NTNU
Institutt for datateknikk
og informasjonsvitenskap

Eksamensoppgave i TDT4120 Algoritmer og datastrukturer

Faglig kontakt under eksamen	Magnus Lie Hetland
Tlf.	91851949
Eksamensdato	11. august 2014
Eksamenstid (fra–til)	0900–1300
Hjelpemiddelkode	D. Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt, enkel kalkulator tillatt.
Målform/språk	Bokmål
Antall sider	9
Antall sider vedlegg	0
Kontrollert av	Pål Sætrom

_____

Dato Sign

Les alle oppgavene før du begynner, disponer tiden og forbered spørsmål til faglærer ankommer lokalet. 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 (med mindre annet er oppgitt), 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. Algoritmer som konstrueres bør generelt være så effektive som mulig, med mindre annet er opplyst. Kjøretider oppgis med asymptotisk notasjon, så presist som mulig. Alle oppgavene teller like mye.

1. Skriv funksjonen

$$T(n) = 492n^3 + 3\left(\frac{n}{4}\right)^8 + \sqrt{n} \cdot \log_2 n$$

med asymptotisk notasjon.

Svar: $\Theta(n^8)$

2. Hva er kjøretiden til den følgende algoritmen, uttrykt som en funksjon av n?

```
x = 0
for i = 1 to n
    for j = i to n
        x = x + 1
    end
end
```

Svar: $\Theta(n^2)$

3. Hva er poenget med å bruke en tilfeldig pivot i randomisert Quicksort?

Svar: Unngå konsekvent worst-case ved bestemte inputs (ferdig sortert, f.eks.).

4. Hvilke antagelser gjør man om input når man bruker Floyd-Warshall?

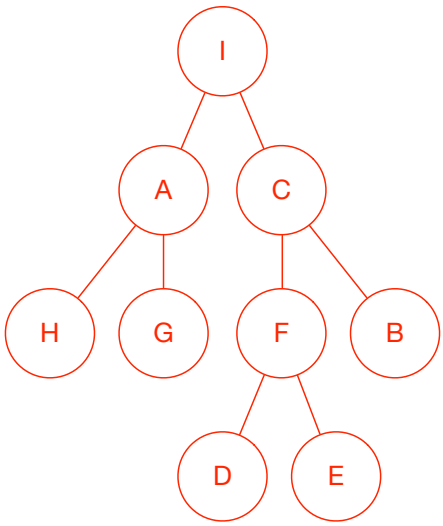
Svar: Ingen negative sykler.

5. Hva er relasjonen mellom Ford-Fulkerson og Edmonds-Karp?

Svar: Edmonds-Karp er Ford-Fulkerson der BFS brukes til traversering.

6. De ni nodene i et binært tre (ikke nødvendigvis et binært søketre) har fått navnene **A**, **B**, **C**, … , **I** i vilkårlig rekkefølge. En *preorder*-traversering (*preorder tree walk*) skriver ut nodene i følgende rekkefølge:
   I A H G C F D E B
   En *postorder*-traversering (*postorder tree walk*) skriver ut nodene i følgende rekkefølge:
   H G A D E F B C I
   Rekonstruer treet, og tegn det nedenfor (med venstre barnenoder tegnet til venstre).

Svar:



7. La **A** være mengden med maksimale asykliske urettede grafer og la **B** være mengden med minimale sammenhengende asykliske urettede grafer, for vilkårlige nodemengder, der «maksimal» og «minimal» refererer til antall kanter. Hva er forholdet mellom **A** og **B**? Forklar kort.

Svar: Både A og B er mengden av trær.

8. Løs rekurrensen $T(n) = 3T(n/3) + n$. Gi svaret i asymptotisk notasjon.

Svar: $\Theta(n \log n)$

9. Din venn Lurvik har funnet opp en algoritme. Algoritmen hans kan ta inn en sekvens av lengde n, der n er et multiplum av k, og sortere den i k segmenter av lik lengde, slik at elementene i et gitt segment ikke nødvendigvis er sortert seg imellom, men alle elementer i ethvert segment er større enn eller like store som elementene i alle segmenter til venstre, og mindre eller like elementene i alle segmenter til høyre. Gi en nedre grense for kjøretiden til algoritmen. Forklar kort.

Svar: Her kan flere svar godtas. Noen studenter har eksplisitt beskrevet best-case (lineær) og fått full score for det. Intensjonen var at man skulle gi en nedre grense for worst-case, altså, hvor bra algoritmen kunne være. For å få til dette reduserer man fra sortering, ved å vise at Lurviks algoritme kan brukes til å sortere med. (Det hjelper ikke å diskutere spesifikke sorteringsalgoritmer e.l.) Det holder f.eks. å vise at for enhver kombinasjon av n og k kan man bruke algoritmen til å sortere k elementer (ved å replisere dem), så man får en grense på $\Omega(n + k \log k)$.

