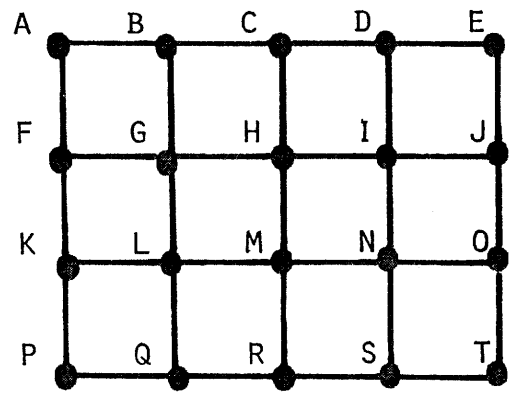
KAPITEL I

RUTEPROBLEMER

I.1 INDLEDNING

De forskellige typer af ruteproblemer blev i mange år betragtet kun som "morskabsmatematik". I de fleste populære samlinger af hovedbrud findes opgaver af disse typer, f.eks. *tegning af figurer i en streg*, *labyrinter* eller *korteste veje*.

Eksempel 1.1 Bogen "Tænke, tænke, tænke" (Fremad 1949) indeholder følgende ruteproblem og forklaring dertil:



"Tegningen forestiller gangene i en mine. Vi kan antage, at hver gang eller passage, A til B , B til C , C til H , H til I osv., er 100 m lang. Det vil ses, at der er 31 sådanne passager. Nu skal en embedsmand inspicere dem alle, og han begynder ved punktet A . Hvor langt skal han gå, og hvilken rute vil du anbefale ham? Måske svarer læseren straks: "Der er 31 passager, hver på 100 m, altså må han gå 3100 m". Men dette forudsætter, at han aldrig behøver at gå mere end én gang gennem hver passage, og det holder ikke stik. Tag en blyant og prøv at finde en korteste rute. Du vil snart opdage, at der er brug for betydelig dømmekraft. I virkeligheden er det en krævende opgave."

De problemer, som vi skal behandle, er endelige: man kan i princippet løse dem ved at undersøge alle muligheder. Så matematisk set kan problemerne måske i første omgang forekomme uinteressante.

Eksempel 1.1 fortsat Det er let at konstruere en mulig rute R af længde allerhøjst 2·3100 m, så vi behøver blot at kontrollere alle ruter af længde mindre end R's længde for at finde en korteste rute. Derfor er problemet endeligt. På den anden side skal de mulige ruter dække alle forbindelser, så vi skal passere alle n knudepunkter mindst én gang hver. Ved hvert knudepunkt er der 2, 3 eller 4 valg for en rutes fortsættelse, så der er mindst 2^n mulige ruter (dette er en grov vurdering - der er i virkeligheden mange flere). Er n=20 som i Eksempel 1.1, og er vi kvikke og kan undersøge 10 muligheder pr. minut, altså 600 muligheder pr. time, vil det tage os allermindst $2^{20}/600 \approx 1748$ timer at løse opgaven ved at undersøge alle muligheder. Havde vi til rådighed en computer, som kunne klare f.eks. 10 muligheder pr. sekund, ville det hjælpe noget: for n=20 ville computeren bruge mindst $2^{20}/36000 \approx 29$ timer, men for n=30 mindst 3 år!

Antallet af samtlige muligheder, selv i små problemer, er ofte så stort, at det ikke er realistisk at løse problemerne ved at undersøge alle muligheder, heller ikke med computer. Der er derfor behov for metoder, der er "bedre end endelige", dvs. metoder, som finder en løsning langt mere effektivt end ved at kigge alle muligheder igennem. I mange (men ikke alle!) tilfælde er dette muligt ved hjælp af en matematisk analyse af problemtypen og dens struktur.

De typer af ruteproblemer, vi skal se på, er:

a) *Tegning af figurer i én streg*. Vi kunne også kalde denne problemtype *turistens problem* ved at formulere opgaven som (om muligt) at finde en rundtur i en by eller bydel, så hver gade gennemgås præcis én gang.

b) *Postbudets problem*. Eksempel 1.1 er af denne type: der ønskes en kortest mulig rute, der gennemgår alle strækninger mindst én gang.

c) *Korteste veje*. Der ønskes en korteste forbindelse mellem to punkter i et netværk af mulige strækninger.

d) *Den handelsrejsendes problem*. En handelsrejsende skal besøge en række byer - problemet er at finde en rækkefølge at besøge byerne i, så den samlede tur bliver kortest mulig (antal muligheder med n byer er n! , hvilket selv for beskedne n-værdier er voldsomt stort).

Som antydet ovenfor, skelner vi mellem begreberne *problem-type* og *problem* således, at en problem-type er en mængde af problemer. Postbudets problem er en problem-type med Eksempel 1.1 som ét problem af denne type. De løsninger, vi søger, er løsninger til problem-typer. Vi er altså ikke tilfredse med blot at løse det konkrete problem i Eksempel 1.1, men ønsker at løse alle problemer af denne type.

Løsningerne angives i form af algoritmer. En *algoritme* til løsning af en problem-type består af en endelig liste af ordrer, som skal udføres på algoritmens input. Et *input* er ét problem af typen. Ordrerne skal være klare og kunne udføres i endelig tid (der må godt være repetitioner af ordrer, forudsat der er ordrer, som angiver repetition, men der må ikke være uendelige repetitioner eller *uendelige løkker*, som man også kalder det). Ordrerne i en algoritme vil vi angive i en blanding af almindelig dansk og den mere formelle form beskrevet i Appendix B. Når listen af ordrer i en algoritme med et givet input er ført til ende, skal der foreligge et *output* som svar. Dette output kan have form af et JA eller et NEJ eller et tal, for eksempel.

Antallet af samtlige muligheder for et problem af størrelse n af en problem-type er oftest mindst eksponentielt i n (dvs. af formen $c \cdot a^n$, hvor $c > 0$ og $a > 1$) eller måske så slemt som $c \cdot n!$. For at kalde en algoritme for en sådan problem-type *effektiv* eller *god* benyttes ofte den definition, at antal skridt algoritmen skal gennemføre er højst en potensfunktion i størrelsen af input (dvs. af formen $c \cdot n^b$, hvor c og b er konstanter, som opfylder $c > 0$ og $b > 0$).

En algoritme, som for et problem af størrelse n, finder output i højst $c \cdot f(n)$ skridt, hvor $c > 0$, siges at være af *kompleksitet* højst $O(f(n))$ (læses store σ af $f(n)$).

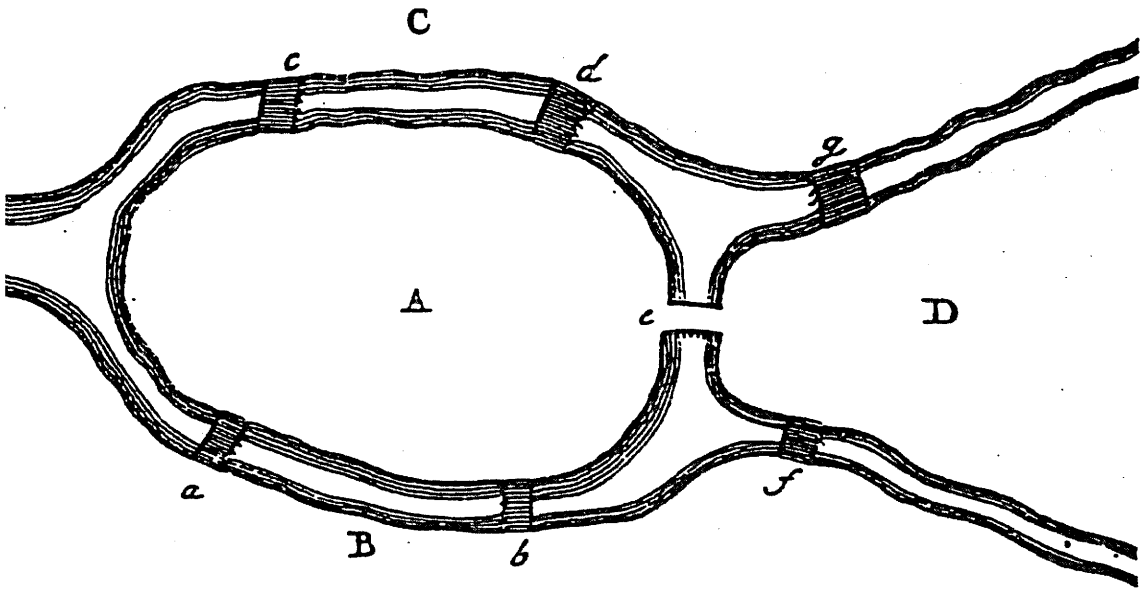
Opgave 1.1 Antag, at din computer kan gennemføre 1000000 skridt pr. sekund. Hvor lang tid vil det tage den at gennemføre henholdsvis n^2 , n^5 , 2^n og $n!$ skridt for $n = 20$, 40 og 60 ?

Definitionen af en god algoritme ovenfor kan synes grov (er en algoritme af kompleksitet $O(n^{1000})$ virkelig god ?). Den er imidlertid i øjeblikket det bedste bud på en rimelig definition. Den har den fordel, at den er *robust* (hvis vi f.eks. tæller det at finde et bestemt element i en mængde med n elementer som ét skridt, hvor andre ville tælle n skridt, så giver denne uenighed ikke anledning til uenighed om algoritmen er god eller ej). Endvidere giver definitionen anledning til interessante resultater, som vi skal se.

En alternativ måde at løse en problem-type på end ved at angive en god algoritme, er at finde en god matematisk sætning i form af en karakterisation. Vi skal senere se eksempler herpå. Ligesom vi ovenfor har præciseret, hvad vi vil forstå ved en god algoritme, skal vi senere præcisere, hvad vi vil forstå ved en *god sætning*. Et af budskaberne bliver så, at gode algoritmer og gode sætninger ofte følges ad, og der er en tæt vekselvirkning mellem de to løsningsformer.

1.2 TEGNING AF FIGURER I ÉN STREG

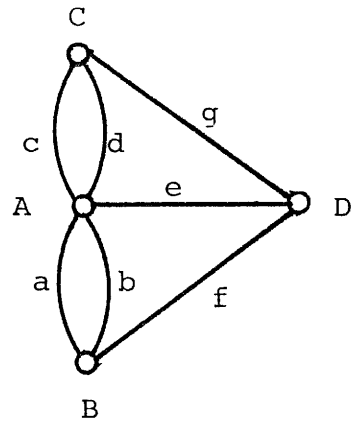
Matematikeren Leonard Euler (1707-1783) skrev i 1736 en artikel om muligheden af at gå en tur i byen Königsberg i Preussen således, at byens 7 broer over floden Pregel hver blev overskredet én gang og kun én gang. Følgende diagram over broernes beliggenhed stammer fra Euler's artikel:



Euler fortæller: "Da der i Königsberg-problemet er 7 broer, kan det løses ved en fuldstændig opregning af alle de mulige veje, som man kan gå. På den måde ville det blive klart, om der fandtes eller ikke fandtes en vej som den i problemet forlangte. Men på grund af det store antal kombinationsmuligheder er denne løsningsmetode både for vanskelig og for besværlig, og i andre problemer med endnu flere broer ville den slet ikke kunne anvendes. Blev en metode af denne art ført til ende, ville man få svar på mange spørgsmål, der slet ikke var stillet; heri ligger utvivlsomt årsagen til mange af vanskelighederne. Efter at have forkastet denne metode gav jeg mig til at søge efter en anden, der blot skulle afsløre om der fandtes en vej eller ikke. Thi jeg havde mistanke om, at en sådan metode ville vise sig langt simplere."

Euler gav løsningen ikke blot på Königsberg-problemet, men på alle problemer af samme type. For at formulere løsningen er det bekvemt at benytte *grafteori*. Se Appendix A for præcise grafteoretiske definitioner.

Eulers diagram ovenfor kan repræsenteres af følgende graf G :



hvor de 4 punkter i punktmængden P(G) svarer til de 4 områder i Königsberg, og de 7 kanter i kantmængden K(G) svarer til de 7 broer. For eksempel svarer kanten g = (C,D) i grafen til broen g mellem områderne C og D i Eulers diagram.

En *rute* i en graf G , også kaldet en *kant-følge*, er en sekvens af kanter

$$[k_1,k_2,k_3,\ldots,k_{n-1},k_n]$$

hvor k_i forbinder x_i og y_i for $i = 1,2,\ldots,n$ og $y_i = x_{i+1}$ for $i = 1,2,\ldots,n-1$, dvs. k_i slutter hvor k_{i+1} starter. Ruten er *lukket* hvis $x_1 = y_n$ og *åben* hvis $x_1 \neq y_n$. Den kaldes en *tur* hvis $k_i \neq k_j$ for $i \neq j$, dvs. alle kanterne er forskellige. Hvis alle kanter i en graf G er med i en tur kaldes turen en *Euler-tur* i G .

