Obligatorisk oppgave 1 i INF 3/4130, høsten 2007

Leveringsfrist er fredag 5. oktober

NB: Les dette først:

Krav til innleverte oppgaver ved Institutt for informatikk

Ved alle pålagte innleveringer av oppgaver ved Ifi - enten det dreier seg om obligatoriske oppgaver, hjemmeeksamen eller annet - forventes det at arbeidet er et resultat av studentens egen innsats. Å utgi andres arbeid for sitt eget er uetisk og kan medføre sterke reaksjoner fra Ifis side. Derfor gjelder følgende: Hvis du tar med tekst, programkode, illustrasjoner og annet som andre har laget, må du tydelig merke det og angi hvor det kommer fra. Det er greit å få hint om hvorledes en oppgave kan løses, men dette skal eventuelt brukes som grunnlag for egen løsning og ikke kopieres uendret inn. Kursledelsen kan innkalle studenter til samtale om deres innlevering.

Gruppearbeid

I noen kurs skal det leveres gruppearbeid. Ifi krever da at alle medlemmer av gruppen kan gjøre rede for hovedtrekkene i det innleverte arbeidet. Dessuten må alle ha utført en rimelig del av det hele, og kunne identifisere og svare i detalj for sin del.

Samarbeid

Reglene om kopiering betyr ikke at Ifi fraråder samarbeid - tvert imot, Ifi oppfordrer studentene til å utveksle faglige erfaringer om det meste. Men det kreves som nevnt at man kan stå inne for det som leveres. Hvis du er i tvil om hva som er lovlig samarbeid, kan du kontakte gruppelærer eller faglærer.

Institutt for informatikk, 27. jan. 2004

Reglementet over er hentet fra: www.ifi.uio.no/studinf/skjemaer/erklaring.pdf

Spesielt gjelder for denne obligatoriske oppgaven:

- Hver student skal levere en egen løsning på oppgavene
- Oppgavene består av tre deler, som alle skal løses
- Det kan komme ytterligere presiseringer om oppgaven eller levering, så følg med!
- Programmet skal i utgangspunktet skrives i Java eller Python uten bruk av programvarebibliotek som ikke er standard. Skriv mest mulig selvforklarende kode, fyldige kommentarer (på "algoritmenivå"), og bruk helst variabelnavn osv. som er konvensjon fra oppgaven eller litteraturen om algoritmen. Om en ønsker å bruke andre språk og/eller bibliotek som ikke er standard kan en spørre gruppelærer om lov.

Oppgave 1 (Optimale søketrær og memoisering)

Lag en implementasjon av "Optimal Search Tree" som bygger på "memoisering". Den skal altså programmeres som en rekursiv metode som først tester om kallet med de samme parameterene er gjort før, og svaret skal i så fall være satt ned på riktig sted i en passelig array.

Angi (med *O*-notasjon) kompleksitet for kjøretid og minnebruk. Gi begrunnelse (ikke nødvendigvis veldig formell).

Input

Programmet skal lese input fra en angitt fil (denne angis til programmet ved oppstart, sammen med navnet for output-fil). Dataene er som følger (en variabel/array per linje, mellomrom som skilletegn i arrayer):

- n – antall noder i treet
- $V[1:n]$ – verdier av nodene i treet som heltall i sortert rekkefølge
- $P[1:n]$ – frekvenser som hver av nodene i treet vil bli aksessert med
- $Q[0:n]$ – frekvenser som mellomrommene mellom nodene vil bli aksessert med

Variabelnavnene over skal også brukes som variabelnavn i programmet, slik at de raskt kan identifiseres (men hvilken indeksering du bruker på arrayene er opp til deg). Eksempel på input:

```
4
0 1 2 3
15 10 20 30
5 5 0 5 10
```

NB: De gitte frekvensene er reelle tall, men er altså ikke nødvendigvis gitt slik at de summerer seg til 1.0. De må altså deles på summen av *P*- og *Q*-verdiene for å bli vanlige sannsynligheter.

