NTNU
Kunnskap for en bedre verden

Institutt for datateknikk og informasjonsvitenskap

# Eksamensoppgave i TDT4120 Algoritmer og datastrukturer

| | |
|---|---|
| **Faglig kontakt under eksamen** | Magnus Lie Hetland |
| **Telefon** | 918 51 949 |
| **Eksamensdato** | 4. desember, 2017 |
| **Eksamenstid (fra–til)** | 09:00–13:00 |
| **Hjelpemiddelkode/tillatte hjelpemidler** | D |
| **Annen informasjon** | Oppgavearkene leveres inn, med svar i svarrute under hver oppgave |
| **Målform/språk** | Bokmål |
| **Antall sider (uten forside)** | 5 |
| **Antall sider vedlegg** | 0 |

**Informasjon om trykking av eksamensoppgave**

**Originalen er**

**1-sidig** ☑ **2-sidig** ☐

**sort/hvit** ☑ **i farger** ☐

**Skal ha flervalgskjema** ☐

**Kvalitetssikret av** Pål Sætrom

**Kontrollert av**

Dato Sign

Merk: Studenter finner sensur i Studentweb. Har du spørsmål om din sensur må du kontakte instituttet ditt. Eksamenskontoret vil ikke kunne svare på slike spørsmål.

## Les dette nøye

(*i*) Les hele eksamenssettet nøye *før* du begynner!

(*ii*) Faglærer går normalt én runde gjennom lokalet. Ha evt. spørsmål klare!

(*iii*) Skriv svarene dine i svarrutene og levér inn oppgavearket. Bruk gjerne blyant! Evt. kladd på eget ark først for å unngå overstrykninger, og for å få en egen kopi.

(*iv*) Ekstra ark kan legges ved om nødvendig, men det er meningen at svarene skal få plass i rutene på oppgavearkene. Lange svar teller ikke positivt.

**Merk:** Varianter av de foreslåtte svarene nedenfor vil naturligvis også kunne gis uttelling, i den grad de er helt eller delvis korrekte.

## Oppgaver med løsninger

---

(5%) 1. (a) Hvilken traverseringsalgoritme i pensum kan brukes til å finne korteste vei i uvektede grafer?

BFS

(5%) (b) Én nøkkelingrediens i dynamisk programmering er optimal substruktur. Hva er den andre?

Overlappende delproblemer.

(5%) (c) Hva er kjøretiden til RANDOMIZED-SELECT i verste tilfelle (*worst-case*)?

$\Theta(n^2)$

(5%) (d) RADIX-SORT sorterer etter ett og ett siffer. I hvilken rekkefølge?

Fra minst til mest signifikante.

(5%) (e) Hva har grafen hvis BELLMAN-FORD feiler (dvs., returnerer FALSE)?

Negative sykler.

(5%) (f) Hva representerer attributten $v.\pi$ i (for eksempel) PRIM og DIJKSTRA?

Forgjengeren (*predecessor*) eller forelderen (*parent*) til $v$.

(5%) (g) Hva er input til kretsen i NPC-beviset for CIRCUIT-SAT (dvs. $y$, der $C(y) = A(x, y)$)?

Et sertifikat.

---

2. I de følgende deloppgavene, sørg for å ikke «kaste bort» informasjon under forenklingen. Gi tette grenser og bruk den mest presise asymptotiske notasjonen du kan (av O, $\Omega$ eller $\Theta$).

(5%) (a) Forenkle uttrykket $\Theta(n) + O(n^2) + \Omega(n^3)$.

$\Omega(n^3)$

(5%) (b) Forenkle uttrykket $\Theta(n + \sqrt{n}) + O(n^2 + n) + \Omega(n \cdot (n + \lg n))$.

$\Omega(n^2)$

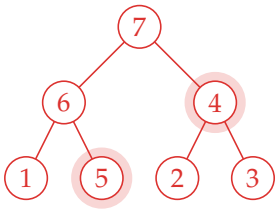
(5%) (c) Løs rekurrensen $T(n) = T(n/3) + \lg n$. Skriv svaret med $\Theta$-notasjon.

