Institut for Matematik og Datalogi
Syddansk Universitet, Odense

24. oktober 2002
LMF

DM02 eksamen – Obligatorisk Opgave
Et Planlægningsproblem

Lene M. Favrholdt
Efteråret 2002

1 Problemet

Et firma har m identiske produktionsmaskiner. En ordre fra en kunde består af en mængde af n produkter, som skal leveres til en bestemt dato. Da der er en udgift forbundet med at starte en maskine op, ønsker firmaet at bruge så få maskiner på ordren som muligt.

Nogle gange er der simpelt hen ikke maskiner nok til at udføre ordren inden kundens deadline. I dette tilfælde beregner firmaet, hvornår ordren kan være udført, hvis alle maskiner sættes ind på opgaven, og spørger så kunden, om det vil være tilfredsstillende.

2 Lidt flere detaljer

Vi bekymrer os ikke om, præcis hvilke produkter kunden ønsker. Det, vi er interesserede i, er, hvor lang tid det tager at producere de enkelte produkter. Da maskinerne er identiske, afhænger dette ikke af, hvilken maskine det enkelte produkt bliver produceret på. Det at producere et produkt kalder vi fra nu af at *udføre et job*, og den tid, det tager at udføre et job, kalder vi jobbets *længde*.

Input til problemet er en sekvens af n heltal, som angiver jobbenes længder, et heltal m, der angiver, hvor mange maskiner, der er til rådighed, samt et heltal d, som angiver hvor mange tidsenheder, der er til rådighed inden deadline.

For hver ordre er der to delproblemer, hvoraf det første altid løses. Om det andet løses, afhænger af løsningen til det første delproblem.

Delproblem 1 Først forsøges det at lægge en plan for, hvordan samtlige jobs kan afvikles inden deadline ved brug af så få maskiner som muligt. Hvis det ikke kan lade sig gøre med højst m maskiner, går man videre til Delproblem 2.

Delproblem 2 Nu lægges en plan for afvikling af de n jobs på de m maskiner, der er til rådighed, sådan at hele ordren er klar hurtigst muligt.

Lad os se på de to delproblemer hver for sig.

3 Delproblem 1

I dette delproblem skal vi undersøge, om deadline kan overholdes uden at bruge flere maskiner end de m, vi har til rådighed. I så fald ønsker vi at angive en plan for, hvordan samtlige jobs kan afvikles inden deadline på så få maskiner som muligt.

3.1 Algoritmen

Desværre tager det generelt meget lang tid at løse Delproblem 1 eksakt. Derfor anvendes en approksimationsalgoritme — d.v.s. en algoritme, som ikke altid finder den bedste løsning, men som finder en løsning, der ikke er "alt for langt fra at være optimal".

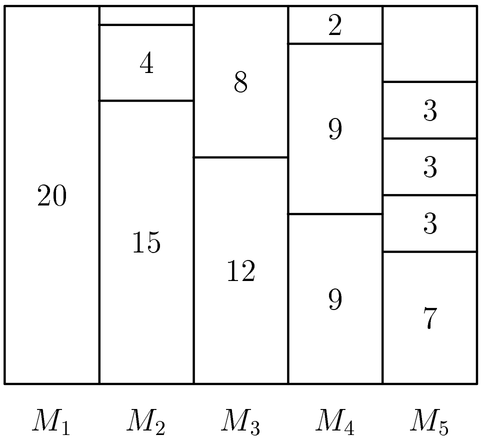
Algoritmen, vi skal bruge, hedder First-Fit Decreasing (FFD). Navnet skyldes, at algoritmen først sorterer jobbene efter længde i ikke-voksende rækkefølge. Derefter bliver jobbene et for et — i ikke-voksende rækkefølge — tildelt en maskine. Hvert job tildeles den første maskine, hvor jobbet "passer", d.v.s. den første maskine, hvor jobbet kan udføres uden at overskride deadline. Det kan bevises, at den løsning, man får v.h.a. FFD, bruger højst 11/9 gange så mange maskiner som en optimal løsning.

Et lille eksempel

Antag, at jobbene har følgende længder $\langle 9, 4, 3, 7, 20, 3, 3, 8, 9, 2, 12, 15 \rangle$, og at deadline er 20. Sekvensen sorteres først: $\langle 20, 15, 12, 9, 9, 8, 7, 4, 3, 3, 3, 2 \rangle$. Derefter tildeles jobbene et for et den første maskine, det passer på. Vi kalder maskinerne $M_1, M_2, \ldots, M_m$, og ser på dem i denne rækkefølge.

