Løsningsforslag for eksamen i fag
SIF8010 Algoritmer og Datastrukturer
Tirsdag 18. Desember 2000, kl 0900-1500

Faglig kontakt under eksamen: Arne Halaas, tlf. 73 593442.
Hjelpemidler: Alle kalkulatortyper tillatt. Alle trykte og håndskrevne hjelpemidler tillatt.
Rubrikksvar: Alle svar skal avgis i angitte svar-ruter. Ikke legg ved ekstra ark som svar.
Krav: Det kreves "bestått" både på de ordinære og på de øvingsrelaterte spørsmål.
Husk: Fyll inn rubrikken "Student nr." øverst på alle ark.

Oppgave 1. (20%)

Følgende 10 påstander skal besvares og begrunnes:

a) Alle algoritmer MinMax for å finne både et største og et minste element i en to-dimensjonal heltalls-tabell med **n** rader og **n** kolonner har tidskompleksitet $\mathbf{\Omega(n^2)}$.

b) Sortering ved innsetting (n elementer) har tidskompleksitet $\mathbf{\Omega(n)}$.

c) Boblesortering (**n** elementer) har tidskompleksitet $\mathbf{O(n^2 \log n)}$.

d) Spredt lagring (hash-teknikk) kan effektivt brukes ved sortering.

e) Dijkstras algoritme kan brukes til å finne alle-til-alle (noder) korteste vei.

f) Prefiks-traversering av et binært søketre svarer til sortert rekkefølge.

g) I Ford-Fulkerson's algoritme kan en velge å gi kantene både en øvre og en nedre flytgrense > 0.

h) Dersom en graf **G=(V,E)** er et tre, kan en i **O(1)** tid avgjøre om **G** er tofargbar.

i) Kruskals og Prims algoritmer gir alltid like korte (minimale), men generelt forskjellige spenntrær.

j) Dersom et gitt problem i sin generelle form er **NP**-komplett, har det ingen hensikt å prøve og løse dette når problemets størrelse øker over en viss terskel.

Svar: (Stryk "Ja" eller "Nei". Begrunnelsen må fylles ut. Hvert delsvar teller 2%)

a) Ja/~~nei~~ Begrunnelse: Alle de n^2 elementene må inspiseres minst én gang.

b) Ja/~~nei~~ Begrunnelse: Som a), men her har vi n elementer.

c) Ja/~~nei~~ Begrunnelse: Boblesortering er $O(n^2)$. $O(n^2 \log n)$ er et slappere krav enn dette.

d) Ja/~~nei~~ Begrunnelse: Spredt lagring er den mest effektive måten å sortere på, i de tilfeller der nøkkelverdiene gjør det mulig å benytte teknikken.
(F.eks: Plasser tall i én av hundre båser, basert på første to siffer ($O(n)$). Deretter Sortér deretter hver bås for seg.)

e) Ja/~~nei~~ Begrunnelse: Ved å kjøre algoritmen $|V|$ ganger – en gang for hver startnode.

f) ~~Ja~~/nei Begrunnelse: Infiks-traversering svarer til sortert gjennomløp.

g) ~~Ja~~/nei Begrunnelse: Nedre flytgrense i Ford-Fulkerson er alltid lik 0.

h) Ja/~~nei~~ Begrunnelse: Ja, fordi alle trær *er* tofargbare (dvs., vi behøver ikke undersøke trær med hensyn til tofargbarhet.)

i) Ja/~~nei~~ Begrunnelse: Generelt forskjellige trær, men ofte like.
Alltid samme sum/lengde, den minimale.

j) ~~Ja~~/nei Begrunnelse: Den spesielle formen på problemet (datasettet) kan være slik at problemet likevel lar seg løse effektivt. (F.eks: Grafen er et tre.)

Oppgave 2. (15%)

Et meget stort datasett (forskjellige positive heltall) **T[1..n]** skal gjøres tilgjengelig på internett for mange hyppige brukere. Datasettet oppdateres ukentlig. Du blir gitt oppgaven å lage en så effektiv som mulig algoritme som gjør følgende:

Innverdier (kun heltall): **T,n,a** og **b** (der **a** og **b** er gitt av brukeren)
Utverdier (heltall): **k** og **M** = Medianen (midtverdien) blant alle **k** verdier **{T[i]}** som tilfredsstiller kravet **a ≤ T[i] ≤ b**.

