Avsluttende eksamen i TDT4125 Algoritmekonstruksjon, videregående kurs

Eksamensdato	3. juni 2010
Eksamenstid	0900–1300
Sensurdato	24. juni
Språk/målform	Bokmål
Kontakt under eksamen	Magnus Lie Hetland (tlf. 91851949)
Tillatte hjelpemidler	Alle trykte/håndskrevne; bestemt, enkel kalkulator

Vennligst les hele oppgavesettet før du begynner, disponer tiden og forbered evt. spørsmål til faglærer kommer til eksamenslokalet. Gjør antagelser der det er nødvendig. Skriv kort og konsist. Lange forklaringer og utledninger som ikke direkte besvarer oppgaven tillegges liten eller ingen vekt. Alle deloppgaver teller like mye.

Oppgave 1

a. Gi et eksempel på en rettet graf der alle noder kan nås fra en start-node r, men der en variant av Prims algoritme som tar hensyn til kantretninger, utført fra r, ikke gir den billigste arboresensen (*min-cost arborescense*).

Enkelt eksempel med to sykler som går hver sin vei (innom de samme nodene, f.eks. med antiparallelle kanter.) Den ene starter dyrt men fortsetter billig, mens den andre starter billig og slutter dyrt.

Merk: Her var det ment "den billigste arboresensen fra r," men siden det ikke står eksplisitt godtas også svar som ikke gjør denne antakelsen. Løsninger som baserer seg på ikke-determinisme (dvs der Prims algoritme *kan* gi optimalt svar, men ikke *nødvendigvis* gjør det) gir noe, men ikke full, uttelling.

b. Vis at hvis *vertex cover* er i co-NP så er NP = co-NP.

Vertex cover (VC) er i NPC. Hvis VC er i co-NP så er co-VC i NP. Siden alle problemer i NP kan reduseres til VC (gjelder både ja- og nei-svar) kan man avgjøre komplementet til alle problemer i NP i ikkedeterministisk polynomisk tid, det vil si NP $\subseteq$ co-NP.

Av dette følger at settet av komplementer av problemene i NP en delmengde av settet av komplementer av problemene i co-NP – det vil si, co-NP $\subseteq$ NP.

Siden de er delmengder av hverandre må de være like.

Oppgave 2

a. Beskriv kort den generelle virkemåten til genetisk programmering.

Stikkord: Tilfeldig populasjon, evaluering, selection basert på fitness, crossover, gjenta til fornøyd.

b. Beskriv kort den viktigste forskjellen mellom genetisk programmering og andre evolusjonære algoritmer.

I GP er individene kjørbare programmer, det er de typisk ikke ved andre evolusjonære algoritmer.

c. Beskriv kort hvordan man gjør genetisk programmering der søk utgjør en del av fitness-beregningen, og hvordan det skiller seg fra vanlig genetisk programmering.

Poenget her er at ved genetisk programmering ved søk så er individene (dvs. "programmene") søkeuttrykk, altså queries i et eller annet query-språk.

I lengste vei-problemet skal du finne den lengste veien i en graf G uten å besøke samme node mer enn én gang.

d. Beskriv en heuristisk søkealgoritme for å finne lengste vei i en graf med sykler.

Det viktigste å få med her er løsningsrepresentasjon S, kostnadsfunksjon $c(S)$ og naboskapsrelasjon $S{\sim}S'$. En mulig løsning er følgende:

S: S er en permutasjon av et subsett av nodene i G.

$c(S)$: $c(S)$ er $\sum E$ – (lengden av veien definert av S), hvor $\sum E$ er summen av alle kantene.

$S{\sim}S'$: $S{\sim}S'$ hvis $S = S'$ + en ubesøkt nabonode til S' sin startnode eller sluttnode, eller $S = S' - S'$ sin startnode eller sluttnode.

En enda bedre naborelasjon vil også kunne sette inn og ta ut noder i midten av nåværende løsning (om mulig).

Oppgave 3

a. Forklar begrepene *recall* og *precision* i søk, og hva de brukes til.