10. En gruppe med n riddere har turnert hele uka, og har nå endelig alle hatt en match med hver av de andre, der hver match hadde en vinner, og du har blitt spurt om å lage en rankingliste for ridderne. Du har tro på at du skal kunne løse dette – du kunne jo bare bruke en sorteringsalgoritme, tenker du. Selv om noen matcher var uavgjorte kunne du bare ha brukt topologisk sortering. Null problem! Men når du ser på dataene oppdager du … at det er sykler der! For eksempel: Den Sorte Ridder vant over Den Røde Ridder, som igjen slo Den Grønne Ridder … som i sin tur overvant Den Sorte Ridder! Hva skal du gjøre? Du bestemmer deg for følgende løsning: Lag en rankingliste der ridderen i hver posisjon vant over ridderen i neste posisjon ned på lista. Etter å ha klødd deg litt i hodet kommer du frem til at dette alltid må være mulig. Du mistenker at bedre løsninger er mulige, men bestemmer deg for å prøve å få en kjøretid på $O(n^2)$. Beskriv en algoritme med denne kjøretiden som løser problemet. Forklar kort hvorfor den er korrekt.

Svar: Sett hver ridder inn på første lovlige plass, etter tur. Forklaring: Hvis vi har en lovlig ranking, så vil innsettingen også føre til en lovlig ranking.

11. Du har oppgitt et flytnettverk med en tilhørende flyt (dvs., en flytfunksjon – ikke bare flytverdien). Beskriv kort en algoritme for å avgjøre om flyten er maksimal. Hva blir kjøretiden?

Svar: Let etter en augmenting path (én iterasjon av Ford-Fulkerson). Om du finner en er flyten ikke optimal; ellers er den det. Kjøretid $\Theta(E + V)$ med DFS, BFS, e.l.

12. Du leker deg med Dijkstras algoritme. Heller enn å legge alle nodene inn i køen Q ved starten, så bruker du Q som en traverseringskø, litt som i BFS, og legger inn nodene etter hvert som de oppdages, *hvis* de potensielt kan gi deg snarveier. (Du trenger jo tross alt bare besøke en node hvis den kan gi deg snarveier til andre noder.) For å gjøre dette har du skrevet om RELAX slik at den returnerer **true** hvis den endrer avstanden og **false** ellers. Du dropper rett og slett mengden S, som i pseudokoden nedenfor:

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  Q = {s}
3  while Q ≠ ∅
4      u = EXTRACT-MIN(Q)
5      for each vertex v ∈ G.Adj[u]
6          if RELAX(u, v, w)
7              Q = Q ∪ {u}
```

(*Oppg. 12, forts.*)

Du er ganske sikker på at dette bør fungere, og så lenge *Q* er en haug (*heap*) bør du få samme kjøretid òg. Du vet at ved forrige eksamen så handlet én av oppgavene om å erstatte *Q* med en uordnet tabell (*array*) som ble brukt som prioritetskø. Du lurer på om kanskje din modifiserte versjon av Dijkstras algoritme ikke *trenger* en prioritetskø i det hele tatt! Du blir nysgjerrig, og prøver å erstatte *Q* med en enkel FIFO-kø. Hva skjer? Når (dvs., for hvilke instanser) vil denne algoritmen fungere som en én-til-alle (*single source*) korteste-vei-algoritme? Hva blir kjøretiden? Hvilket forhold har denne algoritmen til andre algoritmer i pensum? Diskutér kort.

Svar: Vi vil nå relaxe kantene gjentatte ganger, helt til vi ikke får noen forbedringer. Algoritmen vil være svært lik Bellman-Ford, og vil fungere også for negative kanter, så lenge vi ikke har negative sykler. Algoritmen kan i noen tilfeller unngå noen Relax-operasjoner som Bellman-Ford gjør, men i verste tilfelle er de ekvivalente. Denne algoritmen oppdager ikke negative sykler slik som Bellman-Ford.

13. I en urettet graf, gitt tre noder *u*, *v* og *w*, skal du avgjøre om det finnes en sti fra *u* til *w* som går via *v*. Beskriv kort en algoritme som løser problemet.

Svar: Bruk flyt med kapasitet 1, *v* som kilde og *u* og *w* koblet til et nytt sluk. Får vi flyt 2 finnes en slik sti.

14. Gitt en sammenhengende vektet urettet graf skal du finne et spenntre (ikke nødvendigvis et minimalt spenntre) som har en minimal tyngste kant. Det vil si, det finnes ingen andre spenntrær som har en høyest vekt som er lavere. Selv om du mistenker at det finnes mer effektive måter å løse problemet på, så bestemmer du deg for å løse det ved å rett og slett finne et minimalt spenntre. Beskriv kort hvorfor løsningen vil være korrekt.

Svar: Den tyngste kanten kobler sammen to nodesett. Hvis det gikk an å koble dem sammen med en lettere kant, ville vi også kunne få en lavere totalvekt.

(Her har vi også godtatt en del svar som beskriver atferden til spesifikke algoritmer, selv om det egentlig ikke besvarer spørsmålet generelt.)

15. Du har gitt to sekvenser **A** og **B** med respektive lengder k og n og vil telle hvor mange ganger **A** forekommer som en subsekvens av **B**. Med andre ord, hvis alle elementene er identiske skal svare ditt være antall måter man kan velge k elementer fra en mengde på n elementer, dvs., binomialkoeffisienten

$$\binom{n}{k}.$$

Beskriv kort en algoritme som løser problemet.

Svar: For hver forekomst av første element av A i B, løs problemet rekursivt med memoisering for resten av A og B.

(Evt. løs med iterativ DP og en tabell.)