$\Theta(\lg^2 n)$

3. Dine venner Smartnes og Lurvik har laget hver sin algoritme som tar inn en binær maks-haug (*binary max-heap*) som input. Haugen er representert som en tabell $A[1..n]$ på vanlig måte. Alle elementene er forskjellige og haugen er *komplett*.[1]

(5%) (a) Lurvik sin algoritme baserer seg på antagelsen at de $\lceil n/2 \rceil$ minste elementene alltid vil ligge sist (dvs., i løvnodene). Stemmer det? Begrunn svaret kort.

Nei, det stemmer ikke. Her er f.eks. 5 en løvnode mens 4 ikke er det:




Dette tilsvarer $A = [7, 6, 4, \underline{1, 5, 2, 4}]$, der de siste $\lceil n/2 \rceil$ elementene er understreket.

(5%) (b) Smartnes mener hun har en algoritme som utnytter haug-strukturen og bygger et balansert binært søketre for elementene til A i lineær tid. Tror du hun kan ha rett? Begrunn svaret kort.

Nei. Hun kunne også bygge haugen og inorder-traversere treet i lineær tid i verste tilfelle, og dermed bryte sorteringsgrensen på $\Omega(n \lg n)$.

Følgende algoritme ville altså være en slik umulig sorteringsalgoritme:

SMARTNES-SORT(A)

1 BUILD-MAX-HEAP(A)
2 Lag treet T fra A med Smartnes sin algoritme
3 INORDER-TREE-WALK(T.*root*)

Dette gjelder selv om vi begrenser hvilke verdier $n$ kan ha; om vi får inn et annet antall elementer, kan vi alltids legge til noen svært store verdier som uansett havner sist; det vil i verste tilfelle doble størrelsen på A.

(5%) 4. Du skal simulere én iterasjon av TRANSITIVE-CLOSURE. I figur 1 ser du den rettede grafen som tilsvarer $T^{(3)}$. Du skal beregne $T^{(4)}$ og tegne den tilhørende rettede grafen.

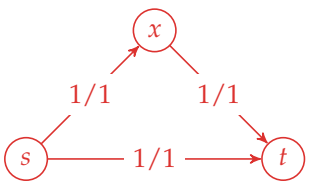
Fyll inn kantene som mangler i figur 2. Angi retning tydelig.

(Merk: Hver $T^{(k)}$ er en helt ny tabell. Dvs., vi gjenbruker ikke den samme i hver iterasjon.)

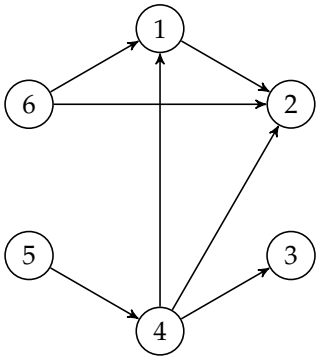
Det er ikke nødvendig med en grundig simulering for å løse denne oppgaven. I iterasjon 4 trenger man bare se på kantene inn til 4 og ut fra 4. Det er bare én kant inn fra 5, så 5 «arver» 4 sine ut-kanter.

(5%) 5. (a) Tegn et flytnettverk der kapasitetene er oddetall men der maksimal flytverdi er et partall.

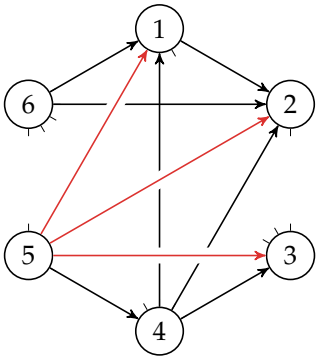
For eksempel:




[1] Alle interne noder har to barn og alle løvnoder er på samme nivå, dvs., $n = 2^k - 1$ for et heltall $k \geq 1$.




Figur 1: $T^{(3)}$ – brukt i oppgave 4




Figur 2: $T^{(4)}$ – svar på oppgave 4

(5%) (b) I et flytnettverk der kapasitetene er partall vil maksimal flytverdi være et partall. Hvorfor?

Maks-flyt = min-cut, og kapasiteten til et snitt er en sum av kapasiteter, og dermed et partall.

Her godtas flere forklaringer. F.eks.: $(i)$ Om vi deler kapasitetene på 2 får vi heltallskapasiteter. Maksimal flytverdi blir et heltall (jf. heltallsteoremet), og vi kan så gange med 2 igjen. $(ii)$ Flaskehalsen/oppdateringer i residualnettverket vil alltid være partall, og sum av partall er et partall.

