UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF 3130/4130: *Algoritmer: Design og effektivitet***
Eksamensdag:	**Fredag 14. desember 2007**
Tid for eksamen:	**Kl. 09.00 til 12.00**
Oppgavesettet er på:	**4 sider**
Vedlegg:	**Ingen**
Tillatte hjelpemidler:	**Alle trykte og skrevne**

Kontrollér at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Les oppgavene nøye, og lykke til!

Med svar-forslag

Oppgave 1 Strengsøk (14%)

a) Tegn et suffix-tre for strengen `GAACAACT`.

b) Beregn shift-verdier for strengen `GAACAACT` slik det gjøres i den forenklede Boyer-Moore-algoritmen (Horspool-algoritmen). Ta utgangspunkt i at alfabetet består av de vanlige norske, store bokstavene (A—Å).

Svarforslag, oppgave 1

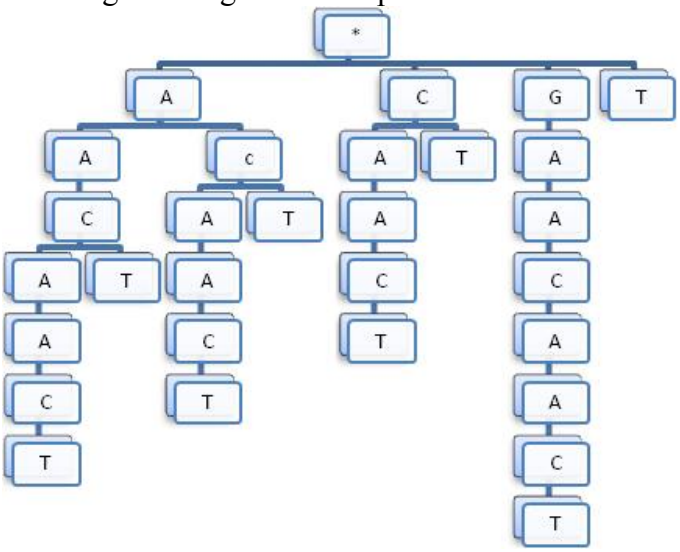
a) Vi er altså gitt følgende streng:

```
GAACAACT
```

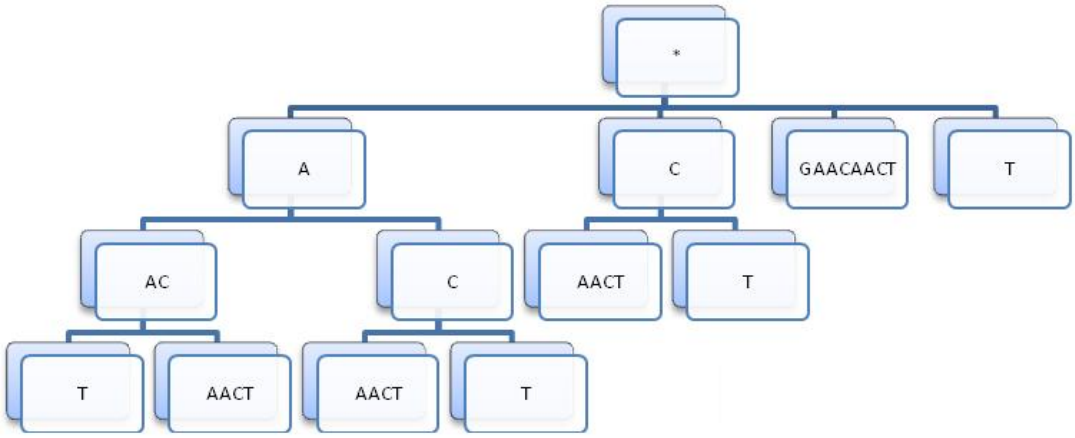
Strengen har følgende suffixer:

```
GAACAACT
 AACAACT
  ACAACT
   CAACT
    AACT
     ACT
      CT
       T
```

Som vel skulle gi oss følgende ukomprimerte suffix-tre:



Og altså i komprimert form, som suffix-tre:



b) Vi skal her altså finne shift-avstandene som brukes i Horspool-algoritmen. Det dreier seg bare om å finne (korteste) avstand en bokstav har fra strengens (patterents) ende.

