NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk

Institutt for datateknikk og informasjonsvitenskap

BOKMÅL

LØSNINGSFORSLAG, AVSLUTTENDE EKSAMEN I

TDT4120/IT1105

ALGORITMER OG DATASTRUKTURER

Mandag 5. desember 2005
Kl. 09.00 – 13.00

Faglig kontakt under eksamen:

Arne Halaas, tlf. 416 61 982

Magnus Lie Hetland, tlf. 918 51 949

Hjelpemidler:

Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.

Sensurdato:

5. januar 2006. Resultater gjøres kjent på http://studweb.ntnu.no og sensurtelefon 81548014.

Viktig:

Les hele oppgavesettet før du begynner. Les oppgaveformuleringene grundig. Det er angitt i poeng hvor mye hver deloppgave teller ved sensur. Gjør nødvendige antagelser der dette er nødvendig. Skriv kort og konsist. Skriv fortrinnsvis i rutene på oppgavearket.

Oppgave 1 (18%)

Anta gitt en heltallsvektor $V[1…n]$

a) Finn en mest mulig effektiv algoritme for å avgjøre om det eksisterer en undermengde (et utvalg) av k elementer i V med sum høyst lik ($\leq$) verdien T.

Grunntanken er å effektivt finne de k minste tallene i V.

Det er flere muligheter:

— 3 poeng: Bruk SELECT (eller RANDOMIZED-SELECT) for å finne det k-te minste tallet. Da vil også de som er mindre være plassert "til venstre" for dette, klare til summering. (Om en student oppgis at PARTITION skal brukes i tillegg trekkes det ikke for det.) Kjøretiden blir $\Theta(n)$.

— 2 poeng: Bruk en maks-heap med k elementer og iterer over V (forkast hele tiden det største for å holde størrelsen på k). Kjøretiden blir $\Theta(n \log k)$ i verste tilfelle og $\Theta(n)$ i beste tilfelle.

— 2 poeng: Legg alle elementene inn i en min-heap og trekk ut de k minste. Kjøretiden blir $\Theta(n + k \log n)$.

— 1 poeng: Gå igjennom alle elementene k ganger for å plukke ut det minste av de gjenværende elementene, eller bruk en enkel struktur (sortert eller usortert liste eller tabell) for å plukke ut alle k i én iterasjon. Kjøretiden blir $\Theta(kn)$.

— 1 poeng: Sortér alle elementene og velg de k første. Kjøretiden blir $\Theta(n \log n)$.

(3 poeng)

Det er gitt at $n/4 \leq k \leq n/2$ og at $n > 1000$.

b) Hva er tidskompleksiteten for ditt algoritmeforslag i a) i verste tilfelle (*worst case*)?

Kommer an på valgt løsning i a). (Sett inn n i stedet for k i kjøretidsuttrykkene.) Her gis det kun poeng hvis algoritmen i a) faktisk er en løsning (men det kan gis full score her om løsningen i a) ikke er en *god* løsning).

(2 poeng)

c) Skisser kort en så effektiv som mulig algoritme for å finne de $k < 6$ minste verdiene i V.

Man kan rett og slett iterere over V og hele tiden ta vare på de k minste verdiene. Akkurat hvordan man gjør det er ikke essensielt. (En enkel, uordnet tabell kan være en naturlig løsning. Man kan også bruke en heap.)

(3 poeng)

SORT100($V[i…j]$) er en gitt $O(j–i)$-metode som sørger for at de inntil 100 minste verdiene i $V[i…j]$ blir plassert i sortert rekkefølge i $V[i, i+1, i+2, \ldots, i+k]$, der $k = \min(i+99, j)$, mens de øvrige elementene, hvis disse eksisterer, forblir usorterte og plasseres i $V[i+k+1, i+k+2, i+k+3, \ldots, j]$.

d) Hva er tidskompleksiteten til en algoritme som bruker SORT100 gjentatte ganger for å sortere $V[1\ldots n]$, der n er mye større enn 100? Begrunn svaret.

