Oblig 3 for INF 4130, 2009

Leveringsfristen er fredag. 20. november

NB: Det blir nokså strengt med leveringsfrister denne gangen, siden all godkjenning må være klar til slutten av november, slik at Fakultetet kan sette opp eksamenslister etc. Det kan komme presiseringer til oppgaven, så følg med.

Merk at noe av stoffet til Oppgave 1 ikke er forelest når denne legges ut, men det blir forelest 5. november.

Levering skjer som for tidligere obliger (spesielt skal programmet ha navn nøyaktig "Oppgave1", på den måten det blir mest naturlig innenfor språket en bruker).

Oppgave 1 (Flyt i nettverk)

Denne oppgaven går ut på å implementere FordFulkerson-algoritmen, med bruk av kortest mulig forbedringsveier i hvert skritt (Edmonds og Karps variant). Gitt en graf med kapasiteter, skal programmet skrive ut verdien av en optimal flyt, flyten vi da har på hver kant, samt et kutt (det som algoritmen gir) som beviser at dette er optimal flyt.

Grafen er rettet, så mellom to noder *u* og *v* kan kapasiteten fra *u* til *v* være forskjellig fra den fra *v* til *u*. Alle kapasiteter er heltallige og ikke-negative.

Input

Programmet skal lese input fra en angitt fil, denne angis til programmet ved oppstart, sammen med navnet for output-fil. Input-filen inneholder :

- Først en linje med antall noder *m*.
- Så *m* linjer med *m* tall i hver (en matrise/tabell) som angir kapasitetene mellom hvert par av noder. Dersom *i* er linjen og *j* er kolonnen til et tall, så angir tallet kapasiteten som kan gå fra node *i* og til node *j*, altså kapasiteten til kanten (*i, j*) i grafen.

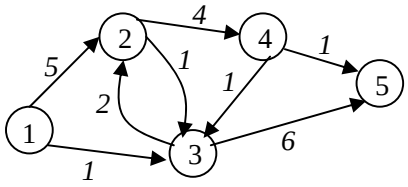
Nodene er altså nummerert fra 1 til *m*, og node 1 er kilde og node *m* er sluk. Merk at det altså kan være positiv kapasitet både fra en node *u* til en node *v*, og fra *v* til *u*. Langs diagonalen angis tallet null. Det vil aldri gå kanter inn i kilden eller ut av sluket (og dermed vil første kolonne og siste linje alltid ha bare nuller).

Output

- Først en linje med ett enkelt tall som angir størrelsen av optimal flyt.
- Så *m* linjer med *m* tall i hver der flyten på kantene skrives ut, etter samme format som kapasitetene ble angitt på i input (vertikal indeks er *fra* og horisontal indeks er *til*).
- Til slutt en egen linje som angir de nodene som er på kildesiden av et kutt med kapasitet lik flyten, i sortert rekkefølge. Kildenoden skal her tas med.

Eksempel-input	Eksempel-output
5 0 5 1 0 0 0 0 1 4 0 0 2 0 0 6 0 0 1 0 1 0 0 0 0 0	4 0 3 1 0 0 0 0 1 2 0 0 0 0 0 3 0 0 1 0 1 0 0 0 0 0 1 2 4

Figur 1 under viser nettverket i eksempel-inputen, med kapasiteter angitt på kantene (node 1 er kilde og node 5 er sluk)



Figur 1. Nettverk med kapasiteter.

Tips til implementasjon

Vi er ikke kritiske til minnebruk, en kan fint unne seg tre-fire $m \times m$-arrayer her uten at det teller negativt. For eksempel kan en bruke én for det opprinnelige problemet (N), én for den flyten man i øyeblikket har på hver kant (f), og én med de flytforandringene som er mulig (Nf), som i figur 14.8 i læreboka.

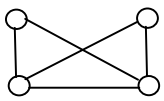
Ellers er det jo bare i hvert steg å gjennomføre et bredde-først søk med en FIFO-kø, og det er kanskje lurt å ha en boolsk array som angir om noden er sett i dette søket..

Oppgave 2 (*NP*-kompletthet)

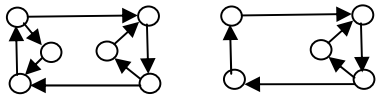
2.a

Bevis at 2-RETTET-HAMILTONBARHET (definert under) er *NP*-komplett. Du kan anta som kjent at vanlig (urettet) HAMILTONBARHET er *NP*-komplett. Vi definerer det slik at en graf med én enkelt node, eller en graf med to noder med kant mellom, er HAMILTONBARE.

Forklar den transformasjonen du vil bruke, og hvorfor den virker. Vis også hvordan transformasjonen vil virke på grafen:



Definisjon av 2-RETTET-HAMILTONBARHET: Gitt en rettet graf G som input.
Spørsmål: Finnes det to rettede enkle løkker i grafen, som hver har minst tre noder, og som til sammen inkluderer alle nodene? Eksempler:



Figuren over viser en ja- og en nei-instans av 2-RETTET-HAMILTONBARHET.

2.b

Vår definisjon av "*NP*-hard" er som følger: NP-harde problemer kan spørre om hva som helst (et tall, en liste etc.) i motsetning til *NP*-komplette problemer, som bare kan være ja/nei-problemer. Kriteriet på at et problem er *NP*-hardt er at om det kunne løses i polynomisk tid, så ville det føre til at alle *NP*-komplette problemer kunne løses i polynomisk tid. Se foilene fra 22/10 (og man kan også slå opp på "*NP*-hard" i Wikipedia, der man finner setningen: "If there is a polynomial algorithm for any *NP*-hard problem, then there are polynomial algorithms for all problems in *NP*, and hence *P = NP*".

Dermed er alle *NP*-komplette problemer også *NP*-harde, men ikke omvendt. For oppgaven nedenfor skal vi snakke om "Ekte *NP*-harde" problemer som de som er *NP*-harde, men ikke *NP*-komplette. Vi skal også snakke om "Polynomiske" problemer, som alle slags problemer (også ja/nei-typen) som kan løses i polynomisk tid.

Klassifisér ut fra dette følgende problemer, og skisser et bevis for hvert av svarene. En enkel vei ("simple path") i en graf er altså en vei hvor ingen noder forekommer to ganger. Lengden av en vei defineres som antall kanter langs veien. Som i 2.a kan du anta at HAMILTONBARHET er *NP*-komplett.

(i) Finn lengden av den lengste enkle vei mellom *x* og *y*.

(ii) Finn lengden av den korteste enkle vei mellom *x* og *y*.

(iii) Avgjør om det finnes en enkel vei mellom *x* og *y* som inneholder minst halvparten av nodene (inklusive *x* og *y*).

(iv) Finn en enkel vei mellom *x* og *y* i *G*, slik at denne veien er så lang som mulig. Algoritmen skal returnere en sekvens noder som svarer til en eller annen lengste enkle vei fra *x* til *y*.

(v) Finnes det en klikk i G som inneholder *x* og *y* og tre noder til? Angi i så fall en slik.

[slutt]