Det første job tildeles altid den første maskine. I dette tilfælde betyder det, at M_1 kun kommer til at køre ét job, fordi det første job kun akkurat kan udføres inden deadline. Det næste job bliver derfor tildelt maskine M_2, som derefter har 5 tidsenheder tilovers. Det er ikke nok til at køre det tredje job til ende. Derfor placeres det tredje job på maskine M_3. Tilsvarende placeres det fjerde job på maskine M_4, hvorefter M_4 har 11 ledige tidsenheder. Det femte job kræver ligesom det fjerde job 9 tidsenheder. Den første maskine med så mange ledige tidsenheder er M_4. Derfor tildeles det femte job også M_4. Det sjette job kan køre på M_3 sammen med det tredje job. O.s.v..

Figur 1 viser den færdige plan. Som det ses, bruger algoritmen 5 maskiner. D.v.s. i dette tilfælde leverer FFD en optimal løsning, for jobbene kan ikke afvikles inden for 20 tidsenheder på færre end 5 maskiner.



Figur 1: Resultatet af FFD anvendt på job-sekvensen $\langle 9, 4, 3, 7, 20, 3, 3, 8, 9, 2, 12, 15\rangle$ med deadline $d = 20$.

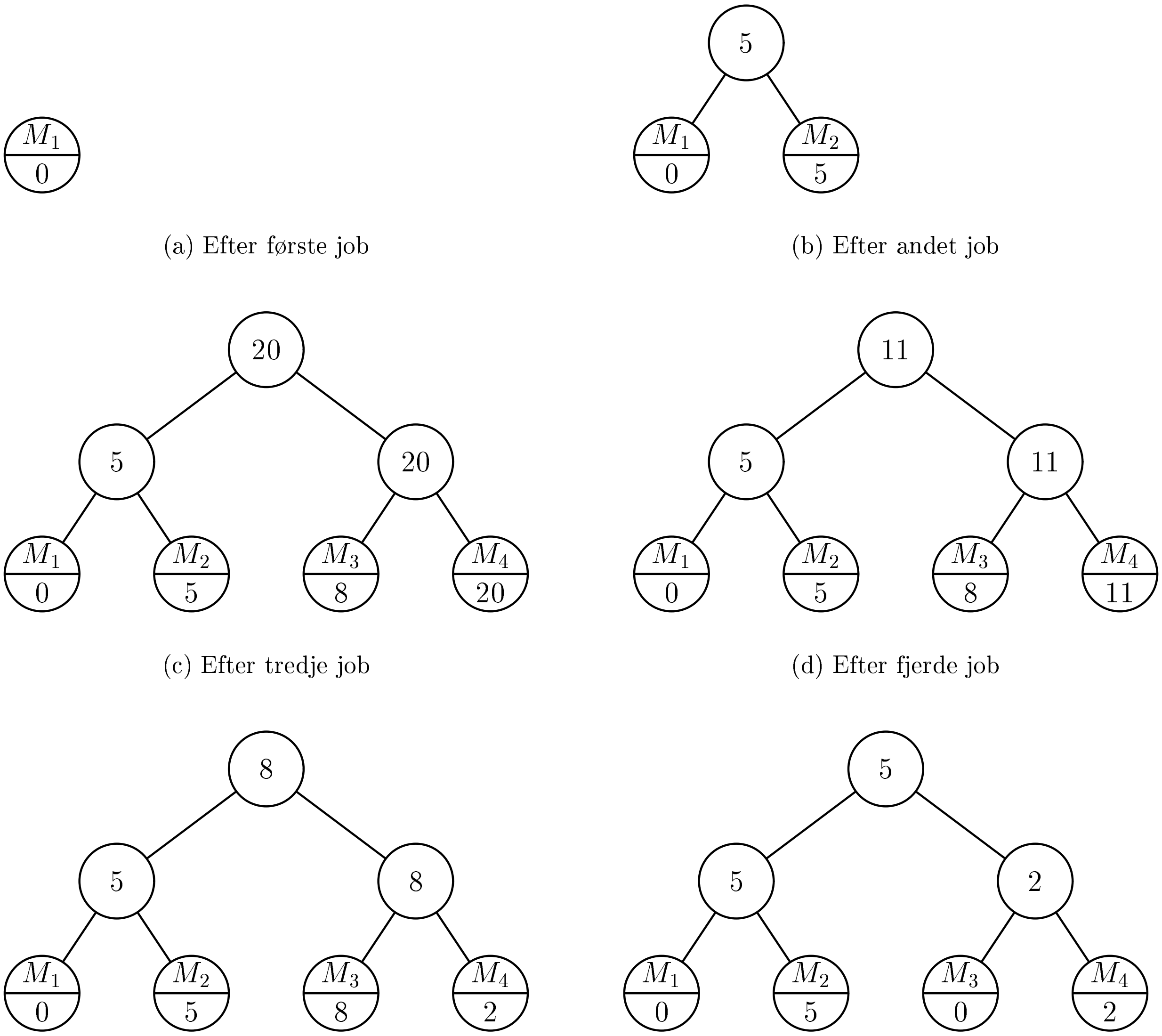
3.2 Datastrukturen

Det er ikke svært at se, hvordan FFD kan implementeres, så den har worst case kompleksitet $O(n^2)$. Man sorterer jobbene, hvorefter man fordeler dem på maskiner på følgende måde. For hvert job løbes maskinerne igennem fra en ende af, startende med M_1, indtil man finder en maskine, hvor jobbet kan afvikles inden deadline.

Vi vil imidlertid gerne opnå en samlet køretid på $O(n \log n)$. Det er ikke muligt med ovenstående strategi. Hvis jobbene er så store i forhold til deadline, at n bliver proportional med det antal maskiner, man anvender, vil det nemlig tage tid $\Theta(n^2)$ at fordele jobbene på maskinerne. Vi tager derfor en træstruktur til hjælp.