Eksempel 1.2 I grafen ovenfor er [a,b,c,d,e,f,g] ingen rute, men [a,b,c,d,e,f,a,b] og [a,e,f,b] er åbne ruter, hvoraf kun den sidste er en tur.

Königsberg-problem-typen kan nu formuleres som spørgsmålet, om en givet graf har en Euler-tur eller ej. Mere uformelt er et problem af den type at *tegne en figur i én streg*, dvs. uden at løfte blyanten fra papiret og uden at tegne den samme streg to gange.

Euler viste, at eksistensen af en Euler-tur afhænger af grafens valenser. *Valensen* $v(x,G)$ af et punkt x i grafen G er antal kanter i G fra x (også kaldet antal kanter *incidente* med x). Euler viste først

SÆTNING 1.1 (Euler 1736) Summen af alle valenser i en graf G er lig med to gange antal kanter :

$$\sum_{x \in P(G)} v(x,G) = 2 \cdot |K(G)|$$

Bevis: Summen af alle valenser medregner hver kant (x,y) to gange, nemlig én gang i hver af $v(x,G)$ og $v(y,G)$. ∎

Opgave 1.2 Prøv at tegne en graf med ét punkt med ulige valens, f.eks. 3, og alle andre punkter med lige valens. Hvorfor lykkes det ikke?

SÆTNING 1.2 (Euler 1736) En graf har altid et lige antal punker af ulige valens

Bevis: Følger af Sætning 1.1, da summen af alle valenser er lige ∎

Eulers løsning af problem-typen om Euler-ture er

SÆTNING 1.3 (Euler 1736) Lad G være en sammenhængende graf.

G har en lukket Euler-tur

⇕

Alle punkter i G har lige valens.

Bevis:

⇓ Ved hver passage af et punkt i gennemløbet af en lukket Euler-tur benyttes to kanter (for begyndelsespunktet modsvares den første kant på turen af sidste kant på turen). Da hver kant benyttes præcis én gang følger heraf at alle punkter har lige valens.

⇑ Lad x_1 være et punkt i G, og lad os starte en tur i G fra x_1, dvs. en følge af forskellige kanter:

$$[(x_1,x_2),(x_2,x_3),(x_3,x_4),\ldots$$

hvor ingen kant gentages. Da der kun er et endeligt antal kanter i G kan vi ikke fortsætte i det uendelige, men standser i et punkt x_n efter at have gennemløbet turen T:

$$T = [(x_1,x_2),(x_2,x_3),(x_3,x_4),\ldots,(x_{n-2},x_{n-1}),(x_{n-1},x_n)] .$$

Lad G' være den delgraf af G, som disse kanter og deres ende-punkter danner. Da G' kan gennemløbes som turen T, har alle punkter i G' lige valenser, bortset fra at x_1 og x_n i tilfældet $x_1 \neq x_n$ har ulige valenser.

Da T standser i x_n er alle kanter incidente med x_n i G med i T, dvs. x_n har lige valens i G', da den har det i G. Heraf følger at $x_1 = x_n$ og at alle punkter i G' har lige valens.

Hvis T omfatter alle kanter i G er vi færdige. Hvis ikke, fås ved fjernelse af alle kanter i T fra G, at alle de resterende kanter danner én eller flere sammenhængende grafer $G_1,G_2,\ldots,G_k$ med alle valenser lige, dvs. de opfylder samme forudsætning som G.

Da G er sammenhængende vil et af punkterne x_i fra G' ligge i G_1. Når vi gennemløber T kan vi nu standse i x_i, derefter foretage en lukket tur T_1 i G_1 (fundet med samme metode som T i G), for så endelig at fortsætte med resten af T fra x_i.

Ved ovenstående fremgangsmåde er T ændret til en tur T^* indeholdende flere af G's kanter end T. Ved at starte forfra med T^* i stedet for T og fortsætte således opnås til slut en tur indeholdende alle G's kanter. ∎

Selv om den formulering beviset for ⇑ har fået måske ikke er den matematisk set mest elegante, udmærker den sig ved at være *konstruktiv*, idet der i beviset umiddelbart ligger en algoritme til bestemmelse af en Euler-tur:

```
ALGORITME EULER-TUR I

INPUT:    Sammenhængende graf G med mindst
          en kant og med alle valenser lige.
          Et fast punkt x i G .

OUTPUT:   En lukket Euler-tur i G startende
          i x .

BEGIN
  K:= K(G) ;   (*K skal betegne mængden af kan-
               ter endnu ikke gennemløbet *)

  P:= P(G) ;   (*P skal betegne mængden af punk-
               ter incidente med kanter fra K*)

  T:= Ø ;      (*T skal betegne følgen af kanter
               allerede gennemløbet *)

  WHILE  K≠Ø  DO
  BEGIN        incident med kant fra T hvis T≠Ø
    Find  y ∈ P ∨(for T=Ø  vælges  y=x);

    Gennemløb en tur  T1  fra  y  med kanter
    fra  K  indtil det ikke er muligt at fort-
    sætte;

    K:= K\{kanterne i T1};

    P:= P\{punkter hvorfra alle kanter er i T1};

    T:= T med T1  "indsat ved y"(for T=Ø sættes T:=T1)

  END

END.
```

Denne algoritme er første skridt på vejen til et effektivt computer-program. Idet den mængde af operationer hver kant deltager i (inkludering i T_1 , fjernelse fra K , evt. fjernelse af punkt fra P , indsættelse i T) er konstant, dvs. uafhængig af G's størrelse, bliver antal skridt højst konstant $\cdot$ $|K(G)|$. Altså siger vi også, at algoritmen er af kompleksitet $O(|K(G)|)$. Der er således tale om en god algoritme.

Euler's sætning 1.3 er en god sætning, idet den præcist forklarer, hvad der er galt, når der ikke er en Euler-tur. Algoritmen EULERTUR I er en god algoritme. Beviset for sætningens og algoritmens korrekthed er det samme! Dette afsnit er derfor et eksempel på den filosofi, der er udtrykt nederst side 1.4.

↓ Matematisk set kan beviset for ⇑ gøres elegantere ved enten [at benytte *induktion*, dvs. vise ⇑ for G med ≤ 2 kanter, og derefter vise ⇑ for G med n kanter under forudsætning af at ⇑ er korrekt for alle grafer med $< n$ kanter. I dette tilfælde kan så i beviset benyttes at $G_1, G_2, \ldots, G_k$ har Euler-ture] eller [at lade T være en længst mulig tur i G fra x_1 , og så bevise, at T må indeholde alle G's kanter].

↑ Opgave 1.3 Gennemgå beviset på begge de antydede måder.

↓ Det første af de to nævnte versioner (induktion) kan også oversættes til en algoritme. Denne algoritme bliver *rekursiv*, idet metoden til bestemmelse af turen undervejs benytter (eller *kalder*) sig selv.

Rekursive metoder er et vigtigt redskab ved udvikling af algoritmer. Ofte kan en algoritme gøres elegant og overskuelig ved benyttelse af rekursion, men prisen er at en rekursiv metode kan stille lidt større krav til plads (til "bogholderi") og tid.

ALGORITME EULER-TUR II

INPUT: Sammenhængende graf G med mindst en kant og med alle valenser lige. Et fast punkt x i G.

OUTPUT: En lukket Euler-tur i G startende i x.

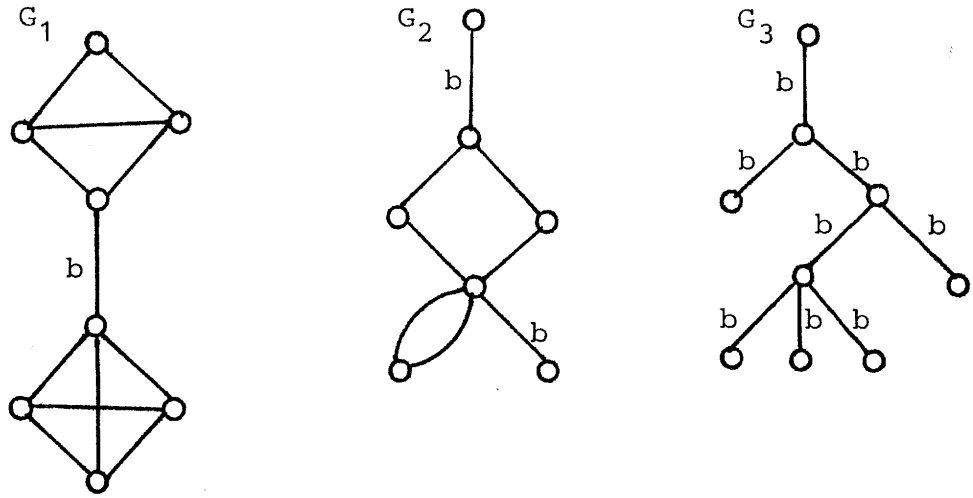
```
BEGIN
  Gennemløb en tur T i G fra x indtil
  det ikke er muligt at fortsætte;
  K:= K(G)\{kanterne i T};
  P:= P(G)\{punkter hvorfra alle kanter er i T};
  IF K≠Ø THEN
    BEGIN
      Find sammenhængskomponenterne
      G1,G2,...,Gk i grafen med punktmængde
      P og kantmængde K, og find punkter
      x1,x2,...,xk, hvor xi ∈ P(Gi) for
      i = 1,2,...,k;
      FOR i:=1 TO k DO
      Find v.h.a. ALGORITME EULER-TUR II
      en lukket Euler-tur Ti i Gi
      startende i xi;
      T:= T med T1,T2,...,Tk indsat ved
      x1,x2,...,xk
    END
END.
```

↑

↓ Den sidste version af beviset for ⇑ (T længst mulig) ser på forhånd ud til ikke at kunne oversættes til en algoritme. En nærmere analyse viser dog, at vi ved at være mere omhyggelige, når kanterne til det første T vælges, kan sikre, at T selv bliver hele Euler-turen. Vi skal nu se hvordan.

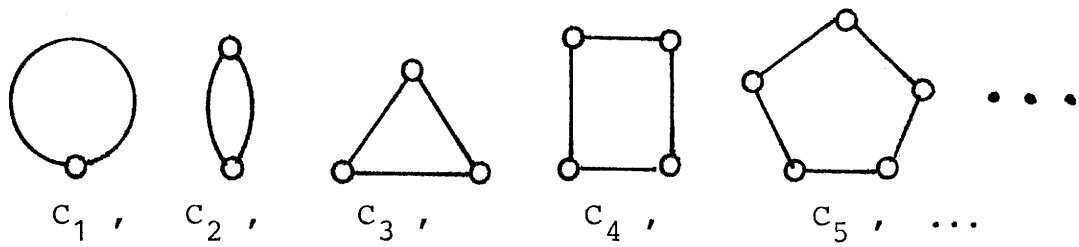
En *bro* i en graf G er en kant (x,y), hvor fjernelse af kanten fra G giver en graf $G-(x,y)$ med x og y i hver sin sammenhængende del af $G-(x,y)$.

Eksempel 1.3 I de tre grafer



er alle kanter mærket med b broer. G_3 er et eksempel på et *træ*, dvs. en sammenhængende graf, hvor alle kanter er broer.

Opgave 1.4 Vis at en kant k i en graf G er en bro hvis og kun hvis k ikke er med i en *kreds* i G , dvs. en graf af følgende type:



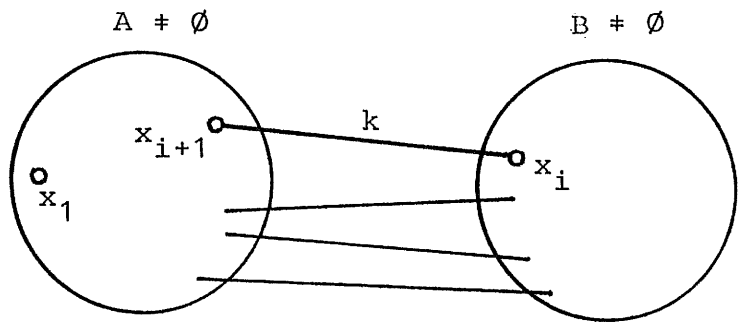
Opgave 1.5 Vis at en graf med en bro altid har punkter med ulige valens (vink: man kan benytte enten Sætning 1.2 eller Sætning 1.3).

Opgave 1.6 Lad G være en sammenhængende graf med mindst én kant. Vis at G har en lukket Euler-tur hvis og kun hvis G er en kant-disjunkt forening af kredse (dvs. G kan opdeles i kredse, som to og to ingen kanter har fælles).

Opgave 1.7 Vis at et træ med ≥ 2 punkter altid har to punkter med valens 1 (vink: enten pr. induktion eller ved at se på endepunkterne i en længst mulig tur). Vis dernæst, at et træ med n punkter altid har $n-1$ kanter.

