LØSNINGSFORSLAG
EKSAMEN I FAG 45011 ALGORITMER OG DATASTRUKTURER

Fredag 12. januar 1996, kl 0900 - 1300

Løsningsforslag ved: Arne Halaas

Oppgave 1 (20 %).

(a) (5 %)

$S(L,n) = 4L + 2L\sqrt{2} + S(L/2,n-1)$

(b) (5 %)

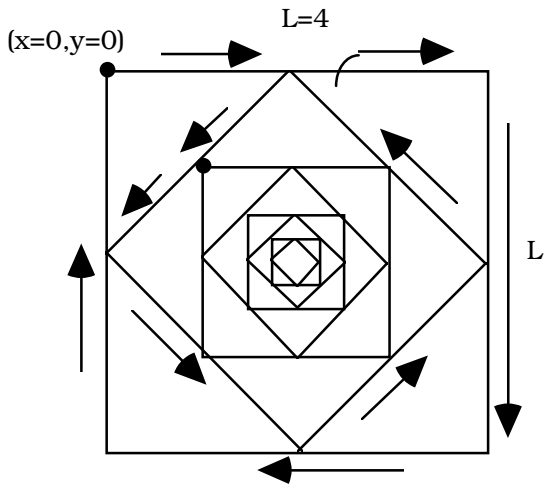
$$S(L,n) = 4L + 2L\sqrt{2}\ (1 + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^{(n-1)}}) = 4L(2 + \sqrt{2})\ (1 - \frac{1}{2^n})$$

Altså blir : $\lim_{n \to \infty} S(L,n) = 4L(2 + \sqrt{2}) = 2\ S(L,1)$

Streken får en endelig samlet lengde selv om uendelig mange likedannede figurer er nøstet. Total lengde blir kun det doble av "grunnfiguren" S(L,1)!
Dette er den gamle historien om sjokoladen som varer evig lenge dersom en bare spiser halvparten av resten ved neste bit.

(c) (10 %)

Dette er **tegning** (0,0,4,4):

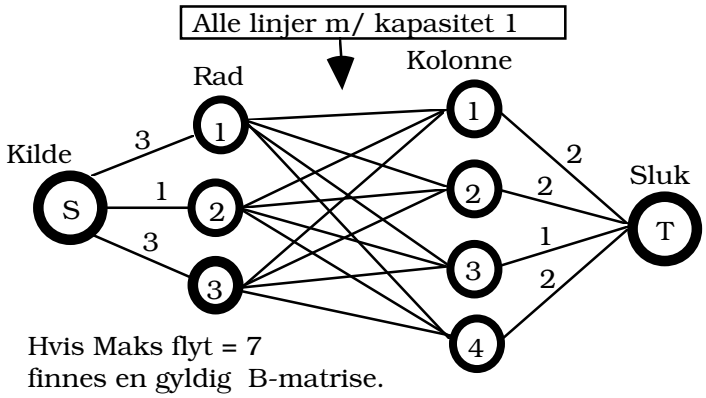


Prosedyren avslutter med å lage en strek tilbake til utgangs-punktet (x,y). Det er derfor lett å fortsette tegningen uten å nøste rekursivt underveis.

Oppgave 2 (30 %)

(a) (15 %)

Det generelle tilfellet lar seg lett avlede fra probleminstansen i oppgaveteksten:



Hvis Maks flyt = 7
finnes en gyldig B-matrise.
En vil da ha:
B(i,j) = flyten fra Radnode #i til Kolonnenode #j.

Merk at tidskompleksiteten her blir θ (r(1) + r(2) + ... + r(m)), altså bedre enn O(n*m). Slik er det fordi antall gjennomløp (=antall ganger vi kan finne en flytforøkende vei) er begrenset av (verdien av) den maksimale flyten (=7 i vårt eksempel). Alle parametre i flyt-maks-algoritmen er heltall, og hver flytforøkende vei vil derfor gi minst 1 enhet økning. Når kapasiteten er 1 for *alle* rad til kolonne-noder, kan vi heller ikke finne flytforøkende veier som gir mer enn 1 enhet flytøkning. Merk at tilbakesporing "automatisk takles" av flyt-maks-algoritmen ved at en verdi kan sendes "motstrøms", det vil si at en "allerede sendt" enhet kan "sendes tilbake". I vårt eksempel blir dette tilfellet ved den siste flytforøkende vei som blir funnet.

