Institut for Matematik og Datalogi
Syddansk Universitet, Odense

6. november 2006
LMF

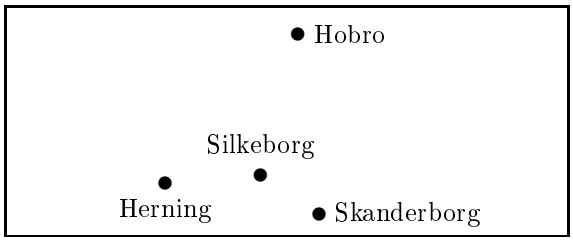
DM02/DM507 Eksamen — Obligatorisk Opgave
Den Handelsrejsendes Problem

1 Indledning

I denne note beskrives den obligatoriske opgave, der skal løses i forbindelse med DM02/DM507, efteråret 2006. 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 utroligt 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. Til sidst beskrives afleveringsproceduren. Det er vigtigt at læse hele opgaven igennem, før man begynder sit arbejde.

2 Den Handelsrejsendes Problem

Opgaven drejer sig om den handelsrejsendes problem. Den handelsrejsende skal fra sin hjemby rundt til en række byer, hvorefter han skal hjem igen. Han vil besøge hver by præcis én gang, og han vil gerne gøre rundturen så kort som muligt. I Figur 1 ses et eksempellandkort fra Midtjylland (kun byerne er indtegnet).



Figur 1: Kort over byerne Herning, Hobro, Silkeborg og Skanderborg.

Til hjælp i sin planlægning har han en afstandstabel. Et eksempel på en sådan kan ses i Figur 2.

	Herning	Hobro	Silkeborg	Skanderborg
Herning		82	37	70
Hobro	82		72	87
Silkeborg	37	72		34
Skanderborg	70	87	34	

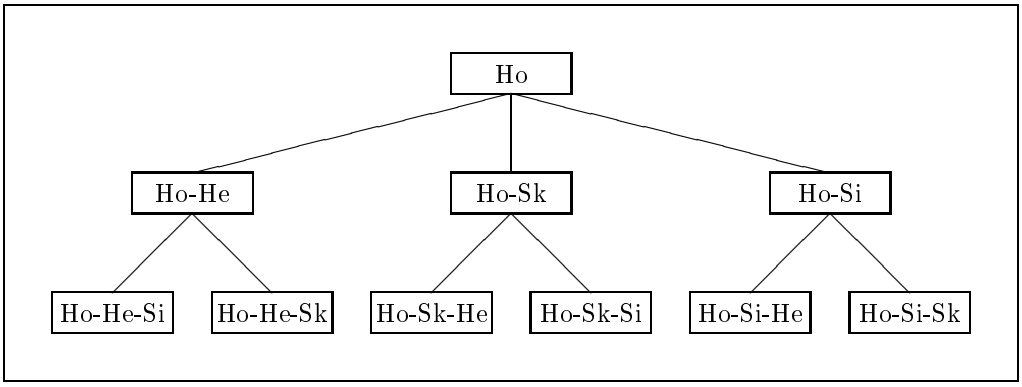
Figur 2: Et eksempel på en afstandstabel.

Et godt bud på en rute rundt til disse fire byer og tilbage igen kunne være: Herning, Silkeborg, Skanderborg, Hobro, Herning med en samlet længde på $37 + 34 + 87 + 82 = 240$, men der er jo også andre muligheder, og vi vil gerne finde den bedste.

Nu er det desværre sådan, at hvis der er mange byer, så er det meget tidskrævende at konstruere den med sikkerhed bedst mulige rute. Problemet er, at man for dette problem og andre lignende problemer ikke kender algoritmer, der med sikkerhed løser dem mere effektivt end ved at prøve alle muligheder. Der findes dog teknikker, der *ofte* kan føre frem til en løsning hurtigere. En sådan er *Branch-and-Bound*, som skal anvendes i denne opgave.

2.1 Anvendelse af Branch-and-Bound

Generelt er der i størrelsesordenen $n!$ måder, hvorpå n byer kan besøges. Man kan skrive disse muligheder op på følgende form – eksemplificeret ved Midtjylland i Figur 3.



Figur 3: Træ med mulige ture og delture i Midtjylland.