`G`	7	(Nærmeste (eneste) `G` i avstand 7 fra enden)
`A`	2	(Nærmenste `A` i avstand 2 fra enden)
`C`	1	(Nærmeste `C` i avstand 1 fra enden)
`T`	8	(Endetegnet får ingen lavere verdi fra strengen)
`{A, …, Å}\{G,A,C,T}`	8	(Resten får strengens lengde)

Oppgave 2 A-søk og heuristikk-funksjoner (16%)*

a) Vi er gitt et enkelt brikkespill bestående av n hvite (**H**) og n sorte (**S**) brikker, samt en åpen rute, som vist på figuren under.

1	2	…	n	$n+1$	$n+2$	$n+3$	…	$2n+1$
S	**S**	**…**	**S**		**H**	**H**	**…**	**H**

Målet er å flytte alle de hvite brikkene over på venstre side av brettet og de sorte på høyre side. Hvor den åpne ruten havner er uinteressant, og vi bryr oss heller ikke om den innbyrdes rekkefølgen av de hvite og de sorte brikkene. I slutt-tilstanden skal det altså ikke være noen sorte brikker til venstre for (med lavere indeks enn) noen hvit brikke. Vi kan gjøre to slags flytt:

- Flytte en nabo-brikke av den åpne ruten inn i den åpne ruten. Dette har kostnad 1.
- Hoppe over nøyaktig én brikke, inn i den åpne ruten. Dette har kostnad 2.

Vurder om følgende forslag til heuristikker, angitt for en generell spill-situasjon, er monotone eller ikke:

1. Antall enkelt-flytt (av kostnad 1) som trengs for å flytte alle H-brikkene helt til venstre (om det ikke var noen S-brikker) pluss tilsvarende antall enkeltflytt det tar å flytte alle S brikker helt til høyre.

2. For hver S-brikke teller vi opp antall H-brikker til høyre for denne. Så summerer vi opp disse verdiene for alle S-brikkene.

b) Når vi arbeider med A*-søk og heuristikk-funksjoner, hender det ofte at vi kan bruke forenklede versjoner av vårt opprinnelige problem til å lage heuristikk-funksjonene våre. Professor Max har på denne måten laget tre ulike heuristikk-funksjoner for et problem han arbeider med: h_1, h_2 og h_3. Han har verifisert at alle de tre funksjonene er monotone, men han har ikke klart å avgjøre hvilken han skal bruke. For noen noder ligger heuristikk h_1 nærmest den faktiske kostnaden, for andre h_2 eller h_3. Professorens datter foreslår at han benytter funksjonen

$$h(x) = \max\{h_1(x), h_2(x), h_3(x)\} .$$

Kan han det? Forklar hvorfor/hvorfor ikke.

Svarforslag, oppgave 2

a) For at en heuristikk-funksjon h skal være monoton, må følgende holde:

i. For enhver måltilstand X skal vi ha $h(X) = 0$

ii. For enhver lovlig overgang fra M til N skal vi ha $h(M) \leq h(N) + \text{cost}(M, N)$.

1) Denne er ikke monoton, for eksempel vil tilstanden **HHHHSS_SS** , som er en lovlig slutt-tilstand, ha heuristikk-verdi 2. Et brudd på monotonitetskravet (punkt i).

2) Denne er monoton. La oss kalle heuristikken h, og se på kravene hver for seg:

i. Dette kravet er trivielt oppfylt, ettersom det i en måltilstand, uansett hvor den åpne ruten befinner seg, ikke vil være noen hvite brikker til høyre for noen svart brikke.

ii. Dette kravet er også oppfylt ettersom et enkeltflytt av kostnad 1 aldri vil endre på heuristikk-verdien, slik at vi alltid vil ha

$h(N) = h(M)$

for en overgang fra M til N, og altså

$h(M) < h(M) + 1$ (dette holder jo alltid).