Output

Programmet skal skrive til filen angitt i andre argument til programmet. Output skal være en linje der alle nodene skrives ut i stigende rekkefølge (infiks-traversering) med (dybde, verdi). Eksempel, om svaret blir et balansert tre med syv noder, med verdiene 11 - 17:

```
(2,11) (1,12) (2,13) (0,14) (2,15) (1,16) (2,17)
```

Tips til implementasjon

For å lese og skrive til/fra fil kan det være greit å bruke henholdsvis `java.util.Scanner` og `java.io.PrintWriter`. Følgende eksempel leser ett tall fra input og skriver ut strengen "<tall> * 2 = <tallet ganger to>" til output på den måten vi vil ha det:

```
public static void main(String[] args) throws FileNotFoundException
{
   Scanner in = new Scanner(new File(args[0]));
   int input;
   try {
      input = in.nextInt();
      // se dokumentasjon til java.util.Scanner for flere eksempler
   } finally {
      in.close();
   }

   int output = input * 2;

   PrintWriter out = new PrintWriter(new File(args[1]));
   try {
      out.printf("%d * 2 = %d\n", input, output);
      // evt. out.println som flere er vant med
   } finally {
      out.close();
   }
}
```

For å slippe at den rekursive metoden må få med seg alle dataene som parametere i hvert kall, kan den kan passelig ligge inne i et objekt der inputdataene er attributter. Også den arrayen som husker svaret av tidligere kall bør ligge inni her. Dette objektet kan passelig settes opp av en hoved-metode, som også starter selve rekursjonen.

Det kan lønne seg å bygge opp selve søketreet, for greit å kunne skrive det ut til slutt på den angitte form.

Til uttesting kan man bruke de to eksemplene fra kap. 9 i læreboka. Verdiene i figur 9.6 (side 276) skulle gi treet nederst i figuren, skrevet ut: (1,0) (2,1) (0,2) (1,3). Disse dataene ligger på fila: `~inf3130/Oblig1/Opg1-test1.txt` Eksempelet i figur 9.7 skulle komme ut som akkurat det samme treet som i 9.6. Merk at indeksene i oppgaven og i boka er forskjellige, og at det er *mange* trykkfeil i matrisene i figur 9.7. Input-dataene til dette eksempelet ligger på `~inf3130/Oblig1/Opg1-test2.txt.`

Ellers kan man jo kjøre noen ekstrem-tilfeller, f.eks. bruke 7 noder med verdier 1, 2, …, 7, og med alle *P*-verdier like og alle *Q*-verdier lik null. Det skulle bli det fullt balanserte treet. Om man bruker 8 noder, og har *P* = 1000, 1, 1, 1, 1, 1, 1, 1 og alle *Q*-verdier lik null skulle man få: (0,1) (3,2) (2,3) (3,4) (1,5) (3,6) (2,7) (3,8). Disse input-dataene ligger på filene `~inf3130/Oblig1/Opg1-test3.txt` og `~inf3130/Oblig1/Opg1-test4.txt.`

Oppgave 2 (Dynamisk programmering)

Problemet "Lengste Økende Utplukk" (LØU) er som følger: Man har gitt en sekvens *T* av tall t_1, t_2, ..., t_N, og man skal gjøre et størst mulig utplukk av indeksene 1, ..., *N* slik at utplukket fra *T* med disse indeksene (sett som en subsekvens av *T*) danner en (ekte) stigende sekvens. Indeksene behøver altså ikke være *ett* intervall av tallene 1, …, *N*.

Oppgaven er å lage en algoritme som løser LØU med dynamisk programmering. Beskriv og begrunn først en rekursjons-formel som man kan bygge en slik løsning på, og programmer så en metode som løser problemet. Metoden får altså de *N* tallene som input og skal finne det lengste økende utplukk. Sporten er jo her å kunne finne fram løsningen fra ende til annen helt selv, og virkelig stor blir sports-prestasjonen om man da, med en liten vri på problemet, finner en løsning som bare behøver $O(N)$ plass. Løsningen skal ikke i noe tilfelle bruke mer enn $O(N^2)$ plass og $O(N^2)$ tid.

Input

Programmet skal lese input fra en angitt fil (denne angis til programmet ved oppstart, sammen med navnet for output-fil). Filen inneholder lengden av *T*, og deretter verdiene i *T* (alt heltall, med blanke mellom).
Eksempel (*T* av lengde 16):

```
16 10 1 11 3 12 4 5 11 14 15 6 7 11 5 12 17
```

Bruk *T* som variabelnavn i koden.

