APPENDIX A

GRAFTEORETISKE DEFINITIONER

En *graf* G består af et tripel

$$G = (P(G), K(G), \psi_G)$$

hvor P(G) og K(G) er mængder, og ψ er en funktion, som til hvert $k \in K(G)$ tilordner et par $\psi(k)$ af elementer fra P(G).

Mængden P(G) kaldes *mængden af punkter* og K(G) kaldes *mængden af kanter*. For en kant k kan $\psi_G(k)$ bestå af enten

1) et ikke-ordnet par (x,y) af punkter x og y. I dette tilfælde siges kanten at være *ikke-orienteret*,

eller

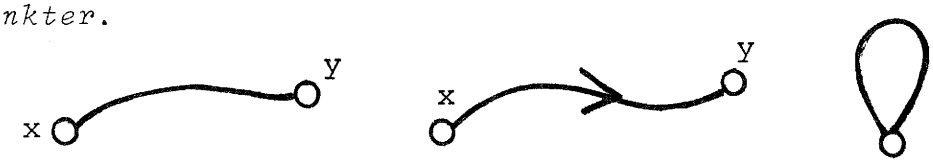
2) et ordnet par (x,y) af punkter x og y. I dette tilfælde siges kanten at være *orienteret fra* x *til* y ,

eller

3) et par (x,x) bestående af det samme punkt to gange. I dette tilfælde siges kanten at være en *sløjfe*.

Ofte benyttes (x,y) som navn for en kant k med $\psi_G(k) = (x,y)$.

En graf kan anskueliggøres vha. et diagram, hvor punkterne repræsenteres af små cirkler, og kanterne repræsenteres af kurver eller linier mellem de pågældende par af punkter. Man siger også, at en kant *forbinder* de pågældende punkter, som kaldes kantens *endepunkter*.



ikke-orienteret kant | orienteret kant fra x til y | sløjfe

Såfremt en graf har forskellige kanter k_1 og k_2 med $\psi(k_1) = \psi(k_2)$, siges der at være *multiple kanter* i grafen. Såfremt alle kanter i en graf er ikke-orienterede, siges grafen at være *ikke-orienteret*. Såfremt alle kanter i en graf er orienterede, siges grafen at være *orienteret*. Såfremt en graf har såvel ikke-orienterede som orienterede kanter, siges den at være *blandet*.

Ofte benyttes ordet *graf* i betydningen en ikke-orienteret graf, evt. uden sløjfer og multiple kanter. Vi betragter kun *endelige grafer*, dvs. grafen G hvor såvel $P(G)$ som $K(G)$ er endelige.

Såfremt $P(G_1) \subseteq P(G_2)$, $K(G_1) \subseteq K(G_2)$ og $\psi_{G_1}(k) = \psi_{G_2}(k)$ for alle $k \in K(G_1)$, siges G_1 at være en *delgraf* af G_2, skrevet $G_1 \subseteq G_2$. Såfremt $G_1 \subseteq G_2$ og $P(G_1) = P(G_2)$ siges G_1 at være en *udspændende* delgraf.

Såfremt G er en graf og $K' \subseteq K(G)$ betegner $G-K'$ delgrafen i G opnået fra G ved at fjerne alle kanter i K' fra G (men ingen punkter fjernes). Såfremt $P' \subseteq P(G)$ betegner $G-P'$ delgrafen i G opnået fra G ved at fjerne alle punkter i P' og alle kanter med mindst et endepunkt i P' fra G. Når x er en enkelt kant eller et enkelt punkt, skrives $G-\{x\}$ ofte blot $G-x$.

Valensen $v(x,G)$ af et punkt x i en graf G er

> antal ikke-orienterede kanter, som har x som præcis det ene endepunkt,
>
> \+
>
> $2 \cdot$(antal sløjfer af typen (x,x) i G).

Indgangsvalensen $v^-(x,G)$ af et punkt x i G er antallet af orienterede kanter i G til x. *Udgangsvalensen* $v^+(x,G)$ af et punkt x i G defineres tilsvarende som antallet af orienterede kanter i G fra x.

En *rute* i en graf mellem punkter x_1 og y_n, 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 hvor $y_i = x_{i+1}$ for $i=1,2,\ldots,n-1$. Hvis $x_1 = y_n$ kaldes ruten *lukket*, og hvis $x_1 \neq y_n$ kaldes den *åben*. Hvis k_i for alle i er orienteret fra x_i til y_i er ruten *ensrettet*.