Med likheten over får vi:

$h(M) \leq h(N) + 1$,

som er det som må vises.

For et hopp vil vi aldri endre heuristikken med mer enn 1, ettersom vi bare kan hoppe over 1 rute, mens kostnaden er 2. Vi vil altså ha

$h(N) \geq h(M) - 1$, eller

$h(N) + 1 \geq h(M)$.

Dette gir oss:

$h(M) < h(M) + 1 \leq h(N) + 1 + 1$

$h(M) \leq h(N) + 2$,

som er det som må vises.

b) Ja, professoren kan benytte datterens foreslåtte funksjon. Denne vil også være monoton om delfunksjonene er det.

i-kravet er trivielt oppfylt når delfunksjonene er monotone. Vi må vise at ii-kravet også er oppfylt for h når h_i-ene er monotone. Vi kan uten tap av generalitet se på bare to av delfunksjonene, h_1 og h_2. Vi antar (får å ende opp med en selvmotsigelse) at:

$h_1(M) \leq h_1(N) + \text{cost}(M, N)$,

$h_2(M) \leq h_2(N) + \text{cost}(M, N)$ (delfunksjonene er monotone),

men at

$h(M) > h(N) + \text{cost}(M, N)$ (sammensatt funksjon ikke er monoton). *

La så $h(N)$ være "dominert" av h_1 og $h(M)$ være "dominert" av h_2, slik at

$h(N) = h_1(N) > h_2(N)$ og **

$h(M) = h_2(M) > h_1(M)$. ***

Da vil vi ha

$h_2(M) > h_1(N) + \text{cost}(M, N)$ (fra * med ** og ***),

som gir oss

$h_2(M) > h_2(N) + \text{cost}(M, N)$ (fra **),

en selvmotsigelse når h_2 er monoton i utgangspunktet.

Alternativt: Vi konstaterer først at om $a1 \leq b1$ og $a2 \leq b2$, så er også $max(a1, a2) \leq max(b1, b2)$. Dette konstateres ved enkel case-by-case-gjennomgang. Vi vet altså fra forutsetningen (se over):

$$h_1(M) \leq h_1(N) + \text{cost}(M, N) \quad \text{og} \quad h_2(M) \leq h_2(N) + \text{cost}(M, N)$$

Dermed er også $max(h_1(M) , h_2(M)) \leq max(h_1(N) + cost(M, N), h_2(N) + cost(M, N))$
Men siden $cost(M,N)$ er en konstant kan den holdes utenfor, og vi får vi det vi vil ha, nemlig:

$$max(h_1(M) , h_2(M)) \leq max(h_1(N), h_2(N)) + cost(M, N))$$

Oppgave 3 Dynamisk programmering (13%)

En lang tømmerstokk av lengde l skal kappes i mindre deler på et sagbruk. På grunn av kvist, stokkens tykkelse, hva delene skal brukes til ol., er posisjonene hvor stokken skal kappes bestemt på forhånd. Disse er: $p_1, p_2, ..., p_{n-1}$, målt fra den venstre enden av stokken. I tillegg lar vi $p_0=0$ være den venstre enden av stokken, og $p_n=l$ den høyre. Etter å ha delt den opprinnelige stokken én gang har vi to nye stokker, etter å ha delt den to ganger har vi tre, osv.

På sagbruket går stokkene på transportbånd, slik at det tar tid t å kappe en stokk som er t lang uansett hvor den kappes. Stokken kan bare kappes én gang per gjennomkjøring, de to nye delene må eventuelt prosesseres på nytt, hver for seg, om de skal kappes ytterligere. Stokker som ikke skal kappes mer kjøres ikke gjennom saga, disse tar altså tid 0.