Mere præcist skal vi bruge et fuldstændigt binært træ, d.v.s. et binært træ, hvor hver indre knude har præcis to børn og alle blade har samme dybde. Bladet længst til venstre repræsenterer M_1, det næste blad repræsenterer M_2 o.s.v.. I hvert blad opbevares et heltal t. t angiver, hvor mange tidsenheder der er tilbage inden deadline på den tilsvarende maskine, når de jobs, maskinen allerede har fået tildelt, er udført. I hver indre knude opbevares det største heltal, som findes i knudens undertræ.

Træet udvides, efterhånden som flere og flere maskiner tages i brug. Sålænge kun én maskine er i brug, har træet højde 0 — d.v.s. det består af kun én knude. Når der er to maskiner i brug, har træet højde 1. Når der er mellem tre og fire maskiner i brug, har træet højde 2. O.s.v.. Generelt har træet højde i, $i \geq 1$, når der er mellem $2^{i-1} + 1$ og 2^i maskiner i brug. Figur 2 viser træet for eksemplet fra afsnit 3.1, efter at hvert af de første seks jobs er blevet tildelt en maskine.



(a) Efter første job

(b) Efter andet job

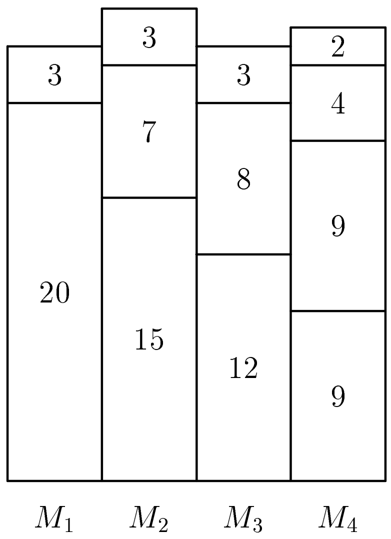
(c) Efter tredje job

(d) Efter fjerde job

(e) Efter femte job

(f) Efter sjette job

Figur 2: Træet fra afsnit 3.2, efter hvert af de første seks jobs er blevet tildelt en maskine.



Figur 3: Resultatet af LPT anvendt på sekvensen $\langle 9, 4, 3, 7, 20, 3, 3, 8, 9, 2, 12, 15\rangle$, når $m = 4$. Efter 25 tidsenheder er alle jobs afsluttet.

