OBLIGATORISK OPPGAVE 4 - INF110, høsten 2003

Prosjektplanlegging

Det tillates at man arbeider to og to sammen om denne oppgaven, men det er selvfølgelig også helt i orden å gjøre den alene. De som jobber sammen bør eventuelt være fra samme gruppe.

Oppgaven går ut på å lage et hjelpemiddel for planlegging av større prosjekter. Dette hjelpemiddelet, som skal være et programsystem, skal ta imot informasjon om de forskjellige deler som det planlagte prosjekt kan deles opp i. Hver slik del skal vi kalle en *aktivitet*, og den informasjon som er knyttet til hver slik aktivitet er følgende:

- Et tekstlig navn på denne aktiviteten, til bruk i utskriften.
- Et nummer på aktiviteten.
- Hvor stort mannskap som tenkes brukt på denne aktiviteten?
- Hvor lang tid aktiviteten er beregnet å ta.
- Hvilke aktiviteter som må være avsluttet før denne aktiviteten kan starte. (Dette oppgis som en sekvens av aktivitetsnumre)

Ut fra disse opplysningene skal så programmet finne svar på følgende spørsmål:

a) Er prosjektet gjennomførbart, eller finnes det aktiviteter som gjensidig vil komme til å vente på hverandre?

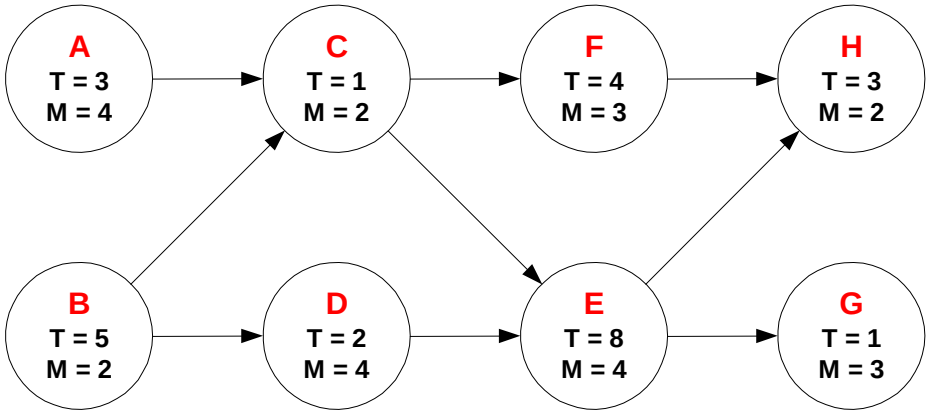
b) Om det kan gjennomføres: Hvordan kan prosjektet gjennomføres på kortest mulig tid om man har ubegrenset stort mannskap å ta av, og starter hver aktivitet så snart det er mulig? Hvor lang tid vil denne gjennomføringen ta? Hvor stort mannskap trengs til enhver tid med en slik gjennomføring.

c) Hvor kritisk er det at den enkelte aktivitet holder den oppgitte tid, når det er viktig at den totale tid for prosjektet ikke skal øke?

d) Dersom man bare har et gitt antall arbeidere til disposisjon, hvordan kan man da gjennomføre prosjektet på en fornuftigmåte? (Det er vanskelig her å finne den "beste" måte, men man har til oppgave å finne en måte som utnytter arbeidsstokken rimelig bra).

For å lette oversikten kan man tegne hver aktivitet som en sirkel med angivelse av tids og mannskapsbehov skrevet inni. Videre kan man markere at en aktivitet A ikke kan startes før en aktivitet B er ferdig, ved å tegne en pil fra B til A. Pilen går altså i tidsretningen. Vi har

dermed en rettet graf, der aktivitetene er noder og avhengighetene er kanter. Et eksempel på en graf som kan representere et lite prosjekt er gitt i figur 1 nedenfor.



Figur 1. Eksempel på aktivitetsgraf. T er tidsbehov og M er mannskapsbehov

Denne grafen må vi på en eller annen måte ha representert i maskinen når vi skal besvare spørsmålene over. Vi skal her bruke en nokså enkel representasjon, som bare bruker lister.

Angående spørsmål (a) over skulle det være greit å innse at prosjektet er gjennomførbart hvis og bare hvis det ikke er løkker (sykler) i grafen, og at det å gjennomføre prosjektet er å foreta noe som likner på topologisk sortering. Etter innlesningen av grafen skal det derfor sjekkes om grafen har løkker, og dette skal gjøres ved en standard "dybde først søking" gjennom grafen. Dersom det finnes slike løkker skal man skrive ut *en* av dem (som en sekvens av aktiviteter), sammen med en beskjed om at videre behandling er umulig.