Læg mærke til, at det er ligegyldigt, hvor vi starter, og så snart vi har bestemt de første $n-1$ byer, er hele turen bestemt. I stedet for blot at lave alle bladene på dette træ, vil vi lave (dele af) hele træet – lidt ad gangen. Vi håber på, at vi undervejs kan slutte os til, at dele af træet ikke behøver at blive undersøgt. Til det formål definerer vi en *nedre grænse*, der associeres med en delløsning, og som angiver, at alle fuldstændige løsninger, der konstrueres ud fra den omtalte delløsning, må have længder mindst så store.

Den nedre grænse, vi vil bruge her, defineres som følger. Antag, at vi ser på en delløsning, hvor vi har bestemt os til at besøge byerne $x_1, x_2, \ldots, x_k$ umiddelbart efter hinanden i den givne rækkefølge. Lad $d(x_i, x_j)$ betegne afstanden mellem byerne x_i og x_j. Det er klart, at enhver fuldstændig løsning, der konstrueres som en udvidelse af delløsningen $x_1, x_2, \ldots, x_k$, må have længde mindst $\Sigma_{i=1}^{k-1} d(x_i, x_{i+1})$. Men de andre byer skal jo også besøges. Lad $\{y_1, \ldots, y_{n-k}\}$ være de $n-k$ byer, der ikke allerede er med i $x_1, x_2, \ldots, x_k$. At besøge by y_i må koste os noget. Hvis de to korteste forbindelser fra y_i til andre byer har længde $l^1_{y_i}$ og $l^2_{y_i}$, skulle man måske umiddelbart tro, at det ville koste mindst $l^1_{y_i} + l^2_{y_i}$. Men så har vi jo allerede forbundet videre til to andre byer i samme omgang. For at kunne behandle hver by separat, benytter vi derfor følgende trick: Vi vil betragte alle veje som bestående af to halvdele hørende til hver sin by. Da kan vi sige, at det koster mindst $\frac{l^1_{y_i}}{2} + \frac{l^2_{y_i}}{2}$ at besøge y_i, og så har vi *ikke* forbundet os med andre byer.

Der er yderligere en uhensigtsmæssighed i ovenstående. Vi ved, at vi skal lave en tur, der kun besøger hver by én gang. Dermed kan y_i umuligt skulle forbindes til nogen af byerne $x_2, \ldots, x_{k-1}$. Vi ændrer derfor definitionen af $l^1_{y_i}$ og $l^2_{y_i}$ til at være længden af de to korteste forbindelser fra y_i til andre byer *bortset fra* $x_2, \ldots, x_{k-1}$.

Endeligt er byerne x_1 og x_k jo kun forbundet med andre byer i den ene ende. At få den anden ende med må med samme trick som ovenfor koste mindst $\frac{l^{1'}_{x_1}}{2} + \frac{l^{1'}_{x_k}}{2}$, hvor $l^{1'}_{x_1}$ og $l^{1'}_{x_k}$ angiver mindste

afstand fra henholdsvis x_1 og x_k til andre byer bortset fra $x_1, \ldots, x_k$ (læg mærke til, at der kan ses bort fra lidt flere byer her, end vi kunne for y_i'erne).

Grænsen er altså

$$\left\lceil \left(\sum_{i=1}^{k-1} d(x_i, x_{i+1}) \right) + \left(\sum_{i=1}^{n-k} \frac{l^1_{y_i} + l^2_{y_i}}{2} \right) + \frac{l^{1'}_{x_1} + l^{1'}_{x_k}}{2} \right\rceil$$

Notationen $\lceil x \rceil$ angiver oprunding af x til nærmeste heltal. Da alle afstande i input er heltal, må længden af en bedste tur jo også være det.

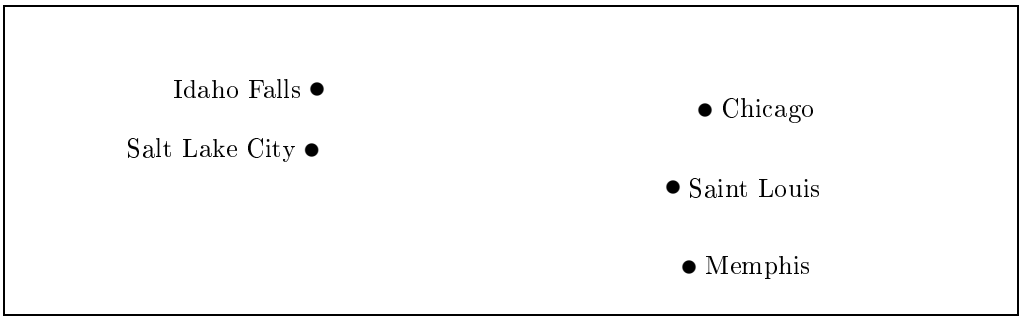
Den nedre grænse kan nu anvendes på to måder. Dels kan vi håbe, at den er et godt bud på den rigtige værdi og dermed altid vælge at fortsætte med den delløsning, der har den mindste nedre grænse. Der er dog en anden anvendelse, der er endnu vigtigere. Når vi først har bestemt længden l af én fuldstændig tur, så kan vi, hvis vi ser en delløsning d med nedre grænse g, hvor $g \geq l$, undlade at undersøge de løsninger, der kan konstrueres ved at udvide d. Da g jo netop er en *nedre* grænse, kan vi aldrig finde noget bedre end l ved at gå den vej.

