NTNU

KANDIDAT

10629

PRØVE

TDT4120 1 Algoritmer og datastrukturer

Emnekode	TDT4120
Vurderingsform	Skriftlig eksamen
Starttid	15.12.2025 14:00
Sluttid	15.12.2025 18:00
Sensurfrist	12.01.2026 23:59
PDF opprettet	16.12.2025 17:40

Oppgave	Oppgavetype
i	Informasjon eller ressurser
i	Informasjon eller ressurser
1	Langsvar
2	Langsvar
3	Langsvar
4	Langsvar
5	Langsvar
6	Langsvar
7	Langsvar
8	Langsvar
9	Langsvar
10	Langsvar
11	Langsvar
12	Langsvar
13	Langsvar
14	Langsvar
15	Langsvar
16	Langsvar
17	Langsvar
18	Langsvar
19	Langsvar

20 Langsvar

1 Hva er kjøretiden til prosedyren for å bygge en maks-haug, **Build-Max-Heap**?
Skriv svaret ditt her. Endringer blir lagret automatisk.

Kjøretiden er Theta(n)

Ord: 3

Maks poeng: 5

2 Din venn Lurvik har klart å vise at to av følgende beskrivelser av kjøretiden til en algoritme er korrekte, men han husker ikke hvilke to:
1. $T(n) = O(n^3)$
2. $T(n) = \Omega(n^3)$
3. $T(n) = \Theta(n^3)$

Sjefen hans har bedt ham velge ut én av beskrivelsene, og nå har Lurvik bedt deg om hjelp. Hvilken av de tre velger du (nummer 1, 2 eller 3)? Forklar kort hvorfor dette er et godt valg.
Skriv svaret ditt her. Endringer blir lagret automatisk.

Han bør velge nummer 3 T(n) = Theta(n^3) siden den gir mest informasjon.

Det er fordi alle uttrykkene uansett er riktig:

O(n^3) og Omega(n^3) <=> Theta(n^3)

O(n^3) og Theta(n^3) <=> Theta(n^3)

Omega(n^3) og Theta(n^3) <=> Theta(n^3)

Ord: 36

Maks poeng: 5

3 Hva er det som gjør at **Counting-Sort** har lavere asymptotisk kjøretid enn f.eks. **Merge-Sort**? Forklar kort.
Skriv svaret ditt her. Endringer blir lagret automatisk.

Det er fordi vi antar noe om inputen. Vi antar at alle verdier i tabellen er lavere eller lik k, som gir kjøretid Theta(n + k). Vi må også anta at k = o(nlogn) for at kjøretiden skal bli bedre enn Merge-Sort sin, som er Theta(nlogn)

Ord: 46

Maks poeng: 5

4 Lurvik og Smartnes har en heftig diskusjon. Smartnes mener man kan ha en algoritme med ulik asymptotisk kjøretid i beste og verste tilfelle, der den gjennomsnittlige kjøretiden likevel er lik én av de to. Lurvik mener dette er umulig, siden gjennomsnittet y av x og z alltid vil ligge imellom dem, altså x < y < z.

Bruk et eksempel fra pensum for å vise at Smartnes har rett. Forklar kort hvorfor det er mulig. (Smartnes mener altså at *average-case* kan være lik enten *best-case* eller *worst-case*, selv om *best-case* og *worst-case* er forskjellige.)

Skriv svaret ditt her. Endringer blir lagret automatisk.

Det er flere, en av dem er Quick-Sort (ikke randomisert eller medians of medians), som har kjøretid Theta(nlogn) i best tilfellet og gjennomsnittlig, men kjøretid Theta(n^2) i verste tilfellet. Grunnen til dette er at verste tilfellet er at listen er nesten sortert (eller nesten reversert sortert), som er ekstremt sjeldent. Det beste tilfellet er egentlig at listen er i en helt tilfeldig rekkefølge, som er det man vil forvente for tilfeldig input og derfor det som vil dominere i gjennomsnitt.