Eksempel: En 10 meter lang stokk skal kappes ved p_1=2 meter, p_2=5 meter, p_3=8 meter. Kapper vi først ved p_3, så ved p_2, og så ved p_1, vil dette ta tid 10+8+5=23. Kapper vi først ved p_2, og deretter p_1 og p_3, vil det ta tid 10+5+5=20, altså noe raskere.

Oppgaven er: Definér en formel (en rekurrensrelasjon) som kan brukes i dynamisk programmering for å finne raskeste måte å kappe den opprinnelige stokken på, og vis hva initialbetingelsene vil være. Skissér gjerne et program, om du syns det er enklere enn å definere en rent matematisk formel.

Svarforslag, Oppgave 3

For $0 \le i \le j \le n$, lar vi c_{ij} være kostnaden av å kutte (del)stokken med endepunkter i og j. Vi kan bruke følgende rekurrensrelasjon, som vi altså fyller ut i en to-dimensjonal tabell $c[0,i][0,j]$:

$$c_{ij} = \min_k\{c_{ik} + c_{kj} + (p_j - p_i) : 0 \le i < k < j \le n\}.$$

Initialbetingelsene vil være k_{ii+1}=0.
Cluet er jo å se at dette er det samme som matrisemultiplikasjon, men at det her vil være en litt annen måte å beregne kostnaden på nå vi setter sammen to deler. Her vil det være lengden av den aktuelle stokken, avstanden mellom endepunktene, som vi finner ved $p_j - p_i$.

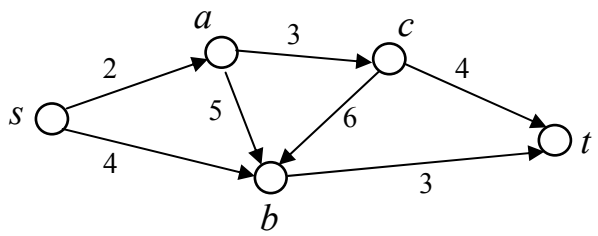
Om man heller vil skrive kode, kan det se noe slikt ut:

```
for i = 0 to n-1 do
  c[i, i+1] = 0                    // initialbetingelsene (diag=1)
od
for diag = 2 to n do
  for i = 0 to n-diag do           // beregner ihht
    j = i+diag                     // rekurrensrelasjonen
    min = ∞                        // angitt over (finner beste k)
    tempcut = i                    // på samme måte som på side
    for k = i+1 to j-1 do          // 271 i læreboka
      temp = c[i,k] + c[k,j] + (p[j]-p[i])
      if temp < min then
        min = temp
        tempcut = k
      fi
    od
    c[i,j] = min
    firstcut[i,j] = tempcut
  od
od
```

Oppgave 4 Flyt i nettverk (21%)

Vi skal arbeide med flyt i (rettede) nettverk av den typen som er beskrevet i kap 14.2. Tallene på kantene, som der brukes som kapasiteter (øvre grense for flyt) skal vi her tolke som *nedre* grense for flyt i den kanten, og for kanten *K* skal vi kalle denne *minflyt(K)*. Dette kan f.eks. være aktuelt for rør-systemer i bakken, der det må være en viss flyt så de ikke fryser. Vi ønsker imidlertid at den totale flyten fra *s* til *t* skal være så lav som mulig. En flyt som ikke er mindre enn *minflyt(K)* for noen kant *K* sies å være *lovlig*.