Når vi i begyndelsen af beviset for $\Uparrow$ vælger T således, at den næste kant (x_i,x_{i+1}) ikke er en bro i $G-(x_1,x_2)-(x_2,x_3)-\cdots-(x_{i-1},x_i)$, medmindre dette ikke kan undgås, så bliver T en Euler-tur i G.

I beviset herfor vil vi benytte notationen fra beviset for $\Uparrow$. Antag at T ikke bliver en Euler-tur. Lad os dele punkterne i G i to klasser: klassen A er de punkter, hvor alle kanter fra et sådant punkt er med i T, og klassen B er resten. Startpunktet x_1 ligger i A (thi ellers kan T udvides) og alle punkter fra $G_1,G_2,\ldots,G_k$ giver netop B.



Lad k være den sidste kant fra T, som har et punkt i B som et endepunkt. Vi kan skrive $k = (x_i,x_{i+1})$, hvor $x_i \in B$ og $x_{i+1} \in A$, da T ender i $x_1 \in A$. Så er (x_i,x_{i+1}) en bro i $G-(x_1,x_2)-\cdots-(x_{i-1},x_i)$. Men da vi stod i x_i og valgte (x_i,x_{i+1}) kunne vi i stedet have valgt en kant fra G_j for et eller andet j, og en sådan kant ville ikke have været en bro, da G_j ikke har broer (jfr. Opgave 1.6). Det giver en modstrid, altså er T en Euler-tur. ∎

Lidt mere formelt opstillet ser algoritmen således ud:

ALGORITME EULER-TUR III

INPUT: Sammenhængende graf G med mindst en kant og med alle valenser lige. Et fast punkt x i G.

OUTPUT: En lukket Euler-tur i G startende i x.

```
BEGIN
  m:= |K(G)| ;
  y:= x ; (*y skal betegne det punkt vi er nået
           til på turen *)
  K:= K(G) ; (*K skal betegne mængden af kanter
              endnu ikke gennemløbet *)
  T:= Ø ; (*T skal betegne følgen af kanter alle-
           rede gennemløbet *)
  FOR i:= 1 TO m DO
    BEGIN
    Vælg kant k_i = (y,u) fra K incident med
    y således, at k_i ikke er en bro i den graf,
    som kanterne i K og deres endepunkter dan-
    ner, medmindre dette ikke kan undgås;
    Tilføj k_i til T ;
    K:= K\{k_i} ;
    y:= u
    END
END.
```

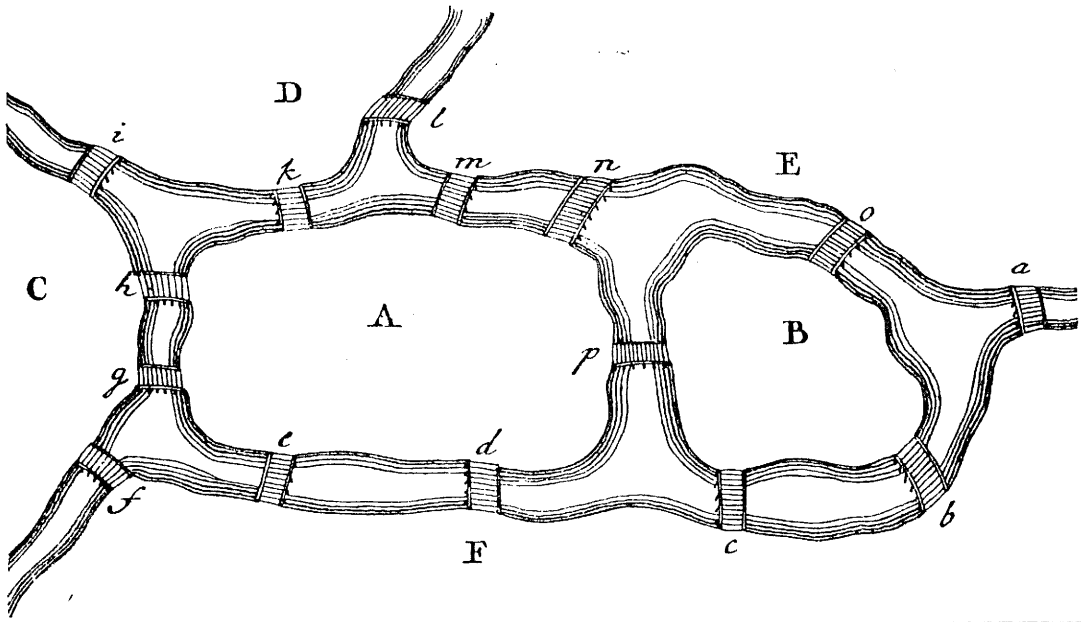
Ved videreudvikling af denne algoritme til et computer-program fås en algoritme, som i kompleksitet er dårligere end den tidligere givne algoritme, idet vi nu for hver kant skal afgøre om den er en bro eller ej, dvs. vi skal se om en vis graf minus kanten er sammenhængende eller ej. Dette arbejde er ikke "konstant" for den enkelte kant, idet det afhænger af grafens størrelse.

For små eksempler med håndkraft er EULER-TUR III dog mere tilfredsstillende end I ved at give turen "fra en ende af" uden senere "reparationer".

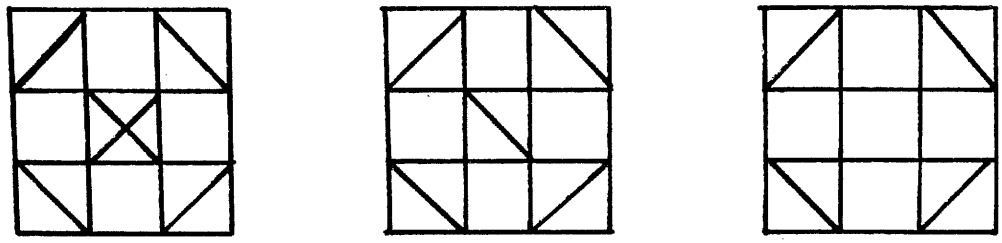
Opgave 1.8 Lad G være en sammenhængende graf med mindst én kant og alle valenser lige, og lad x være et punkt i G. Under hvilke omstændigheder kan vi være helt sikre på, at WHILE-løkken i EULER-TUR I kun gennemløbes én gang ? (Vink: Vis, at dette sker hvis og kun hvis G-x ikke indeholder kredse, d.v.s. hver sammenhængende del af G-x er et træ, jvf. Opgave 1.5).

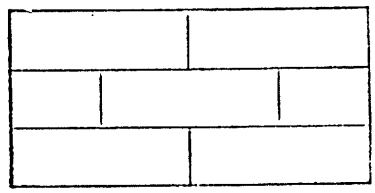
Vi har indtil nu kun beskæftiget os med lukkede Euler-ture. En sætning om åbne Euler-ture svarende til Sætning 1.3 (nu skal der være præcis to punkter af ulige valens) kan vises meget let ved at benytte Sætning 1.3.

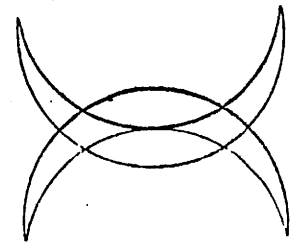
Opgave 1.9 Kan man gå en Euler-tur i denne by fra Euler's artikel (1736) ?

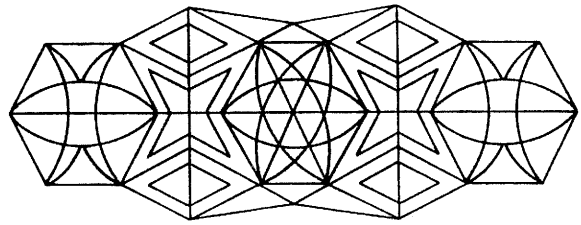


Opgave 1.10 Hvilke af følgende figurer kan tegnes i én streg ? I hvilken figur skal blyanten løftes flest gange ?









Også for *orienterede grafer*, hvor hver kant skal gennemløbes i en på forhånd fastsat retning, kan problemet løses à la Sætning 1.3. Betingelsen er nu, at udgangsvalensen $v^+(x,G)$ skal være lig indgangsvalensen $v^-(x,G)$ for alle punkter x.

Et andet eksempel er behandlingen af Euler-ture i *blandede* grafer, d.v.s. grafer hvor nogle kanter er ikke-orienterede, mens andre er orienterede. Her blev en god sætning,så vidt jeg ved, først fundet i 1962:

SÆTNING 1.4 (Ford og Fulkerson 1962) Lad G være en sammenhængende blandet graf.

G har en lukket Euler-tur

$\Updownarrow$

1) $v^+(x,G) + v^-(x,G) + v(x,G)$ lige for alle $x \in P(G)$.

2) $|v^+(S,G) - v^-(S,G)| \leq v(S,G)$ for alle delmængder S af $P(G)$.

Her betyder $v^+(S,G)$ antal orienterede kanter, som starter i S og slutter udenfor S; $v^-(S,G)$ antal orienterede kanter, som starter udenfor S og slutter i S; $v(S,G)$ antal ikke-orienterede kanter mellem punkter i og udenfor S.

Opgave 1.11 Vis den lette vej $\Downarrow$. Vis med et eksempel, at det ikke er nok for $\Uparrow$ kun at have 1) og 2) for alle S med $|S| = 1$.

Sætningen er en god sætning. Lad os antage, at en lærer giver sine elever en kæmpe blandet graf og beder dem afgøre, om den har en lukket Euler-tur eller ej. For eleverne er det at benytte sætningen ikke umiddelbart let: hvis G har n punkter, er antal mulige mængder S lig med 2^n. Når vi alligevel kalder sætningen *god*, er det fordi den giver læreren en let mulighed for at kontrollere, om et givet svar er korrekt: hvis svaret er "ja , der er en Euler-tur", så skal eleverne blot vise, hvordan den gennemløbes; hvis svaret er "nej, der er ingen Euler-tur", så skal eleverne blot vise et x eller et S , som ikke opfylder 1) eller 2).

At der er en god sætning betyder altså ikke umiddelbart, at der er en god algoritme. Imidlertid er beviset for Sætning 1.4 igen samtidig et bevis for, at en bestemt god algoritme giver korrekt svar. Vi behandler ikke dette her.

I.3 POSTBUD-PROBLEMET

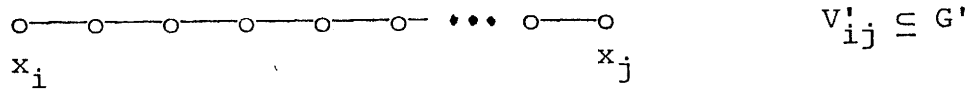
Postbud-problem-typen kan formuleres som spørgsmålet om i en givet sammenhængende graf G at finde en lukket rute, som indeholder hver kant mindst én gang, og som gennemløber et mindst muligt antal kanter ialt. I en mere generel udgave af problemet er hver enkelt kant (x,y) i G mærket med et tal $\ell(x,y) > 0$, kaldet længden af (x,y). Problemet er så at finde en lukket rute, som indeholder hver kant mindst én gang og som er af minimal samlet længde.

Såfremt grafen har en lukket Euler-tur, er denne naturligvis en rute af minimal samlet længde, idet hver kant er indeholdt i ruten præcis én gang.

Såfremt grafen ikke har en lukket Euler-tur, vil det være nødvendigt på en korteste rute at gennemløbe mindst én af kanterne mere end én gang. Benyttes en kant k ialt p gange, kunne vi registrere det ved til grafen at tilføje $p-1$ kopier af k. Hverved opstår en graf G^*, som har en lukket Euler-tur.

Vha. Sætning 1.2 kan vi derfor omformulere postbud-problemtypen således: Tilføj til en sammenhængende graf G kopier af kanter fra G således, at den derved fremkomne graf G^* har alle valenser lige, og således, at de tilføjede kopier er af minimal samlet længde.

Vi vil nu analysere, hvordan grafen G' bestående af de tilføjede kopier ser ud. Hvis G kun har lige valenser, er $K(G') = \emptyset$. Ellers har G iflg. Sætning 1.2 et lige antal punkter $x_1, x_2, \ldots, x_{2i-1}, x_{2i}$ $(i \geq 1)$ af ulige valens. I G' har $x_1, x_2, \ldots, x_{2i-1}, x_{2i}$ også som de eneste punkter ulige valens. Hver sammenhængende del af G' indeholder et lige antal af $x_1, x_2, \ldots, x_{2i-1}, x_{2i}$ (igen iflg. Sætning 1.2), og to af dem x_i og x_j er derfor forbundet med en *vej* V'_{ij} i G', d.v.s. G' har en delgraf af formen



Fjernes kanterne i V'_{ij} fra G' har den resterende graf $2i-2$ punkter med ulige valens. Er $i \geq 2$ er to af disse punkter derfor forbundet med en vej i den resterende del af G', o.s.v. Da foreningen af de i veje, som opstår på denne måde, præcis har $x_1, x_2, \ldots, x_{2i-1}, x_{2i}$ som punkter af ulige valens, og G' har minimal samlet længde, består G' af præcis disse i veje.