Dersom grafen viser seg å ikke ha løkker, skal vi gå videre til spørsmål (b). Resultatet av dette skal presenteres som følger: Man skal for det første tenke seg at man gjennomfører prosjektet ved å starte opp hver aktivitet så snart som mulig etter at alle de aktivitetene den er avhengig av er ferdig. Hvordan denne "simuleringen" kan gjennomføres skal vi se på lenger ned.

Som en utskrift fra denne simuleringen skal man lage en kronologisk liste over alle "interessante tidspunkter", nemlig når en eller flere aktiviteter starter eller slutter. Mellom hvert interessant tidspunkt skal det skrives ut hvor stort mannskap som er i arbeid. Tiden fra starten fram til alle aktiviteter er ferdig er den korteste tiden prosjektet overhodet kan gjennomføres på. For prosjektet overfor kunne denne utskriften være f.eks. som i figur 2.

Beregningene i punkt (c) bygger så på følgende betraktninger. Ut fra det totale bildet av hvordan aktivitetene må vente på hverandre, kan man beregne hvor kritisk det er at den enkelte aktivitet overholder sin tidsplan, dersom prosjektet fremdeles skal kunne utføres på kortest mulig tid.

Dette kan f.eks. uttrykkes ved et *seneste starttidspunkt* for aktiviteten, slik at tidsrammen fremdeles kan holdes om *alle de andre* aktivitetene ikke bruker lenger tid enn angitt. Alternativt kan dette uttrykkes som en "slakk" for hver aktivitet, som er *differansen mellom seneste og tidligste starttidspunkt*, og det er denne slakken som skal skrives ut. Legg her merke til at de aktiviteter som har slakk lik null danner en slags "kritisk vei" gjennom grafen. Om en av disse tar lengre tid en forutsatt, så forsinkes hele prosjektet tilsvarende.

Seneste starttidspunkt ville man kunne finne ved å "gjennomføre prosjektet bakfra" etter samme mønster som i del (b) (ved da å tenke seg at hvert prosjekt blir avsluttet på det senest mulige tidspunktet før starten av alle de som er avhengig av den). Da ville man imidlertid være avhengig av også å lagre forgjengermengden i hver node, og bl.a. for å unngå dette skal vi bruke en annen metode. Vi skal gjennomføre en "dybde først søking" framover gjennom grafen, der det rekursive kall for hver aktivitet returnerer det seneste tidspunkt denne aktiviteten må startes, for at ikke totaltiden for prosjektet skal økes. Flere detaljer lenger ned.

```
Tid: 0      Starter: A
            Starter: B
I arbeid: 6

Tid: 3      Slutter: A
I arbeid: 2

Tid: 5
            Slutter: B
            Starter: C
            Starter: D
I arbeid: 6

Tid: 6      Slutter: C
            Starter: F
I arbeid: 7

Tid: 7      Slutter: D
            Starter: E
I arbeid: 7

Tid: 10     Slutter: F
I arbeid: 4

Tid: 15     Slutter: E
            Starter: H
            Starter: G
I arbeid: 5

Tid: 16     Slutter: G
I arbeid: 2

Tid: 18     Slutter: H

**** Minste gjennomføringstid for prosjektet er 18 ***
```

Figur 2. Eksempel på utskrift fra del (b) for prosjektet i aktivitetsgrafen

Utskriften fra del (c) skal være en liste over alle aktivitetene, sortert på nummer, der vi for hver aktivitet angir:

- Nummer
- Beskrivende tekst (navn)
- Tidsbehov og mannskapsbehov
- Slakk
- De aktivitetene som er avhengige av denne (ved nummer).

Beregn for hånd seneste starttidspunkt og slakken for aktivitetene i prosjektet gitt over, og angi hvordan utskriften kan bli.

Angående punkt (d) så skal vi her skrive ut en ny gjennomføringsplan slik som over, men denne gangen med den begrensning at antall i arbeid ikke skal overstige et gitt tall. Programmet som produserer en slik gjennomføringsplan vil på mange måter være likt det som finner gjennomføringen under (b), men man kan ikke nå uten videre sette i gang en aktivitet så fort alle den er avhengig av er avsluttet. Det må sjekkes om det er ressurser nok til igangsettingen.