(5%) (c) Et team med forskere skal gjennomføre et sett med prosjekter.

- Hvert prosjekt består av flere oppgaver, og et visst antall av prosjektets oppgaver (f.eks. 7 av 12) må gjøres. Ulike prosjekt kan ha ulike antall.
- Hver oppgave skal utføres av én forsker.
- Hver forsker har en viss kapasitet, dvs., et antall oppgaver hun rekker å gjøre.
- Hver forsker er kompetent til å gjøre noen av oppgavene, men ikke nødvendigvis alle.

Du skal avgjøre hvem som skal gjøre hvilke oppgaver, eller finne ut at det ikke går.
Beskriv hvordan du kan løse dette som et maks-flyt-problem. Tegn gjerne et flytnettverk.

For eksempel:

- Noder for forskere, oppgaver og prosjekter.
- Kanter fra kilde til forskere har kapasitet lik forskerens kapasitet.
- Kanter fra forskere til oppgaver de er kompetente til å gjøre har kapasitet 1.
- Kanter fra oppgaver til prosjekter har kapasitet 1.
- Kanter fra prosjekter til sluk har kapasitet lik antall oppgaver som må gjøres i prosjektet.

Dersom alle kantene til sluket er fulle når man har funnet maksimal flyt, så lar problemet seg løse, og fulle kanter fra forsker til oppgave angir hvem som skal gjøre hva.

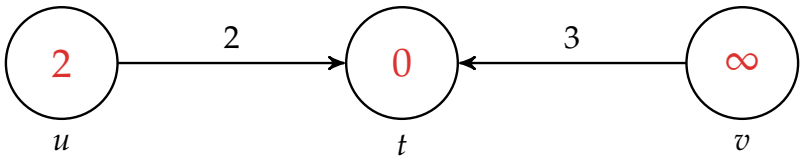
---

6. Din venn Klokland vil finne korteste vei *til* en node $t$ (*single-destination shortest-path*) i en rettet graf $G = (V, E)$ med vektfunksjon $w : E \to \mathbb{R}$. Han har endret på RELAX så den oppdaterer $u$ basert på $w(u, v)$ og $v.d$, i stedet for omvendt.

   Han bruker så denne nye RELAX′ i en modifisert versjon av DFS: Først initialiserer han grafen som vanlig, og når den rekursive traverseringen returnerer til $u$ etter å ha besøkt $v$ (og farget den svart), bruker han sin RELAX′ på kanten $(u, v)$. (Om ønskelig, se pseudokode på s. 5 for presisering.)

   Klokland påstår at algoritmen fungerer for rettede, asykliske grafer. Du er skeptisk, og vil lage et moteksempel. Resultatet er grafen som er vist nedenfor.

(3%) (a) Hva er resultatet av algoritmen om $u$ besøkes først? Fyll inn $u.d$, $t.d$ og $v.d$ nedenfor.




(7%) (b) Beskriv kort hvordan algoritmen kan endres så den blir korrekt for (vektede, rettede) asykliske grafer. (Her holder det med en svært kort forklaring.)

Kall RELAX også du når treffer svarte noder.
(Dvs., redusér indenteringen på linje 5 i DFS-VISIT′ med ett hakk; flytt RELAX′ ut av **if**-utsagnet.)
Synkende *finish-time* gir en topologisk sortering, og vi får slakket alle ut-kanter fra en node før den får tildelt *finish-time*, og vi vil kun slakke til svarte noder, som altså allerede har fått tildelt en *finish-time*. Dette tilsvarer en variant av DAG-SHORTEST-PATH der vi slakker inn-kanter i stedet for ut-kanter, altså grunnleggende dynamisk programmering.

---

7. Du får oppgitt et spillbrett som vist i figur 3 (på side 5), med $n \geq 3$ ruter, der du starter helt til venstre og skal flytte mot høyre. I rutene står det hvor langt du maksimalt kan flytte i ett trekk når du står på ruten. Målet er å komme frem til den siste ruten med så få trekk som mulig.

   (Du kan anta input er slik at at det alltid er mulig å komme frem til den siste ruten.)

(7%) (a) Beskriv en algoritme som effektivt finner ut hvor mange trekk du må bruke. (Du trenger *ikke* finne ut *hvilke* trekk.) Beskriv den gjerne med egne ord, eller bruk formler, figurer, kode eller pseudokode etter eget ønske.