Konklusionen er altså, at hvis G har $2i$ punkter af ulige valens, så består G' af kopier af i veje fra G, hvor disse veje i G forbinder punkterne af ulige valens to og to. Enhver af disse veje er en korteste vej mellem sine endepunkter i G, thi ellers kunne vi fjerne kopien af vejen fra G' og erstatte den med en kopi af en korteste vej i G mellem de samme to punkter. Da G' har samlet minimal længde, er det ikke muligt.

Denne analyse viser, at vi kan løse postbud-problem-typen med følgende algoritme:

ALGORITME POSTBUD

INPUT: Sammenhængende graf G med mindst én kant. En længde $\ell(x,y) > 0$ tilknyttet hver kant (x,y).

OUTPUT: En lukket rute R i G, som benytter hver kant mindst én gang, og som har minimal samlet længde.

BEGIN

Find korteste veje V_{ij} i G mellem alle par af punkter x_i og x_j af ulige valens;

Find en pardannelse P i mængden af punkter af ulige valens således, at den samlede længde af de veje V_{ij}, hvor x_i og x_j danner par, er minimal;

Lad G^* være grafen opnået fra G ved at tilføje kopier af kanterne i de veje V_{ij},

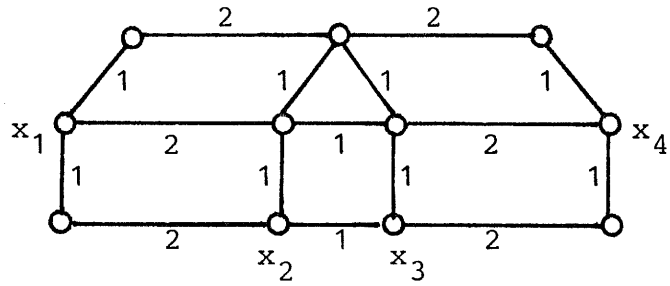
hvor x_i og x_j danner par i P ;

Find en lukket Euler-tur T i G*
vha. ALGORITME EULER-TUR ;

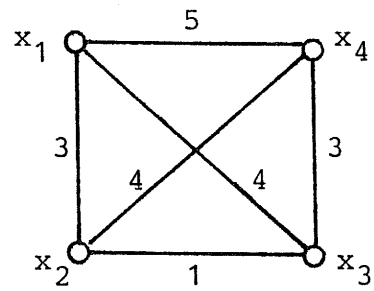
Erstat i T enhver kopi af en kant i G
med den oprindelige kant i G. Derved
dannes ruten R i G

END.

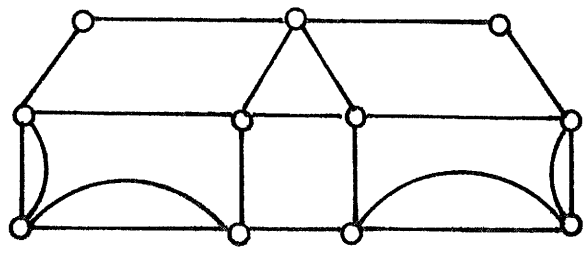
Eksempel 1.4 Lad G være grafen



G har 4 punkter x_1, x_2, x_3, x_4 med ulige valens, og længderne af korteste veje mellem alle par af disse er angivet på kanterne i følgende *komplette 4-graf* K_4 :



For de tre mulige pardannelser $\{(x_1,x_2), (x_3,x_4)\}$, $\{(x_1,x_3), (x_2,x_4)\}$ og $\{(x_1,x_4), (x_2,x_3)\}$ er den samlede længde af de korteste veje henholdsvis 6, 8 og 6. En mulig graf G* er derfor



og den korteste lukkede rute R for et postbud har længde 28. R kan findes som en lukket Euler-tur i G*.

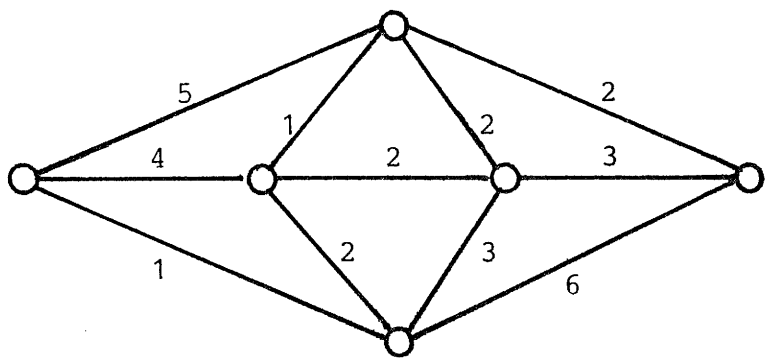
Det ses, at for at ALGORITME POSTBUD kan kaldes en god algoritme, er det nødvendigt at etablere gode algoritmer for korteste veje og for vægtet pardannelse. Det kan gøres, og vi skal vende tilbage til det i senere afsnit.

Vi kan også udtrykke ovenstående ved at sige, at postbud-problem-typen er blevet *reduceret* til problem-typerne "korteste vej" og "vægtet pardannelse". Sådanne reduktioner af problemtyper er en ofte anvendt metode, enten til at vise, at en problem-type kan løses v.h.a. en god algoritme, eller til at vise, at en problem-type er vanskelig (vi har ovenfor set, at hvis postbud-problemet er meget vanskeligt, så må enten "korteste vej" eller "vægtet pardannelse" også være det).

Ønskes en åben rute i stedet for en lukket, evt. mellem to på forhånd fastlagte punkter, kan dette opnås ved en simpel modifikation af metoden ovenfor.

Opgave 1.12 Løs problemet i Eksempel 1.1. Svaret afhænger af, om der kræves en lukket eller åben rute, og af hvilke krav der stilles til endepunkterne af en åben rute.

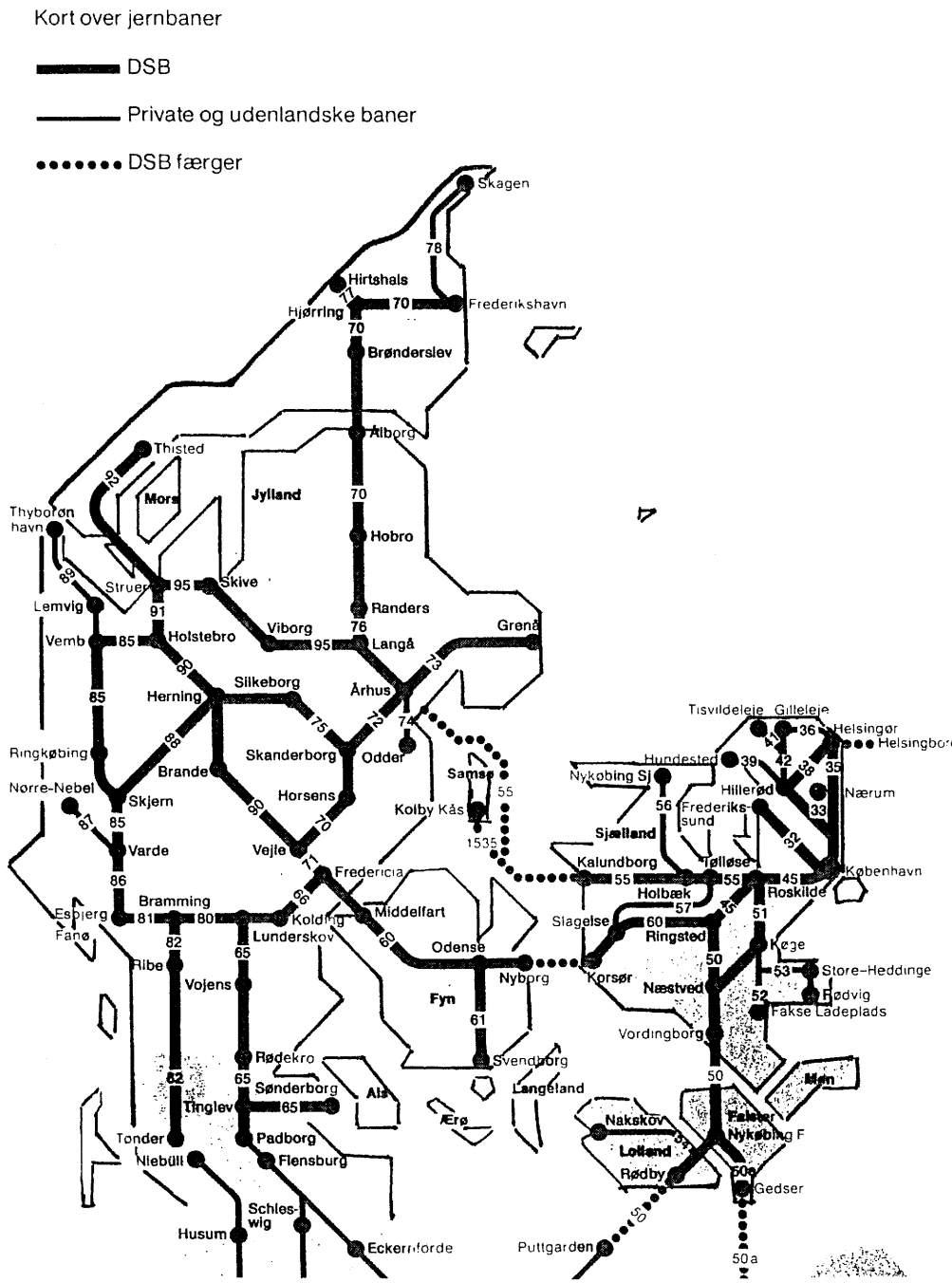
Opgave 1.13 Find en kortest mulig lukket rute for et postbud i grafen



Opgave 1.14 UD AT SE MED DSB

En rejse rundt i Danmark (med tog og færge) ønskes planlagt, så den starter og ender samme sted, så samtlige DSB-strækninger dækkes mindst én gang, og så den samlede rute bliver kortest mulig. Hvilke strækninger (bortset fra de trivielle som f.eks. Struer-Thisted, Odense-Svendborg, Roskilde-København o.s.v.) skal tilbagelægges to gange ?

(N.B. Tallene på de enkelte strækninger er ikke afstande, men henvisninger til køreplanen!)



Opgave 1.15 Vis, at to forskellige af vejene V_{ij} i ALGORITME POSTBUD ikke kan have en fælles kant (Vink: vis, at hvis to af vejene har en fælles kant, så eksisterer der en bedre pardannelse).

Vis, at i en korteste rute for postbudet benyttes en kant aldrig tre eller flere gange).

For orienterede grafer kan postbud-problem-typen løses helt tilsvarende. Men for blandede grafer, selv med alle kantlængder lig med 1, kendes ingen god algoritme. Det vides, at dette problem tilhører den meget vanskelige klasse af NP-komplette problemer, som vi skal møde senere, og det er mest nærliggende at tro, at postbud-problemet for blandede grafer slet ikke har nogen god algoritme som løsning.

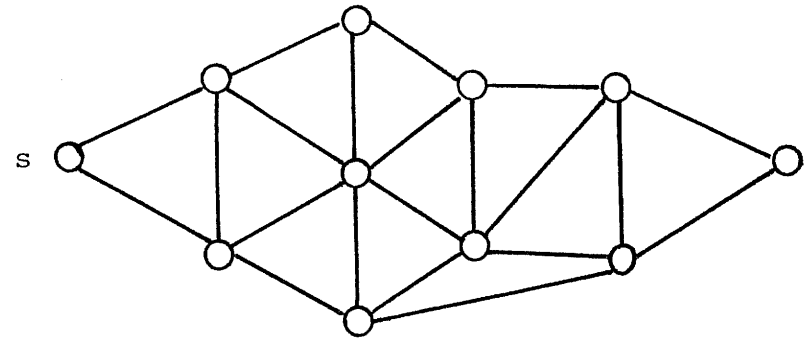
1.4 KORTESTE VEJE.

Vi starter med en ikke-orienteret sammenhængende graf G uden sløjfer, og vi ønsker fra et bestemt punkt s at finde veje med så få kanter som muligt til alle andre punkter i grafen. Antal kanter i en vej kaldes *længden af vejen*. Som output af en algoritme ønskes altså for hvert punkt i grafen et tal $m(x_i)$ og evt. en vej fra s til x_i.

