Institut for Matematik og Datalogi
Syddansk Universitet, Odense

3. marts 2008
LMF

DM507 Eksamen — Obligatorisk Opgave, Del 1–2 af 3
Et Pakningsproblem

1 Indledning

I denne note beskrives første og anden del af den obligatoriske opgave, der skal løses i forbindelse med DM507, foråret 2008. Tredje og sidste del af opgaven stilles mandag d. 21. april.

Afleveres de to første delopgaver samlet senest mandag d. 7. april, vil instruktorerne kigge dem igennem og give kommentarer. Hele opgaven (1. – 3. del) skal afleveres samlet senest tirsdag d. 13. maj kl 12:00. Det er frivilligt at aflevere delopgave 1 + 2 til instruktoren inden den endelige aflevering, men det anbefales kraftigt.

Første delopgave gik ud på at implementere pakningsalgoritmen Next-Fit. Anden delopgave går ud på at implementere en anden pakningsalgoritme, som hedder First-Fit.

Efter en indledende beskrivelse af det problem, der skal arbejdes med, beskrives input/output-formater mm. Programmerne vil blive afprøvet automatisk, og det er derfor *vigtigt at overholde kravene til input/output-format* med videre. I teksten beskrives det, hvordan man selv kan kontrollere, at ens format er i orden.

2 Paknings-problemet

I denne opgave ser vi på et pakningsproblem, hvor vi er givet en mængde af kasser og en mængde af elementer, som skal pakkes i kasserne. Kasserne har alle størrelse B, men elementerne kan have varierende størrelse. Det gælder om at bruge så få kasser som muligt uden at overfylde nogen kasse; dvs. den samlede størrelse af elementerne pakket i en enkelt kasse må ikke overstige B.

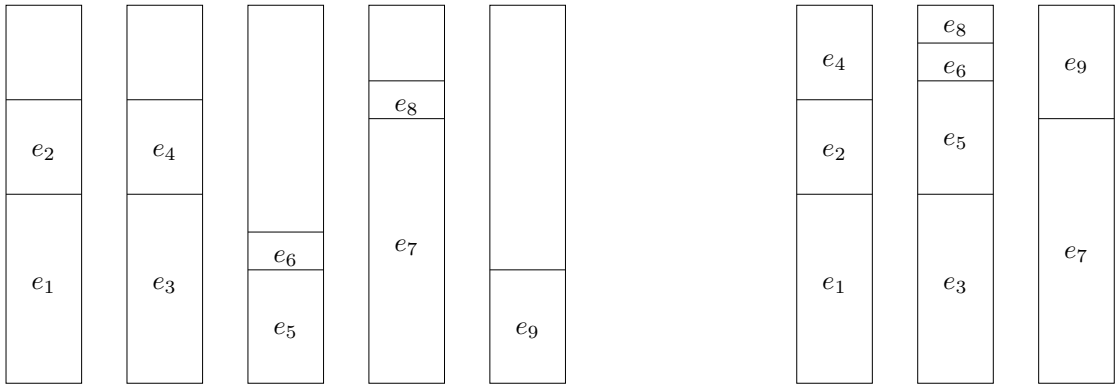
3 Algoritmerne

Det beskrevne pakningsproblem er NP-fuldstændigt. Det betyder, at man regner med, at der ikke findes nogen polynomiel algoritme, som løser problemet optimalt. (At en algoritme er polynomiel betyder, at der findes en konstant c, så algoritmen har køretid $O(n^c)$.) Derfor må vi ty til algoritmer, som ikke altid finder den bedst mulige løsning. Algoritmerne, som vi arbejder med i denne opgave, hedder Next-Fit (NF) og First-Fit (FF).

Next-Fit pakker elementerne et efter et, og holder kun én kasse åben ad gangen. Kan det aktuelle element være i den åbne kasse, anbringes det der. Ellers lukkes den åbne kasse, en ny, tom kasse åbnes, og elementet lægges heri.

First-Fit nummererer kasserne efter den rækkefølge, den tager dem i brug. Den ser på elementerne et for et, og pakker hvert element i den første kasse, som har plads nok. Er der ingen af de allerede anvendte kasser, som har plads til elementet, tages en ny kasse i brug.

Figur 1 illustrerer med et lille eksempel, hvordan de to algoritmer fungerer.



