Eksamen i

Algoritmer og Datastrukturer

IAI 20102

Høgskolen i Østfold
Avdeling for informatikk
og automatisering

Lødag 5. juni 2004, kl. 09.00 - 13.00

LØSNINGSFORSLAG

Del 1 – 60%

Oppgave 1.1 - 10%

Forklar kort (helst ved hjelp av passende figurer) hvordan innsettingssortering virker.

Konseptuelt har man to mengder, en som er usortert,
Og en som er sortert. Man starter med tom sortert mengde, og i hver iterasjon tar man ett vilkårlig element fra den usorterte mengden og setter inn på rett plass i den sorterte.

Angi algoritmens orden med O-notasjon og gi en kort vurdering av metodens brukbarhet.

Verste tilfelle kvadratisk. Hvis input er tilnærmet ferdig sortert, vil arbeidsmengden bli tilnærmet lineær. Generelt dårlig ved store datamengder. Bra ved "nesten" sortert input, eller ved sortering av små mengder pga enkel implementasjon og ikke overhead i form av hjelpestruturer etc.

Oppgave 1.2 - 5%

Vis hvert steg i algoritmen (dvs. hvordan tabellen ser ut i hver iterasjon) brukt på sekvensen

```
45 - 12 – 8 – 13 – 97 – 83  |->  45
     12 – 8 – 13 – 97 – 83  |->  45
          8 – 13 – 97 – 83  |->  12 – 45
              13 – 97 – 83  |->  8 - 12 – 45
                   97 – 83  |->  8 – 12 - 13 – 45
                        83  |->  8 – 12 - 13 – 45 – 97
                        -   |->  8 – 12 - 13 – 45 – 83 -
97
```

Oppgave 1.3 - 5%

Implementerer metoden **`Del1.skriv`**`(Node liste)` *som skriver ut verdiene av tallene i en enkeltlenket liste som starter med noden* `liste`*.*

```
    void skriv(Node liste) {
        if (liste == null) return;

        Node n = liste;
        while (n != null) {
            System.out.println(n.tall+" ");
            n = n.neste;
        }
        System.out.println();
    }
```

Oppgave 1.4 - 5%

Implementerer metoden **`Del1.tilfeldigListe`**`(int n)` *som genererer en enkeltlenket liste med* `n` *noder med tilfeldige tallverdier mellom 0 og 1 (du kan gjerne bruke* `Math.random()` *som returnerer en tilfeldig double verdi mellom 0 og 1).*

```
    Node tilfeldigListe(int n) {
        if (n == 0) return null;

        Node liste = new Node(Math.random());
        Node node = liste;
        for (int i = 1; i < n; i++) {
            node.neste = new Node(Math.random());
            node = node.neste;
        }
        return liste;
    }
```

Oppgave 1.5 - 10%

Implementer metoden **`Del1.settInn`**`(Node sortert, Node inn)` *som plasserer noden* `inn` *på rett plass i den lenkede listen som starter med noden* `sortert`*, og som vi antar er korrekt sortert på stigende verdi av tall. Metoden returnerer en referanse til den første noden i den nye listen (som da også vil være sortert). Du kan anta at alle verdier er distinkte (ingen duplikater).*

```
    Node settInn(Node sortert, Node inn) {
        if (sortert == null || inn == null) return sortert;

        if (inn.tall < sortert.tall) {
            inn.neste = sortert;
            return inn;
        }

        Node n = sortert;
        while (n.neste != null) {
            if (n.tall < inn.tall && n.neste.tall > inn.tall) {
                inn.neste = n.neste;
                n.neste = inn;
                return sortert;
            }
            n = n.neste;
        }

        inn.neste = null;
        n.neste = inn;

        return sortert;
    }
```

Oppgave 1.6 - 10%

Implementer den rekursive metoden **`Del1. sorterRekursivt`** `(Node usortert, Node sortert),` *som blir kalt i driverrutinen* **`Del1.sorter`**`(Node liste).` *Driverrutinen skal returnere en referanse til den første noden i en sortert versjon av listen (med stigende tallverdi).*

```
Node sorterRekursivt (Node usortert, Node sortert) {
    if (usortert == null) return sortert;

    Node nesteUsortert = usortert.neste;
    sortert = settInn(sortert , usortert);

    return sorterRekursivt (nesteUsortert, sortert);
}
```

Oppgave 1.7 - 15%

En medstudent forteller deg at han har funnet en måte som han tror kanskje kan gjøre innsettingssortering mer effektiv. Når man skal sette inn et element i den ferdig sorterte delen, mener han det er mulig å bruke prinsippet for binærsøk.

Diskuter kort denne ideen, både anvendt på innsettingssortering av tabeller (arrays) og referansestrukturer. Angi kompleksiteten i O-notasjonen i begge tilfellene.

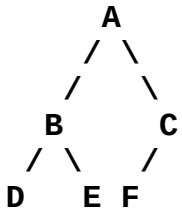
Tabeller: Å finne rett plass blir en logaritmisk operasjon…men: å flytte elementene krever i verste tilfelle 1 + 2 + 3 + … flyttinger … altså den samme arbeidsmengden som i originalutgave.