4 Delproblem 2

I dette delproblem har vi m maskiner til rådighed, og opgaven er at fordele jobbene på de m maskiner, så det tidspunkt, hvor alle jobs er udført, indtræffer tidligst muligt. Igen står vi med et problem, som det tager meget lang tid at løse eksakt. Derfor anvendes approksimationsalgoritmen Longest Processing Time first (LPT). Ligesom FFD behandler LPT jobbene i rækkefølge efter ikke-voksende længde. Algoritmen holder hele tiden styr på, hvor lang tid hver maskine vil være om at afvikle de jobs, der indtil videre er tildelt den. Hvert job tildeles en maskine, som indtil videre er blevet tildelt jobs med kortest samlet længde. Det kan bevises, at den tid, LPT bruger til at afvikle jobbene, aldrig er mere end 7/6 gange så lang som den kortest mulige tid, jobbene kan afvikles indenfor.

Figur 3 viser, hvordan LPT fordeler jobbene fra eksemplet i afsnit 3.1, når der er $m = 4$ maskiner til rådighed. De første m jobs vil altid blive fordelt ud på hver sin maskine. D.v.s. i eksemplet vil jobbet af længde 20 blive tildelt M_1, jobbet af længde 15 vil derefter blive tildelt M_2, jobbet af længde 12 vil blive tildelt M_3, og det ene job af længde 9 vil blive tildelt M_4. Herefter er M_4 den maskine der tidligst afslutter sit job. Derfor bliver det næste job af længde 9 også tildelt M_4, hvorefter M_3 er den maskine, der afslutter afviklingen af sine jobs tidligst. Dvs. jobbet af længde 8 bliver tildelt M_3. O.s.v..

Da det nu er lykkedes os at løse Delproblem 1 i tid $O(n \log n)$, ønsker vi også at løse Delproblem 2 i tid $O(n \log n)$, så den samlede køretid bliver $O(n \log n)$. Til dette formål skal en prioritetskø implementeres. Prioriteterne er selvfølgelig den tid, det tager den enkelte maskine at afvikle de jobs, den indtil nu har fået tildelt.

5 Input, output og kørsler

I dette afsnit beskrives formatet for input og output samt præcis, hvordan programmet skal kunne afvikles.

5.1 Input

Sekvensen af joblængder lægges i en fil, et job pr. linie. En "jobfil" er altså en fil, hvor hver linie indeholder netop ét positivt heltal. Jobfilen for eksemplet i afsnit 3.1 ville se således ud:

```
9
4
3
7
20
3
3
8
9
2
12
15
```

Programmet skal tage tre argumenter: en jobfil, et positivt heltal, som angiver antallet af maskiner, samt et positivt heltal, som angiver deadline, i nævnte rækkefølge.

5.2 Output

Planen for afviklingen af jobbene skal skrives til standard output på følgende måde. For hver maskine skrives en linie med længderne af jobbene, som er tildelt denne maskine. Joblængderne på de enkelte linier adskilles af mellemrum.

For eksemplet i afsnit 3.1 med 5 maskiner og deadline 20, ville output altså se sådan ud:

```
20
15 4
12 8
9 9 2
7 3 3 3
```

Med 4 maskiner og deadline 20, ville output se sådan ud:

```
20 3
15 7 3
12 8 3
9 9 4 2
```

Bemærk, at der altid kommer output fra netop et af de to delproblemer; fra Delproblem 1, hvis samtlige jobs kan udføres på de m maskiner inden deadline, og fra Delproblem 2 ellers.

5.3 Kørsler

Der er følgende krav til afviklingen af det færdige JAVA-program. Ens hovedklasse skal hedde Plan (stort 'P'). Man skal kunne afvikle programmet på følgende måde:

```
java Plan jobfil m d
```

hvor `jobfil`, `m` og `d` er som beskrevet i afsnit 5.1.

Hvis I gerne vil have en ide om, hvad vi senere vil sige til det output, I producerer, kan I skrive `DM02check`. I skal, før I afgiver kommandoen, placere jer i det katalog, alle jeres oversatte `Java`-programmer ligger i. Så vil jeres program blive kørt på samtlige testfiler i `/home/IMADA/courses/dm02/Tests`. Hvis der findes fejl, bliver dette skrevet på standard output, ellers skrives der intet. Hvis der ikke findes fejl, betyder det ikke nødvendigvis, at jeres program fungerer helt korrekt, da testen ikke er udtømmende. Programmet `DM02check` kan afbrydes med `Ctrl-c`.