Generelt vil vi dermed her sitte med en liste av aktiviteter som er klare for igangsetting, men som ikke er kommet i gang på grunn av personalmangel. Hvilke av disse aktivitetene man skal sette i gang når det blir frigjort nye personer er et vanskelig spørsmål, men vi skal ikke kreve noe genialitet her. Om man VIL være litt lur kan man ta hensyn til at man må se å få i gang aktiviteter som fikk liten slakk under punkt (c), og at det lett kan bli til at aktiviteter som krever stort mannskap kommer i gang svært sent.

Detaljer i programutformingen

Innlesing og opprettelse av grafen

For å representere grafen internt skal vi bruke følgende klasser:

```
class Aktivitet
{  int arbAnt, tid, nr;
   String navn;
   int tidligsteStart, senesteStart;
   Kant utKanter;
   int antForgj;
   int hjelp;          // Brukes til litt ymse

   Aktivitet(int n)
   {  nr = n;
   }
}
```

```
class Kant
{  Kant neste;
   Aktivitet aktivi;

   Kant(Kant n, Aktivitet a)
   {  neste = n;
      aktivi = a;
   }
}
```

Den datastrukturen vi skal ha er følgende: For hver aktivitet skal vi ha et objekt av klassen "Aktivitet", og etter innlesningen skal den ha riktig nummer, navn, arbAnt, tid og antForgj. Vi skal regne med at alle aktivitetsnumrene ligger i et område fra 1 til maxnr (men ikke nødvendigvis tett), der maxnr oppgis helt i starten i dataene. Vi skal ha en global "Aktivitet [maxnr +1] akt", der de indeksene som tilsvarer brukte aktivitetsnumre har en peker til det tilsvarende objekt, mens resten av arrayen skal inneholde **null**.

Knyttet til hver aktivitet skal det så være en liste av Kant-objekter, som angir de utgående kanter til aktiviteten (til de aktiviteter som ikke kan startes før denne er avsluttet). Rekkefølgen av kantobjektene i disse listene er uten betydning.

Når man leser inn dataene kan denne datastrukturen opprettes etter hvert. Det eneste trikset er at man må opprette et aktivitetsobjekt første gang denne aktiviteten "nevnes" i dataene, og dette kan godt være i en forgjengerliste (se under) før selve definisjonen av aktiviteten kommer. Man må da bare la objektet stå "tomt" til definisjonen kommer.

Formatet på dataene er en sekvens av aktivitetsdefinisjoner, og hver slik definisjon er en sekvens av følgende (skilt med blanke):

- Nummer på aktiviteten (heltall)
- Navn på aktiviteten (som en tett sekvens av tegn avsluttet av blank)
- Tidsforbruk for aktiviteten (heltall)
- Personkrav til aktiviteten (heltall)
- En sekvens av aktivitetsnumre som angir de aktiviteter som må være ferdige før denne kan startes ("forgjengerlisten"). Denne listen avsluttes med en null.

Foran dataene til første aktivitet ligger altså tallet "maxnr", nevnt over. Det er ingen spesiell rekkefølge på aktivitetsdefinisjonene. Innlesningen av dataene skal ikke gjøres spesielt robust, men det *skal* sjekkes at alle aktiviteter som noensinne nevnes, blir definert en og bare en gang. Så fort man finner noe feil i dataene skal man bare gi en feilmelding å avslutte.

Det kan være lurt å lage en global liste av kantobjekter som angir de aktiviteter som ikke har noen forgjengere, og som dermed kan startes umiddelbart når prosjektet settes i gang.

Om å finne og skrive ut løkker

Det å undersøke om det finnes løkker i grafen kan man gjøre ved å la en rekursiv prosedyre gjennomsøke grafen på standard "dybde først" maner, og se om vi finner en "bakoverkant". Den rekursive prosedyren skal dermed kalles en gang for hver node, og man må passe på å starte opp kall i alle noder som ikke har forgjengere. Til å kontrollere utbredelsen av denne prosedyren kan man bruke attributtet "hjelp". Initielt kan den være null i alle aktivitetene, og vi kan la den bli noe annet så fort vi kaller prosedyren i en aktivitet.

I aktiviteter der prosedyren er kalt kan man skille mellom to tilstander, nemlig at prosedyren fremdeles er i gang, eller at den har terminert. Vi kan markere dette med at hjelp er hhv. positiv og negativ. I det siste tilfellet kan vi rett og slett la hjelp ha verdien -1,mens det i det første tilfellet er behagelig at hjelp inneholder nummeret til den aktiviteten der prosedyren i denne aktiviteten ble kalt fra.

