Avsluttende eksamen i TDT4125 Algoritmekonstruksjon, videregående kurs

Eksamensdato	15. mai 2009
Eksamenstid	0900–1300
Sensurdato	5. 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

Du skal lage en visualiseringsløsning med tilsvarende funksjonalitet som Google Earth for bruk i en flysimulator. Til enhver tid observeres et utsnitt av jordkloden, og du ønsker å lagre terrengdataene du trenger i en forhåndsallokert cache-struktur. Denne strukturen har en fastsatt minnekapasitet C og skal ha metodene SET og GET, hvor SET plasserer data i cachen og GET henter disse ut igjen. Dersom dataene ikke finnes i cachen, returnerer GET en null-verdi. Anta at cache-blokkene har samme størrelse.

For denne cache-strukturen kan du velge mellom en utkastelsesstrategi basert på det elementet det er lengst siden du aksesserte (*least recently used*) eller en basert på det laveste antallet forespørsler (*least frequently used*).

a. Velg en av disse strategiene, og forklar kort hvorfor den sannsynligvis vil fungere best for dette problemet.

b. Ta utgangspunkt i cache-strategien du valgte i oppgave **a**. Angi hvilke datastrukturer du trenger, og skisser implementasjonen for funksjonen GET.

Scan resistance kan defineres som motstandsdyktighet mot en serie av forespørsler til cachen etter datanøkler man bruker bare én gang, og med flere forespørsler enn cachens kapasitet C.

c. Beskriv en utkastelsesstrategi som er *scan resistant*, med samme forventede kjøretid som i oppgave **a**.

Oppgave 2

Du har blitt kontaktet av en biolog som studerer nettverk av proteiner. Biologen er spesielt interessert i å finne store grupper av proteiner hvor alle proteinene interagerer med hverandre. «Aha! Klikker,» tenker du, og ber raskt biologen om å sende deg en beskrivelse av nettverket som du oversetter til en graf G med proteiner som noder V og interaksjoner som kanter E. Siden du nettopp har fått gode resultater ved å bruke simulert størkning (*simulated annealing*) til å løse TSP, ønsker du å gjenbruke programmet ditt for simulert størkning for å finne

den maksimale klikken i G. Det eneste du trenger å gjøre er å definere input, i form av løsningsrepresentasjon, kostnadsfunksjon og naboskapsrelasjon.

a. Skisser en egnet løsningsrepresentasjon S, gi et eksempel på en egnet starttilstand S_0, og definer en kostnadsfunksjon $c(S)$ og naboskapsrelasjon $S \sim S'$.

Etter å ha presentert din løsning for biologen, viser det seg at han er litt misfornøyd med resultatet. Det han egentlig var på jakt etter var store grupper med proteiner der de aller fleste, men ikke nødvendigvis alle, interagerer med hverandre.

b. Definer en ny kostnadsfunksjon $c'(S)$ som kan løse dette problemet. Kan du gjenbruke den eksisterende naboskapsrelasjonen (fra **a**)? Hvis ikke, definer en ny naboskapsrelasjon.

Oppgave 3

Betrakt følgende sorterte sekvens med 16 tall:

29867, 29869, 29880, 30119, 30265, 30473, 30476, 30477,
30478, 30481, 30503, 30514, 30672, 30682, 31033, 31034

Anta at hvert tall i utgangspunktet representeres med 32 bits.

a. Komprimer sekvensen med PFOR-DELTA-koding. Anta en blokkstørrelse på 16 tall og bruk to elementer i unntakslista. Hvor mange bits trenger hvert element i *code section*? Hvor mange bytes er det totalt behov for, inkludert unntakslista? Ikke regn med noen *header*. Vis de viktigste trinnene i utregningen.

b. Hvorfor vil PFOR-DELTA være mye raskere enn vByte?

Oppgave 4

Du skal finne en approksimeringsalgoritme for en variant av SET-COVER-problemet der man har lagt til ett krav til probleminstansene: Hvert element i mengden X forekommer i maksimalt k av mengdene i $\mathcal{F}$, for en gitt k.

a. Hvordan vil du løse problemet for $k = 2$? Hva blir kjøretid og *approximation ratio* (ρ) for algoritmen? Begrunn svaret.

b. Hvordan vil du løse problemet generelt? Hva blir kjøretid og *approximation ratio* (ρ) for algoritmen?

Oppgave 5

Anta at du skal finne maksimum av n elementer med en enkel, trådbasert splitt-og-hersk-algoritme (med en todeling av datasettet, og to rekursive kall som kan kjøres i parallell).

a. I hvilken grad vil denne algoritmen kunne dra nytte av en økning i antall prosessorer? Begrunn svaret.

b. Vis hvordan du kan finne maksimum av av n elementer i $O(1)$ parallell tid med n^{1+c} prosessorer, der $0 < c < 1$. Du kan selv velge verdien til c (oppgi denne i svaret). Hvilken PRAM-modell trenger du?

c. Skisser hvordan du kan generalisere løsningen så den vil fungere for alle verdier av c (der $0 < c < 1$). Detaljerte utregninger/bevis kreves ikke.

Oppgave 6

Du skal lage en algoritme for å tilnærme støyfylte tidsserier med glatte, konvekse funksjoner. En tidsserie er gitt ved:

$$X = x_1, x_2, \ldots, x_n,$$

der x_i er heltall i verdiområdet 0 til M ($0 \leq x_i \leq M$).

Tilsvarende kan den glatte, konvekse funksjonen som din algoritme skal beregne ut fra X beskrives som:

$$Y = y_1, y_2, \ldots, y_n,$$

der y_i er heltall i verdiområdet 0 til M ($0 \leq y_i \leq M$).

Du skal beskrive en algoritme som finner den konvekse funksjonen Y som er en best mulig tilnærming til X. Tilpasningen mellom Y og X måles med total kvadratfeil (som altså skal minimeres):

(A) $E = \sum_{i=1\ldots n} (y_i - x_i)^2$

Kravet om at Y skal være konveks (dvs «krummer oppover») betyr at den skal ha ikke-negativ andre derivert. Numerisk gir dette følgende krav:

(B) $2 \cdot y_i \leq y_{i-1} + y_{i+1}$ for $i=2, \ldots, n-1$,

det vil si, hvert element er mindre eller likt gjennomsnittet av sine to naboverdier.

a. Beskriv en algoritme som så effektivt som mulig ut fra X finner den tidsserien Y som både tilfredsstiller krav B ovenfor og har minst mulig kvadratfeil gitt ved A. Angi også hvilken kjøretid og hvilket plassbehov algoritmen har som funksjon av n og M.

Merk: Vi antar at M har en «rimelig» verdi, så en pseudopolynomisk algoritme (som altså er polynomisk som funksjon av n og M i stedet for n og $\log_2 M$) vil være en fullgod løsning.