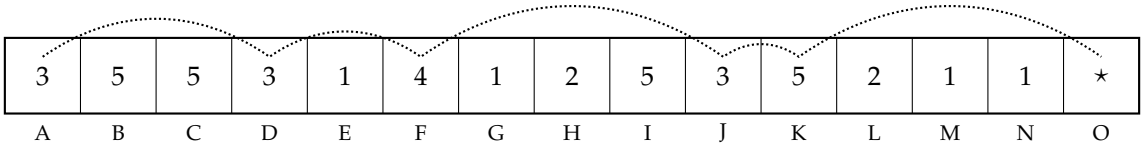
**Versjon 1**:
La $r[1, \ldots, n-1]$ være input og $t[i]$ antall trekk frem til rute $i$. Da har vi $t[1] = 0$ og, for $i > 1$,

$$t[i] = \min\{t[j] + 1 : j = 1 \ldots i-1,\ \ i - j \leq r[j]\}\,.$$

Svaret er $t[n]$. Løses med memoisering eller «bottom up», ved iterasjon ($i = 2 \ldots n$).
Dette er ekvivalent med å bruke DAG-SHORTEST-PATH med kantvekter 1.
**Versjon 2**:
Tolk tabellen som en graf, der lovlige hopp er kanter, og kjør BFS fra første rute. Det finner korteste vei til siste rute.
**Versjon 3**:
La antall trekk $t$ være 1, la $k = m = r[1]$ og gjenta det følgende for $i = 2$ til $n-1$:
La $k = k - 1$. La $m$ være maks av $m$ og $i + r[i]$. Hvis $k = 0$, la $k = m - i$ og $t = t + 1$.
Forklaring: $m$ er hvor langt vi kan komme med dette eller neste trekk, og $k$ er hvor mye vi har igjen av dette trekket. Hvis vi trekket «går tomt» ($k = 0$), så må vi skifte til neste trekk ($t = t + 1$). Vi tillater at dette skiftet *kan ha skjedd tidligere*, uten at vi vet hvor; vi vet uansett at vi har $m - i$ trinn igjen av trekket, dvs., $k = m - i$.
(Den samme ideen kan naturligvis implementeres på flere måter.)

(3%) (b) Hva blir kjøretiden i verste tilfelle? Oppgi svaret i $\Theta$-notasjon. Forklar kort.

**For versjon 1 og 2**:
Vi får $i - 1$ iterasjoner/kanter for $t[i]$, der $i = 2 \ldots n$, så $\mathrm{T}(n) = \Theta(1 + \cdots + (n-1)) = \Theta(n^2)$.
**For versjon 2**:
Vi går igjennom tabellen én gang, så $\mathrm{T}(n) = \Theta(n)$.




Figur 3: Eksempel til oppgave 7. I første trekk kan du flytte fra rute A til B, C eller D. Flytter du til D kan du så flytte til E, F eller G i ett trekk. (Merk: Eksempel-løsningen er ikke optimal. Dette er en feil. Eksemplet er vilkårlig, og parentesen ble lagt til så man ikke skulle legge vekt på det – men tilfeldigvis så *er* det optimalt.)

---

**Pseudokode til oppgave 6**

Det følgende er en mer detaljert beskrivelse av algoritmen som omtales i oppgave 6. INITIALIZE-SINGLE-SOURCE er som beskrevet i pensum. Pseudokoden er ment som et supplement, og det er mulig å besvare oppgaven uten den. Merk at det *ikke* er nødvendig å skrive pseudokode, eller referere til denne koden, i besvarelsen.

RELAX$'(u, v, w)$

```
1  if u.d > v.d + w(u, v)
2      u.d = v.d + w(u, v)
3      u.π = v
```

DFS$'$(G, $t$)

```
1  INITIALIZE-SINGLE-SOURCE(G, t)
2  for each vertex u ∈ G.V
3      u.color = WHITE
4  for each vertex u ∈ G.V
5      if u.color == WHITE
6          DFS-VISIT'(G, u)
```

DFS-VISIT$'$(G, $u$)

```
1  u.color = GRAY
2  for each v ∈ G.Adj[u]
3      if v.color == WHITE
4          DFS-VISIT'(G, v)
5          RELAX'(u, v, w)
6  u.color = BLACK
```

---