Avsluttende eksamen i TDT4120 Algoritmer og datastrukturer

Eksamensdato	7. august 2009
Eksamenstid	0900–1300
Sensurdato	28. august
Språk/målform	Bokmål
Kontakt under eksamen	Magnus Lie Hetland (tlf. 91851949)
Tillatte hjelpemidler	Ingen 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 på angitt sted. Lange forklaringer og utledninger som ikke direkte besvarer oppgaven tillegges liten eller ingen vekt.

Algoritmer kan beskrives med tekst, pseudokode eller programkode, etter eget ønske, så lenge det klart fremgår hvordan den beskrevne algoritmen fungerer. Korte, abstrakte forklaringer kan være vel så gode som utførlig pseudokode, så lenge de er presise nok. Kjøretider oppgis med asymptotisk notasjon, så presist som mulig.

Oppgave 1 (30%)

a) Fyll inn følgende tabell. (Det holder med stikkord i problemkolonnen.)

Svar (20%):

Algoritme	**Problem + evt. ekstrakrav til problem**	**Kjøretid**
DIJKSTRA	Korteste vei, én til alle, pos. vekter	$O(E \lg V)$
HUFFMAN	Prefikskoder	$O(n \lg n)$
KRUSKAL	MST	$O(E \lg V)$
EDMONDS-KARP	Maksimal flyt	$O(VE^2)$
FLOYD-WARSHALL	Korteste vei, alle til alle (uten neg. sykler)	$O(V^3)$
COUNTING-SORT	Sortering, heltall i $[0 \ldots O(n)]$	$O(n)$

b) Løs følgende rekurrens. Oppgi svaret i asymptotisk notasjon. Begrunn svaret kort.

$$T(n) = 2T(n/3) + 1/n$$

Svar (10%): Kan løses med master-metoden, tilfelle 1 ($a = 2$, $b = 3$, $f(n) = n^{-1}$): $O(n^{.63})$

Oppgave 2 (24%)

a) Hvilken grunnleggende algoritme brukes når man skal finne sterke komponenter?

Svar (8%): DFS. (Her godtas også TOPOLOGICAL-SORT.)

b) Hva er fordelen med red-black-trees i forhold til vanlige binære søketrær?

Svar (8%): Red-black-trær er automatisk balanserte, som gir O(lg n) worst-case.

c) Hva er fordelen med B-trær i forhold til red-black-trees?

Svar (8%): B-trær har større aritet og dermed større noder, som passer til disk-I/O.

Oppgave 3 (23%)

Vi kjører BFS fra node s i den uvektede grafen G. Det er en kant e mellom nodene u og v i G, men denne kanten er ikke med i bredde-først-treet. La d være avstandstabellen fra BFS.

a) Argumenter kort for at differansen mellom $d[u]$ og $d[v]$ ikke kan være mer enn 1.

Svar (10%): I så fall ville e ha vært en snarvei som ga en kortere vei til u eller v.

Merk: Her har noen påpekt at situasjonen *kan* oppstå dersom grafen er rettet (noe som ikke var spesifisert i oppgaven), og e går «bakover». Det er korrekt, og har gitt full uttelling.

Det kan være rimelig å tenke at negative sykler kan gi «uendelig korte» stier. Men: korteste-vei-algoritmer leter etter *simple paths*, altså stier *uten sykler*.

b) Hvorfor har vi likevel problemer med å finne korteste vei når en graf inneholder negative sykler?

Svar (6%): Å finne korteste vei i det generelle tilfellet er et NP-komplett problem (ekvivalent med å finne lengste vei). Her kan det også gis en viss uttelling om man bare forklarer at problemet ikke har optimal substruktur.

Du har en sortert sekvens med n unike ID-nummer (positive heltall). Du vil finne det første (laveste) ledige ID-nummeret.

Eksempel: I sekvensen [1, 2, 3, 5, 7, 8] er 4 det første ledige nummeret.

c) Beskriv en effektiv algoritme for å finne det første ledige ID-nummeret. Hva blir kjøretiden? Begrunn svaret svært kort.

Svar (7%): Hvis $S[j] - S[i] > j - i$ inneholder intervallet $S[i \ldots j]$ et ledig tall. Bruk binærsøk med dette kriteriet, og velg det venstre av de to intervallene hvis begge inneholder ledige tall. Kjøretid: $O(\log n)$ fra binærsøk.

Oppgave 4 (23%)

Du har en streng med lengde n (og konstant alfabetstørrelse) lagret i en tabell med samme størrelse.

a) Beskriv en effektiv algoritme for å reversere strengen med konstant lagringsplass ut over strengen i seg selv (dvs *in-place*). Hva blir kjøretiden? Begrunn svaret kort.

Svar (10%): Bytt elementpar fra ytterst til innerst. Kjøretid: Enkel traversering gir $O(n)$.

Du har en liste L med n strenger (med konstant alfabetstørrelse) av varierende lengde (maks-lengde m; datasettet inneholder totalt M tegn) og ønsker å avgjøre hvorvidt listen inneholder minst ett strengpar A, B, der A er et anagram for B (det vil si, A består av de samme tegnene som B, men i en annen rekkefølge).

b) Beskriv en effektiv algoritme for å finne ut om L inneholder et slikt ordpar. Hva blir kjøretiden? Begrunn svaret kort.

Svar (7%): Sorter tegnene i hver streng (f.eks. med tellesortering) og se etter duplikater, for eksempel med en hashtabell. Kjøretid: Sortering og hash-innsetting gir $O(m) + O(1)$ for hver streng. Totalt $O(nm)$, eller, mer presist, $O(M)$. Eventuelt kan man bruke sortering og traversering for å finne duplikater. Ved bruk av en (litt optimalisert) RADIX-SORT kan man

likevel få til $O(M)$. Et alternativ kan være å lage en hashfunksjon som ignorerer rekkefølgen på tegnene i strengene. Kjøretiden blir uansett den samme, siden alle tegnene må behandles.

Et *addagram* er en serie med ord av økende lengde, der hvert ord kan gjøres til et anagram av det forrige ved å fjerne en bokstav. For eksempel:

```
to, sto, stor, store, sorten, stormen, matrosen, samordnet,
motstander, demonstrant, motstanderne, demonstranten,
demonstrantene, demonstrantenes
```

Du ønsker nå å finne lengden på den lengste sekvensen av slike ord som finnes i ordlisten L. (Merk at ordene godt kan forekomme i en annen rekkefølge i ordlisten enn i sekvensen.)

c) Beskriv en effektiv algoritme for å finne det lengste addagrammet i L. Hva blir kjøretiden? Begrunn svaret kort.

Svar (6%): Sorter tegnene i hver streng og bruk dem som nøkler i en hashtabell (duplikater kan droppes) med addagram-lengden som verdi. Bergn lengden «nedover» fra hver streng, rekursivt ved å prøve å fjerne hvert tegn. Lagre og gjenbruk lengdene underveis (memoisering). Vi får et rekursivt kall for hvert slettede tegn i hver streng, som gir en kjøretid på $O(nm)$ eller, mer presist, $O(M)$.