Ord: 80

Maks poeng: 5

5 Dette er den rekursive formuleringen av lengden til den lengste felles delsekvensen (*longest common subsequence*, LCS) for prefiksene X_i og Y_j av X og Y.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ \blacksquare\blacksquare\blacksquare\blacksquare\blacksquare & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

Hva er det som mangler? Forklar kort.

Skriv svaret ditt her. Endringer blir lagret automatisk.

'c[i-1, j-1] + 1' mangler. Da har man funnet en karakter som er lik for begge, og man velger denne da grådig, så man henter da beste løsning med de i-1 første karakterene av første streng og de j-1 første karakterene av andre streng c[i-1, j-1] og legger til 1, for man har funnet en til karakter.

Ord: 57

Maks poeng: 5

6 Felles for følgende algoritmer er at hver node v har et bestemt felt av typen v.x (men med et annet navn), som representerer en del av løsningen:

- **DFS-Visit** (altså dybde-først-søk fra én node)
- **BFS**
- **Dag-Shortest-Paths**
- **Dijkstra**
- **Bellman-Ford**
- **Prim**

Hvilket felt er det snakk om (dvs., hva skal stå i stedet for «x» i «v.x»)? Hva representerer det? Forklar kort.

(Det er altså snakk om det samme feltet i alle algoritmene.)

Skriv svaret ditt her. Endringer blir lagret automatisk.

Det er snakk om v.d. Den er som oftest brukt som korteste avstand fra startnoden til noden v (målt i antall kanter for BFS og vekt for de andre).

Utenom disse:

Prim bruker den som 'kostnad' for å legge til en node i det minimale spenntreet som den bygger.

DFS-Visit bruker den som tid når noden ble oppdaget 'discovery time'.

Ord: 60

Maks poeng: 5

7 I skog-implementasjonen av disjunkte mengder (*disjoint-set forests*), som brukt bl.a. i **Kruskal**, hva skjer med foreldrepekerne når man bruker **Find-Set**?

Skriv svaret ditt her. Endringer blir lagret automatisk.

Find-Set tar i bruk path compression, den finner representanten rekursivt for hver node og setter deretter foreldrepekeren lik representanten før den returnerer svaret.

Ord: 23

Maks poeng: 5

8 Dette er definisjonen av restkapasiteten (*residual capacity*) $c_f(u, v)$ i et flytnett (*flow network*) med flyt f:

$$c_f(u,v) = \begin{cases} [redacted] & \text{if } (u,v) \in \mathrm{E}, \\ [redacted] & \text{if } (v,u) \in \mathrm{E}, \\ 0 & \text{otherwise}. \end{cases}$$

Hva er det som mangler? Forklar kort.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Den første linjen skal være: c_f(u, v) - f(u, v), dette representerer gjenstående kapasitet

Den andre linjen skal være: f(u, v), dette representerer flyt som kan oppheves med å gå tilbake

Ord: 31

Maks poeng: 5

9 Du studerer et problem P, og vil sammenligne det med et annet, kjent problem Q ved å finne en effektiv reduksjon. Hvilken vei vil du redusere for å vise hva? Forklar kort.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Hvis du vet at Q er lett og vil vise at P er lett , så vil du redusere fra P til Q

Hvis du vet at Q er vanskelig og vil vise at P er vanskelig, så må du redusere fra Q til P

Viktig at redukjsonene er effektive ellers vil ikke resonnementet holde

Ord: 55

Maks poeng: 5

10 Hvordan kan antall sterke komponenter (*strongly connected components*) i en rettet graf endres hvis man legger til en ny kant? Forklar kort.

Skriv svaret ditt her. Endringer blir lagret automatisk.

SCC (sterke komponenter) er komponenter hvor alle noder kan nå hverandre. Anta at alle noder i en sterk komponent kan nå en node A. Hvis du da legger til en kant fra A tilbake til denne komponenten så kan A nå alle og alle kan nå A, så A blir med i denne. Hvis A var i en sterk komponent fra før så vil disse to bli slått sammen, siden du kan gå via A for å nå alle andre. Så antallet kan reduseres med 1.