Alle disse overvejelser fører frem til følgende algoritmeskitse:

```
best ← ∞
Q ← PriorityQueue()
Q.insert(initialRoute)
while not Q.isempty()
    route ← Q.deletemin()
    if route.lowerBound < best
        for each r in nextRoutes(route)
            lb ← lowerBound(r)
            if lb < best
                if Complete(r)
                    best ← lb
                    solution ← r
                else
                    r.lowerBound ← lb
                    Q.insert(r)
```

Her kunne `[Hobro]` eksempelvis være `initialRoute`, og `nextRoutes([Hobro,Herning])` ville bestå af de to muligheder `[Hobro,Herning,Silkeborg]` og `[Hobro,Herning,Skanderborg]`.

For at se nogle interessante udregninger af nedre grænser, må vi se på et lidt større eksempel givet i Figur 4 og 5.



Figur 4: Kort over byerne Chicago, Idaho Falls, Memphis, Saint Louis og Salt Lake City.

	Chicago	Idaho Falls	Memphis	Saint Louis	Salt Lake City
Chicago		1437	544	299	1403
Idaho Falls	1437		1754	1523	208
Memphis	544	1754		311	1605
Saint Louis	299	1523	311		1364
Salt Lake City	1403	208	1605	1364	

Figur 5: Afstandstabel for USA-eksemplet.

Den nedre grænse for `[Chicago,Saint Louis,Salt Lake City]` er $(299 + 1364) + (\frac{208+1437}{2} + \frac{544+1605}{2}) + \frac{544+208}{2} = 3936$. I den anden parentes optræder altså de to korteste forbindelser fra byerne Idaho Falls og Memphis, der *ikke* går til Saint Louis.

Den nedre grænse for `[Chicago]` er $(\frac{208+1437}{2} + \frac{311+544}{2} + \frac{299+311}{2} + \frac{208+1364}{2}) + \frac{299+299}{2} = 2640$.

2.2 Forbedring af køretiden

Som man kan se på algoritmeskitsen, kan man ikke se bort fra nogen delløsninger, før man har fundet den første længde på en fuldstændig tur. Dermed får man i starten prioritetskøen fyldt op med en masse dårlige delløsninger. For at råde bod på det, kan man prøve at give `best` en bedre startværdi end ∞.

Det kunne man gøre ved at lave nogle tilfældige ture og give `best` værdien lig med den mindste længde, man opnåede.

Man kunne også lave nogle ture efter den "grådige" model: start med en vilkårlig by og fortsæt nu altid ad den korteste vej, der fører til en ny by. Man kunne prøve at starte i alle byerne og igen huske det bedste resultat.

I skal gøre begge dele og bruge minimum af de længder, der opnås.

3 Programmet

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

Prioritetskøen skal implementeres som en hob (heap), som beskrevet i lærebogen.

3.1 Input, output og kørsler

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

Input

Inputformatet er beskrevet i Figur 6. Som en konkret illustration kan man se på input til eksemplet fra Jylland i Figur 7. Sammenlign med tabellen i Figur 2.

Læg mærke til, at da tabellen i Figur 2 er symmetrisk omkring diagonalen, indeholder input'et kun data til den del af tabellen, der ligger over diagonalen.

Bynavnene er sekvenser af tegnene fra a–z, A–Z samt 0–9. Derudover kan der forekomme blanke, hvis et bynavn er i flere ord som f.eks. Idaho Falls eller Salt Lake City.