Verdien *minflyt(K)* er for alle *K* et heltall som er null eller positivt. Vi har ingen øvre grense for hvor stor flyten kan være i en kant. De nettverkene vi arbeider med her har ikke rettede løkker, og alle kanter ligger på en rettet vei fra *s* til *t*. Eksempel på nettverk, der *minflyt*-verdiene er satt på kantene:



a) Vi vil lage en algoritme som finner minimal flyt, slik den er angitt over. Algoritmen får ved starten oppgitt en lovlig flyt fra *s* til *t* (som altså tilfredsstiller minflyt-kravet på hver kant, men kan være unødvendig stor). Forklar hvordan vi kan forandre stegene i FordFulkerson-algoritmen så vi kan arbeide oss mot en mindre og mindre, men lovlig, flyt. Det er helt greit å bruke ord til å forklare forandringene, men en programskisse er også OK.

b) Vi vil se på situasjonen når algoritmen fra oppgave a) stopper. Vi får da en situasjon med min-flyt/maks-kutt tilsvarende max-flyt/min-kutt for FordFulkerson. Hvordan vil et maks-kutt generelt se ut, og hvordan kan man finne et slikt når algoritmen stopper?

c) Som angitt over trenger algoritmen fra oppgave a) en lovlig startflyt. Forklar hvordan man kan finne en slik. Du behøver ikke legge vekt på at denne startflyten skal være spesielt lav.

Svarforslag Oppgave 4

a)

Vi kaller det gitte nettverket for N, og refererer til kantene i dette som N-kanter. Vi lager nettverket N*f* ut fra den filosofi at flyten på hver kant K må være mellom *minflyt(K)* og +∞. Vi merker oss da at alle "foroverkanter" i N*f* (altså de som går i samme retning som den underliggende kanten i N), alltid har forøkningsmulighet +∞. De motsatte kantene i N*f* ("bakoverkantene") har imidlertid alltid endelig forøkningsmulighet (nemlig det flyten *f* i N-kanten kan minskes med uten at flyten blir lavere enn *minflyt(K)*).

Vi er ute etter å minske flyten, og vi leter derfor etter en vei V i N*f* fra t til s. Om vi finner en slik, kan vi bruke den slik: Finn den kanten i N*f* med minst forøkningsmulighet (fra *t* til *s*, altså *senknings*-mulighet av flyten fra s til t). Denne kanten er helt sikkert en bakoverkant, og tilsvarer den N-kanten langs V der vi ville gjøre flyten ulovlig om vi senket flyten mer enn dette.

Det er viktig at ikke alle kantene langs V i N*f* kan ha uendelig forøkningsmulighet. Det ville tilsvare at det gikk en vei fra t til s i N. En slik finnes ikke fordi det ikke finnes kanter inn til s eller ut av t (og det ville også lage løkker i grafen). Dermed er minimumet nevnt over alltid veldefinert.

b)
Siden minflyt-verdiene bare er heltallige vil algoritmen til slutt stoppe, siden det ikke kan bli mindre enn null flyt fra s til t, og siden forandringen hver gang blir minst 1 (men dette var det egentlig ikke spurt etter). Når det stopper er det fordi vi ikke finner noen vei fra t til s i N*f*. Vi kan da se på mengden T av noder som vi kan nå fra t i N*f*, og si at de er den ene delen av kuttet, mens den andre delen av kuttet, S, er resten av nodene og inneholder altså noden s.

Hva karakteriserer så dette kuttet? Tenker vi på kanter i N så kan det ikke gå slike fra T til S. Disse ville representert en kant fra T til S i N*f*, og den ville være en foroverkant, og dermed ha forøkningsmulighet lik $+\infty$. Da ville søket fra t ikke stoppet på denne måten, denne kanten i N*f* ville også kommet med, og endenoden ville vært i T og ikke i S. Når det gjelder kanter K i N fra S til T så må alle disse ha flyt lik minflyt(K), ellers ville heller ikke søket ha stoppet på denne måten (det ville da vært en bakover-kant i N*f* fra T til S). Altså, et maksimalt kutt består av to mengder S (med s) og T (med t) slik at det i N ikke går kanter fra T til S, og slik at alle N-kanter fra S til T har minimal flyt. Når algoritmen slutter har vi nettopp en slik situasjon.