Svar (algoritmeskisse): 15%

Ukentlig: Sorter T; La T[0] = –∞, T[n+1] = ∞. (Kompleksitet: $O(n \log n)$)
Ved hver bruk: (m ganger per uke)

1. Binærsøk i T med a og b for å finne nærmeste øvre indeks i og nærmeste nedre indeks j slik at $T[i] \le a < T[i+1]$ og $T[j-1] < b \le T[j]$. (Kompleksitet: 2 $O(\log n)$)
2. Sett $k = j - i + 1$.
   Hvis $k \ge 1$: M = T[*floor*((i+j)/2)] (hvor funksjonen *floor* runder nedover)
   (Kompleksitet: $O(1)$)

Tidskompleksitet: $O(\log n)$ per medianfinning, eller $O(n \log n) + mO(\log n)$ per uke
($= O((m+n)\log n)$)
Begrunnelse for algoritmevalget: Antar at sortering lønner seg fordi m er "stor".
For små m finnes $O(n)$-metode (Randomized Select) som kan brukes til medianfinning.

Oppgave 3. (20%)

Du har gitt **n** punkter i **x-y**-planet, **P1:(x1,y1),P2:(x2,y2),...,Pn:(xn,yn)**. Du skal finne en beste vei fra **P1** til **Pn**, via et utvalg av de øvrige **(n-2)** punktene, som er slik at den lengste avstanden mellom nabopunkter på veien er så kort som mulig. (Tenk deg at du skal hoppe på stener over en elv og at det lengste hoppet skal være så kort som mulig.)

(a) Skisser en så effektiv som mulig algoritme som løser dette beste-vei problemet ved å bruke Dijkstras algoritme (uendret) som subrutine.

Svar: 8%
Sett $D = dist(P1, Pn)$ (Euklidsk avstand).
Finn den minste "største-kanten" D som gir en løsning vha. binærsøk:
1. Sett $i = 0$, $j = D$.
2. Finn korteste vei (med Dijkstra) når bare kanter med lengde mindre eller lik $(i+j)/2$ tas med.
3. Fantes en slik vei?
   Ja: Sett $j = (i+j)/2$
   Nei: Sett $i = (i+j)/2$
4. Hvis $|i-j| < \varepsilon$ (hvis vektene er reelle) eller hvis $i=j$ (hvis vektene er heltall), sett $D = i$.
   Hvis ikke, gå til 1.

(*Dijkstras algoritme i et minimalt spenntre vil også gi beste vei – men det bør begrunnes godt hvorfor dette er riktig. Man kunne da like godt ha brukt DFS eller BFS.*)

Algoritmens tidskompleksitet: $O((n \log n) \cdot \log D)$ (Antar her at $\varepsilon=1$)
Begrunnelse: Ikke den mest effektive algoritmen, men bruker Dijkstras uendret.

(b) Finn en mer effektiv algoritme for å løse vårt beste-vei-problem ved å modifisere koden i Dijkstras algoritme. Vis modifikasjonene i detalj.

Svar: 12%

Lager ny versjon av Relax:
Relax-Max(u,v,w):
1. **if** $d[v] > \max\{d[u], w(u,v)\}$
2. **then** $d[v] \leftarrow \max\{d[u], w(u,v)\}$
3. $\pi[v] \leftarrow u$

(Altså – kun 2 linjer må endres)

(*Er treet som bygges av denne modifiserte algoritmen et minimalt spenntre?*)

Algoritmens tidskompleksitet: $O(n \log n)$ (Samme som Dijkstra)
Begrunnelse: Enkel modifisering – bevis som for den ordinære algoritmen med "max" istedenfor "+"

Oppgave 4. (20%)

La **G=(V,E)** være en sammenhengende, urettet og vektet graf. Anta at kantenes vekter er forskjellige positive heltall. Vi benevner **G**'s minimale spenntre for **MST**.

(a) Anta at vi øker alle vektene for kanter i **E** med en konstant verdi **c > 0**. Er **MST** fortsatt et minimalt spenntre for **G**? Hvis ikke, hvordan finner vi mest effektivt **G**'s nye spenntre?

Svar: 6%

Spenntre-algoritmene påvirkes kun av de *relative* lengdene kantene imellom.
($a \leq b \Rightarrow a+c \leq b+c$)
MST er altså *uendret* selv om alle lengdene økes med en konstant $c>0$.

(b) La nå **e** være en vilkårlig kant i **E**, og la **T(e)** være det spenntre i **G** som har minst kostnad av alle spenntrær som inneholder kanten **e.** Beskriv en så effektiv som mulig algoritme som finner **T(e)** for samtlige $|\mathbf{E}|$ kanter **e** ε **E.** Finn også den foreslåtte algoritmens kompleksitet.