(b) (15 %)

Det er *mulig* at det finnes enklere, spesialiserte algoritmer for løsning av problemet, men disse vil trolig være ganske like flyt- maks- løsningen fra (a). Dersom disse algoritmene ikke har behov for tilbakesporing er det grunn til å mistenke forslagene for ikke å være korrekte. Dersom, nemlig, en "rett-fram"-algoritme uten noen form for tilbaksesporing virker, vil dette kunne bety viktige forenklinger i.f.t. den veletablerte løsningen (a). Det må uansett følge en overbevisende argumentasjon for at en enklere alternativ algoritme generelt er korrekt.

Oppgave 3 (30 %).

(a) (10 %)

Dette vises lettest ved et eksempel der en grådighetsalgoritme vil finne for mange mynter.
Sett at myntenhetene består av M = (1,4,6) og B = 8.
En grådighetsalgoritme vil finne at 3 mynter (6,1,1) trengs, mens det minimale antallet mynter åpenbart er 2 (4,4).

(b) (20 %)

Dette er standard, enkleste sort DP. Anta at c(i,j) er det minimale antall mynter som trengs dersom beløpet er j og kun myntsorter M(1),M(2), ... , M(i) tillates brukt. Vi vil da ha 2 alternativer, enten å ikke benytte mynttype nr. i, altså få løsningen c(i-1,j), eller benytte mynttype nr. i og derved få løsningen c(i,j-M(i)) + 1. Best resultat oppnås da ved å velge

c(i,j) = min(c(i-1,j), c(i,j-M(i)) + 1)

Det er ikke spurt etter et eksempel, men et slikt kan være:

	Beløp B								
	0	1	2	3	4	5	6	7	8
M(1) = 1	0	1	2	3	4	5	6	7	8
M(2) = 4	0	1	2	3	1	2	3	4	2
M(3) = 6	0	1	2	3	1	2	1	2	2

Vi har

```
function Myntantall(B,k) : integer;
....erklæringer;
begin
for  i : = 1 to k do c(i,0) : = 0 ;
for  j : = 0 to B do c(1,j) : = j ; / Her brukes at M(1) = 1/

for  i : = 2 to k do
        for j : = 1 to B do
                c(i,j) : =  if j < M(i) then c(i-1,j)
                                else min(c(i-1,j), c(i,j-M(i)) + 1)
Myntantall  : =  c(k,B)
end;
```

Oppgave 4 (20 %).

a) (10 %)

Ja, dette er mulig ved generelt å benytte en algoritme som er "for generell" i forhold til den aktuelle problem-instansen:

Sett at vi har en generell algoritme for å løse f.eks. k-fargbarhets-problemet, eller k-klikk-problemet, begge NP-komplette. Begge disse problemene er trivielle og løst i polynomisk tid dersom k = 2. En generell, og god, algoritme vil arbeide korrekt og effektivt ved enkle problem-instanser. Forenklede problem-instanser kan kreve kun polynomisk tid der det generaliserte problem er NP-komplett. Dersom det generaliserte problem nevner f.eks. "nettverk", kan dette i en spesiell problem-instans være degenerert til "tre" eller "liste".

b) (10 %)

Anta at det finnes en polynomisk-tid algoritme A0 for å løse problemet S. Vi kunne da med A0 avgjøre om grafen G = (V,E) har en Hamiltonsykel ved å kjøre algoritmen *et polynomisk antall ganger*, en gang for hvert par (u,v) blant V's noder: Dersom G har en Hamiltonsykel HC vil vi garantert finne denne ved at det for (minst) ett par (u0,v0), under gjentatte løsninger av problemet S, finnes en kant fra v0 til u0.

Hvis vi altså kan kjøre A0 et polynomisk antall ganger (en gang for hvert (u,v)-par), og ved dette løse Hamilton-sykel-problemet i polynomisk tid, har vi "knust" klassen NPC, med andre ord vist at P = NP. At så er tilfellet er, som kjent, lite sannsynlig.

Påstanden i oppgaveteksten er derfor korrekt.