Såfremt alle kanter i en rute er forskellige, kaldes den en *tur*. Såfremt også $x_1, x_2, \ldots, x_n$ og y_n alle er forskellige, kaldes turen en *vej*, som *forbinder* x_1 og y_n. Såfremt $x_1, x_2, \ldots, x_n$ alle er forskellige, og $x_1 = y_n$, kaldes turen en *kreds*. Veje og kredse opfattes også ofte som delgrafer. *Længden* af en vej eller en kreds er antal kanter i den. En sløjfe er således en kreds af længde 1.

Såfremt en graf G indeholder en rute mellem to forskellige punkter x og y, så vil en kortest mulig sådan rute i G være en vej mellem x og y.

En graf, hvor der for vilkårlige to punkter x og y er en rute mellem x og y, siges at være *sammenhængende*. Såfremt en graf G ikke er sammenhængende, kan den opdeles i disjunkte delgrafer $G_1, G_2, \ldots, G_k$, som tilsammen indeholder alle G's punkter og kanter og som hver for sig er sammenhængende. Disse delgrafer kaldes G's *sammenhængskomponenter*.

En sammenhængende ikke-orienteret graf T uden kredse som delgrafer, kaldes et *træ*. En bro i en graf er en kant, som ikke er med i nogen kreds. Et træ er således en sammenhængende graf, hvor alle kanter er broer.

Da et træ T er sammenhængende, har vilkårlige to punkter x og y en vej i T mellem sig. Der er præcis én sådan vej, thi to forskellige veje mellem x og y ville give anledning til en kreds. Et træ T med ≥ 2 punkter kan derfor også defineres som en graf, hvor vilkårlige to punkter er forbundet med præcis én vej i T.

Beviset for at to forskellige veje mellem x og y vil give anledning til en kreds, kan gives på følgende måde: Lad de to veje hedde V_1 og V_2. Følg V_1 fra x indtil vi første gang møder en kant k, som ikke er på V_2 (k eksisterer, da de to veje er forskellige). Betragt den delvej V_1' af V_1, som starter med kanten k og ender i punktet y. Lad x' være det første punkt på denne vej, og lad y' være det første punkt efter x', som også ligger på V_2 (y' eksisterer, da y er et punkt efter x' på V_1' og y ligger på V_2). Den delvej V_1'' af V_1, som forbinder x' og y' har så kun de to punkter x' og y' og ingen kanter fælles med V_2. Sammen med delvejen

af V_2 , som forbinder x' og y' , danner V_1'' en kreds.

Hvis T er et træ med $\geq$ 2 punkter, så vil en længst mulig vej i T forbinde to punkter x og y med v(x,T) = v(y,T) = 1. T-x er et træ med ét punkt mindre end T. Ved induktion følger, at et træ T med n punkter har præcis n-1 kanter.

En *to-delt* graf er en graf, hvor P(G) kan deles i to disjunkte mængder A og B således, at enhver kant har ét endepunkt i A og det andet i B.

En *pardannelse* P i en graf G er en mængde af kanter i G , hvor vilkårlige to kanter i P ikke har et fælles endepunkt.

En *dækning* D i en graf G en mængde af punkter i G , hvor en vilkårlig kant i G har mindst ét af sine endepunkter i D.

APPENDIX B

ALGORITMISK NOTATION

Algoritmerne i kapitel I og II beskrives i et pseudo-sprog. En algoritme beskrevet i dette sprog kan opfattes som et trin på vejen mod udviklingen af et egentligt computer-program i f.eks. Pascal.

Algoritmerne i pseudo-sproget er angivet som en række *ordrer* med simikolon imellem.

På vilkårlige steder i disse ordrer kan der indsættes *kommentarer* mellem tegnene "(*" og "*)". Disse kommentarer er ikke en del af selve algoritmen, men har karakter af forklaringer.

Ordrerne i en algoritme kan være af følgende typer:

1. *Tilordning*

Variabel := udtryk.

Et udtryk kan bestå af en aritmetisk formel, af ordet "sand" eller "falsk", eller en beskrivelse af en mængde.

2. IF-THEN-ELSE-*ordre*

IF betingelse THEN ordre 1 ELSE ordre 2.