(a) Next-Fit (b) First-Fit

Figur 1: De to algoritmer anvendt på sekvensen $\langle e_1, e_2, e_3, e_4, e_5, e_6 \rangle$, hvor $|e_1| = |e_3| = 50$, $|e_2| = |e_4| = 25$, $|e_5| = |e_9| = 30$, $|e_6| = |e_8| = 10$, $|e_7| = 70$ og $B = 100$.

4 Datastrukturen

Det er ikke svært at se, hvordan man kan implementere First-Fit, så den har køretid $O(n^2)$, hvor n er antallet af elementer, der skal pakkes. For hvert element løber man blot kasserne igennem fra en ende af, indtil man finder en, som kan rumme det nye element.

Vi vil imidlertid gerne opnå en worst-case køretid på $O(n \log n)$. Det er ikke muligt med ovenstående simple strategi; hvis elementerne er så store i forhold til kasser, at hver kasse kun indeholder nogle få elementer, vil man i gennemsnit undersøge $\Theta(n)$ kasser per element. Vi tager derfor en træstruktur til hjælp.

Vi skal bruge et fuldstændigt binært træ; dvs. et binært træ, hvor hver indre knude har præcis to børn, og alle blade har samme dybde. Bladet længst til venstre repræsenterer den første kasse, det næste blad repræsenterer den anden kasse, o.s.v.. I hvert blad opbevares et heltal t, som angiver, hvor meget ledig plads der er i den tilsvarende kasse. I hver indre knude opbevares det største heltal, som findes i knudens undertræ.

Man ved selvfølgelig ikke fra starten, hvor mange kasser, der bliver brug for, men man må gerne bruge et statisk træ med 2^k blade, hvor k er et heltal, så $2^{k-1} < n \leq 2^k$.

Figur 1 viser træet, som det ser ud, efter at Next-Fit har pakket elementerne fra eksemplet i figur 4

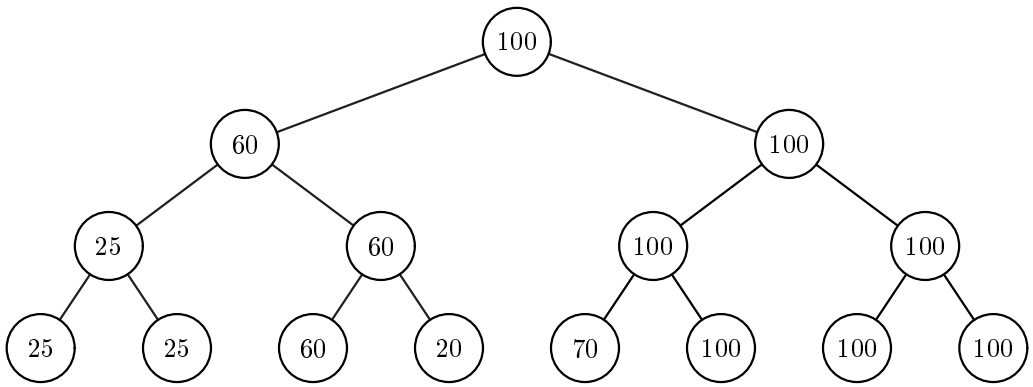
5 Programmet

Programmet skal skrives i JAVA og skal køre på IMADA's maskiner. Programmet skal være velstruktureret og kommenteret i passende omfang.

Algoritmerne skal implementeres, så Next-Fit har køretid $O(n)$, og First-Fit har køretid $O(n \log n)$, hvor n er antallet af elementer. Skriv pseudokode for algoritmerne, inden du begynder at implementere dem.

5.1 Input, output og kørsler

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



Figur 2: Træet svarende til pakningen i Figur 1(a)

Input

Første linie i input-filen er et heltal, som angiver størrelsen B af kasserne. På de næste n linier står størrelserne af de n elementer, der skal pakkes. Disse størrelser er heltal mellem 1 og B. Som en konkret illustration kan man se på input til eksemplet fra Figur 1.

```
100
50
25
50
25
30
10
70
10
30
```

Figur 3: Illustration af input-formatet.

Man må gerne gå ud fra, at input overholder det beskrevne format. Dvs. *programmet behøver ikke at kunne håndtere forkert input.*

Output

Output består af to dele svarende til de to algoritmer Next-Fit og First-Fit, i denne rækkefølge. Output fra de to algoritmer adskilles med en blank linie.