Hvis du legger til en kant mellom to noder som ikke kunne nå hverandre fra før av, så vil ikke antallet endres. Det vil heller ikke endres hvis en node som ikke var med i en SCC ble med i en.

Det kan også økes med 1 ved at for eksempel: A -> B og B -> C eksisterer fra før, og så legger du til C -> A

Så antallet kan enten reduseres med 1, økes med 1, eller ikke endres.

Ord: 168

Maks poeng: 5

11 Hvorfor gir ikke **Dijkstra** nødvendigvis riktig svar dersom grafen vi bruker den på inneholder kanter med negativ vekt? Forklar kort.
(Her må forklaringen være mer presis enn f.eks. at «valgene algoritmen gjør blir gale». Hvorfor blir de gale? Bruk gjerne et eksempel, om du har behov for det.)
Skriv svaret ditt her. Endringer blir lagret automatisk.

Det er fordi den bygger på et grådighetsprinsipp som ikke gjelder for negative vekter. Den oppdaterer alltid med RELAX ut ifra noden med lavest avstandsestimat (som ikke er prossesert enda), for at dette skal fungere så må det umulig finnes sti til denne noden med lavere vekt. Hvis du går fra A->B og så B->C, så må d(A, B) < d(A, C). Enkelt eksempel: A -> B med vekt 1, A -> C med vekt 2, og C->B med vekt -2. Da vil den først gå til B og sette avstanden til 1 og si at den noden er ferdig, men dette er feil siden man kan gå via C og få avstand 0. Så hvis den har videre oppdatert kanter ut ifra B før oppdateringen fra C så vil disse få feil verdi

Ord: 134

Maks poeng: 5

12 Hvor mye øker m om du utfører følgende prosedyre?

```
1  for i = 1 to n
2      m = m + 1
3      for j = 1 to n
4          m = m + 1
5          for k = 1 to n
6              m = m + 1
```

Oppgi svaret i Θ-notasjon, som funksjon av n. Forklar kort.
Skriv svaret ditt her. Endringer blir lagret automatisk.

Theta(n^3). Du trenger bare den innerste løkken siden de andre vil øke med Theta(n) og Theta(n^2), så den innerste vil dominere siden den øker med Theta(n^3) og Theta(n) + Theta(n^2) + Theta(n^3) = Theta(n^3).

Ord: 34

Maks poeng: 5

13 Hvor mye øker m om du utfører følgende prosedyre?

```
1  k = 1
2  for i = 1 to n
3        m = m + k
4        k = k + k
```

Oppgi svaret i Θ-notasjon, som funksjon av n. Forklar kort.
Skriv svaret ditt her. Endringer blir lagret automatisk.

k vil dobles for hver iterasjon og vil være 2^i for hver iterasjon når m skal økes. m vil da bli økt med 2^0 + 2^1 + 2^2 ... 2^n = 2^(n+1) - 1. Dette vil bli Theta(2^n)

Ord: 38

Maks poeng: 5

14 Løs følgende rekurrens eksakt:
T(n) = T(n−1) + 2n−1
T(0) = 0
Oppgi svaret uten asymptotisk notasjon. Forklar kort.
Skriv svaret ditt her. Endringer blir lagret automatisk.

Du kan snu rekkefølgen og tenke på det som iterasjon, vil kjøre n ganger, og for hver iterasjon:

s = 0

for i=1 to n

  s += 2*i - 1

Dette er summen av alle oddetallene opp til 2n - 1, så 1 + 3 + 5 + 7 + 9 + ... + 2n-1. Dette er en aritmetisk rekke, som betyr at for å finne summen kan man ta gjennomsnittet av første og siste element, og gange med antall elementer. Gjennomsnitt = (2n-1 + 1) / 2 = n; antall elementer = n; total sum = n^2.

