NTNU

Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Eksamensoppgave i IDATT2101 Algoritmer og datastrukturer

Faglig kontakt under eksamen: Helge Hafting

Tlf.: 924 386 56

Eksamensdato: 24. november 2020

Eksamenstid (fra–til): 09:00–13:00

Hjelpemiddelkode/Tillatte hjelpemidler: Alle

Annen informasjon: Løsningsforslag

Målform/språk: bokmål

Antall sider (uten forside): 8

Antall sider vedlegg: 1

Informasjon om trykking av eksamensoppgave			
Originalen er:			
1-sidig	☐	2-sidig	☒
sort/hvit	☐	farger	☒
skal ha flervalgskjema			☐

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.

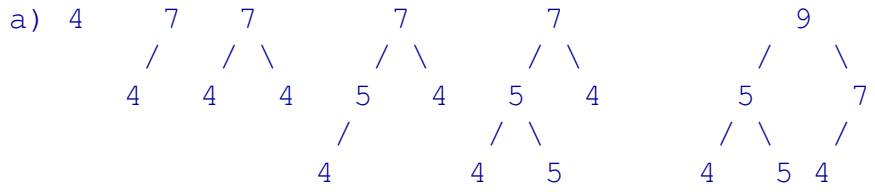
Oppgave 1 — 10%

Du har fått et kandidatnr for å skrive på oppgaven og vedlegg. Gang dette kandidatnummeret med 47. (Kalkulator kan brukes.) Tallet du får, blir sifre for denne oppgaven.

a) Sett sifrene inn i en max-heap. Sett dem inn ett om gangen, i den rekkefølgen de har i tallet. Tegn opp heapen en gang for hvert tall du setter inn.

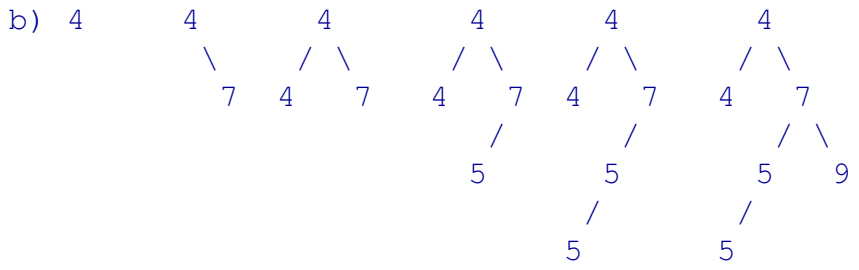
Individuell løsning. De fleste greide dette, så jeg lager ikke alle de 84 løsningene…

Eksempel for kandidatnr 10097: $10097 \cdot 47 = 474559$. Tallserie: 4,7,4,5,5,9



b) Sett de samme sifrene inn i et binært søketre. Sett dem inn i den rekkefølgen de har i tallet, og tegn opp søketreet en gang for hvert tall du setter inn.

Her er det flere like sifre. Jeg velger å plassere like sifre i venstre subtre. (Noen valgte høyre i stedet. Noen tillot ikke duplikater. Alle varianter ble godtatt.)



Oppgave 2 — 20%

Analyser de følgende programmene og finn kjøretiden. Regn med at alle parametre er større enn eller lik 0. Bruk Θ om mulig, ellers O og Ω.

```
pubic void oppg_a(int m, int [][] tab) {
  for (int i = 1; i < m; ++i) {
        for (int j = m; j > 0; --j) tab[i][j] = i*j;
    }
}
```

$\Theta(m^2)$

```
public int oppg_b(int n, int p) {
    int res;
    if (n >=p) return -1;
  for (int i = 0; i < p; i+=n) res += i;
    return res;
}
```

$\Omega(1), O(p/n)$

```
public float oppg_c(int n, float[] tab) {
    float sum = 0.0;
    if (n <= 0) return sum;
  for (int i=0; i<n; ++i) sum += tab[i];
  return sum + oppg_c(n/2, tab);
}
```