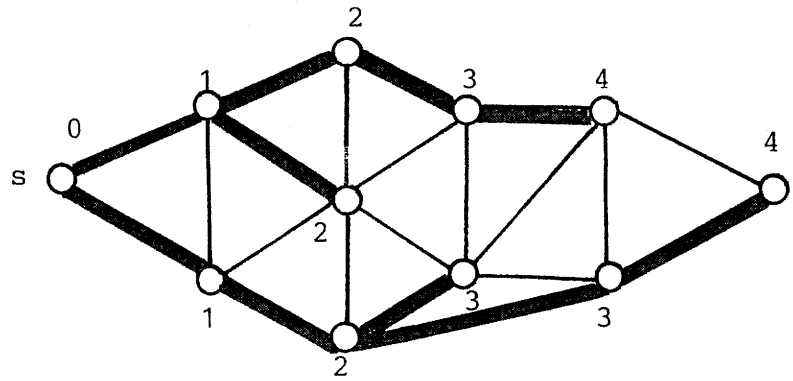
Det er klart, at vi skal starte med at sætte $m(s) = 0$. Derefter mærkes alle punkter x_i forbundet til s med en kant med $m(x_i) = 1$. Alle ikke-mærkede punkter y_i har så afstand ≥ 2 fra s. Alle ikke-mærkede y_i, som er forbundet til et allerede mærket punkt, gives så mærket $m(y_i) = 2$, osv.

Denne form for mærkning kaldes *bredde først søgning* (BFS). For at holde øje med, hvor en korteste vej er, angives for hvert punkt y med mærke $m(y) \geq 1$ en kant $k(y)$, som er en kant fra et punkt med mærket $(m(y) - 1)$ til y. En sådan kant er sidste kant på en mulig korteste vej fra s til y.

Eksempel 1.5 På følgende graf G



giver den antydede algoritme følgende mærkning af punkterne (hvor kanterne k(x) er tegnet tykke):



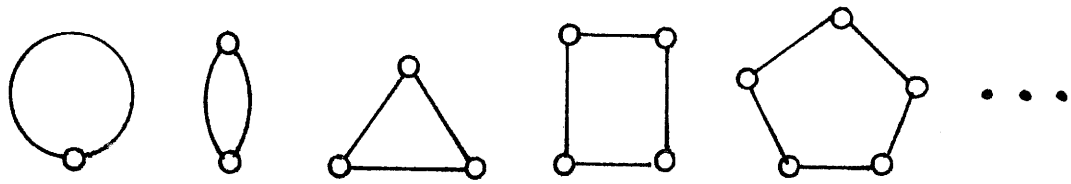
De tykke kanter danner et såkaldt udspændende træ i G

Når vi mærker et punkt v i den følgende mere præcise udformning af algoritmen, så tilføjer vi samtidig til en liste L alle de naboer til v , som vi ikke tidligere har mødt. Det næste punkt, der skal mærkes, bliver så hele tiden det punkt, som står forrest i L. Listen L kaldes også en *kø* , idet et punkt, der kommer før ind i L end et andet punkt også bliver mærket og dermed fjernet fra L før det andet ("først ind - først ud").

```
ALGORITME KORTESTE VEJ I

INPUT:   Sammenhængende graf  G  uden sløjfer
         med mindst ét punkt  s .

OUTPUT:  En mærkning af  G's  punkter med
         antal kanter i en korteste vej
         fra  s  i  G .

BEGIN
  L:= [s]; (*L skal betegne en liste af ikke-mærkede
           punkter, der i  G  er forbundet til alle-
           rede mærkede punkter *)
  WHILE  L ≠ Ø  DO
    BEGIN
      v:= første punkt i  L ;
      L:= L  med  v  fjernet;
      m(v):= m(u)+1 , hvor  k(v) = (u,v)
             (for  v=s  sættes  m(v)=0);
      FOR  alle  punkter x ikke i L,
           der er forbundet til  v  med
           en kant i  G
      DO
        BEGIN
          Tilføj  x  til  L ;
          k(x):= kant fra v til x
        END
    END
END.
```

Benyttes ALGORITME KORTESTE VEJ I på en graf G, som ikke er sammenhængende, standser algoritmen med $L = \emptyset$ før alle punkter er mærkede. Hvis vi i algoritmen tæller antal mærkede punkter, og hvis dette tal til slut er mindre end antal punkter i G, så er G ikke sammenhængende. ALGORITME KORTESTE VEJ I kan derfor benyttes til at undersøge, om en graf er sammenhængende eller ej.

Antag nu igen, at G er sammenhængende. Kanterne k(x) for alle punkter $x \neq s$ danner en delgraf T i G, som er et såkaldt *udspændende træ i G*. At T er *udspændende* i G betyder, at T er en delgraf af G indeholdende alle G's punkter. At T er et *træ* betyder, at T er sammenhængende og at T ikke indeholder nogen *kreds*, d.v.s. en graf af følgende type:



C_1, C_2, C_3, C_4, C_5, ...

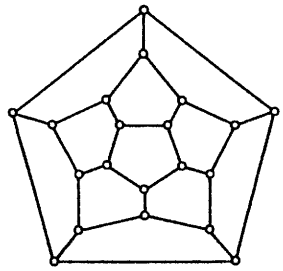
Af grafen T, der dannes af kanterne k(x) for alle punkter $x \neq s$ og deres endepunkter, virkelig har egenskaberne beskrevet ovenfor kan indses således:

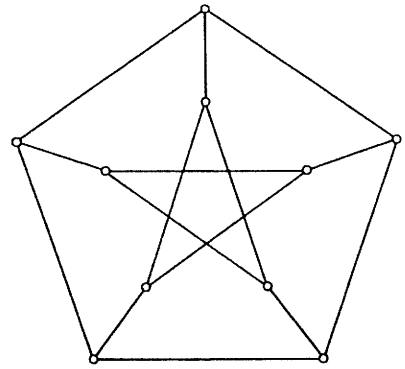
For det første forbinder kanten k(x) for alle $x \neq s$ punktet x med et punkt v med $m(v) = m(x) - 1$. Ved at fortsætte "baglæns" med k(v) nås et punkt w med $m(w) = m(v) - 1 = m(x) - 2$. Fortsættes således, nås til sidst et punkt med $m = 0$, og dette punkt må være s, idet s er det eneste punkt med $m = 0$. Kanterne k(x) har altså tilsammen alle punkter i G som endepunkter, og ethvert punkt er forbundet med en vej i T til s. Altså er T en sammenhængende og udspændende delgraf i G.

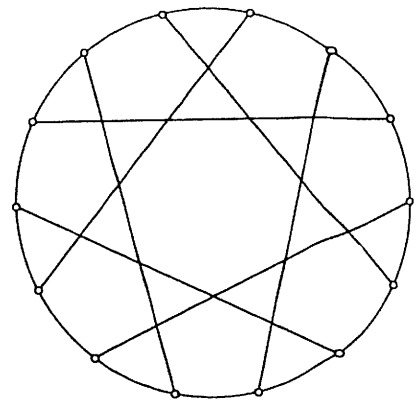
For det andet indeholder T ingen kreds. Hvis T nemlig indeholder en kreds C, så lad x være det punkt på C, hvor m(x) er størst. Så er x forbundet med to kanter til punkter med mindre mærker end m(x). Men ethvert punkt x i G med $x \neq s$ er forbundet til præcis ét punkt v med et mindre mærke end x, nemlig det andet endepunkt af k(x). Dette giver en modstrid.

Altså er T virkelig et udspændende træ i G. En korteste vej fra s til x i G er netop forbindelsen fra s til x i T (i et træ T er der præcis én vej mellem to punkter, idet ingen vej ville gøre T ikke sammenhængende og mindst 2 forskellige veje ville skabe en kreds). Det kunne derfor være naturligt at lade k(x) for alle $x \neq s$ være en del af ALGORITME KORTESTE VEJ I's output.

Opgave 1.16 Anvend ALGORITME KORTESTE VEJ I på følgende grafer. Find i hvert tilfælde et udspændende træ T af korteste veje.







De punkter som får det samme mærke j ved ALGORITME KORTESTE VEJ I siges at ligge i *den j'te afstandsklasse mht. punktet* s . Disse afstandsklasser spiller en vigtig rolle i grafteori. Ét enkelt eksempel er nævnt i den følgende opgave:

Opgave 1.17 Vis

G er *to-delt*, dvs. G's punkter kan deles i to klasser A og B , så alle kanter forbinder et A-punkt med et B-punkt

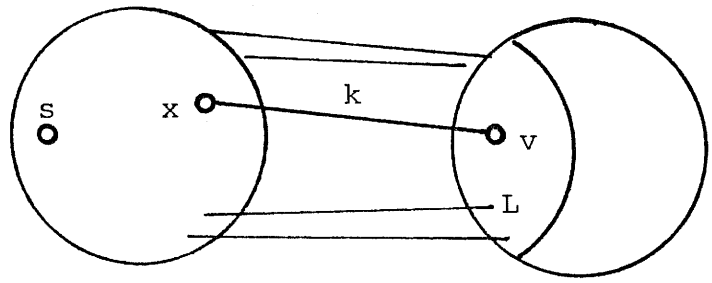
$\Updownarrow$

G har ingen ulige kredse.

(Vink: $\Downarrow$ er let . $\Uparrow$ vises ved først at mærke G's punkter som angivet i ALGORITME KORTESTE VEJ I. Vis så at to punkter med samme mærke j aldrig er forbundet i G . Klassen A kan vælges som alle punkter med et lige mærke).

Vi gør nu situationen mere generel, idet der til hver kant k i grafen G knyttes en længde $\ell(k) > 0$. Vi vil forsøge at videreudvikle ALGORITME KORTESTE VEJ I til at finde en korteste vej fra ét punkt s til alle andre punkter i G, hvor "korteste" nu betyder, at de kanter, der indgår i vejen, tilsammen har mindst mulig længde.

Det som bl.a. kendetegner ALGORITME KORTESTE VEJ I er, at punkterne markeres ét efter ét med længden af en korteste vej. Forsøger vi det samme i den situation, vi nu betragter, er situationen før et gennemløb af WHILE-løkken følgende:



Mærkede punkter Ikke-mærkede punkter

Vi forudsætter, at de mærkede punkter alle er mærkede med de korrekte længder af korteste veje fra s. Endvidere forudsætter vi, at L består af alle ikke-mærkede punkter forbundet med kanter til mærkede punkter.

For et punkt v i L forbundet til et mærket punkt x med en kant k er der en forbindelse fra s til v af længde $m(x) + \ell(k)$. En korteste vej til v har altså længde $\leq m(x) + \ell(k)$. Vælges v, x og k nu, så $m(x) + \ell(k)$ er mindst muligt, er tallet $m(x) + \ell(k)$ længden af en korteste vej fra s til v, thi en vilkårlig vej V fra s til v skal benytte en kant $k' = (x',v')$ fra et mærket punkt x' til et ikke mærket punkt v'. Det giver:

længden af V

$=$ længden af delen af V fra s til x' + $\ell(k')$ +
længden af delen af V fra v' til v

$\geq m(x') + \ell(k') + 0$

$\geq m(x) + \ell(k)$

Det viser, at vælger vi $v \in L$ som angivet, så er længden af en korteste vej fra s til v lig med $m(x) + \ell(v)$, og vi kan mærke v med $m(x) + \ell(v)$.

Algoritmen bliver derfor næsten som ALGORITME KORTESTE VEJ I. Ændringerne bliver

a) v vælges ikke længere som første element i L, men som det v, der gør $m(x) + \ell(k)$ mindst mulig blandt alle kanter $k = (x, v)$ fra et mærket x til et ikke-mærket v,

b) $m(v)$ sættes til $m(x) + \ell(k)$,

c) for ikke at udregne tallene $m(x) + \ell(k)$ flere gange end højst nødvendigt beholder vi for et ikke-mærket $v \in L$, dette tal som et ikke-permanent mærke $M(v)$ på v.

En algoritme af denne type for korteste veje blev først beskrevet af den hollandske datalog E.W. Dijkstra i 1959 og kaldes derfor af og til *Dijkstra's algoritme*.

ALGORITME KORTESTE VEJ II

INPUT: Sammenhængende graf G uden sløjfer med mindst et punkt x og en længde $\ell(k) > 0$ tilknyttet hver kant k.