Svaret på rekurrensen er n^2 eksakt, T(n) = n^2

Ord: 107

Maks poeng: 5

15 Hva blir T(n), om du løser følgende rekursive ligningssett?
$T(n) = 2S(n/2) + (n \lg n)^2$
$S(n) = 8T(n/2) + n^2$
Oppgi svaret i Θ-notasjon. Forklar kort.
(Bruk vanlige antagelser for algoritmiske rekurrenser.)
Skriv svaret ditt her. Endringer blir lagret automatisk.

Bytter først inn uttrykket for S(n) i T(n), får da:

T(n) = 2* (8*T(n/4) + (n/2)^2) + (nlgn)^2

T(n)= 16T(n/4) + n^2/2 + n^2*lg^2(n)

T(n) = 16T(n/4) + Theta(n^2 * lg^2(n))

Kan nå bruke masterteoremet:

log4(16) = 2, siden 4^2 = 16

Dette er andre tilfellet i masterteoremet og svaret blir da:

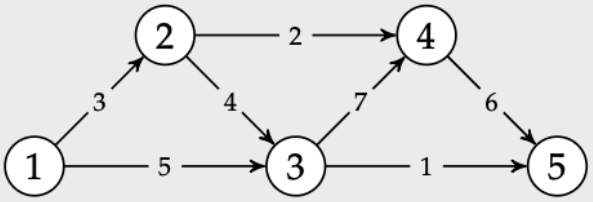
T(n) = Theta(n^2 * lg^3(n))

Ord: 57

Maks poeng: 5

16

Figur 1



Utfør den første av de |V|−1 iterasjonene til **Bellman-Ford** på grafen i figur 1, under følgende betingelser:

1. Bruk node 1 som startnode.
2. Anta at 8 er brukt i stedet for ∞ under initialiseringen.
3. Der du kan velge mellom kanter, velg den med lavest vekt.

Hva blir v.d for hver node v = 1, … , 5 etterpå? Forklar kort.

(Oppgi verdien for hver node, i rekkefølge.)

Skriv svaret ditt her. Endringer blir lagret automatisk.

v1.d = 0

v2.d = 3

v3.d = 5

v4.d = 5

v5.d = 6

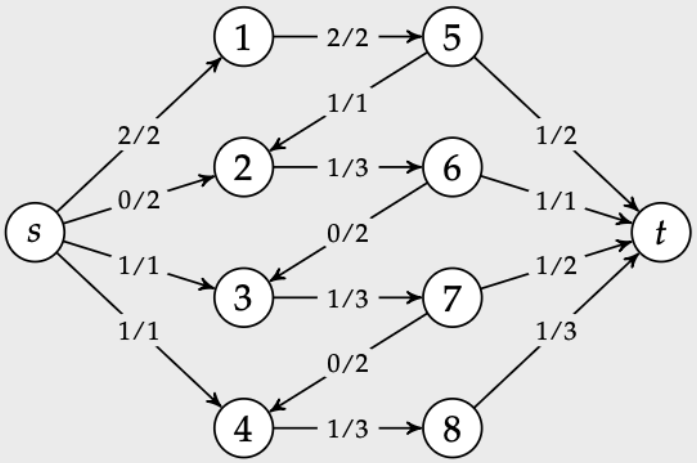
Den vil få riktig svar etter bare en iterasjon siden den oppdaterer den med lavest vekt, det finnes ikke negative kantvekter og det finnes ingen sykler.

Ord: 41

Maks poeng: 5

17

Figur 2



Figur 2 viser et flytnett (*flow network*) med flyt. Utfør én iterasjon av **Edmonds-Karp** på flytnettet for å finne en forøkende sti (*augmenting path*). Oppgi nodene i den resulterende stien, i rekkefølge. Forklar kort.

Skriv svaret ditt her. Endringer blir lagret automatisk.

s, 2, 5, t. Den vil finne den korteste stien fra s til t målt i antall kanter (bruker BFS). Viktig å huske på at den bruker restnettet for traversering som har bakoverkanter hvor det går flyt, for eksempel 2 -> 5.