En betingelse er et logisk udtryk, som kan være enten sand eller falsk, f.eks. "$x \in L$" eller "$x < N$ og $y < M$". Såfremt betingelsen er sand, udføres ordre 1 og i modsat fald ordre 2. Den sidste del af ordren, dvs. ELSE-delen, kan udelades således, at der ikke udføres nogen ordre, når betingelsen er falsk.

3. FOR-*løkke*

FOR liste DO ordre 1.

Listen angiver alle værdier af en variabel for hvilke ordre 1 skal udføres, f.eks. "j := 1,2,...,n". Af og til skrives også i det nævnte eksempel

FOR j := 1 TO n DO ordre 1

Vi tillader også lister af typen "alle $x \in L$, hvor x har egenskaben E".

4. WHILE-*løkke*

WHILE betingelse DO ordre 1

Her udføres ordre 1 gentagne gange, så længe betingelsen er sand.

5. *Beskrivende ordre*

Her tillader vi næsten hvad som helst, hvis det blot er læseligt og nogenlunde klart, f.eks. "Vælg v, så m(v) er mindst mulig" eller "Find en pardannelse P i grafen bestående af ...".

6. *Sammensat ordre*

BEGIN ordre 1 ; ordre 2 ; ... ; ordre k END

Ordrerne i en sådan sammensat ordre kan være af en vilkårlig af typerne 1-6. Læg også mærke til, at de enkelte ordrer i typerne 2-6 kan være sammensatte.

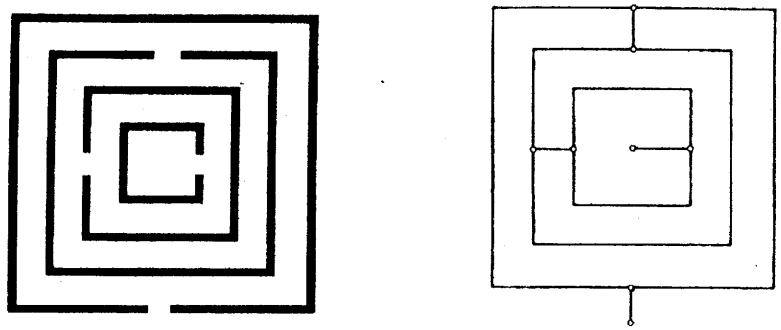
APPENDIX C

AFSNIT SOM RØG UD AF BOGEN.

I.4 LABYRINTER.

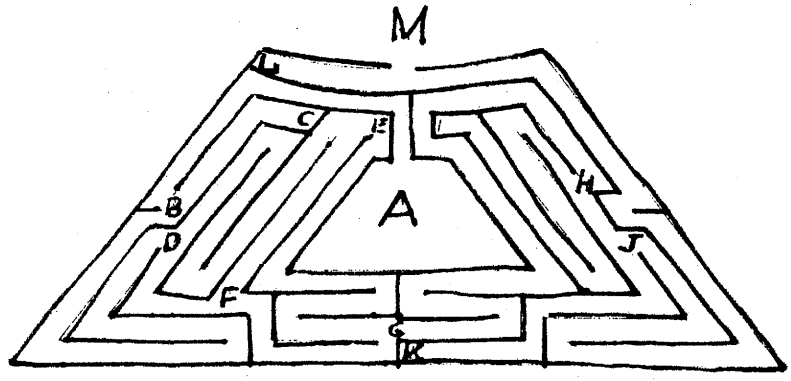
En labyrint består af en række gange og forgreningspunkter og repræsenteres matematisk mest nærliggende af en graf.

Eksempel 1.5 Følgende labyrint og graf svarer til hinanden

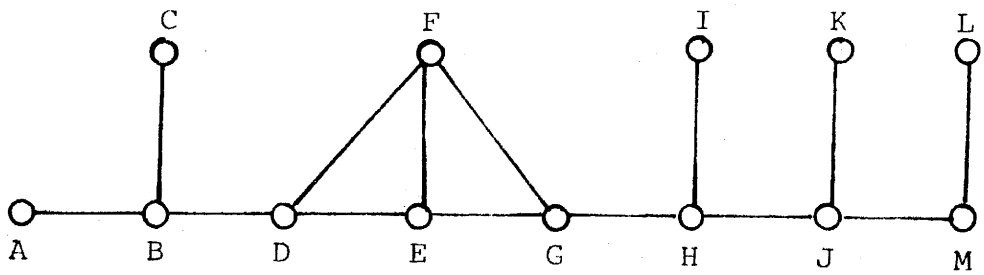


Grafen kan tegnes simplere således:

Eksempel 1.6 Den berømte labyrint af hække i haven på Hampton Court i England ser således ud:



Den tilsvarende graf er



At finde vej i en labyrint er således problemet at finde en rute fra et punkt i en graf til et andet. Dette andet punkt kan være et vilkårligt sted i grafen, og vi vil derfor forsøge at finde en algoritme til at bestemme en rute, som starter i et givet punkt x og indeholder alle grafens kanter og punkter.

Ved at indsætte en ekstra kopi af alle kanter i grafen fås en graf, som har en Eulertur. Vi vil imidlertid ikke acceptere ALGORITME EULERTUR I eller II og heller ikke ALGORITME POSTBUD, som mulige løsninger, idet disse algoritmer forudsætter et globalt kendskab til hele grafen. Det der kendetegner labyrint-problem-typen er netop, at vi ikke kender grafen. For at problemet skal have mening, er det dog nødvendigt at forudsætte én eller anden form for "lokalt" kendskab til grafens struktur på det sted, vi er nået til på ruten.

Dette "lokale" kendskab kan have flere former. Én mulighed (Ariadnes tråd til Theseus) er, at man på et vilkårligt sted på ruten kan gennemløbe den allerede tilbagelagte rute baglæns til udgangspunktet. En anden mulighed, som vi vil benytte, er: For ethvert af grafens punkter P, vi ankommer til på ruten, ved vi

a) hvilke kanter der tidligere på ruten har været benyttet væk fra P,

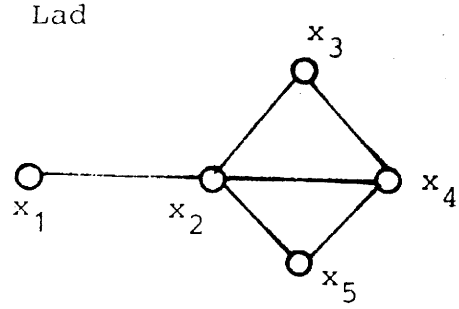
b) hvilken kant, der blev benyttet, da vi første gang kom til P. Denne kant vil vi kalde en *ind-kant* for P.

Det viser sig, at denne information er nok til at sikre, at vi med en simpel regel kommer rundt i hele grafen (under forudsætning af at den er sammenhængende). Reglen er først formuleret af en franskmanden Gaston Tarry i 1895: Den siger, at vi ved P skal vælge en kant bort fra P, som ikke tidligere på ruten har været benyttet bort fra P, dog skal P's ind-kant kun benyttes, hvis der ikke er andre muligheder.

Mere formelt:

```
ALGORITME LABYRINT

INPUT:    Sammenhængende graf G med punkter
          x1,x2,...,xn og mindst én kant.

OUTPUT:   En lukket rute R i G fra punkt x1,
          hvor hver kant i G er gennemløbet
          præcis én gang i hver retning.

BEGIN
  R:= Ø ; (*R skal betegne følgen af kanter alle-
          rede gennemløbet *)
  y:= x1 ; (*y skal betegne det punkt vi er nået
           til *)

  FOR  i:= 1 TO n DO
    BEGIN
    I(i):= mængden af kanter incidente med punkt
           xi  i  G ,
     (* I(i) skal betegne mængden af kanter inci-
     dente med punkt Xi og ikke tidligere benyttet
     bort fra xi *)
    k(i):= Ø ;
     (* k(i) skal betegne ind-kanten for punkt xi *)
    END
  WHILE I(y) ≠ Ø DO
    BEGIN
       Vælg kant k=(y,u) ∈ I(y), hvor  k≠k(y)
       medmindre dette ikke kan undgås;
       I(y):= I(y)\{k} ;
       R:= R med  k  tilføjet;
       IF k(u)=Ø og u≠x1 THEN k(u):= k ;
       y:= u

    END
END.
```

Eksempel 1.7 Lad



Algoritmen anvendt på G giver f.eks.:

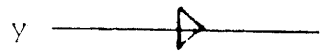
$R = [(x_1,x_2), {}^*(x_2,x_3), (x_3,x_4), {}^*(x_4,x_2), {}^*(x_2,x_5), (x_5,x_4),$

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

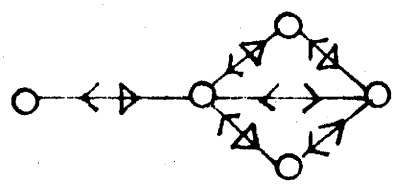
Ved de med * mærkede steder er der flere valgmuligheder. De understregede kanter er ind-kanter.