Her vil man naturlig sortere 100 og 100 tall. Den totale kjøretiden blir en aritmetisk rekke:
$\sum_{i=0}^{n/100} \Theta(n-100i) = \Theta(n+(n-100)+\ldots+100+0) = \Theta(n^2)$. (Her holder en kort, tekstlig begrunnelse.)

(5 poeng)

Anta nå at heltallsvektoren $V[1\ldots n]$ er sortert. Dersom vi søker etter verdien x i V kan vi undersøke midt-verdien i V i forhold til x og derved eliminere halvparten av V i det fortsatte søket. Binærsøk er en algoritme som gjentar denne operasjonen ved å halvere størrelsen på den resterende sekvensen inntil x enten er funnet eller inntil en kan slå fast at x ikke er blant V-verdiene.

e) Skriv en kort, men detaljert, (pseudo)kode for Binærsøk og angi kjøretiden for algoritmen i verste tilfelle.

```
BINARYSEARCH(V, v, l, r)
if r < l
        ERROR("Not found")
mid ← ⌊(l + r) / 2⌋
if V[mid] = v
        return mid
if v < V[mid]
        return BINARYSEARCH(V, v, l, mid–1)
else
        return BINARYSEARCH(V, v, mid+1, r)
```

(Merk: En iterativ løsning aksepteres også.)

Kjøretid: O(log n)

(5 poeng)

Oppgave 2 (17%)

a) La H være den binære haugen (*binary heap*) [2, 4, 3, 4, 7, 6, 4, 5, 6, 7, 9, _]. Hvordan ser H ut etter at du setter inn verdien 1? (Den siste posisjonen er i utgangspunktet ledig.)

H = [1, 4, 2, 4, 7, 3, 4, 5, 6, 7, 9, 6]

(3 poeng)

b) Vis hvordan en tabell (*array*) A = [5, 9, 7, 4, 0, 2, 8, 8] ser ut etter hver SWAP-operasjon (ombytting av to tall) fram til fullført sortering ved Quicksort. Den siste verdien (den lengst til høyre) i tabellen (eller deltabellen under betraktning) skal alltid velges som pivot-element. Anta at PARTITION er implementert som oppgitt under. Fyll ut tabellen til høyre. (4 poeng)

```
PARTITION(A, p, r)
x ← A[r]
i ← p – 1
for j ← p to r – 1
        if A[j] ≤ x
                i ← i + 1
                SWAP(A[i], A[j])
SWAP(A[i + 1], A[r])
return i + 1
```

5	9	7	4	0	2	8	8
5	7	9	4	0	2	8	8
5	7	4	9	0	2	8	8
5	7	4	0	9	2	8	8
5	7	4	0	2	9	8	8
5	7	4	0	2	8	9	8
5	7	4	0	2	8	8	9

Merknad til sensur: Denne oppgaven er uheldig formulert, ettersom det ikke er mulig å fullføre Quicksort slik som beskrevet. Om studentene har gjort noe som viser at de har skjønt Quicksort så gis det full uttelling (4 poeng). Løsningen til høyre er en begynnelse på en sortering, der hver SWAP er vist. (Om studenten har skrevet første rekkefølge to ganger er det også riktig.)

Stipendiat Smartens hevder å ha klekket ut en sorterings-algoritme som kan ta inn en gyldig binærhaug (*binary heap*) med n verdier og sortere disse på $O(n)$ tid. Smartens hevder at han unngår den fundamentale $\Omega(n \log n)$-tid laveste grense for sammenligningsbasert sortering fordi elementene i den binære haugen allerede er delvis sortert.

c) Gi en så overbevisende begrunnelse som mulig for at Smartens dessverre tar feil.

Her kan man ha flere mulige argumenter; for eksempel:

— Hvis Smartens bruker $O(n)$ tid på en heap kan han bare bruke HEAPIFY først på $O(n)$ tid, og har da brutt $\Omega(n \log n)$-grensen.

— Anta at halvparten av verdiene er like (og minimale). Den andre halvparten vil da være totalt usorterte i en haug (og man har dermed et nytt sorteringsproblem av størrelse $n/2 \in O(n)$).