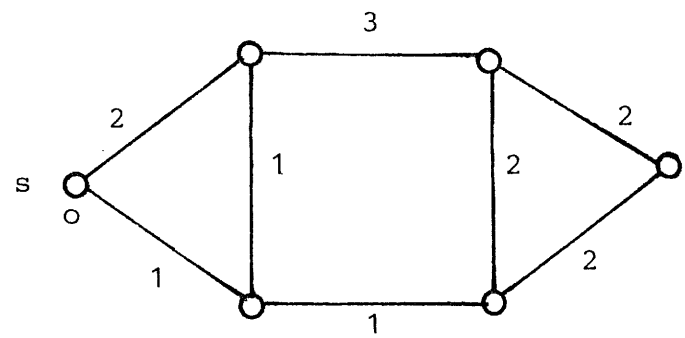
OUTPUT: En mærkning af G's punkter med længden af en korteste vej fra s i G.

```
BEGIN
  L := {s};
  (* L skal betegne mængden af ikke-mærkede
  punkter, der i G er forbundet til allerede
  mærkede punkter *)
```

```
M(s) := 0 ;
WHILE  L ≠ ∅  DO
   BEGIN
     Vælg  v ∈ L  så  M(v)  er mindst mulig;
     L := L\{v};
     m(v) := M(v);
     FOR  alle ikke-mærkede punkter  x , der er
     forbundet til  v  med en kant  k  i  G .
     DO
       BEGIN
         IF (x ∉ L) eller
            (x ∈ L   og  m(v) + ℓ(k) < M(x))
         THEN
           BEGIN
             M(x) := m(v) + ℓ(k);
             k(x) := k;
             IF x∉L THEN L:= LU{x}
           END
       END
   END
END.
```

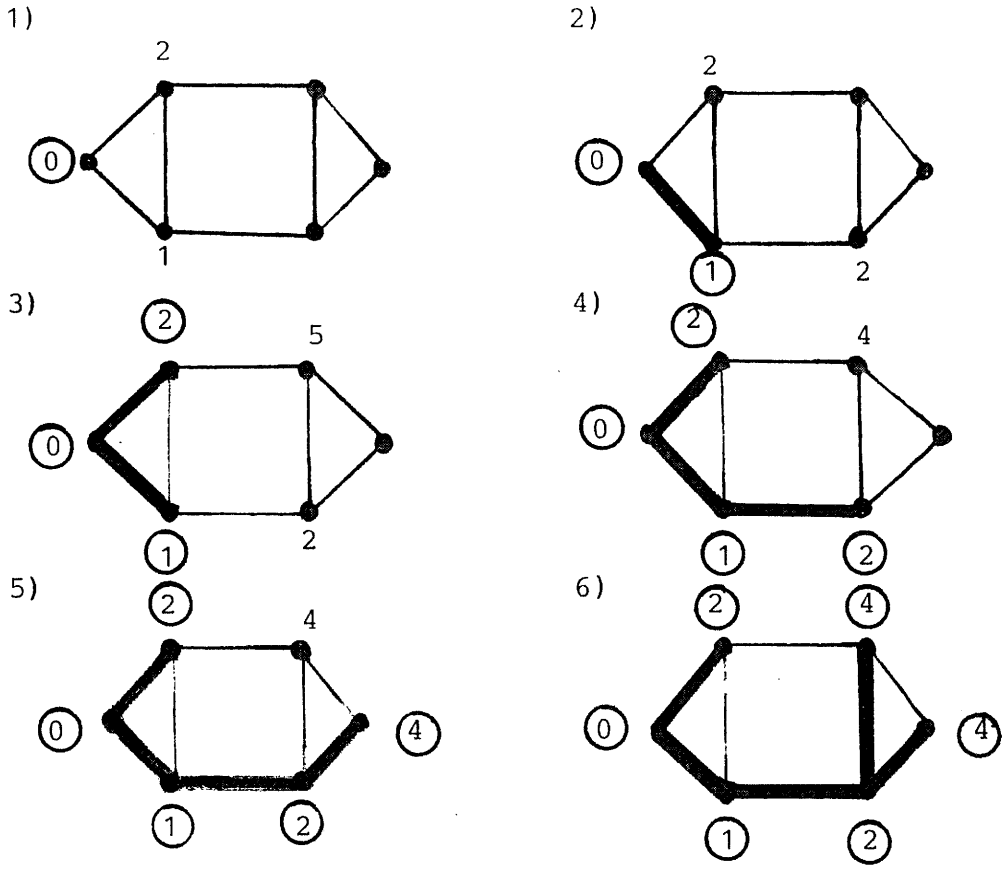
Kanterne $\{k(x) \mid x \in P(G)\setminus\{s\}\}$ danner igen et udspændende træ T indenfor hvilket korteste veje fra s til alle andre punkter findes.

Eksempel 1.6 Lad G være grafen

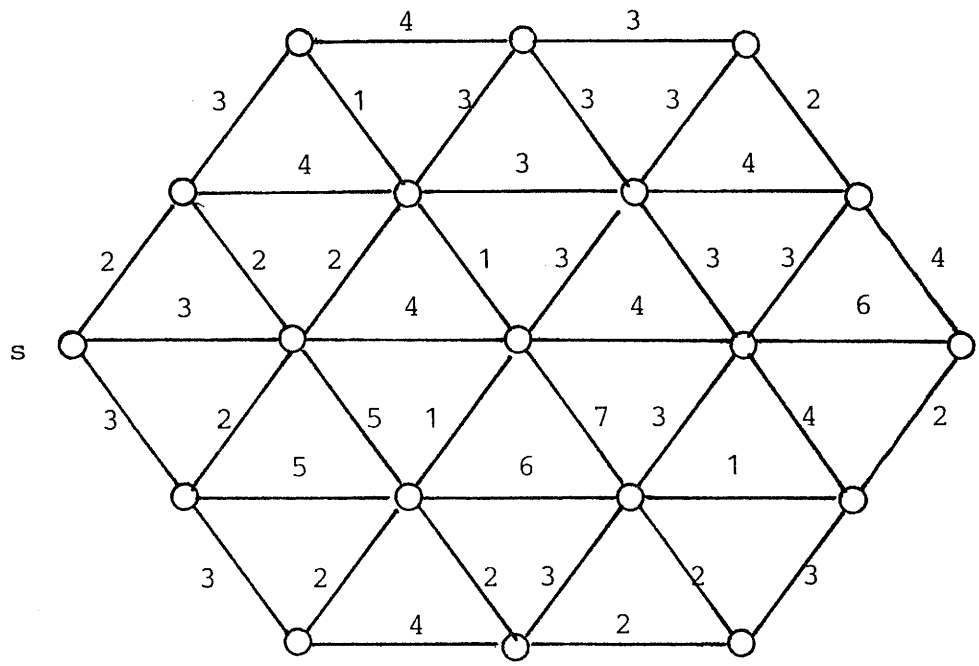


Korteste veje fra s bestemmes v.h.a. algoritmen, idet permanente mærker på punkterne er skrevet i en cirkel

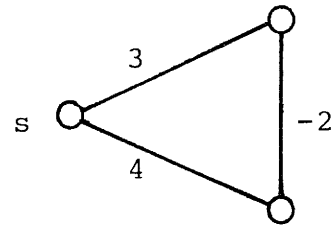
og kanterne k(x) tegnes tykt:



Opgave 1.18 Bestem v.h.a. ALGORITME KORTESTE VEJ II korteste veje fra s til alle andre punkter i følgende graf, og find et udspændende træ af sådanne veje:



Opgave 1.19 Anvend ALGORITME KORTESTE VEJ II på grafen



Hvad går galt ? Hvor benyttede vi under udviklingen af algoritmen, at $\ell(k) > 0$ for alle kanter k ?

ALGORITME KORTESTE VEJ II er en god algoritme. WHILE-løkken gennemløbes $|P(G)|$ gange. For hvert gennemløb skal der foretages højst $|P(G)|$ sammenligninger for at finde v, og FOR-løkken gennemløbes ligeledes højst $P(G)$ gange (én gang for hvert ikke-mærket x). Et gennemløb af FOR-løkken kræver højst et konstant antal skridt (uafhængigt af G's størrelse). Alt i alt følger, at det samlede antal skridt højst er konstant $\cdot |P(G)|^2$, d.v.s. algoritmen er højst af kompleksitet $O(|P(G)|^2)$.

Antal mulige veje fra s til t i en graf med n punkter og en kant mellem hvert par af punkter er mindst $(n-2)!$. ALGORITME KORTESTE VEJ II er derfor langt overlegen den metode at undersøge alle mulige veje og tage den korteste, jvf. opgave 1.1 side 1.4.

↓ Ønskes en korteste vej mellem alle par af punkter, kan dette opnås ved at udføre ALGORITME KORTESTE VEJ II ialt $|P(G)|$ gange. Det giver en algoritme af kompleksitet $O(|P(G)|^3)$.

En helt anden type algoritme til løsning af dette sidste problem er følgende algoritme, også af kompleksitet $O(|P(G)|^3)$. Algoritmen formuleres mest naturligt for orienterede grafer:

ALGORITME KORTESTE VEJ III

INPUT: Sammenhængende orienteret graf G med $P(G) = \{x_1, x_2, \ldots, x_n\}$. Ialt n^2 tal $\ell(i,j) \geq 0$, hvor $\ell(i,j)$ er en længde tilknyttet kanten fra x_i til x_j. Hvis der ingen kant er fra x_i til x_j i G sættes $\ell(i,j) = \infty$. Endelig sættes $\ell(i,j) = 0$ for $i = j$.

OUTPUT: Ialt n^2 tal $d(i,j)$, hvor $d(i,j)$ er længden af en korteste vej fra x_i til x_j i G.

```
BEGIN
  FOR alle i og j DO d(i,j):= ℓ(i,j);
  FOR j:= 1 TO n DO
    FOR i:= 1 TO n DO
      FOR k:= 1 TO n DO
      IF d(i,j) + d(j,k) < d(i,k)
      THEN d(i,k):= d(i,j) + d(j,k)
END.
```

Opgave 1.20 Vis at ALGORITME KORTESTE VEJ III giver det ønskede OUTPUT (Vink: Vis at når vi har gennemløbet den yderste FOR-løkke for j, så er værdien $d^j(i,k)$ af $d(i,k)$ lig med længden af en korteste vej i G mellem x_i og x_k, hvor alle indre punkter på vejen er blandt $x_1, x_2, \ldots, x_j$ (dette vises pr. induktion over j)).

Opgave 1.21 Anvend ALGORITME KORTESTE VEJ III på eksemplet i Eksempel 1.6 med passende orienteringer.

Opgave 1.22 Hvordan kan ALGORITME KORTESTE VEJ III bringes til at fungere for en ikke-orienteret graf ?

Opgave 1.23 Hvad er det præcise antal sammenlægninger og sammenligninger ved en udførelse af ALGORITME KORTESTE VEJ III på en graf G med n punkter ?

I.5 DEN GRÅDIGE ALGORITME

Vi har i det foregående afsnit set udspændende trær for sammenhængende grafer dukke op i forbindelse med korteste veje. Vi vil nu betragte den opgave i en sammenhængende (ikke-orienteret) graf med vægte $\ell(k)$ på de enkelte kanter k at bestemme et udspændende træ af minimal samlet vægt.

Det viser sig her, at den simplest tænkelige metode (for en gangs skyld!) virker: man kan bygge træet op fra en ende af ved hele tiden at tilføje en kant af lavest mulig vægt. En sådan algoritme, hvor det, der er bedst i hvert skridt, d.v.s. på kort sigt, også bliver det bedste på langt sigt, kaldes en *grådig algoritme*.

Fremgangsmåden til bestemmelse af et minimalt udspændende træ minder meget om ALGORITME KORTESTE VEJ II. Den eneste ændring er faktisk, at mærket "$m(v) + \ell(k)$" overalt erstattes af "$\ell(k)$". Da tallet $m(v)$ ikke længere spiller en rolle, vil det dog være naturligt også at lave andre justeringer.

Bl.a. viser det sig, at det ikke længere er væsentligt at bygge træet op fra en ende af som vi har gjort det i de tidligere eksempler, hvor der dukkede udspændende trær op. Men en opbygning fra en ende af således, at den del af træet, vi har hele tiden, er sammenhængende, er også korrekt. De to måder svarer til to forskellige måder at karakterisere udspændende trær på i en sammenhængende graf G :

a) et udspændende træ T i G er karakteriseret ved, at $K(T)$ er en maximal mængde af kanter i G uden kredse,

b) et udspændende træ T i G er karakteriseret ved, at $K(T)$ er en minimal mængde af kanter i G , så T er sammenhængende.

Den første synsmåde giver følgende grådige algoritme (kompleksiteten vil bl.a. afhænge af hvordan betingelsen efter IF testes, men det kan gøres så kompleksiteten bliver højst $O(|P(G)|^2)$) :

ALGORITME MINIMALT TRÆ

INPUT: Sammenhængende graf G med mindst én kant og en vægt $\ell(k)$ tilknyttet hver kant.