For hver algoritme skal der være en linie for hver kasse, som tages i brug. På linie i står størrelserne af de elementer som pakkes i den i'te kasse. Elementerne skal optræde i den rækkefølge, de blev pakket i. Der skal være præcis et mellemrum mellem hvert par af nabotal. Der må ikke være mellemrum efter det sidste tal på en linie. Hver linie (også den sidste) afsluttes med "\n".

I Figur 4 kan man se det korrekte output til input'et fra Figur 3.

Kørsler

Ud over input/output-format er der følgende krav til programmet:

```
50 25
50 25
30 10
70 10
30

50 25 25
50 30 10 10
70 30
```

Figur 4: Output svarende til input fra Figur 3.

- Alle filer skal ligge i ét katalog (directory) hos jer selv på IMADA's system.
- Hovedprogrammet skal hedde `BP.java` (for Bin Packing). Dvs. man skal kunne afvikle programmet på følgende måde:

  `java BP <filename>`

  hvor filen `filename` indeholder input.
- Programmet skal fungere på IMADAs system.

6 Test

Testen skal designes, inden du skriver programmet. Overvej, hvilke specialtilfælde man kan komme ud for, og hvilke fejl der kan opstå. *Det er en god ide at teste de enkelte komponenter i programmet, efterhånden som du laver dem.*

Du kan få et fingerpeg om, hvad vi vil sige til det output, dit program producerer, ved at skrive `DM507check` (`DM507check` kalder `/home/IMADA/courses/dm507/Tests/check.pl`). Du skal, før du afgiver kommandoen, placere dig i det katalog, som alle dine oversatte `Java`-programmer ligger i. Så vil dit program blive kørt på testfilerne i `/home/IMADA/courses/dm507/Tests`. Bemærk, at testen kun indeholder nogle få testeksempler, så hvis der ikke findes fejl, betyder det ikke nødvendigvis, at dit program fungerer korrekt.

7 Rapporten

Rapporten skal indeholde pseudokode og gerne et klassediagram. Argumentér for, at algoritmerne har de ønskede køretider.

Rapporten skal desuden indeholde en udskrift af hele programmet. Denne udskrift skal være identisk med det elektronisk afleverede program. Der skal være en beskrivelse af de væsentligste valg, der er truffet i forbindelse med implementeringen, samt begrundelser herfor.

Endelig skal der være en overordnet beskrivelse af din teststrategi. Dvs. du skal *uden at referere til programmet* beskrive dine overvejelser i forbindelse med design af testen. Derudover skal selve testen dokumenteres, med reference til den overordnede teststrategi.

8 Aflevering

Programmet skal afleveres elektronisk. Den elektroniske aflevering foregår på følgende måde: Opret et katalog (directory), som indeholder alle dine `.java`-filer til opgaven *og intet andet*. Stil dig i dette katalog. Brug først `ls -la` for at sikre, at du står det rigtige sted, og at de rigtige filer er der. Afgiv derefter kommandoen `aflever DM507`.

D.v.s. gør følgende.

```
cd <opgave-katalog>
ls -la
aflever DM507
```

hvor `<opgave-katalog>` indeholder alle dine `.java`-filer til opgaven og intet andet.

Bemærk, at du (inden afleveringsfristen) kan aflevere flere gange. Kun den sidste aflevering tæller (de andre slettes).

Rapporten afleveres til forelæseren eller instruktoren. Denne opgaveformulering er vedhæftet en forside, som skal anvendes ved afleveringen.

9 Yderligere formalia

Bemærk, at den obligatoriske opgave er en del af eksamen, så reglerne for en eksamenssituation gælder også her. Dvs. du må ikke modtage hjælp fra andre, og du skal sørge for, at andre ikke kan læse dine filer. En god måde at beskytte sine filer på er følgende.

```
mkdir DM507projekt
chmod 700 DM507projekt
```

I må gerne snakke sammen om den overordnede løsning og lære af hinanden, men når I går i gang med at skrive ting ned, såvel program som rapport, skal I arbejde selvstændigt. Overtrædelse af dette er eksamenssnyd.

God arbejdslyst ☺

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

3. marts 2008
LMF

DM507 eksamen, Foråret 2008
Obligatorisk Opgave, Del 1 og 2

Skriv tydeligt (maskinskrift eller blokbogstaver)

Navn: .....

Fødselsdato: .....

Brugernavn (login): .....

Instruktor	Jacob	Nikolaj
Sæt kryds		

Besvarelsen omfatter ..... nummererede sider.