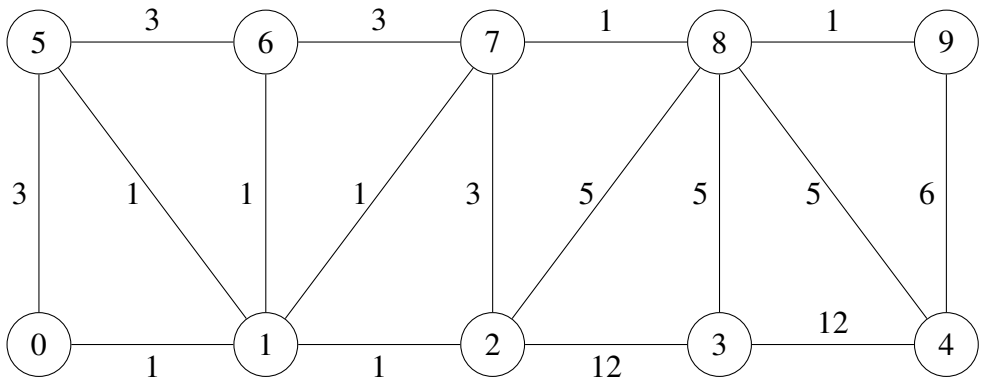
$T(n) \in \Theta(n)$, i følge mastermetoden.

```
public float oppg_d(int m, float[] tab) {
    float sum = 0.0;
    if (m <= 0) return sum;
    for (int i=0; i<m; ++i) {
        for (int j=i; j<m; ++j) {
            sum += tab[i] * tab[j];
        }
    }
    sum += oppg_d(m/2, tab);
    return sum + oppg_d(m/2, tab);
}
```

$T(x) \in \Theta(m^2)$. etter mastermetoden

Oppgave 3 **30%**

Gitt denne grafen:



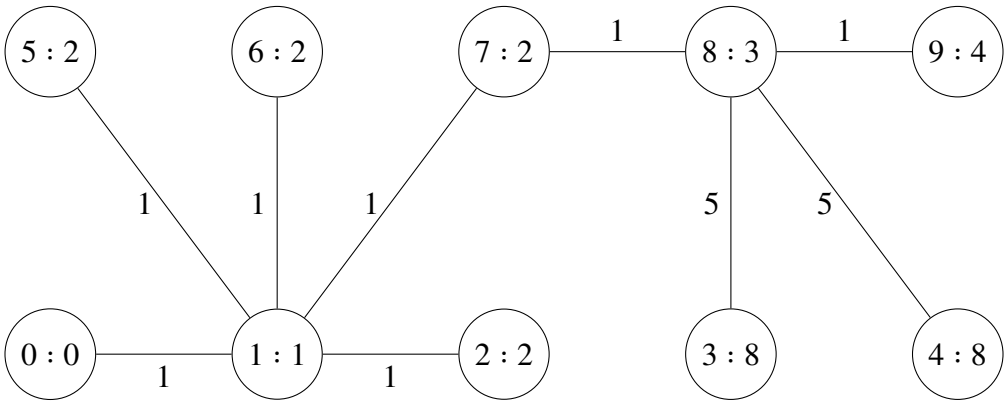
a) Velg noden med samme nr. som siste siffer i kandidatnummeret ditt som startnode. Finn korteste vei fra startnoden til alle andre noder i grafen, og tegn korteste-vei treet.

b) Forklar hvorfor Dijkstras algoritme ikke fungerer korrekt på grafer som har kanter med negative vekter.

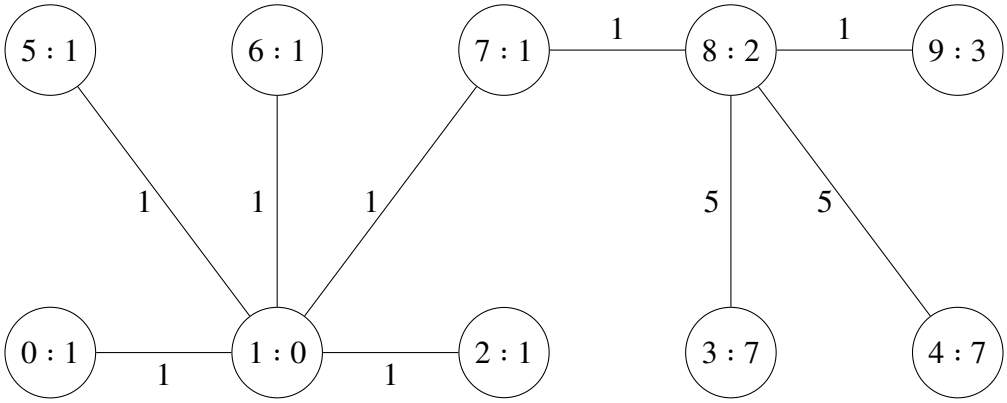
DELOPPGAVE 3a

10 ulike løsninger, avhengig av hva siste siffer i kandidatnummeret var.