Det er rett fram å vise at ingen lovlig flyt kan ha flyt mindre enn "nedad-kapasiteten" til dette kuttet, og siden vi nettopp har en flyt som er lik denne nedad-kapasiteten, gjelder et max-kutt/min-flyt-forhold her, og algoritmen viser oss et slikt kutt/flyt-par når den stopper.

c)
For å finne en startflyt som for alle kanter K ikke er mindre enn *minflyt(K)*, kan vi gjøre som følger: Sett først flyten i alle kanter til 0. Gå så gjennom alle kantene K= (u->v) i vilkårlig rekkefølge, og gjør følgende: Om K ikke allerede har flyt større eller lik *minflyt(K)*, så finn en vei fra s til u og fra v til t. Slike skal ut fra forutsetningen finnes. Sammen med K vil disse danne en vei i N fra s til t, og vi øker så flyten langs denne så mye at flyten over K blir *minflyt(K)*. Når dette er gjort for alle K vil vi opplagt ha en lovlig flyt, siden vi i hvert steg aldri minker flyten over en kant.

Oppgave 5 Uavgjørbarhet (20%)

Hvilke av følgende språk er avgjørbare? Skissér kort et bevis for hvert av svarene.

a) $L_1 = \{M, x$: Med input x stopper maskin M etter ikke mer enn $|x|$ skritt $\}$

b) $L_2 = \{M$: Maskin M stopper for enhver input med lengde høyst 10 $\}$

c) $L_3 = \{M, x$: Maskin M stopper ikke med input x $\}$

d) $L_4 = \{M$: Maskin M flytter aldri lesehodet til venstre når den startes med blank input $\}$

e) $L_5 = \{M$: Maskin M aksepterer eksakt en streng $\}$

Svarforslag Oppgave 5

a)
L1 er avgjørbar, og dette kan vises ved eksistens av en Turing-maskin som avgjør problemet: En Turing-maskin kan greit lese over x-delen av input uten å modifisere for å telle lengden av det og lagre den i K. Siden det er vist at en universll Turing-maskin eksisterer (UTM, en Turing-maskin som kan emulere andre Turing-maskiner) kan maskinen vår så emulere M på x i maks K steg -- om maskinen stopper før det, svar Y, ellers N.

b)
L2 er ikke avgjørbar, via standardreduksjonen fra stoppeproblemet der en bruker følgende M':

```
Simuler M på x
Stopp
```

M' ser ikke på sin input i det hele tatt (M og x er kodet i tilstandene til M', og er ikke en del av input), så M' stopper på input av lengde <= 10 (og alle andre input også...) hvis og bare hvis M stopper på x, som er det en trenger i standardreduksjonen fra stoppeproblemet for å vise at L2 er uavgjørbart.

c)
L3 er komplementproblemet til stoppeproblemet, og er dermed ikke avgjørbart siden stoppeproblemet ikke er avgjørbart. (Dersom det finnes en maskin M3 som avgjør L3 så kunne en bare snudd svaret den maskina gir for å løse stoppeproblemet. Siden dette er umulig kan ikke en slik M3 eksistere, og da er L3 uavgjørbart.)

d)
L4 er avgjørbar. Merk at L4 innbefatter f.eks. både maskiner som går til høyre i det uendelige, og maskiner som umiddelbart stopper.

Dette kan bevises ved å oppgi en algoritme: Betrakt overgangsfunksjon delta(state, tapesymbol). Så vi kan starte i start-tilstanden og følge overgangsfunksjonen (som en graf). Om vi kommer tilbake til en allerede besøkt tilstand svarer vi Y, om vi kommer til en tilstand som stopper maskinen svarer vi Y, men om vi kommer til en tilstand der maskinen vil ta et steg til venstre svarer vi N.

Mens en går gjennom overgangsfunksjonen kan en hele tiden anta at tapesymbolet er blankt, siden maskinen ikke kan se på input den allerede har skrevet ut med mindre den går til venstre.

