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*).

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

Oppgave 2

a. Beskriv kort den generelle virkemåten til genetisk programmering.

b. Beskriv kort den viktigste forskjellen mellom genetisk programmering og 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.

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.

Oppgave 3

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

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

doc1: "this is the first one"

doc2: "this is not"

doc3: "neither is this"

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

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.

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.

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 størrelse 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 størrelse 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.

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

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.

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)?

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).