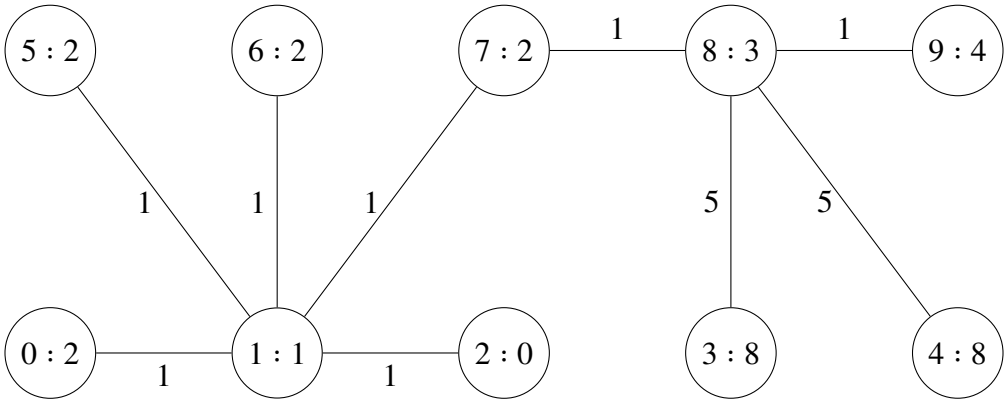
Løsning 0:



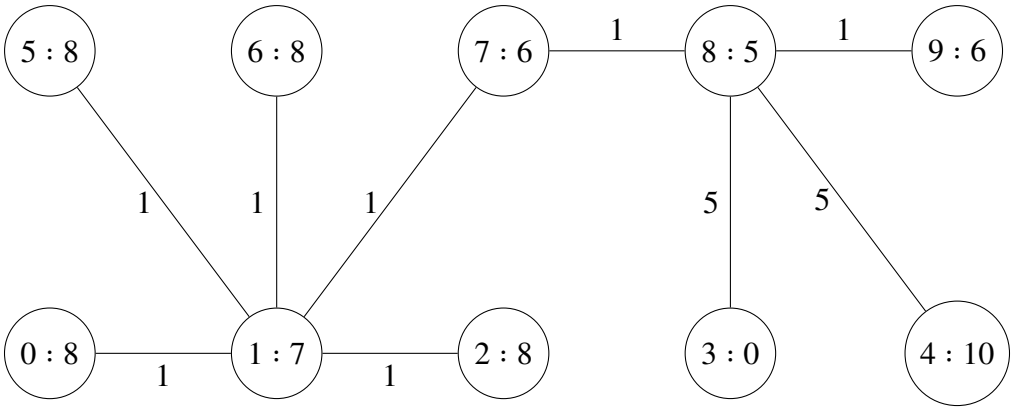
Løsning 1:



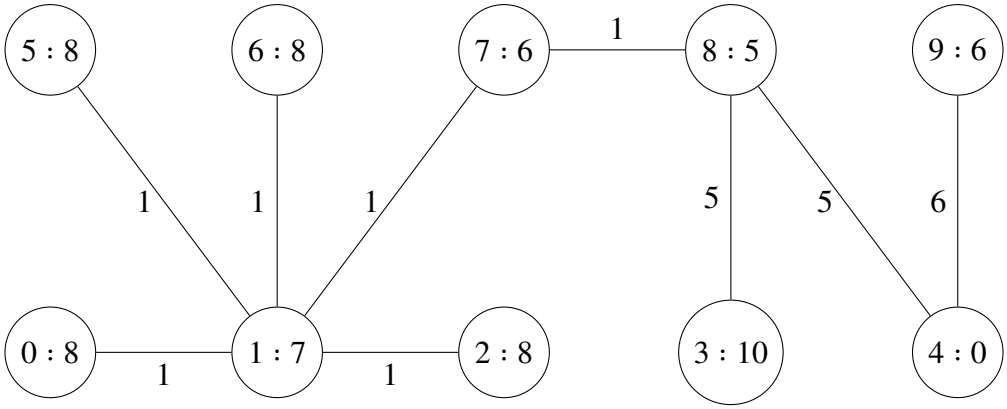
Løsning 2:



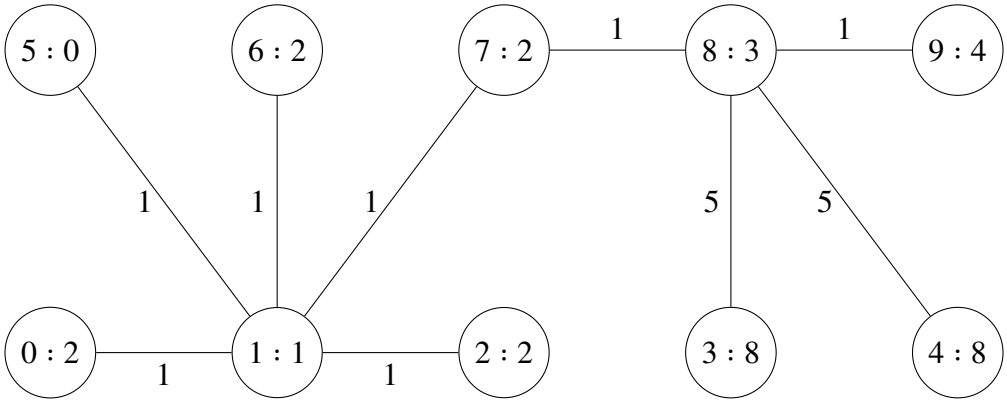
Løsning 3:


5 : 8
6 : 8
7 : 6
8 : 5
9 : 6
0 : 8
1 : 7
2 : 8
3 : 0
4 : 10
1
1
1
1
1
1
1
5
5

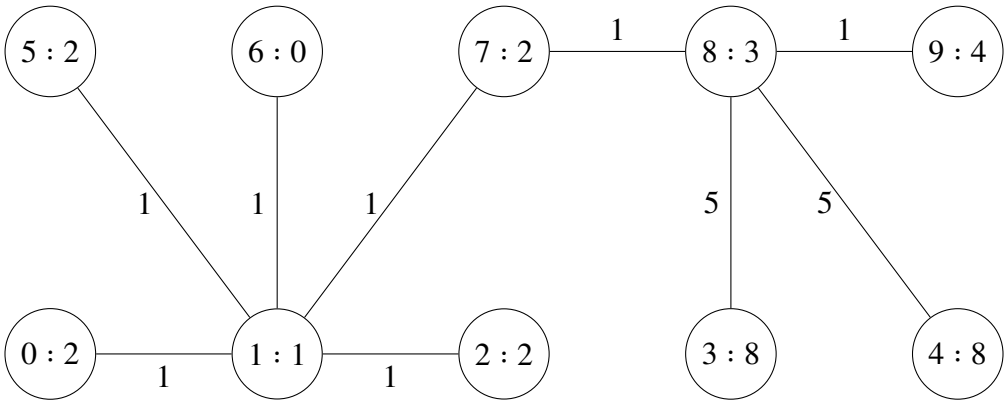
Løsning 4: (kan bruke kanten 8–9 i stedet for 4–9, det gir samme vekt på spenntreet)



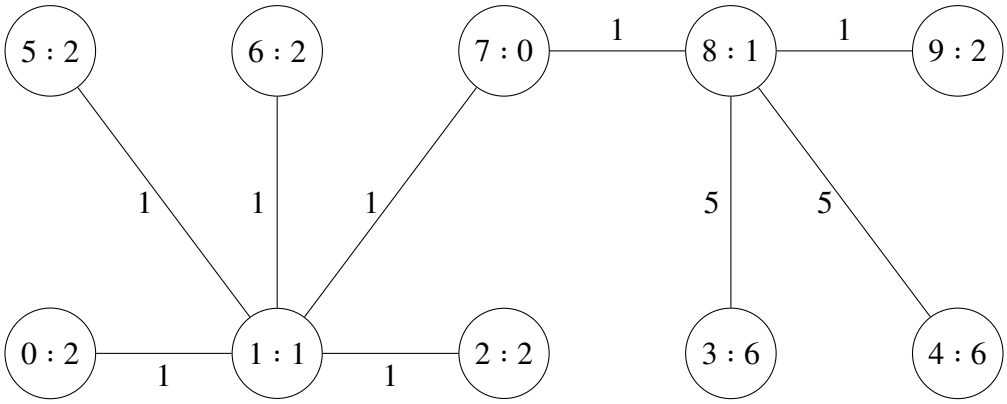
Løsning 5:



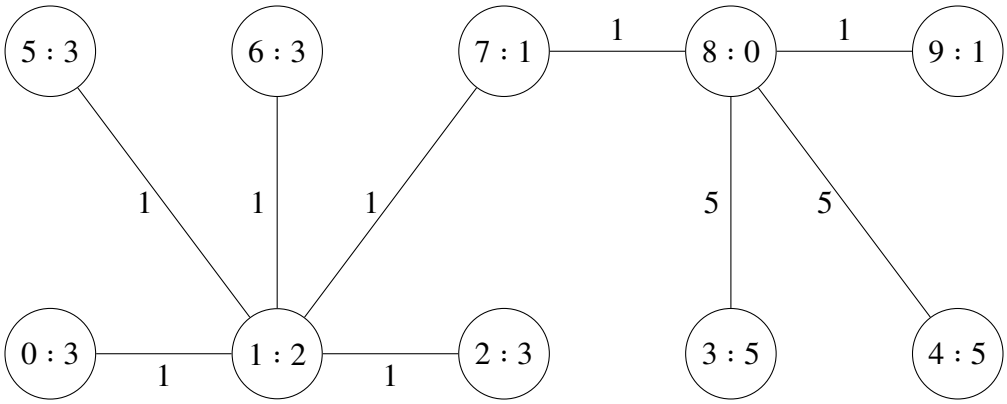
Løsning 6:



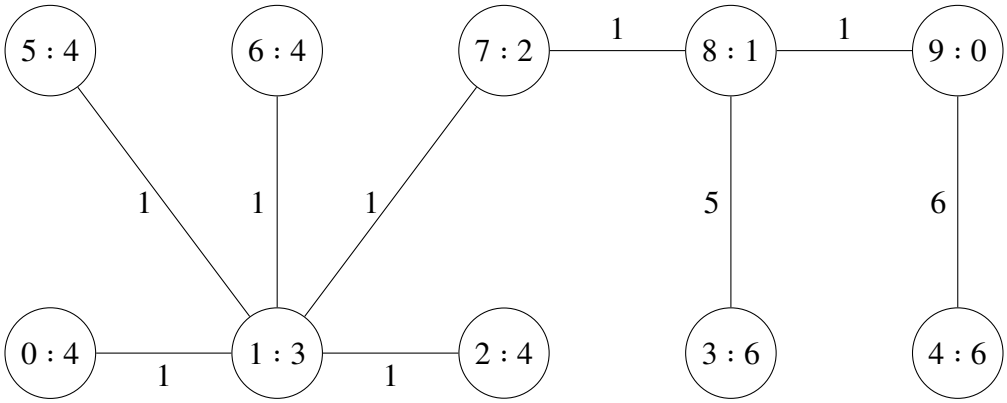
Løsning 7:



Løsning 8:



Løsning 9: (Kan bruke kanten 8–4 i stedet for 9–4, det gir samme vekt på spenntreet)



DELOPPGAVE 3b

Dijkstras algoritme ser på hver node bare én gang, og regner deretter noden som «ferdig behandlet». For at dette skal gi rett resultat, må en node med kort avstand gjøres ferdig før alle noder med lengre avstand. Algoritmen bruker en prioritetskø for å finne den uferdige noden med kortest avstand, og sjekker om kantene ut fra den gir kortere veier til naboene. Alle ferdige noder har kortere avstand enn den noden som er under behandling. Alle uferdige noder har lenger avstand.

Med positive kantvekter kan vi aldri få kortere vei inn mot en ferdigbehandlet node, fordi noden vi jobber med allerede har høyere avstand. (En positiv kantvekt kan bare gjøre avstanden enda større.)