Når algoritmen gennemføres på en graf er det smart at sætte pile på kanterne, når de gennemløbes:

og sætte en speciel pil på, hvis kanten er ind-kant for u:



Ovenstående graf kommer så til at se således ud:



hvoraf det fremgår, at den opnåede rute er som beskrevet i output.

Opgave 1.14 Benyt ALGORITME LABYRINT på Eksemplerne 1.5 og 1.6.

Vi skal nu bevise, at algoritmen virkelig giver det ønskede resultat.

Algoritmen standser når $L(y) = \emptyset$ for ét eller andet y. Er $y \neq x_1$ er vi kommet ind til y én gang mere end vi er gået bort fra y, men da vi aldrig benytter samme kant to

gange i samme retning, er det en modstrid med at $L(y) = \emptyset$. Altså standser vi i punkt x_1 og har brugt alle kanter fra punkt x_1 bort fra x_1 én gang hver og dermed også alle kanter ind til x_1 én gang hver.

Lad $y_1 = x_1, y_2, y_3, \ldots$ være punkterne i den rækkefølge i hvilken vi møder dem første gang på R. Antag at der er kanter, som ikke er blevet gennemløbet i begge retninger, og lad y_j være det første punkt i rækkefølgen incident med en sådan kant. Iflg. ovenstående er $j \geq 2$. Endvidere løber R ind og ud af y_j det samme antal gange, så der er kanter bort fra y_j, som ikke er gennemløbet.

Lad ind-kanten for y_j være (y_i, y_j). Så er y_i mødt tidligere på R end y_j, dvs. $i < j$. Men så er (y_i, y_j) gennemløbet i begge retninger på R, da det gælder for alle kanter incidente med y_i.

Ind-kanten (y_i, y_j) er altså gennemløbet bort fra y_j, samtidig med at der er andre kanter fra y_j, ikke gennemløbet bort fra y_j. Det er i modstrid med den måde næste kant skal vælges på i algoritmen. Altså er alle kanter gennemløbet i begge retninger. ∎

Mængderne $I(i)$ indeholder i starten tilsammen alle kanter to gange. Ved hvert gennemløb af WHILE-løkken fjernes mindst et element fra et $I(y)$ og hvert gennemløb består af højst et konstant antal skridt (uafhængig af G's størrelse). Alt i alt er ALGORITME LABYRINT en god algoritme af kompleksitet $O(|K(G)|)$.