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

n
By_0
By_1
.
.
By_{n-1}
$\text{Afstand}_{0,1}$
$\text{Afstand}_{0,2}$
.
.
$\text{Afstand}_{0,n-1}$
$\text{Afstand}_{1,2}$
$\text{Afstand}_{1,3}$
.
.
$\text{Afstand}_{1,n-1}$
$\text{Afstand}_{2,3}$
$\text{Afstand}_{2,4}$
.
.
$\text{Afstand}_{2,n-1}$
.
.
$\text{Afstand}_{n-3,n-2}$
$\text{Afstand}_{n-3,n-1}$
$\text{Afstand}_{n-2,n-1}$

Figur 6: Inputformatet.

4
Herning
Hobro
Silkeborg
Skanderborg
82
37
70
72
87
34

Figur 7: Eksempel på input.

Output

Output'et fra programmet skal være en sekvens af bynavne, som angiver en kortest muligt tur, der besøger hver by præcis én gang. Til sidst skal turens længde udskrives. I Figur 8 kan man se et eksempel på et korrekt output til input'et fra Figur 7.

```
Skanderborg
Silkeborg
Herning
Hobro
240
```

Figur 8: Eksempel på output.

Der må ikke være blanke linier, og i bynavne bestående af flere ord (som f.eks. Idaho Falls og Salt Lake City) skal de enkelte ord i bynavnet adskilles af præcis én blank. Læg mærke til, at der er flere output, der vil blive accepteret som korrekte. Der kan jo være flere forskellige rigtige ture, og man behøver ikke starte med en bestemt by. Turens længde er dog entydigt bestemt.

Kørsler

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

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

  ```
  java TSP <filename>
  ```

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

4 Test

Testen skal designes, inden du skriver programmet.

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 `DM02check`. 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/dm02/Tests`. Bemærk, at testen kun indeholder nogle få testeksempler, så hvis der ikke findes fejl, betyder det ikke nødvendigvis, at dit program fungerer korrekt.

Det er en rigtig god ide at køre DM02check, inden du afleverer. På den måde sikrer du bl.a., at dit program bruger det korrekte input- og output-format.

Programmet `DM02check` kan afbrydes med `Ctrl-c`.

5 Rapporten

Rapporten skal 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.

Du skal give en overordnet beskrivelse af din teststrategi; dvs. du skal *uden at referere til programmet* forklare, hvilke specialtilfælde man kan komme ud for, og hvordan du tester, at programmet virker korrekt i alle tilfælde.

6 Aflevering

Programmet skal afleveres elektronisk senest mandag d. 27/11 kl. 12:00.

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 DM02`.

D.v.s. gør følgende.

```
cd <opgave-katalog>
ls -la
aflever DM02
```

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 skal afleveres på IMADAs sekretariat senest fredag d. 1/12 kl. 12:00. Denne opgaveformulering er vedhæftet en forside, som skal anvendes ved afleveringen.

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

7 Evaluering

For at kunne gå til eksamen i DM02/DM507 til januar skal man have bestået den obligatoriske opgave. Har man tidligere lavet en obligatorisk DM02-opgave og fået den godkendt, er det tilstrækkeligt.

Der er tre mulige bedømmelser: Godkendt, Ikke Godkendt og Genafleveres. Bedømmelsen Genafleveres gives kun til opgaver, som er så tæt på at være i orden, at vi har svært ved at afgøre, om de skal godkendes eller ej. Dvs. Genafleveres er en "undtagelses-karakter".

De rettede opgaver leveres tilbage ved forelæsningen mandag d. 11/12. Fristen for genaflevering er fredag d. 15/12 kl. 12:00. *Husk, at det er dit eget ansvar at afhente din rettede besvarelse og undersøge, om den er godkendt, eller du eventuelt skal genaflevere.*

8 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 DM02projekt
chmod 700 DM02projekt
```

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 og behandles som sådan.

Programmet skal kunne køre på IMADAs maskiner. Man må gerne lave det hjemme, men det er helt på eget ansvar. Tekniske problemer derhjemme eller problemer med at flytte sine filer er ikke gyldig grund til at aflevere for sent. Udsættelse af aflevering kræver særlige omstændigheder, som vil kunne holde i studienævnet — vi har pligt til at behandle jer ens.

God arbejdslyst ☺

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

6. november 2006
LMF

DM02/DM507 eksamen, Efteråret 2006
Obligatorisk Opgave

Skriv tydeligt (maskinskrift eller blokbogstaver)

Navn: .....

Fødselsdato: .....

Brugernavn (login): .....

Instruktor	Mikkel	Morten
Sæt kryds		

Besvarelsen omfatter ..... nummererede sider.