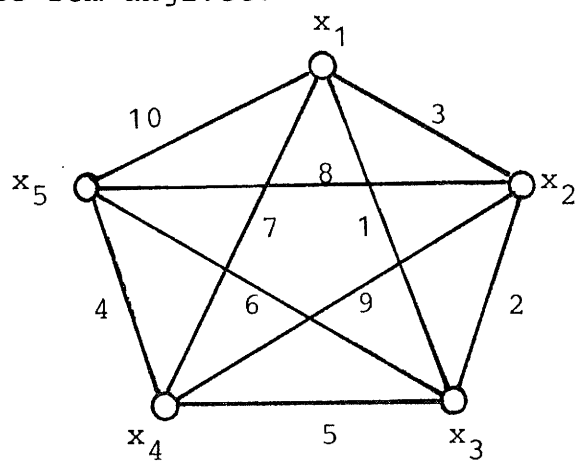
OUTPUT: Mængde T af kanter, som danner et minimalt udspændende træ i G.

```
BEGIN
   T: = Ø;
   K: = K(G);
   WHILE  K ≠ Ø  DO
     BEGIN
      Vælg kant  k = (x,y) ∈ K  med mindst mulig
      ℓ(k);
      IF ∄ kreds af kanter fra  TU{k}
      THEN T: = TU{k};
      K: = K\{k}
     END
 END.
```

Da et træ med n punkter altid har $n-1$ kanter kan betingelsen "K≠Ø" i WHILE-løkken erstattes af "$|T| < |P(G)|-1$".

ALGORITME MINIMALT TRÆ dur også til at finde et maximalt træ, blot skal ordene "minimalt" og "mindst mulig" erstattes af "maximalt" og "størst mulig".

Eksempel 1.7 Lad G være den komplette 5-graf K_5 med kant-vægte som angivet:



ALGORITME MINIMALT TRÆ ser på kanterne i rækkefølge efter stigende vægt og inkluderer næste kant, hvis der ikke opstår en kreds. Man får følgende kanter i træet:

(x_1,x_3) , (x_2,x_3) , (x_4,x_5) , (x_3,x_4).

Udvælges kanterne så træet opbygges sammenhængende, fås (startende i punktet x_5):

(x_5,x_4) , (x_4,x_3) , (x_3,x_1) , (x_3,x_2)

altså samme træ.

Da K_5 har ialt 125 forskellige udspændende trær (K_n har n^{n-2}) er metoden her naturligvis langt overlegen den metode at prøve alle muligheder.

Vi viser nu at algoritmen virker. Lad T være output af algoritmen.

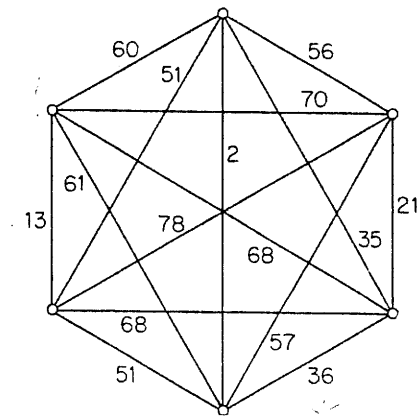
Først vil vi overbevise os om, at kanterne i T og deres endepunkter danner en graf G' , som er et udspændende træ i G . For det første indeholder G' alle punkter i G , thi hvis punktet x ikke var med, så ville en kant k incident med x i G med mindst vægt helt sikkert blive tilføjet til T når den betragtes i algoritmen, og x ville så alligevel være med i G' . For det andet indeholder G' ingen kredse, thi hvis k var den sidste til T tilføjede kant fra en kreds, så skulle k ikke have været tilføjet. For det tredie er G' sammenhængende, thi hvis vi kunne dele G' op i to grafer G'_1 og G'_2 uden nogen kant imellem, så ville en kant k fra G mellem G'_1 og G'_2 med mindst vægt helt sikkert blive tilføjet til T når den betragtes i algoritmen, og k ville så være med i G'. Alt i alt viser dette, at G' er et udspændende træ.

Antag at kanterne i T blev inkluderet i rækkefølgen $k_1,k_2,k_3,\ldots,k_{n-1}$ (et træ med n punkter har $n-1$ kanter, jfr. Opgave 1.7 eller Appendix A), og antag at T ikke er minimal.

Lad T' være et minimalt udspændende træ, som indeholder kanterne $k_1, k_2, \ldots, k_{j-1}$, men ikke k_j, hvor T' er valgt, så j er størst mulig. $T' \cup \{k_j\}$ indeholder præcis én kreds C. Denne kreds C har en kant $k' \notin T$. Grafen $T' \cup \{k_j\} \setminus \{k'\}$ er et nyt udspændende træ T^* med samlet vægt $\ell(T^*) = \ell(T') + \ell(k_j) - \ell(k')$. Da T' er minimal, er $\ell(k') \leq \ell(k_j)$. På den anden side er $\ell(k') \geq \ell(k_j)$, da k' ellers ville være blevet valgt i stedet for k_j som kant i T. Det følger, at $\ell(k') = \ell(k_j)$, og at T^* har samme vægt som T'. T^* er således at minimalt træ og $k_1, k_2, \ldots, k_j$ er alle i T^*. Det strider mod valget af T'. ∎

Opgave 1.24 Bevis at det er muligt i ALGORITME MINIMALT TRÆ at opbygge træet sammenhængende, d.v.s. der startes med et punkt s, og næste kant $k = (x,y) \in K$ vælges med mindst mulig $\ell(k)$, hvor x er med i T og y ikke er med i T. (Vink: beviset ovenfor kan bruges med små ændringer).

Opgave 1.25 Find et minimalt udspændende træ T i følgende grafer. Prøve både at opbygge T sammenhængende og uden dette krav.



Prøv også i grafen i Opgave 1.18.

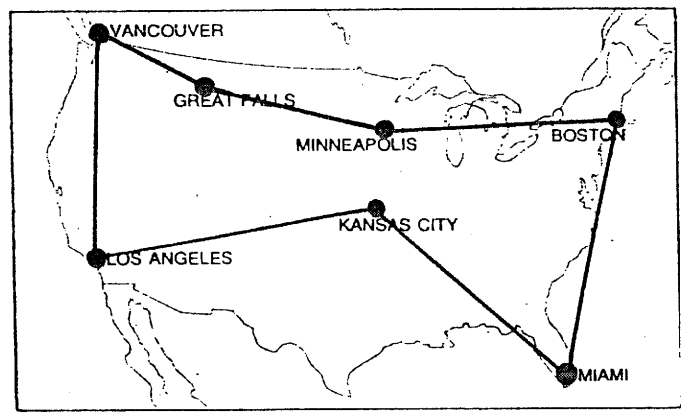
Opgave 1.26 *En udspændende vej* i en graf er en vej, som indeholder alle punkter. Vis ved et eksempel, at det ikke er muligt at finde en minimal udspændende vej ved at være grådig. Hvad med en udspændende kreds ?

I.6 DEN HANDELSREJSENDES PROBLEM

Givet n byer, find en korteste rundrejse for en handelsrejsende, som skal besøge alle byer. Dette problem minder i sin grafteoretiske formulering meget om de problemer, vi tidligere har betragtet. I forhold til postbud-problemet stilles der nu blot det krav, at alle punkterne skal gennemløbes snarere end alle kanter. Og i forhold til minimalt udspændende træ-problemet ønskes nu blot en minimal udspændende kreds, d.v.s. en kreds med alle grafens punkter.

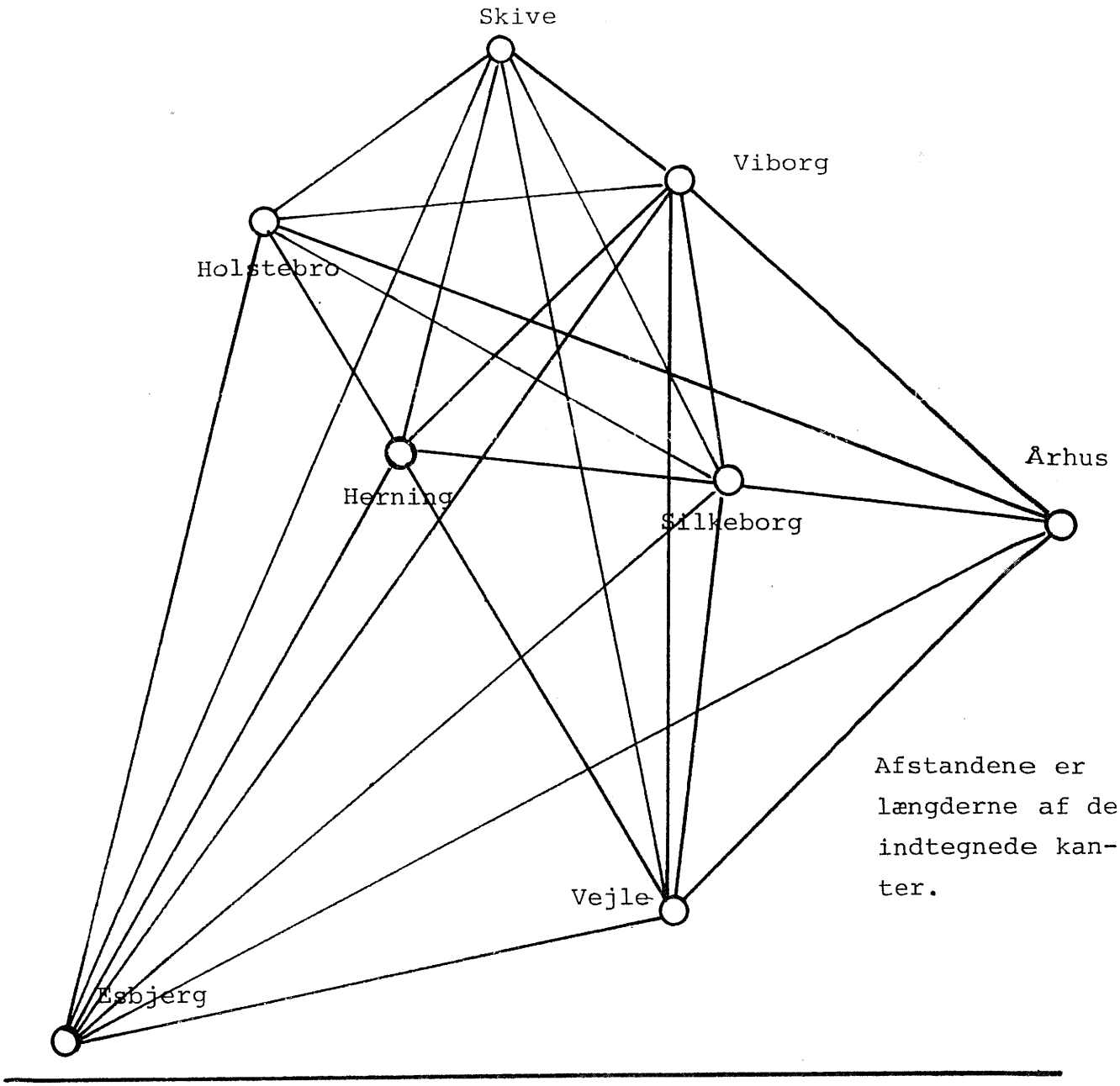
Det er derfor overraskende, at dette problem er så svært. Ingen algoritme bedre end at undersøge alle muligheder kendes! Problem-typen tilhører en klasse af meget vanskelige problemer, kaldet NP-komplette, som vi skal møde i Kapitel III.

Eksempel 1.8 For 7 nord-amerikanske byer er en korteste rundrejse angivet på følgende kort. Denne rejse er fundet gennem undersøgelse af alle muligheder.



Læg mærke til, at den korteste afstand Minneapolis-Kansas City ikke benyttes.

Opgave 1.27 Hvem kan finde en korteste rundrejse for en handelsrejsende, som skal besøge byerne Esbjerg, Herning, Holstebro, Silkeborg, Skive, Vejle, Viborg og Århus ?



I tilfælde som det foreliggende, hvor det ikke synes muligt at opnå nogen god algoritme, forsøger man at finde gode algoritmer, som giver tilnærmede løsninger (*heuristiske algoritmer*). For den handelsrejsende generelt kendes heller ikke heuristiske algoritmer af kvalitet, men hvis længderne opfylder trekantmuligheden

$$\ell(i,j) \leq \ell(i,k) + \ell(k,j)$$

for alle punkter i,k og j , så er der en elegant heuristisk løsning, som skyldes englænderen N. Christofides (1976). Denne løsning skyder højst 50% over målet.

ALGORITME HANDELSREJSENDE

INPUT: Ialt $\binom{n}{2}$ tal $\ell(i,j) \geq 0$, hvor $\ell(i,j)$ er en længde tilknyttet kanten (x_i, x_j) i en komplet graf G med n punkter. Der skal gælde, at $\ell(i,j) \leq \ell(i,k) + \ell(k,j)$ for alle tripler i,j,k.

OUTPUT: En kreds C* med alle G's n punkter, hvor den samlede længde af C* højst er $\frac{3}{2}$ • (længden af minimal kreds).