6 Hjælp

6.1 Indlæsning af filer

Indlæsning af filer kan klares med

```
FileReader fr = new FileReader(fileName);
BufferedReader inFile = new BufferedReader(fr);
```

hvorefter man kan læse en linie ad gangen med

```
s = in.readLine();
```

`s` kan f.eks. konverteres til en `int` ved at skrive

```
i = java.lang.Integer.valueOf(s).intValue();
```

6.2 Sortering

Til sortering af jobbene er det tilladt at bruge en af JAVAs indbyggede sorteringsmetoder. Bemærk dog, at hvis man ikke angiver andet, da vil en sådan funktion sortere jobbene i ikke-voksende rækkefølge, og ikke i ikke-faldende rækkefølge, som er det, de to planlægningsalgoritmer skal bruge.

7 Krav til rapport og program

Først og fremmest skal alle krav beskrevet i denne opgaveformulering naturligvis tilfredsstilles.

Programmet skal være velstruktureret og kommenteret i passende omfang.

Rapporten skal indeholde en udskrift af hele programmet. Denne udskrift skal være identisk med det elektronisk afleverede program.

Rapporten skal indeholde en beskrivelse af de væsentligste valg, der er truffet i forbindelse med implementationen, samt begrundelser herfor.

Man skal argumentere for, at den samlede køretid af ens program er $O(n \log n)$, hvor n er antallet af jobs. Her må man gerne gå ud fra, at JAVAs indbyggede sorteringsfunktioner kører i tid $O(n \log n)$.

Desuden skal det forklares, hvordan programmet er afprøvet.

Eventuelle mangler i program eller rapport skal beskrives.

Endelig skal rapporten underskrives.

8 Aflevering

Der skal afleveres en rapport på sekretariatet, og et program skal afleveres elektronisk.

Den elektroniske aflevering foregår på følgende måde: Opret et katalog (directory), som indeholder alle jeres JAVA-filer til opgaven og intet andet. Stil jer i dette katalog, og afgiv kommandoen `DM02-aflever`. Denne aflevering skal foretages inden torsdag d. 14. november 2002 kl. 12:00. Bemærk, at I (inden denne frist) kan aflevere flere gange, hvis I fortryder en aflevering. Kun den sidste aflevering tæller (de andre slettes).

Rapporten skal afleveres på sekretariatet på Institut for Matematik og Datalogi senest torsdag d. 21. november 2002 kl. 12:00. Denne opgaveformulering er vedhæftet en forside, som skal anvendes ved afleveringen. Husk for jeres egen skyld at få en kvittering (også vedhæftet) på, at I har afleveret.

Husk, at det er et krav, at det elektronisk afleverede program er præcis det samme program, der afleveres en udskrift af i rapporten.

9 Yderligere formalia

Den obligatoriske opgave er et individuelt eksamensprojekt. Der må altså ikke arbejdes i grupper. I må gerne snakke sammen om løsningen og lære af hinanden, men når I går i gang med at skrive ting ned, såvel program som rapport, skal I arbejde selvstændigt. En overtrædelse af dette er eksamenssnyd og vil blive behandlet som sådan. Man har pligt til selv at beskytte sine noter og filer mod læsning af andre. Dette kan f.eks. gøres med `chmod 700 opgavedirectory`. Begge parter involveret i en eventuel plagiering kan blive holdt ansvarlige.

Programmet skal kunne køres på IMADAs maskiner. Man må gerne lave det hjemme, men det er helt på eget ansvar. Tekniske problemer derhjemme er ingen undskyldning. Man er også helt selv ansvarlig for, at ens filer kan flyttes uden problemer.

God fornøjelse!

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

24. oktober 2002
LMF

DM02 eksamen, Efteråret 2002
Obligatorisk Opgave

Skriv tydeligt (maskinskrift eller blokbogstaver)

Navn: ....

Fødselsdato: ....

Brugernavn (login): ....

Instruktor	Elias	Henrik	Lars	Martin
Sæt kryds				

Besvarelsen omfatter .... nummererede sider.

Kvittering for aflevering af obligatorisk opgave i DM02, efteråret 2002

Udfyldes inden afleveringen

Navn: ....

Fødselsdato: ....

Brugernavn (login): ....

Udfyldes af sekretariatet

Modtaget den .... kl. .... af ....
(dato) (klokken) (initialer)