Svar: 14%

1. Lag et minimalt spenntre; kall dette T
2. Hvis *e* ligger i T så har vi at T(*e*) = T
3. Preprosessering: Bruk Dybde-Først-Søk på T for å finne den største kant-kostnaden for (den entydige) veien fra *v* til *w* i T, for alle par (*v*,*w*). (Vi trenger ikke se på *alle* par, men det er enklest.)

4. Finner T(*e*) for alle kanter *utenfor* T ved å føye *e*=(*x*,*y*) til T og deretter fjerne den største kanten funnet i punkt 3 for veien fra *y* til *x*. Denne finner vi lett ved hjelp av preprosesseringen i punkt 3.

Algoritmens kompleksitet: $O(|V|^2)$ (Preprosesseringen dominerer kjøretiden.)

Oppgave 5 (Øvingsrelatert) (25%)

a) Hvor effektivt kan vi sortere n tall i tallområdet $0..n^n$? Begrunn svaret.

3%: n tall kan sorteres i $O(n \log n)$ tid hvis man antar at tallene kan sammenlignes i konstant tid. Hvis dette ikke gjelder (grunnet stort tallområde) kan man bruke radix-sortering med n siffer og man får en kjøretid på $O(n^2)$. (Hvert siffer kan fremdeles sammenlignes i konstant tid.)

b) Hva er tidskompleksiteten til 0-1 ryggsekk-problemet? Begrunn svaret.

2%: Med dynamisk programmering kan problemet løses i $O(nC)$, der n er antallet gjenstander, og C er kapasiteten til sekken. Dette innses fordi vi skal fylle en tabell med størrelse $n \cdot C$.

c) Hvorfor har tidskompleksiteten til innsettings-sortering samme *worst-case* og *average-case*?

2%: I gjennomsnitt må $O(n)$ elementer flyttes på plass, og hvert av dem må flyttes i gjennomsnitt $O(n)$ plasser. Det samme gjelder som en øvre grense, som dermed også gjelder for *worst-case*. (I det beste tilfellet vil hvert element flyttes $O(1)$ plasser, noe som gir $O(n)$ i kjøretid.)

d) Når er det lurt å bruke memoisering?

2%: Når vi har et problem der det er naturlig å bruke rekursjon, men der enkelte delproblemer (rekursive kall) overlapper, slik at vi i utgangspunktet får en eksponensiell kjøretid.

(M.a.o.: Naturlig rekursiv løsning + optimal delstruktur + overlappende delproblemer)

e) Vil du bruke *dybde-først-søk* eller *bredde-først-søk* for å finne én-til-alle korteste vei i en graf der alle kantene har vekt 1? (Begrunn.)

2%: Ved å bruke bredde-først-søk vil dybden i søketreet gi avstanden til start-noden; dette er samtidig lengden av den korteste vei. Hvis en bruker DFS må man foreta en god del ekstra unødvendig bokføring.

f) Gi en praktisk anvendelse av *største uavhengige delmengde*.

3%: Vi skal arrangere et *bli kjent*-møte. Vi har et begrenset budsjett og vil derfor ikke invitere noen som kjenner hverandre til møtet. Av en gitt gruppe mennesker vil vi invitere flest mulig innbyrdes ukjente personer. Hvis vi lager en graf med personer som noder og kanter mellom de som kjenner hverandre, vil grafens største uavhengige delmengde gi oss de som skal inviteres.

g) Hvis du har et sett $\{s_i\}$ med n strenger av ulik lengde, der lengden av strengen s_i er $l(s_i)$, hva er kjøretiden for en mest mulig effektiv algoritme som sorterer strengene?

3%: $O(\sum l(s_i))$. Her brukes den samme metoden som i øving 10, oppgave 2. Sorter strengene først etter lengde, og sorter deretter hver gruppe med lik lengde med radiks-sortering. (Antar naturligvis et alfabet av rimelig størrelse, f.eks. {a ... z})

h) Hva er sammenhengen mellom redigerings-avstand (*edit distance*) og lengste felles subsekvens (*longest common subsequence*)?

4%: Hvis redigerings-avstanden er $d(x,y)$, og den lengste felles subsekvensen er $l(x,y)$, har vi:

$$d(x,y) = |x| + |y| - |l(x,y)|,$$

der $|\cdot|$ er lengde-operatoren.

i) I en bok leser vi at 0-1 ryggsekk-problemet er NP-komplett. Hvordan harmonerer dette med svaret ditt på spørsmål b) ?

4%: Det er lite trolig at vi vil finne en polynomisk løsning på et NP-komplett problem; men løsningen er ikke polynomisk, siden den ikke bare er avhengig av størrelsen n, men også parameteren C. Økes vekt-nøyaktigheten med ett siffer så tidobles (minst) kjøretiden.