Ord: 42

Maks poeng: 5

18 Nøklene (*keys*) i et søketre er vanligvis fra en ordnet mengde (f.eks. tall eller tekststrenger). Din venn Gløgsund har laget et søketre der hver node i stedet inneholder et rektangel. Når hun skal bygge treet, tar hun inn et sett S med ikke-overlappende rektangler, som legges i løvnodene. For hver indre node konstruerer hun et nytt rektangel: det minste som inneholder alle barnas rektangler.

Når hun skal søke i treet, tar hun inn et punkt, og sjekker rekursivt nedover i treet hvilke rektangler som inneholder punktet, og returnerer det av disse som ligger i en løvnode. Gjør følgende antagelser:

• Rektanglene fordeles tilfeldig på løvnodene.

• Treet er perfekt balansert.

Hva blir kjøretiden for et søk, som funksjon av n = |S|? Gi både en øvre og nedre grense, dvs., bruk enten Θ-notasjon eller både O- og Ω-notasjon.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Antar at det tar konstant tid å sjekke om et punkt er innenfor et rektangel. Kallhøyden vil være Theta(lgn) siden det er perfekt balansert, og alle løvnodene vil da være på siste nivå. Det vil bli gjort konstant mengde arbeid i hvert nivå. Kjøretiden for et søk er da Theta(lgn).

Ord: 50

Maks poeng: 5

19 Du utvikler et kodebibliotek for tekstprosessering, og skal implementere funksjonen split(s, x), som deler strengen s ved alle forekomster av tegnet x. For eksempel vil
split("abc;de;fghi;jk", ";")
returnere
["abc", "de", "fghi", "jk"].
Du kan finne indeksene til x i s med en løkke. For en gitt slik indeks, kan du dele strengen i to mindre strenger, som er kopier av hver sin «halvdel». Om strengen du deler inneholder n tegn, må du kopiere n−1 tegn. Konstruer og beskriv en algoritme som finner det minste totale antallet slike kopieringer som trengs. Hva blir kjøretiden? Forklar.
Skriv svaret ditt her. Endringer blir lagret automatisk.

Bruker splitt og hersk med rekursjon, for hvert funksjonskall iterer over strengen og finn x-en som er nærmest midten, splitt den der og løs rekursivt for hver halvdel. Kan eventuelt iterere over en gang i starten og så huske alle indeksene hvor tegnet forekommer, men dette vil ikke føre til bedre asympotisk kjøretid. Du må da kopiere n tegn for hvert kall, men lengden kuttes i beste tilfellet med 2 for hvert kall (hvis den splittes i det hele tatt).

Algoritmen kan fullføre på Theta(n) tid, når x ikke eksisterer i s. Så nedre grense:
Omega(n)

Algoritmen vil som oftest ha rekursjonshøyde på lgn og rekurrensen er da T(n) = 2T(n/2) + n, som er samme som MergeSort. Gjennomsnittskjøretiden blir da Theta(nlogn)

I veldig få tilfeller vil strengen være konstruert slik at det ikke er mulig å få lgn. Hvis den er på formatet axbxa, hvor b ikke inneholder noen x og dominerer størrelsen. Hvis a igjen er på samme format, så blir det vanskelig for algoritmen, den må da splitte i starten eller slutten på axbxa, men vil da ende opp med axbxa igjen, som den igjen må splitte i starten eller slutten. Dette er et eksempel på en slik streng, merk at antall etterfølgende b-er mot midten vokser eksponensielt:

xbbbbxxbbbbbbbbbbbbbbbbbbbbbbbbbbbxxbbbbx

Det at b-ene må vokse eksponensielt redder oss, for når vi har en lang sekvens med b-ere, så vil disse ikke skape flere rekursjonskall, så denne vil bli f(xbbbbx) + f(bbbbbbbbbbbbbbbbbbbbbbbbbbbbxxbbbbx) = f(xbbbbx) + bbbbbbbbbbbbbbbbbbbbbbbbbbb + f(xbbbbx)

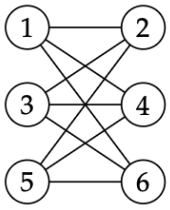
Den har da kjøretid Theta(nlogn) i verste tilfellet.