BEGIN

Find et minimalt udspændende træ T i G v.h.a. ALGORITME MINIMALT TRÆ;

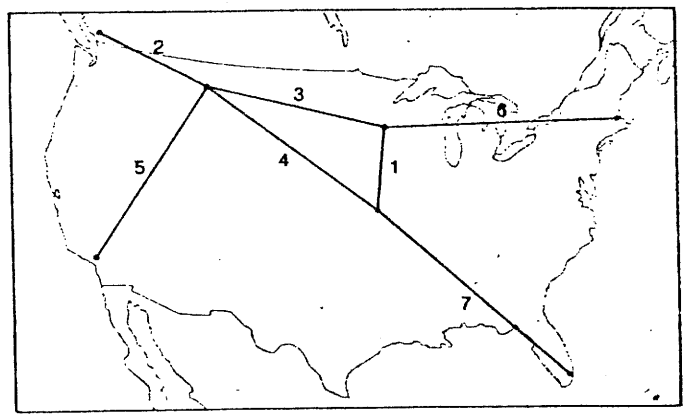
Find en minimal pardannelse P i delgrafen af G bestående af punkterne med ulige valens i T og kanterne mellem dem i G;

Find en Euler-tur E i TU{kanterne i P};

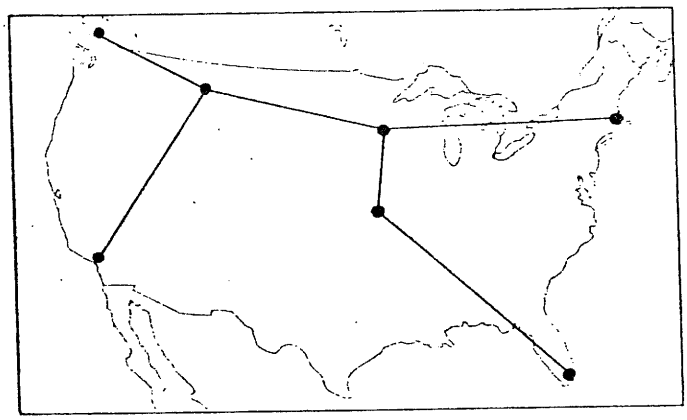
Omdan E til en kreds C* indeholdende alle grafens punkter ved at foretage Euler-turen, men overspringe punkter allerede tidligere besøgt på turen

END

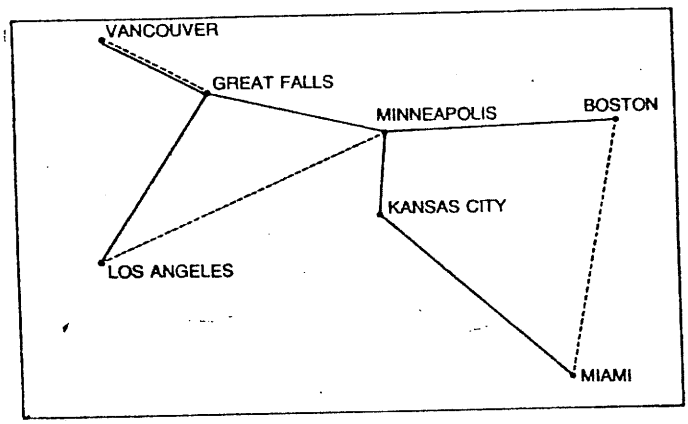
Eksempel 1.9 Lad os løse problemet i Eksempel 1.8 ved at benytte ALGORITME HANDELSREJSENDE:



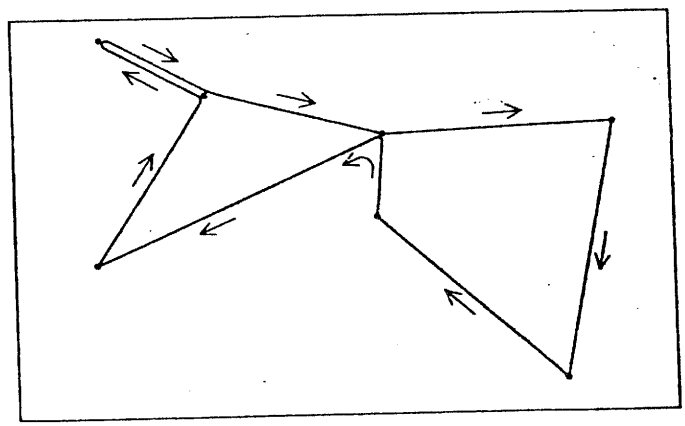
Afstande



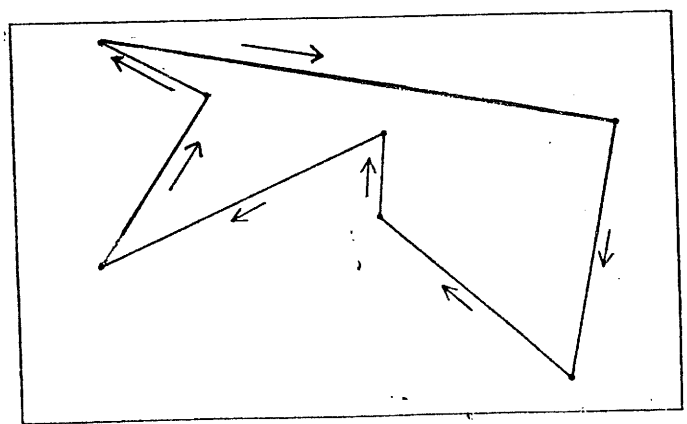
Minimalt udspændende træ



Minimal pardannelse blandt punkterne med ulige valens



Euler-tur



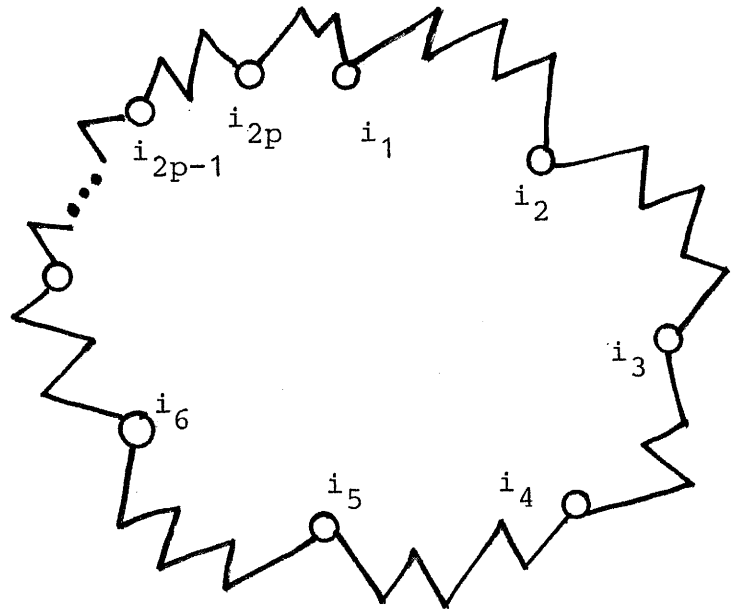
Tur for den handelsrejsende fundet fra Euler-turen ved at starte i Boston

↓ At ALGORITME HANDELSREJSENDE giver et output som beskrevet kan vises således:

Lad C være en minimal tur og lad k være en kant på C. Så er $C-k$ et udspændende træ. Heraf fås at $\ell(C) \geq \ell(T)$, hvor T er et minimalt udspændende træ.

Pardannelsen P danner par mellem nogle af punkterne på C, nemlig punkterne med ulige valens i T. Kald disse punkter $i_1, i_2, \ldots, i_{2p}$, hvor rækkefølgen er den punkterne kommer i på C

C :



Så gælder

$$\ell(P) = \sum_{k \in P} \ell(k)$$

$$\leq \min \{\ell(i_1,i_2) + \ell(i_3,i_4) + \ldots + \ell(i_{2p-1}, i_{2p}),$$

$$\ell(i_{2p},i_1) + \ell(i_2,i_3) + \ldots + \ell(i_{2p-2},i_{2p-1})\}$$

$$\leq \tfrac{1}{2}\ell(C) \quad \text{(p.gr. af trekantuligheden).}$$

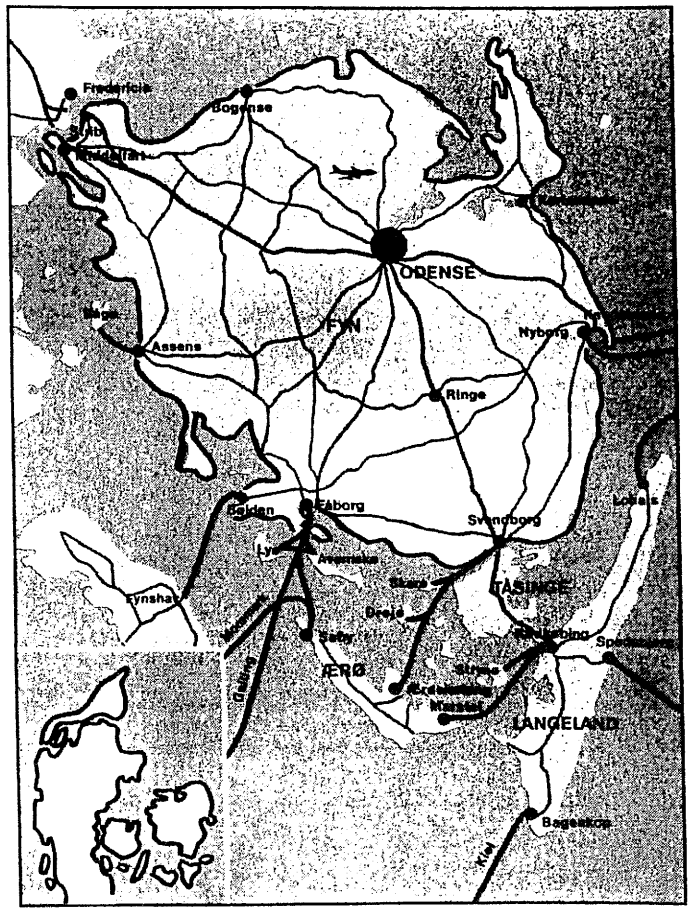
Heraf fås at

$$\ell(C^*) \leq \ell(TUP) = \ell(T) + \ell(P) \leq \tfrac{3}{2}\ell(C)$$

hvor det første ulighedstegn igen er baseret på trekantuligheden. ∎

Opgave 1.28 Anvend ALGORITME HANDELSREJSENDE på eksemplet i Opgave 1.27.

Opgave 1.29 En cykeltur på Fyn planlægges som en rundtur med start og slut i Odense. Den skal gå gennem Assens, Bogense, Fåborg, Kerteminde, Middelfart, Nyborg, Ringe og Svendborg. Planlæg turen v.h.a. ALGORITME HANDELSREJSENDE. Kommer de korteste strækninger Odense-Kerteminde og Odense-Ringe mon med i en korteste rundtur ?



AFSTANDE PÅ FYN	Assens	Bogense	Fåborg	Kerteminde	Middelfart	Nyborg	Odense	Ringe
Assens								
Bogense	42							
Fåborg	36	62						
Kerteminde	59	51	60					
Middelfart	35	31	70	66				
Nyborg	67	59	48	19	75			
Odense	38	30	39	21	46	30		
Ringe	43	50	26	38	65	25	20	
Svendborg	61	73	26	49	84	34	43	23

1.7 LITTERATUR

Definitionen af *god algoritme* i afsnit I.1 dukkede op midt i 1960'erne i flere arbejder, mest eksplicit i

J. Edmonds: Paths, trees and flowers, Canadian Journal of Mathematics 17 (1965), side 449-467.

I artiklen løste Edmonds med en god algoritme og en god sætning det vægtede pardannelsesproblem, som optræder i ALGORITME POSTBUD. Det er også Edmonds, som først har gjort opmærksom på begrebet *god sætning* og givet en definition heraf.

Euler's artikel fra 1736 var skrevet på latin, men er oversat til dansk:

L. Euler: Løsning af et problem fra beliggenhedsgeometrien, i P. Wolff: Højdepunkter i matematikken, Steen Hasselbalchs Forlag 1967, side 187-196.

Idéen i ALGORITME EULER-TUR III stammer fra franskmanden M. Fleury, som publicerede den i 1885, se

É. Lucas: Récréations Mathématiques, udgivet i 1894, genoptrykt af Albert Blanchard Paris i 1960, bind IV, kapitel 6.

Sætning 1.4 stammer fra

L.R. Ford & D.R. Fulkerson: Flows in Networks, Princeton University Press (1962).

En god oversigtsartikel, hvorfra eksemplet med rundturen mellem 7 byer i Nord-Amerika er hentet, er

H.R. Lewis & C.H. Papadimitriou: The efficiency of algorithms, Scientific American, Januar 1978, side 96-109.

Meget af materialet i dette kapitel findes i den fremragende bog

J.A. Bondy & U.S.R. Murty: Graph Theory with Applications, Macmillan 1976.

ALGORITME MINIMALT TRÆ kaldes også KRUSKAL'S ALGORITME, idet den blev kendt gennem artiklen

J.B.Kruskal : On the shortest spanning subtree of a graph and the travelling salesman problem, Proc. American Math. Society 7 (1956), side 48-50.