Obligatorisk oppgave 1 i INF 4130, høsten 2009

Leveringsfrist fredag 2. oktober

Institutt for informatikk

Krav til innleverte oppgaver ved Institutt for informatikk (Ifi)

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 både ulovlig og uetisk og kan medføre sterke reaksjoner fra Ifis og Universitetets side, for eksempel utvisning i ett eller flere semestre; se http://www.uio.no/admhb/reglhb/studier/andre-regelverk/fuskesaker.xml.

Vær derfor oppmerksom på 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.
- Du kan bli innkalt til samtale om dine innleveringer. Du må da kunne forklare innholdet i detalj og redegjøre for hvorledes det innleverte arbeidet er blitt til.

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
Disse kravene 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 kun leverer besvarelser man har produsert selv. Hvis du er i tvil om hva som er lovlig samarbeid, må du kontakte gruppelærer eller faglærer.

http://www.ifi.uio.no/studinf/skjemaer/erklaring.pdf 13. desember 2007

Spesielt gjelder for denne obligatoriske oppgaven:

- Oppgaven består av to deler, som begge skal løses. Hver student skal levere en egen, selvstendig løsning på oppgavene.
- Programmene skal skrives i Java, Python eller C/C++, og de skal kommenteres fyldig. Dog kan man snakke med gruppelærer, som kan tillate enkelte andre språk.
- En leveringsanvisning står til nederst i dokumentet. Før obligene rettes i detalj testes de automatisk mot en rekke testsett, det er derfor viktig at output er ***nøyaktig*** som angitt i oppgaveteksten.
- Det kan komme ytterligere presiseringer om oppgaven eller levering, så følg med!

Oppgave 1 (Dynamisk programmering)

Problemet "Sum Av Utplukk" (SAU) er som følger: Man er gitt en sekvens av n positive heltall $t_1, t_2, \ldots, t_n$ og et positivt heltall K, og man skal gjøre et utplukk av tallene som har sum nøyaktig K.

Eksempel: La n være 5, la tallene være 1, 5, 6, 3, 10 og K = 8. I denne probleminstansen kan vi plukke ut 5 og 3 som har summen 8. Er vi bare interessert i et JA/NEI-svar, kan vi på denne instansen av SAU altså svare JA.

Oppgaven er flerdelt for å illustrere dynamisk programmering, med og uten memoisering, slik at forskjellene på de to metodene kommer fram, og slik at vi ser hvordan begge metodene baserer seg på samme rekursive formel.

a) For å løse oppgaven med dynamisk programmering kan du bruke en boolsk tabell U med format $[1 \ldots n] \times [0 \ldots K]$. Bruk U som variabelnavn i koden. Angi hva du vil at verdier i denne matrisen skal bety, og angi og begrunn hvilken rekursiv formel du vil basere din dynamiske programmerings-algoritme på.

b) Implementer en standard bottom-up dynamisk programmerings-algoritme som løser SAU. Forklar hvorfor du initialiserer tabellen som du gjør.

c) Forklar hvordan man kan løse oppgave b) med bare å bruke $O(K)$ plass (altså at plassen man bruker er uavhengig av lengden av tall-sekvensen).

d) Skriv en rekursiv, memoisert rutine (top-down) som bruker formelen fra a) til å løse en instans av SAU. Tabellverket kan ha samme dimensjon som angitt i a) over, men du trenger nå tre verdier i hver rute (og du kan f.eks. bruke en INT til det). Forklar hvorfor, og hvordan tabellen må være initialisert når du starter algoritmen.

e) Forklar hvorfor det i oppgave d) ikke er like rett fram å redusere plassbehovet som det var i oppgave c). Men anta så at det i d) er ekstremt viktig å redusere plassbruken så mye som mulig og at det ikke er så kritisk med tiden. Hva slags datastruktur ville du da bruke i stedet for den to-dimensjonale tabellen? Forklar!

Input

Programmet skal lese input fra en angitt fil. Navnet på denne angis til programmet ved oppstart, sammen med navnet for en output-fil. Filen inneholder tallet n deretter tallet K, og til slutt t_i-ene (alt heltall, med blanke mellom).
Eksempelet over vil altså se slik ut:

```
5 8 1 5 6 3 10
```

Output

JA eller NEI avhengig av om et utplukk med sum *K* var mulig å gjøre, et mellomrom, *K*, et mellomrom og deretter selve utplukket som en sekvens av (<t_i>,<0/1 avhengig av om t_i er med i utvalget>) skal skrives til filen angitt i andre argument til programmet.
Eksempel (utplukket består av 5 og 3):

```
JA 8 (1,0)(5,1)(6,0)(3,1)(10,0)
```

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.

Angående uttesting skulle det her være greit å sette opp tastdata selv, men det vil bli lagt ut noen testdata på kurssiden.

Oppgave 2 (Trier og suffiks-trær)

Trier og suffix-trær er beskrevet i kapittel 20 i læreboka. Skriv et program som bygger et ukomprimert trie-tre fra en samling *C* av strenger, og som så søker i dette trie-treet for å se om et gitt pattern finnes i *C*. 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(ally)(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 det vil bli lagt ut et datasett på kurssiden.

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_kjetimh.tar.gz oblig1_kjetimh`

- Send fila til: inf4130-1@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).

[slutt]