(10 poeng)

Oppgave 3 (15%)

Vi har gitt 3 vektorer $A[1\ldots n]$, $B[1\ldots n]$, $C[1\ldots n]$, alle med verdier som er fødselsdatoer på formen "`ddmmåå`", f.eks.: $A[1342]$ = `180368`.

Vi har som mål å finne ut hvilke datoer som er med i både A, B og C.

Du skal lage en enkel og særdeles effektiv algoritme for å framskaffe en vektor D som inneholder disse felles fødselsdatoene. Tenk på konstantleddene i metoden også.

a) Beskriv en slik algoritme SNITTABC. Uttrykk algoritmens tidskompleksitet ved Θ-notasjonen.

Flere mulige løsninger:

— Bruk en bitvektor med en bit per mulig dato. Fyll én tom vektor X ved å iterere over A. Fyll en ny vektor Y ved å iterere over B og slå opp X. Svaret finnes så ved å iterere over C og slå opp i Y.

— En variant av første løsning kan være å bruke hash-tabeller.

— En tredje mulighet kan være å bruke radiks-sortering og fletting.

Løsninger som ikke oppnår $\Theta(n)$ vil bli gitt færre poeng. (For eksempel vil en $\Theta(n \log n)$-løsning kunne gi inntil 8 poeng.)

Kjøretid: $\Theta(n)$

(15 poeng)

Oppgave 4 (15%)

Vi har gitt et olje-transportnettverk N med 6 noder: A (kilde), B, C, D (sluk), E, F, der oljerørenes kapasitet er gitt ved

AB:3, AC:3, BC:2, BE:3, CF:2, EF:4, ED:2, FD:3.

a) Du skal beregne maksimal flyt fra *A* til *D* ved den generelle Ford-Fulkerson-metoden. For at svaret ditt skal bli entydig i de tilfeller du har flere stivalg, skal du besøke noder i alfabetisk stigende rekkefølge, dvs. *B* før *C*, *E* før *F*, etc. Fyll ut, i tabellen nedenfor, de strømforøkende stiene (*augmenting paths*) i den rekkefølge de blir oppdaget, samt den netto flytøkning stien representerer. Nedenfor er første sti fylt inn. Vær nøye med å følge de beskrevne reglene. (Merk at du ikke nødvendigvis trenger å bruke alle radene i tabellen.)

(8 poeng)

Strømforøkende sti (*augmenting path*)	Netto flytøkningsbidrag
A–B–C–F–D	2
A–B–E–D	1
A–C–B–E–D	1
A–C–B–E–F–D	1

b) Hvilke minimale snitt finnes i nettverket over? (Oppgi hvert snitt som en kantliste.)

[ED, FD], [BE, CF], [AB, CF]

Her er det viktig å få med alle snittene. Har man med 2 snitt får man 5 poeng; har man med 1 snitt får man 3 poeng. (Man får samme uttelling enten bakoverkanter er tatt med eller ikke.)

(7 poeng)

Oppgave 5 (25%)

På en nyttårsfest er det invitert N gutter og N jenter, og arrangøren vil oppnå at så få som mulig får en "ikke spesielt ønsket" bordkavaler/dame til bords. Det er av den grunn blitt gjennomført en spørreundersøkelse blant deltakerne, og data fra denne er samlet i to matriser G og P.

Her er $G(i, j) = 0$ dersom gutt nr. i ikke spesielt ønsker jente nr. j, $j = 1,2, \ldots, N$; ellers er $G(i, j) = 1$.

Tilsvarende er $P(j, i) = 0$ dersom jente nr. j ikke spesielt ønsker gutt nr. i, $i = 1, 2, \ldots, N$; ellers er $P(j, i) = 1$.

Ønskene er ikke nødvendigvis gjensidige; eksempelvis kan $G(5, 19) = 1$ mens $P(19, 5) = 0$.

Vi definerer nå Problem PAR: Finn en par-sammensetting som er slik at så få personer som mulig ender opp med å få en ikke spesielt ønsket person til bords.

a) Formuler problemet PAR som et Lineær-Programmerings-problem (LP-problem). Du må klart definere de variable, målfunksjonen og kravene.

