Obligatorisk oppgave 1 i INF 3/4130, høsten 2005

Presisert utgave 3. oktober. Leveringsfrist: Mandag 17. oktober

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 fire deler, som alle skal løses

- Programmene skal skrives i Java, og de skal kommenteres fyldig.

- Det kommer en leveringsanvisning for denne obligen i god tid før levering, og kanskje også ytterligere presiseringer av oppgavene.

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.

Input til hovedmetoden (som ikke bør være den rekursive) skal være:

```
n      antall noder i treet
V[1:n] heltallsverdier i sortert rekkefølge, som skal inn i treet.
P[1:n] sansynlighetene for at verdiene i treet vil bli aksessert
Q[0:n] sansynligheten for at "mellomrommene" skal bli aksessert.
```

NB: Sansynlighetene er reele tall, men ikke nødvendigvis gitt slik at de summerer seg til 1.0. De er å regne som frekvenser (og må eventuelt deles på summen av P- og Q-verdiene for å bli ekte sansynligheter).

Hovedmetoden skal også bygge opp selve treet, og levere tilbake en peker til roten. Nodene i treet skal være standard søketre-noder uten foreldrepekere.

Anbefaling: 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 settes opp av hovedmetoden, som også starter selve rekursjonen.

Input/output etc.: Det skal lages et hovedprogram omkring dette som får inn to filnavn ved oppstart, og som fra den første fila leser inn verdiene den skal arbeide på i den rekkefølge de er angitt over (uten annet enn blanke mellom). På den andre fila skal den skrive ut det resulterende tre i innfiks rekkefølge med (dybde,verdi) for hver node (skrives greit ut med en rekursiv prosedyre). Altså, om resultatet er det fullt balanserte tre, og verdiene er 11, 12, 13, 14, 15, 16, 17, så skal utskriften bli:

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

NB: Eksrta-spørsmål som skal besvares: Vurder hvor mye/lite det kan være å spare på å bruke memoisering på dette problemet.

Oppgave 2 (Dynamisk programmering)

Problemet "lengste økende utplukk" (LØU) er som følger: Man har gitt en sekvens T av tall t1, t2, ..., tN, 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.

Oppgaven er å finne en algoritme som løser LØU med dynamisk programmering. Beskriv og begrunn en rekursjons-formel som man kan bygge en slik løsning på, og programmer en Java-metode som løser problemet. Input til metoden er en array med sekvensen T, og den skal levere lengden av 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/output etc.: Det skal lages et hovedprogram omkring dette som ved oppstart får inn et filnavn, og som derfra leser inn lengden av T, og deretter verdiene i T (alt heltall, med blanke mellom). Programmet skal skrive lengden av det lengste økende utplukk av T på skjermen (og gjerne også selve det økende utplukket (altså ikke indeksmengden), men det forlanger vi ikke).

Oppgave 3 (Splay-trær)

Lag en implementasjon av innsettings-operasjonen for splay-trær som bygger på å gå rekursivt ned i treet for å sette inn den nye noden. Om en node med den aktuelle verdien allerede finnes, skal ingen ny settes inn. Splay-opersjonen som bringer denne noden opp til roten skal så gjørs mens rekursjonen trekker seg tilbake mot roten (altså, splay-operasjonen gjøres "bottom up").

Det som skal programmeres er altså en variant av det som diskuteres først i annet avsnitt av kap. 12.1. i Weiss. Vi skal ikke ha foreldre-pekere, men om man ønsker kan man bruke en eksplisitt stakk som lagrer veien fra roten og ned til den aktuelle noden (og da kanskje også gjøre splay-operasjonen helt etter rekursjonen).

Verdiene i treet skal være heltall, og nodene i treet skal være standard søketre-noder, altså uten foreldrepekere. Input til metoden er verdien det skal søkes etter, mens peker til roten av treet ligger

f.eks. i en statisk variabel i nodeklassen. Metoden skal levere en boolsk verdi som angir om verdien var i treet fra før (false) eller ble satt inn nå (true).

Input/output etc.: Det skal lages et hovedprogram omkring dette som får inn to filnavn ved oppstart, og som fra den første fila leser sekvensen av heltallsverdier som skal settes inn (eller oppsøkes). På den andre fila skal programmet skrive ut treet på samme format som i oppgave 1 etter hver operasjon. Start utskriften fra hver operasjon på ny linje, og start linja med den verdien som skal settes inn.

Oppgave 4 (Trier og suffiks-trær)

Løs oppgave 20.23 og 20.24 i læreboka. De skal begge programmeres helt ut. 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). Strengene inneholder bare små bokstaver. Vi er ikke voldsomt kritiske til bruk av tid og plass. MEN: Du skal angi hvilke fordeler /ulemper den valgte metoden har.

Input/output etc.: Det skal lages et hovedprogram omkring dette som får inn to filnavn ved oppstart, og som fra den første fila leser et antall strenger som skal settes inn, og et antall strenger som det skal sjekkes om er i treet. Aller først i fila ligger et heltall N som angir hvor mange av strengene som skal settes inn. For de N første strengene skal man, på den andre fila, skrive ut selve strengen, samt hvordan trien ser ut etter innsettingen (se under). For de resterende strengene skal man også skrive ut selve strengen, samt om de ble funnet eller ikke.

Utskrift av en komprimert trie: 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))

Løs så oppgave 20.25. Her behøver du bare beskrive algoritmen, samt gjøre analysen. MEN: I tillegg skal du vurdere og beskrive minst en mulig optimalisering, i forhold til en algoritme som bygger direkte på 20.23 og 20.24. Du må gjerne slå opp i bøker etc., men du skal beskrive og begrunne optimaliseringen(e) med egne ord (men du behøver ikke bevise dem formelt)