Output

Lengden av det lengste økende utplukk av *T* og deretter selve utplukket som en sekvens av (<indeks i *T*>,<verdi i *T*>) skal skrives til filen angitt i andre argument til programmet.
Eksempel (utplukket er 9 langt):

```
9 (2,1)(4,3)(6,4)(7,5)(11,6)(12,7)(13,11)(15,12)(16,17)
```

Tips til implementasjon

Se oppgave 1 angående I/O-håndtering i Java. Angående uttesting skulle det her være greit å sette opp tastdata selv, men datasettet i eksempelet over ligger på fila: `~inf3130/Oblig1/Opg2-test1.txt`.

Oppgave 3 (Trier)

Trier og suffix-trær er beskrevet i kapittel 20 i læreboka. Løs oppgave 20.23 og 20.24. Begge skal programmeres helt ut. Vi lar for enkelthets skyld strengene kun bestå av små bokstaver. Innsetningsprosedyren skal gi en feilmelding om den får en streng som er et prefiks av en streng den har fått før (men skal akseptere samme strengen flere ganger). Du kan selv velge hvordan du vil implementere aksess fra en node til dens subnoder. Vanligvis skal jo rett subnode finnes (eller skapes) ut fra neste tegn i inputstrengen, mens alle subtrærne skal aksesseres i sortert rekkefølge ved utskrift av trærne (se under). Vi er ikke voldsomt kritiske til bruk av tid og plass. MEN: Du skal angi hvilke fordeler/ulemper den valgte metoden har.

Input

Programmet skal lese input fra en angitt fil (denne angis til programmet ved oppstart, sammen med navnet for output-fil). Filen består av ett heltall *N* og en mengde strenger (ord) adskilt med linjeskift. De *N* første strengene skal settes inn i trien/suffix-treet, de resterende strengene skal brukes til oppslag i den ferdige datastrukturen.

Output

Output skal igjen skrives til filen angitt i andre argument til programmet. For hver av de *N* første strengene skal strengen skrives ut, deretter skal datastrukturen skrives ut slik den ser ut etter innsetting av denne strengen. Den skal skrives ut i prefiks format, med en parentes omkring hvert subtre. Subtrærne skal komme i sortert rekkefølge. Trien i figur 20.8 skal dermed komme ut slik:

```
(al(gorithm)(l))(inter(n(allay)(et))(view))(w(eb)(orld))
```

For de resterende strengene skal man også skrive ut selve strengen, samt om de ble funnet eller ikke når man gjør et oppslag i datastrukturen man har bygget opp med de *N* første strengene.

Tips til implementasjon

Se oppgave 1 angående I/O-håndtering i Java. Angående uttesting skulle det her være svært greit å sette opp testdata selv, men et datasett (det fra boka) ligger på fila `~inf3130/Oblig1/Opg3-test1.txt`.

Leveringsanvisning

Det skal kun leveres kildekode og ett enkelt PDF-dokument som besvarelse. Kommentarer til besvarelsen som du ønsker å gi kan du legge på starten av PDF dokumentet. Dersom du er klar over en svakhet i algoritmen som slår ut på et spesifikt tilfelle kan du også (hvis du vil, dvs. ønsker tilbakemelding om hvor feilen ligger) legge ved et datasett som demonstrerer dette og henvise til det i besvarelsen.

- Legg kildekoden (så mange filer du vil, men ikke spre koden over flere mapper) og et dokument kalt oblig1.pdf i en mappe kalt:
  `oblig1_<ditt brukernavn>`.

- Pakk mappa i en tar.gz-fil eller zip-fil med samme navn. For eksempel
  `tar cvzf oblig1_dagss.tar.gz oblig1_dagss`

- Send fila til: inf3130-1@ifi.uio.no.

Dersom du ønsker kan du eventuelt legge ved et plain text-dokument i stedet for PDF, kalt oblig1.txt (men det kan jo konverteres med f.eks. kommandoen a2ps eller a2pdf).

- - - o o o - - -