Lister: Binærsøk på en lenket liste er i utganspunktet en dårlig ide…men hva om vi bruker et binært søketre…som jo er å anvende prinsippet om binærsøk…så blir oppgaven å sette inn alle elementene inn i et binært søketre…og det vet vi er en logaritmisk operasjon, inkludert en eventuell balanseringsoperasjon. Denne metode blir dermed loglineær, og bedre enn vanlig innsettingssortering.

Slutt på del 1

Del 2 - 40%

Oppgave 2.1 - 5%



Skriv ut rekkefølgen på nodene slik de blir behandlet i en postorder traversing.

D – E – B – F – C – A

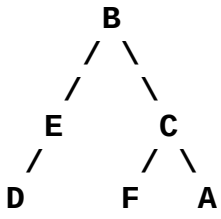
Oppgave 2.2 - 10%

Ved en inorder traversering av et tre får vi følgende rekkefølge:

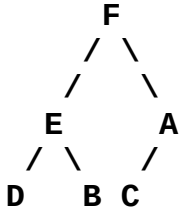
D – E – B – F – C - A

Tegn et binært tre som kan generere denne sekvensen.

Flere alternativer, f.eks:



Eller f.eks:



Oppgave 2.3 - 10%

Høyden i et tre er definert som den største av avstandene fra rota til bladnodene. Høyden til et tre som består av kun rota er 0. Et komplett binærtre er kjennetegnet ved at for enhver node i treet, så gjelder det at enten har nodens venstre og høyre subtre lik høyde, eller så er det venstre subtreet ett nivå høyere enn det høyre subtreet.

Implementer metoden **`Del2.komplett`**`(TreNode rot)` *som sjekker om treet representert ved noden rot er komplett eller ikke.*

OBS: Det finnes flere formelle definisjoner av et komplett tre. Denne definisjonen avviker noe fra den som er brukt i eksemplene i forelesningene, ved at den tillater at det nederste nivået ikke er er helt fyllt opp fra venstre, dvs. at nodene på dette nivået kan forekomme mer spredt (men ikke vilkårlig spredt!).

Her lønner det seg å ha en separat rutine for utregning av høyden i et tre (husk at høyen er lik antall nivåer minus en):

```
int hoyde(Node rot) {
  // Vi benytter oss av at høyden i et tre er lik
  // nivået vi er på (rota) + antall nivåer i subtreet
(venstre/høyre)
  // med størst høyde

  // Vi må sørge for at bladnodene får høyde lik 0.
  if (rot == null) return -1;

  // Finner høyde i venstre/høyre subtre rekursivt
  int v = høyde(rot.venstre);
  int h = høyde(rot.hoyre);

  //Den totale høyden er da lik 1 pluss høyden til det største
subtreet
  if (v>h) {
    return v+1;
  }
  else {
    return h+1;
  }

}


   boolean komplett (TreNode rot) {
        if (rot == null) return true;

        int v = hoyde(rot.venstre);
        int h = hoyde(rot.hoyre);

        if (h > v || v – h > 1) return false;

        return komplett(rot.venstre) && komplett(rot.hoyre);
    }
```

Oppgave 2.4 - 15%

I et komplett binærtre kan nodene indekseres ifølge levelorder traversering, slik at for enhver node med indeks *i*, vil det venstre barnet ha indeks *2*i +1*, og det høyre *2*i +2*. Dermed kan et slikt tre representeres i en tabell med referanser til (eller verdier av) innholdet i nodene med korresponderende indeks. Det første elementet i en slik tabell (indeks = 0) representerer altså rota i dette treet.

Implementer metoden **`Del2.skrivUtPreOrder`**`(char[] tabell, int index)` *som skriver ut subtreet med rot som korresponderer med* `index` *(i preorder rekkefølge).*

```
// Oppgaven spesifiserer ikke om tabellens lengde tilsvarer et
// komplett tre som er fyllt helt opp (perfekt binærtre).
// I dette tilfelle må da de ”tomme” plassene i tabellen være fyllt
// med et tegn som angir dette. Dette gjelder også hvis man antar at
// treet er bygget etter definisjonen gitt i forrige oppgave.

// Hvis man ikke har tenkt på denne detaljen, vil metoder som ikke
// sjekker på ”tomme” noder bli gitt full uttelling (for eksempel
// noe liknende det som er foreslått her.

void skrivUtPreOrder(char[] tabell, int index) {
      // Skriver ut innholdet av subtreet
       // med noden som korresponderer med den gitt indeksen i treet
       // som rot.

      if (index >= tabell.length) return;

     // Hvis vi tar utangspunkt i definisjonen gitt i forrige oppgave,
     // kan

      System.out.println(tabell[index]);
      skrivUtPreOrder(tabell, (2*index)+1);
      skrivUtPreOrder(tabell, (2*index)+2);
  }
```

Slutt på del 2