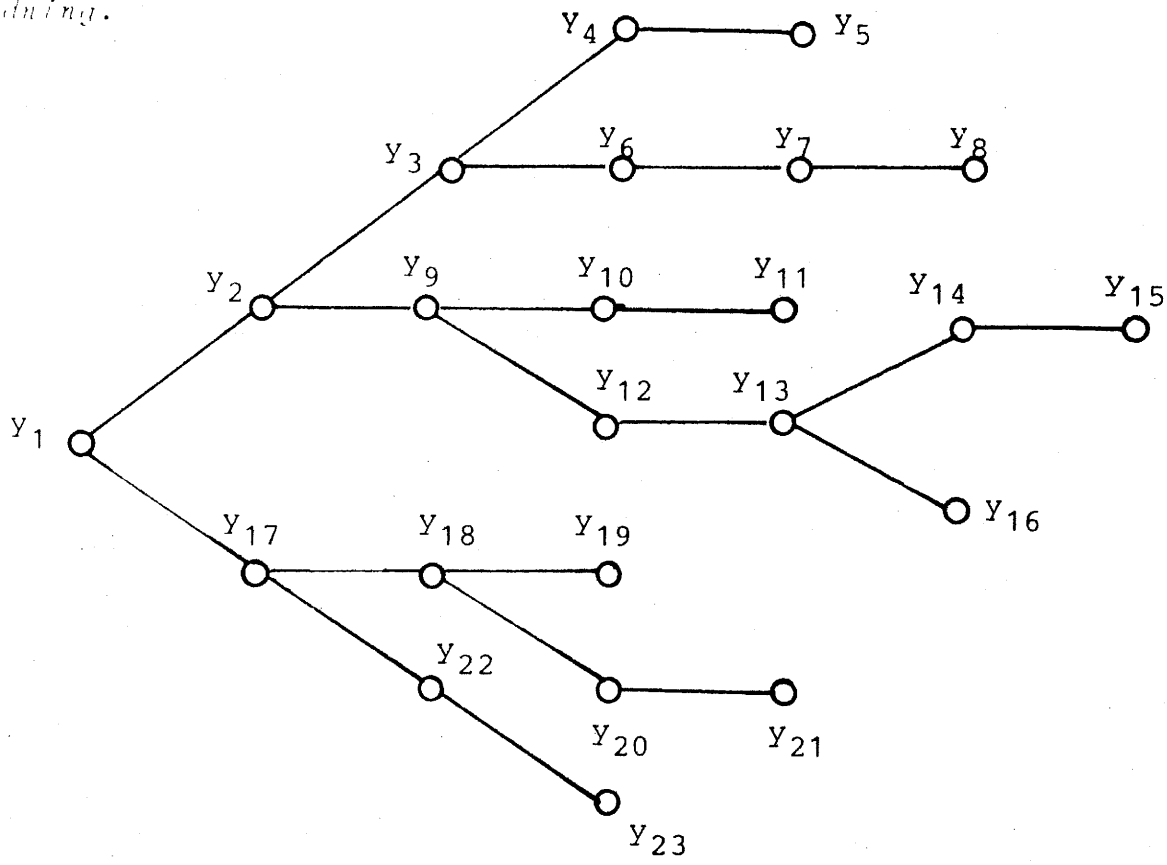
Algoritmen giver præcis én ind-kant for alle punkter $\neq X_1$. Endvidere er det let at se, at ind-kanterne ikke danner nogen kreds (antag at y_j er det første punkt fra rækkefølgen ovenfor, som er med i en kreds af ind-kanter; det giver let en modstrid). Det følger heraf, at den delgraf af G, som ind-kanterne og deres endepunkter danner er et træ i G, som indeholder alle G's punkter. Et sådant træ kaldes *et udspændende træ* i G.

Det er muligt at gøre valget af kanten (y,u) lige efter det sidste BEGIN i ALGORITME LABYRINT lidt mere præcist ved at kræve, at vi så vidt muligt vender tilbage ad den kant (y,u) vi lige er kommet til y ad . De to eneste situationer, hvor det ikke er muligt, er

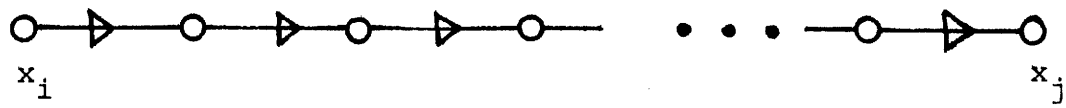
a) (y,u) er u's ind-kant, dvs. det er første gang vi møder u , og u er incident med andre kanter

b) (y,u) er y's ind-kant, som vi altså netop har brugt i retningen bort fra y .

I en sådan situation er den orden $y_1, y_2, \ldots$, vi møder punkterne i, præ- i forhold til træet T af ind-kanter. Mønsteret fremgår (vist) af følgende figur. En ordning $y_1, y_2, \ldots$ af punkterne i et træ som vist på figuren kaldes en *præ-ordning*.



Alle øvrige kanter i G , dvs. alle ikke ind-kanter, går mellem punkter "på samme gren af T ", dvs. hvis $(x_i, x_j) \in K(G)$, så er punkterne x_i og x_j forbundet med en ensrettet vej af ind-kanter



I ovenstående eksempel kan der altså ikke være nogen af kanterne (y_3,y_{18}) , (y_3,y_{10}) , (y_5,y_9) osv. i grafen G . Men (y_2,y_1) , (y_2,y_{10}) , (y_5,y_2) osv. kan evt. være i G .

Kanterne i G deles altså i to grupper: ind-kanterne (alle kanterne i T) og de øvrige kanter, kaldet *bak-kanterne* (alle kanterne ikke i T).