(For å virkelig forstå dette er det verdt å filosofere på hvorfor f.eks. ikke den vanlige reduksjonen fra stoppebeviset fungerer her.)

e)
L5 er uavgjørbar ved standardreduksjon fra stoppeproblemet. Angir som før bare M':

```
Simuler M på x
(M og x er som før her bygd inn i maskinen, og ikke en del av input til M')
Se på input. Dersom det er strengen "010", svar Y
Gå inn i en uendelig løkke.
```

Da vil M' akseptere eksakt strengen 010 hvis og bare hvis M stopper på x, som er det vi behøver å plugge inn i stoppebeviset.

Oppgave 6 Kompleksitet (16%)

Med en vellykket kombinasjon av nanoteknologi, genemanipulering, lim og alkohol, klarte Gustav Hamilton ved Thinking Machines International å skape en spesialisert maskin som løser Hamiltonbarhet (Hamiltonicity) i polynomisk tid. Vurder og argumenter for/imot følgende som mulige konsekvenser av Gustavs oppfinnelse:

a) Tilfredstillbarhet (SAT) kan løses i polynomisk tid.

b) Ikke-tilfredstillbarhet (NON-SAT) kan løses i polynomisk tid.

c) Alle umedgjørlige (intractable) problemer kan løses i polynomisk tid.

d) Optimaliseringsversjonen av Handelsreisendesproblem (TSP-optimization) kan løses i polynomisk tid.

Svarforslag Oppgave 6

Hovedpoenget her er at siden HAM er NP-komplett finnes det en reduksjon på (deterministisk) polynomisk tid fra alle problemer i NP til HAM.

a)
Ja. SAT ligger i NP, så med en vanlig datamaskin kan en på polynomisk tid omforme SAT-instansen til en HAM-instans og fore HAM-instansen til Gustavs maskin.

b)
Kort svar: Ja, ta svaret gitt i a) og bare snu Y-output til N-output og omvendt.

Lenger svar: Dette kommer litt an på hvordan en tolker ordet "løser" i oppgaveteksten. Om en har en maskin som avgjør problemet (dvs. svarer Y eller N), og det virker som om det er det en har her, kan en alltid bare snu på output.

Om Gustavs maskin derimot er av typen som bare aksepterer problemet, dvs svarer Y eller holder på uendelig lenge, så er svaret nei, da kan ikke NON-SAT løses. Det er denne siste typen maskiner vi har holdt på med når en skal vise NP-kompletthet av problemer.

c)
Nei. Det finnes mange problemer som ikke er i NP (men er verre), og disse kan ikke løses. Eksempel: Sjakk.

d)

Ja. Forutsetter heltallskostnader for enkelthets skyld.

Avgjørbarhetsversjonen av TSP ("finnes en TSP av lengde k?") er i NP. Dette kan antas som kjent, eller også holder det å finne finne et sertifikat: En TSP-vei av lengde k er et sertifikat, så avgjørbarhets-TSP er i NP. Dermed har avgjørbarhets-TSP en polynomisk reduksjon til HAM og kan avgjøres av Gustavs maskin.

Optimaliserings-TSP kan så implementeres med to serier av repetisjoner av avgjørbarhets-TSP:

i) Finn kostnaden til veien. Finn først en maksimumkost M (f.eks. ved å summere kostnaden til alle kantene) og så gjøre et binærsøk mellom M og 0 på avgjørbarhetsproblemet. (M er eksponentielt stor med tanke på input, men et binærsøk tar O(log M) som bare polynomisk stor, så dette går bra.)

ii) Når en har funnet den optimale kostnaden k, så spør en fortsatt "Finnes en TSP med lengde k", men fjerner en kant fra grafen for hver gang. Om en får N som svar kan ikke den kanten fjernes uten at den beste veien forsvinner. Til slutt sitter en igjen med akkurat de rette kantene, og dette trengs ikke gjøres flere ganger enn det er kanter i grafen (=polynomisk).

(Slutt på oppgavesettet)