Kjøretiden totalt sett er Omega(n), O(nlogn)

Ord: 263

Maks poeng: 5

20 For to grafer G og H er en grafhomomorfi f : G → H en funksjon f fra G.V til H.V, som bevarer naboskap. For alle u, v ∈ G.V:
(u, v) ∈ G.E ⇒ (f(u), f(v)) ∈ H.E
Det vil si, hvis (u, v) er en kant i G, så er (f(u), f(v)) en kant i H. Å avgjøre om det eksisterer en homomorfi, gitt G og H, er NP-komplett generelt, men kan i visse tilfeller gjøres i polynomisk tid.
Anta at vi bestemmer at H alltid skal være følgende graf, kjent som $K_{3,3}$:



Input er nå bare G, og vi skal avgjøre om det finnes en homomorfi f : G → $K_{3,3}$.
Enten vis at denne begrensede versjonen av problemet fortsatt er NP-komplett, eller vis hvordan problemet kan løses (dvs., avgjøres) i polynomisk tid. Kan du trekke noen generelle konklusjoner fra svaret ditt? Forklar.

Skriv svaret ditt her. Endringer blir lagret automatisk.

Hvis det er flere sammenhengende komponenter i G, så kan man løse problemet rekursivt for hver av dem.

Hvis det finnes en sykel i en sammenhengende komponent av oddetallslengde, så finnes det ikke en homomorfi. Dette er fordi den da ender opp på feil side av grafen, og kan ikke komme seg tilbake til start, siden det ikke kan eksistere en kant der. Hvis det ikke finnes en sykel av oddetalls lengde, så finnes det en homomorfi. For eksempel hvis du har en lang enkel sti av noder, så kan du oversette dem alternerende mellom to node, f.eks 1 og 2. Enkeltnoder kan oversettes til hvilke som helst node. For noder som danner sykler av partalls lengde kan du gå i ring og oversette dem. Første node i sykelen blir 1, andre blir 2, tredje blir 3, fjerde blir 4, femte blir 1 igjen, og sånn fortsetter du. Så det er bare sykler av oddetalls lengde som skaper problemer.

Vi kan løse dette med DFS, som har kjøretid O(V+E). Først kjør gjennom med lignende kode som DFS for å finne sammenhengende komponenter, altså start et søk fra en node som ikke er oppdaget, alle noder som oppdages fra denne vil være i samme sammenhengende komponent, dette gir O(V+E) kjøretid.

Så må du sjekke om det finnes oddetalls sykler i hvert sammenhengende komponent. Bruk DFS for å oppdage sykler. Hvis du møter på en node som er oppdaget, men ikke ferdig, så har du oppdaget en sykel, som i vanlig DFS. Men må ha en måte å vite om denne har oddetalls lengde. Kan legge til et boolean felt på hver node som kan kalles noe som 'isEven'. Når du oppdager en ny node så setter du v.isEven = not p.isEven, så den har motsatt verdi fra forelderen. Hvis du oppdager en sykel, og begge nodene har samme verdi, så har sykelen oddetalls lengde, og du returnerer False. Du må huske på å alltid sjekke hva returnverdien er når du kaller rekursivt, og returnere False hvis den returnerer False.

Det at dette problemet kan løses i polynomisk tid sier oss ikke noe mer generelt enn at noen versjoner av problemet kan løses i polynomisk tid. Hvis du hadde vist at det GENERELLE homomorfi problemet kunne løses i polynomisk tid, så hadde du vist at P=NP.

Ord: 384

Maks poeng: 5