Den beskrevne modifikation af ALGORITME LABYRINT er en meget nyttig søge-algoritme for grafer. Den har navnet *dybde-først-søgning* (DFS). Ved gennemførelse af DFS for en graf G vil output normalt ikke bestå af ruten R , men af en eller flere af

a) rækkefølgen $y_1,y_2,\ldots$, dvs. for hvert x_i et nummer $j = m(x_i)$, hvor $x_i = y_j$,

b) træet T , dvs. for hvert x_j , $j \geq 2$, en kant $k(x_j) = (x_i,x_j)$, som er x_j's ind-kant. Med tanke på stam-trær kaldes x_i af og til x_j's *far* i T ,

c) for hvert x_j et nummer $\ell(x_j)$, som angiver det laveste $m(x_i)$ for et punkt x_i med $m(x_i) < m(x_j)$, som kan nås fra x_j ved først evt. at gå fremad i T for derefter at gå tilbage langs en bak-kant (hvis ingen sådan vej findes sættes $\ell(x_j) = m(x_j)$).

Disse data giver mange oplysninger om G's struktur, f.eks. er en kant (x_i,x_j) i G med $m(x_i) < m(x_j)$ en bro hvis og kun hvis (x_i,x_j) er en kant i T og $\ell(x_j) \leq m(x_i)$. Algoritmen for DFS kan altså bl.a. benyttes til effektivt at finde alle broer i en graf. Disse og mange andre egenskaber ved DFS har specielt to amerikanske dataloger J. Hopcroft og R. Tarjan undersøgt og gjort opmærksom på i 1972.

Opgave 1.15 Tegn et eller flere eksempler på grafer og prøv DFS på disse eksempler. Angiv numrene $m(x_i)$ på punkterne og angiv træet T. Overvej også hvordan $\ell(x_i)$ kan beregnes (vink: når x_i mødes første gang får x_i sit nummer $m(x_i)$, $k(x_i)$ bestemmes, og $\ell(x_i)$ sættes foreløbigt til $m(x_i)$. $\ell(x_i)$ ændres løbende under den videre gennemsøgning hver gang vi vender tilbage til x_i indtil vi sidste gang forlader x_i).

Den mærkning $m(x_i)$, der sker af punkterne i en graf G ved DFS, foregår ved først at mærke x_1 og derefter så vidt muligt at mærke et punkt forbindet til det sidst mærkede af de allerede mærkede punkter. Ser vi bort fra hvilke mærker der sættes, ser metoden mere præcis således ud

```
ALGORITME MÆRKNING VED DFS

INPUT:    Sammenhængende graf G med mindst
          ét punkt x .

OUTPUT:   En mærkning af G's punkter.

BEGIN
  L:= [x]; (*L skal betegne en liste af ikke-
           mærkede punkter, der i G er for-
           bundet til allerede mærkede punk-
           ter *)

    WHILE L ≠ Ø DO

      BEGIN

        v:= sidst tilføjede punkt i L;

        L:= L med v fjernet;

        Mærk v;

        L:= L med alle ikke mærkede punkter,
            der er forbundet til v i G og som
            ikke allerede er i L, tilføjet

      END

END.
```

En liste L , hvor vi tilføjer og fjerner elementer i samme ende af L , kaldes en *stak*, og den ende af stakken hvor elementer fjernes og tilføjes kaldes *stakkens top*. En stak er karakteriseret ved princippet *"sidst ind - først ud"*. Et andet meget brugt princip for en liste er "først ind - først ud", dvs. elementer fjernes i den ene ende og tilføjes i den anden. En sådan liste kaldes en *kø*.

Ændres listen L i ALGORITME MÆRKNING VED DFS fra en stak til en kø fås en metode kaldet *"bredde-først-søgning"* (BFS) i stedet for DFS.

ALGORITME MÆRKNING VED BFS

Som ALGORITME MÆRKNING VED DFS med første ordre efter det sidste BEGIN ændret til:

```
v:= første punkt i L
```

Den mærkning, der foregår i de to algoritmer, kan se ud på mange forskellige måder, og metoderne er i mange sammenhænge nyttige. I næste afsnit skal vi se hvordan mærkning ved BFS kan benyttes til bestemmelse af korteste veje.