Recall er andelen av relevante objeker som returneres, mens *precision* er andelen av objektene som returneres som er relevante. Begrepene brukes til å vurdere kvaliteten til et søkesystem.

b. Lag en invertert fil-indeks for de følgende dokumentene:

doc1: "this is the first one"

doc2: "this is not"

doc3: "neither is this"

first:	doc1
is:	doc1, doc2, doc3
neither:	doc3
not:	doc2
one:	doc1
the:	doc1
this:	doc1, doc2, doc3

Her det selvfølgelig også riktig å inkludere ord-posisjon.

c. Forklar hvordan en søkemotor kan lages slik at den skalerer lineært både med hensyn på datavolum og spørsmålsvolum.

Standard partisjonering og replisering av data (data fordeles på et lineært antall maskiner, og hver av disse dupliseres et lineært antall ganger). Fordeling av queries og innsamling av resultater kan f.eks. struktureres som et tre.

Oppgave 4

For en graf $G = (V, E)$ er en *matching* et subsett av E der ingen av kantene deler noder. En *maksimal* matching er en matching som ikke kan utvides ved å legge til flere kanter fra E.

a. Forklart kort hvordan nodene knyttet til en maksimal matching gir en approksimering med $\rho = 2$ for *vertex cover*-problemet.

Omtrent identisk med kjernen av approksimerings-algoritmen i pensum.

Alle kantene er dekket av minst én node i matchingen. Hvis det fantes en kant som ikke var dekket, så kunne den ha blitt lagt til, men matchingen er antatt maksimal. Alle kantene i matchingen må dekkes, så antallet kanter m er en nedre grense for optimal løsning. Antallet noder i approksimeringen er $2m$, som gir $\rho = 2$.

Anta at du har oppgitt en rettet graf $G = (V, E)$. La problemet A være å finne en størst mulig (altså med maksimal kardinalitet) delmengde av E som er slik at den resulterende delgrafen er asyklisk.

b. Beskriv en approksimeringsalgoritme for A. Oppgi kjøretid (i θ-notasjon) og approksimerings-ratio, ρ. Begrunn svaret.

Ordne nodene i en tilfeldig rekkefølge (f.eks. ved å gi dem en tilfeldig nummerering). Du har da ett sett med kanter som peker *fremover* og ett som peker *bakover*, og begge disse er asykliske. Velg det største, og du har minst halvpartene av kantene med. Den optimale løsningen kan ikke være mer enn dobbelt så stor, så $\rho = 2$.

Oppgave 5

PhD-student Lurvik skal designe en cache for objekter av forskjellig størrelse, og ønsker å bruke et Least Recently Used-skjema – en Størrelses-avhengig LRU (SaLRU). Kravene til cache-strukturen er å hurtig kunne slå opp i cachen, og hurtig kunne bytte ut innholdet i cachen med nye inkommende objekter.

På et tidspunkt i cachens levetid, har cachen C et sett objekter $i \in C$. Hvert objekt har en størelse S_i, og tiden siden forrige gang objektet ble brukt, T_i. Cachen har en kapasitet K (maksimal sum av objektstørrelser). Lurvik tenker at verdien for et cache-item er omvendt proposjonalt med alderen, $1/T_i$. Når det kommer inn et objekt med størelse R, må cachen kaste ut mange nok objekter til å gi rom for dette. Samtidig ønskes det at verdien til objektene som kastes ut er så liten som mulig.

Vi innfører en beslutningsvariabel $q_i \in \{0,1\}$, slik at om q_i=1, ønsker vi å kaste objektet ut av cachen, og om q_i=0 ønsker vi å beholde det. Det vi ønsker er da å optimalisere etter følgende kriterier:

Minimer $\sum_{i \in C} q_i / T_i$

Slik at $\sum_{i \in C} q_i S_i \geq R$

Vi ser bort fra evt. problemer med cachefragmentering.

a. Beskriv en algoritme som utfører optimaliseringen.

Hvis man maksimerer verdien på de gjenværende objektene, med en øvre vektgrense gitt av gjenværende plass, så er dette ekvivalent med 0-1-knapsack. (Verdi = $1/T_i$, vekt = S_i, vektkapasitet = $K - R$.)

b. Er Lurvik på rett spor? Forklar hvorfor dette ikke er et godt valg.

