c);
  snu(c);
}
```

3. Det finnes en effektiv måte å løse dette på. Det kalles tredelt partisjonering:

```
public static void omorganiser(char[] c)
{
  int v = 0, h = c.length-1, k = h;  // v forrest, h og k bakerst

  while (v <= h)
  {
    if (c[h] == 'U') Tabell.bytt(c,v++,h);
    else if (c[h] == 'L') h--;
    else  Tabell.bytt(c,h--,k--);
  }
}
```

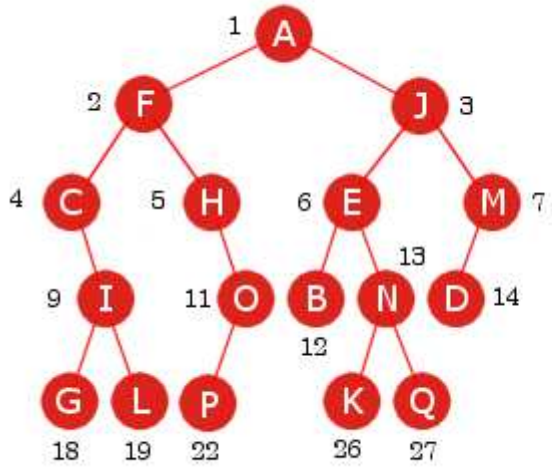
Oppgave 2A

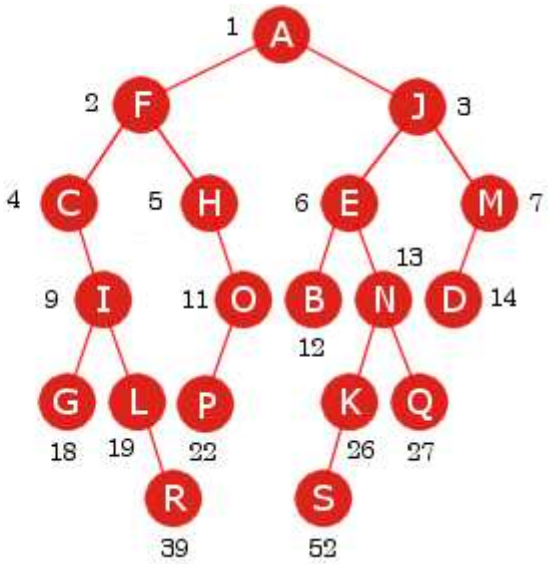
```
public void leggInnFørst(T verdi)
{
  hode = new Node<T>(verdi,hode);
  antall++;
}

public T taUtFørst()
{
  if (antall == 0) throw new NoSuchElementException();

  T temp = hode.verdi;
  hode = hode.neste;
  hode.verdi = null;
  antall--;
  return temp;
}
```

Oppgave 2B

```
public void leggInnSist(T verdi)
{
  if (antall == 0) hode = new Node<T>(verdi,null);
  else {
    Node<T> p = hode;
    while (p.neste != null) p = p.neste;
    p.neste = new Node<T>(verdi,null);
  }
  antall++;
}

public T taUtSist()
{
  if (antall == 0) throw new NoSuchElementException();
  else if (antall == 1)
  {
    T temp = hode.verdi;
    hode = null;
    antall--;
    return temp;
  }
  else
  {
    Node<T> p = hode;
    while (p.neste.neste != null) p = p.neste;
    T temp = p.neste.verdi;
    p.neste = null;
    antall--;
    return temp;
  }
}
```

Oppgave 3A



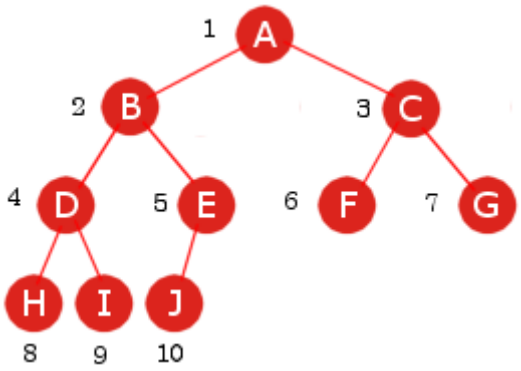


Inorden: C G I L R F H P O A B E S K N Q J D M

Nivåorden: A F J C H E M I O B N D G L P K Q R S

Oppgave 3B

3B, punkt 1.



3B, punkt 2.

I metoden *komplett* lages først et tomt tre. Så legges det inn verdier med posisjoner slik at treet blir komplett. Til slutt returneres det ferdige treet:

```
public static <T> BinTre<T> komplett(T[] a)
{
  BinTre<T> tre = new BinTre<T>();

  for (int i = 1; i <= a.length; i++)
    tre.leggInn(a[i - 1], i);

  return tre;
}
```

En rekursiv teknikk vil gi en mer effektiv metode:

```
private static <T> Node<T> komplett(T[] a, int k)
{
  if (k > a.length) return null;
  return
    new Node<T>(komplett(a,2*k), komplett(a,2*k+1), a[k-1],k);
}

public static <T> BinTre<T> komplett(T[] a)
{
  BinTre<T> tre = new BinTre<T>();
  tre.rot = komplett(a,1);
  tre.antall = a.length;
  return tre;
}
```

Oppgave 3C

Først en hjelpemetode som bytter om barna til en node:

```
private static <T> void byttBarn(Node<T> p)  // hjelpemetode
{
  Node<T> q = p.venstre; p.venstre = p.høyre; p.høyre = q;
}
```

Dette kan løses f.eks. ved hjelp av to rekursive metoder. Den første speilvender treet og den andre går gjennom treet og setter korrekte posisjoner:

```
// rekursiv hjelpemetode
private static <T> void speilvend(Node<T> p)
{
  if (p == null) return;
  byttBarn(p);
  speilvend(p.venstre);
  speilvend(p.høyre);
}

// rekursiv hjelpemetode
private static <T> void settPosisjoner(Node<T> p, int pos)
{
  if (p == null) return;

  p.posisjon = pos;
  settPosisjoner(p.venstre, 2*pos);
  settPosisjoner(p.høyre, 2*pos + 1);
}

public void speilvend()
{
  if (antall > 1)
  {
    speilvend(rot);
    settPosisjoner(rot, 1);
  }
}
```

Det blir imidlertid en bedre løsning hvis en speilvender og setter korrekte posisjoner i samme metode. Da traverseres treet bare én gang:

```
// rekursiv hjelpemetode
private static <T> void speilvend(Node<T> p, int pos)
{
  if (p == null) return;

  p.posisjon = pos;

  byttBarn(p);

  speilvend(p.venstre, 2*pos);
  speilvend(p.høyre, 2*pos + 1);
}

public void speilvend()
{
  if (antall > 1) speilvend(rot,1);
}
```

Oppgave 3D

Hvis noden *p* har et ikke-tomt høyre subtre, så er den neste i inorden den noden som ligger lengst ned til venstre i dette subtreet. Hvis ikke, må vi oppover i treet. Da finner vi først

hvilken posisjon den neste må ha. Deretter kan vi starte i roten og lete oss ned til den posisjonen:

```
private Node<T> nesteInorden(Node<T> p)
{
  if (p.høyre != null)
  {
    p = p.høyre;
    while (p.venstre != null) p = p.venstre;
  }
  else
  {
    int posisjon = p.posisjon;

    while (posisjon % 2 != 0) posisjon /= 2;

    if (posisjon == 0) return null;

    posisjon /= 2;  // posisjonen til neste i inorden

    String s = Integer.toBinaryString(posisjon);

    p = rot;

    for (int i = 1; i < s.length(); i++)
      p = s.charAt(i) == '0' ? p.venstre : p.høyre;
  }

  return p;
}
```