Formulerer dette som et heltallig lineært program:

maksimer $\sum_{i,j=1}^{N} (G(i,j) + P(j,i)) \cdot f(i,j)$

slik at

$f(i,j) \in \{0, 1\}$ for alle $1 \le i, j \le N$,

$$\sum_{j=1}^{N} f(i,j) = 1 \qquad \text{for alle } 1 \le i \le N$$

$$\sum_{i=1}^{N} f(i,j) = 1 \qquad \text{for alle } 1 \le j \le N$$

Her er $f(i,j) = 0$ hvis i og j ikke sitter sammen, og 1 ellers. Alle mulige par der $G(i,j) + P(j,i) = 0$ kan fjernes fra beregningen, ettersom de ikke bidrar til målfunksjonen.

Merknad til sensur: Selv om det ikke er akkurat det oppgaven spør etter er det ikke helt urimelig å anta at en match er like dårlig enten den inneholder 1 eller 2 uønskede personer (altså at vi prøver å redusere antallet "mismatcher"). En slik tolkning av oppgaven kan gi opp til 10 poeng.

(15 poeng)

b) Ofte er det slik at spesielle LP-problemer kan løses på en spesiell måte. Hvordan vil du så effektivt som mulig løse PAR-problemet i praksis? (Husk at N kan være vilkårlig stor.)

PAR kan formuleres (og løses) som et sirkulasjonsproblem. Legg kanter fra (f.eks.) gutt i til jente j hvis ikke begge misliker hverandre. Sett kostnaden til $-(G(i,j) + P(j,i))$, øvre kapasitet til 1 og nedre kapasitet til 0. Legg til kilde med én kant til hver av guttene og et sluk med én kant fra hver av jentene (øvre/nedre kapasitet 1/0 og kostnad 0). Legg til en bakoverkant (med kapasitet $\ge n$ og vilkårlig kostnad) fra sluk til kilde. En optimal sirkulasjon vil nå også gi en billigst (dvs. best) mulig matching mellom gutter og jenter (de som gjenstår matches vilkårlig).

Problemet kan også løses med mer spesialiserte algoritmer for *min-kost-flyt*-problemet (ikke pensum), på en måte som ligner løsningen for bipartitt matching ved hjelp av en maks-flyt-algoritme (Cormen 26.3).

Merk: Vi antar her en sirkulasjonsalgoritme som vil gi heltallig sirkulasjon hvis vi har heltallige kapasiteter. Detaljer rundt dette kreves ikke.

Merknad til sensur: Denne oppgaven viste seg å være vanskelig for de aller fleste.Ved sensur ble det derfor besluttet å gi hver student den mest lønnsomme vurdering av følgende 2 alternativer:

Alternativ 1. Beholde opprinnelig vekt (10 poeng) — mest lønnsomt for studenter som har fått mye til på oppgave 5b i forhold til resten av eksamen.

Alternativ 2. Se bort fra oppgave 5b og skalere opp vekten på de øvrige oppgavene så de summerer til 100 poeng.

(10 poeng)

Oppgave 6 (10%)

Anta at GLAGNAR er en type datastruktur (en klasse) med 4 attributter/felter, *buncha*, *muncha*, *cruncha* og *human*. Anta at disse feltene er udefinerte idet objektet opprettes. Anta at *zoid* er en minimums-heap av GLAGNAR-instanser, der GLAGNAR-instansene sammenlignes basert på sine *muncha*-attributter. Du er usikker på hva *buncha*-feltene er satt til for instansene i *zoid*, men du vet at *muncha*-feltene er satt til ulike tall. *cruncha* og *human*-attributtene for disse instansene er udefinerte. Betrakt følgende pseudokode:

```
while zoid.length > 1
        pop a fra zoid
        pop b fra zoid
        rinds ← en ny GLAGNAR-instans
        rinds.muncha ← a.muncha + b.muncha
        rinds.cruncha ← a
        rinds.human ← b
        sett (dvs. push) rinds inn i zoid
```

a) Hva gjør koden?

Konstruerer Huffman-trær. (*muncha* er frekvens, *cruncha* og *human* er høyre og venstre barn, og *buncha* er objektene/tegnene som frekvensene gjelder for).

(10 poeng)