Den rekursive prosedyren bør derfor som parameter ha nummeret til aktiviteten der kalleren er lokal. En bakoverkant (og dermed en løkke) er funnet dersom man skal til å kalle en node med som har en positiv verdi i "hjelp". Ut fra verdiene i hjelp er det da lett å følge løkka bakover rundt, slik at man kan skrive den ut før man avslutter programmet.

Gjennomgang med oppstarting på tidligste tidspunkt

Vi skal så se på det å "gjennomføre" prosjektet, etter den filosofi at en aktivitet skal startes opp så fort alle de den er avhengig av er ferdige. De aktivitetene som ikke har noen forgjengere skal da selvfølgelig startes opp med en gang.

Et viktig poeng her er at man hele tiden har en liste som angir de aktiviteter som er startet, men ikke avsluttet. Denne listen bør man holde sortert på avslutningstiden til aktivitetene, som man jo kan fastlegge når aktiviteten startes (eller man kan bruke en "prioritetskø" i stedet for bare en liste). Det nærmeste tidspunktet det skal "skje noe" på vil da hele tiden være avslutningstidspunktet til den første aktiviteten i lista.

Når denne lista er tom er prosjektet ferdig utført. Denne lista lages mest behagelig av objekter som likner kantobjekter, men som i tillegg har et heltalls attributt "avslTid", som er den verdien lista skal sorteres etter.

Nok et poeng er viktig, nemlig at man hele tiden holder rede på hvor mange uavsluttede aktiviteter hver av aktivitetene venter på før de kan komme i gang. Til dette kan vi bruke variabelen "hjelp", og den må da altså i denne forbindelse initialiseres i alle aktiviteter til å være lik antall forgjengere. Vi må da passe på å telle variabelen "hjelp" ned i alle etterfølgerne hver gang vi avslutter en aktivitet, og om verdien blir null skal denne etterfølgeren straks settes i gang.

Den kronologiske utskriften produseres rett og slett under veis mens man utfører prosessen beskrevet over. Man må også ha en variabel som holder greie på hvor mange personer som til enhver tid er i arbeid. Vi kan dermed lage utskriften til punkt (b).

Beregning av slakk etc.

For å finne seneste starttidspunkt under punkt (c) skal vi gjøre nok en "dybde-først" søking gjennom grafen, og de rekursive kallene kan da passelig returnere det seneste starttidspunkt for den aktuelle noden (aktiviteten). Seneste starttidspunkt for de aktivitetene som ikke har etterfølgere bestemmes ut fra totaltiden for prosjektet, som ble funnet i fase (b). For andre noder kan den bestemmes ut fra svarene på de rekursive kallene. Vi kan også her passelig bruke "hjelp" til angi hvilke noder man allerede har vært innom, men vi behøver her bare skille mellom to verdier.

Utskriften av en liste med alle opplysninger om alle aktivitetene skulle etter dette være rett fram.

Gjennomføring med begrensning i antall personer

Det programmet man her skal skrive blir svært likt programmet for å gjennomføre prosjektet uten personbegrensning. Det viktige her er at man har en liste over de aktiviteter som er klare til å starte opp, men som enda ikke er igangsatt. I og med personbegrensningen kan ikke disse settes i gang umiddelbart, men man må hele tiden passe på å sette i gang så "mange som mulig". Hvilke som skal settes i gang når man har flere å velge i er imidlertid ikke noe lett spørsmål.

For å gjøre det enkelt kan man holde denne lista sortert på slakk (slik at de med minst slakk blir satt i gang først) eller på personalkrav, etter som hva man tror gir minst total gjennomføringstid. Hver gang en aktivitet avsluttes, må man da sørge for å sette i gang (altså gjøre utskrift, og flytte over i den andre lista) så mange aktiviteter som mulig fra denne lista, innenfor den rammen at personbruken ikke overgår den totale personalbegrensningen.

Uttesting, krav til levering

Det er en rekke testfiler som kan nyttes som inndata til programmet deres på det området hvor programmer fra forelesninger ligger: dbldekt, dblstart, husbygg, husbygg2, husfeil, lokke, stlokke, udeffor, veilbygg og veifeil1. De vil alle teste ulike sider av programmet - bruk disse. For å få oppgaven godkjent, må den virke tilfredsstillende på filen: veibygg.