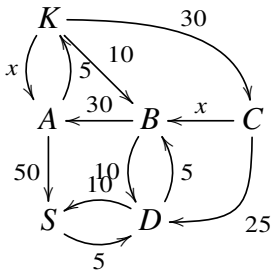
Med negative kantvekter kan en node som ligger lenger unna, likevel gi kortere vei inn til en «ferdig» node. I så fall må kantene ut fra denne noden undersøkes på nytt, men det gjør ikke Dijkstras algoritme. Derfor kan den mislykkes med negative kanter.

Dette kan skrives på mange andre måter. Noen viste hvordan dette går galt, med en eksempelgraf med negativ kant.

Problemer med negative rundturer er et annet problem, som *ikke* er tilstrekkelig svar på 6b.

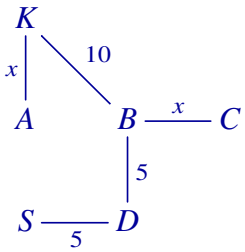
Oppgave 4 — 25%

Bruk denne grafen. Bruk siste siffer i kandidatnummeret ditt, som vekt på de to kantene merket x.



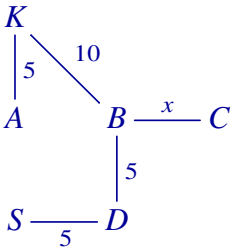
a) Finn og tegn et minimalt spenntre med vekt, eller forklar hvorfor det ikke er mulig. (På denne deloppgaven ser du bort fra pilene, så det blir en urettet graf.)

Løsning for $x \leqslant 5$:



KA:x, BC:x, SD:5, BD:5, KB:10, vekt=20 + 2x

Løsning for $9 \geqslant x \geqslant 5$:



KA:5, SD:5, DB:5, BC:x, KB:10, vekt=25 + x

x	0	1	2	3	4	5	6	7	8	9
vekt	20	22	24	26	28	30	31	32	33	34

b) Finn maksimal flyt fra K til S ved hjelp av flytøkende veier. Skriv opp hvilke flytøkende veier du bruker, og hvor mye du øker flyten langs hver vei. I denne deloppgaven er grafen rettet.

Maksimum flyt: $|f| = 25 + 2x$. Det er mange ulike korrekte løsninger for de flytøkende veiene. Alle som gir rett sum, er ok. F.eks.:
$K \to A \to S : x$
$K \to B \to D \to S : 10$
$K \to C \to B \to A \to S : x$
$K \to C \to D \to B \to A \to S : 15$

x	0	1	2	3	4	5	6	7	8	9
Max flyt	25	27	29	31	33	35	37	39	41	43

Oppgave 5 — 15%

a) Forklar hva det vil si, at et program eller problem er i kompleksitetsklassen **P**. Gi eksempel på et slikt problem.

Problemer i **P** kan *løses* på polynomisk tid.

b) Forklar hva det vil si, at et problem er i kompleksitetsklassen **NP**.

Løsningsforslag til et problem i **NP** kan sjekkes på polynomisk tid. (Problemet trenger ikke kunne *løses* på polynomisk tid.)

c) Kåre vil lagre postnumre i en hashtabell. Tabellen har størrelse 100. Mange postnumre slutter på 0. Vil det være lurt å bruke en hashfunksjon basert på restdivisjon? Hvorfor, eller hvorfor ikke?

Nei.

Ved restdivisjon, deler vi med tabellstørrelsen for å spre nøklene utover. Og den er 100 i dette tilfellet.

Alle numre som slutter på 00 vil hashe til samme plass, det gir unødvendig mange kollisjoner. Tilsvarende vil alle som slutter på 20 hashe til plass 20, alle som slutter på 30 vil hashe til plass 30, osv. Det var opplyst om at mange postnumre slutter på 0, de er altså ikke jevnt fordelt.

Ulikhet i første del av postnummeret nyttes ikke, ved restdivisjon med 100. Et primtall ville vært bedre, men her var størrelsen satt til 100. Restdivisjon med noe annet enn tabellstørrelsen, vil gi dårlig spredning og unødige kollisjoner. Kåre bør bruke en annen hashfunksjon, f.eks. en basert på multiplikasjon.