Formuleringen er ekvivalent med et NP-komplett problem, og lar seg vanskelig implementere effektivt.

Et annet poeng (som ikke kreves for full score) er at man ikke nødvendigvis må rydde plass tilsvarende R, dersom det allerede er noe ledig plass i cachen.

c. Argumentene dine for hvorfor dette ikke er lurt synker sakte inn, og Lurvik foreslår et alternativ: Fjern de objektene som har lavest verdi/kost, $1/(S_i\ T_i)$, inntil det er nok plass i cachen. Beskriv en effektiv algoritme for denne utkastingsstrategien.

Oppgaven var opprinnelig tenkt stilt annerledes, i en form der det ville være mulig å la cachen være en prioritetskø (f.eks. en heap). I oppgavens nåværende form vil det ikke være realistisk. Derfor vil oppgaver som involverer å sortere hele cachen hver gang aksepteres.

Full score gis ved lineær kjøretid. Det kan for eksempel oppnås (1) ved bruke Radix-sortering på $S_i\ T_i$, med passende antagelser, eller (2) å bruke en variant av Select (eller Randomized-Select) lineær (eller forventet lineær) kjøretid (her må man summere den venstre halvdelen etter bruk av Partition, og før hvert rekursive kall).

Oppgave 6

Du har en streng med n karakterer, $s[1..n]$, der hver "karakter" er ekstrahert ved å analysere tale og bryte lyden ned til en sekvens med atomiske og språkspesifikke lydprimitiver, såkalte *fonemer*. I tillegg har du en ordliste-operator tilgjengelig, som finner det mest sannsynlige ordet knyttet til en sekvens med fonemer. Operatoren kan både returnere det matchende ordet og den feilen man får ved å tolke fonemsekvensen som det foreslåtte ordet.

WORD(w): Returnerer en streng med det engelske ordet som er den mest sannsynlige tolkningen av fonemsekvensen $w = s[i, i+1, \ldots, j]$

WORD_ERROR(w): Returnerer et ikke-negativt tall som representerer feilen eller kostnaden ved å tolke fonemsekvensen $w = s[i, i+1, \ldots, j]$ som WORD(w).

Anta at både WORD(w) og WORD_ERROR(w) kan beregnes i konstant tid ($O(1)$).

a. Beskriv med pseudokode en algoritme som finner den mest sannsynlige (dvs, den billigste) sekvensen av engelske ord som kan ekstraheres fra fonemsekvensen $s[1, \ldots, n]$. Matematisk betyr dette at du må dekomponere sekvensen av n fonem-karakterer i en sekvens med k ord, [WORD(w_1), … , WORD(w_k)] (der k i utgangspunktet er ukjent), som er representert av k etterfølgende intervaller i fonemstrengen $s[\ldots]$ sånn at den totale feilen, gitt ved

$$\text{WORD\_ERROR}(w_1) + \ldots + \text{WORD\_ERROR}(w_k)$$

er minimert. Hva blir kjøretiden til algoritmen din (i θ-notasjon)?

Mulig løsning: For hver sluttposisjon i s "spiser" man seg bakover i sekvensen, og finner det ordet som best matcher de i siste fonemene. Tidligere delproblem er allerede løst og lagret, så kostnaden for prefikset før disse i fonemene kan man slå opp i konstant tid. Man lagrer så optimal verdi for denne sluttposisjonen og går videre. Kjøretiden blir $\theta(n^2)$.

b. Algoritmen fra **a** kan produsere ord som er lovlige individuelt, men som gir setninger som ikke er optimale, eller som ikke engang gir mening for en menneskelig leser. Diskuter kort hvordan algoritmen fra **a** kan utvides med heuristikker for å produsere bedre setninger (eller ordsekvenser).

Her er det mange muligheter. Man kan for eksempel ha en matrise som gir en kostnad for to etterfølgende ord, og ha ett delproblem for hver kombinasjon av sluttposisjon og ord, og ta hensyn til denne kostnaden når man velger mest sannsynlige ord på en gitt posisjon.