45011 Algoritmer og datastrukturer
Løsningsforslag eksamen 13. januar 1992

Oppgave 1

a)

Idé til algoritme

Benytter S_n som betegnelse på en tallmengde med n elementer. For at et tall m skal være et majoritetstall blant n tall, må m (etter oppgavens definisjon) forekomme mer enn $(n \operatorname{div} 2) + (n \bmod 2)$ ganger. Dersom det finnes et majoritetstall m i S_n, og det finnes to *vilkårlige* tall x_i og x_j hvor $x_i \neq x_j, x_i \in S_n, x_j \in S_n$, så må m også være et majoritetstall i $S_{n-2} = S_n - \{x_i, x_j\}$. Siden x_i og x_j er ulike, kan maksimalt ett av dem være lik majoritetstallet m i S_n. Når vi tar dem bort, kan antall reprensentanter for majoritetstallet reduseres, men samtidig reduseres også antall andre tall minst like mye. Et majoritetstall m vil derfor bevares.

Ved å ta bort to og to ulike tall i S er vi derfor sikre på at et eventuelt majoritetstall vil bli igjen. Når det så til slutt er igjen ett eller flere like tall, vil denne verdien være det eneste *mulige* majoritetstall. Den opprinnelige tabellen testes derfor for å se om dette tallet faktisk er i majoritet.

PASCAL-kode

```
function majoritet ( x : Stabell; n: integer): integer;
var
   m, antall, i : integer;
begin
   m := x[1];
   antall := 1;
   for i := 2 to n do
      if antall = 0 then
         begin
            m := x[i]; { Ny majoritetskandidat }
            antall := 1;
         end
      else
         if m <> x[i] then
            antall := antall - 1 { Glemmer to forskjellige tall }
         else
            antall := antall + 1; { Øker antall m-kandidater }
   {m er nå eneste mulige kanditat }
   if antall = 0 then
      majoritet = -1
   else
```

```
    begin
      antall := 0;
      for i := 1 to n do
        if x[i] = m then antall := antall + 1;
      if antall > ((n div 2) + (n mod 2)) then
        majoritet := m
      else
        majoritet := -1;
    end;
end;
```

b)
Den foreslåtte algoritmen består av to iterasjoner over n, og er derfor opplagt $\Theta(n)$. Alternative løsningsmetoder kunne være:

Median Dersom det finnes et majoritetstall m vil dette også være medianen i tallmengden. Ved bruk av algoritmen SELECT (s. 190-191 i læreboka) kan dette gjøres i $\Theta(n)$ tid.

Sortering Ved først å sortere tallene, vil eneste kandidat for medianverdi havne midt i intervallet. Sortering med radix-sort vil ta $\Theta(n)$ tid, men vil praktisk sett være tregere enn algoritmen over siden det gjøres mye "unødig" arbeid. Ved bruk av Quicksort vil tidsforbruket økes til $O(n^2)$.

Oppgave 2

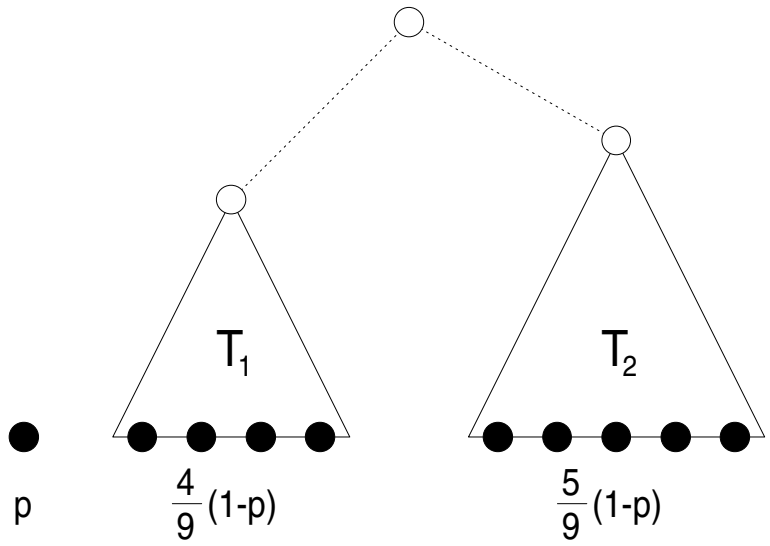
a)
Siden sifferet 0 har større frekvens enn alle de andre til sammen, vil dette bli kodet med en bit. De resterende siffer vil kodes i et tilnærmet balansert binærtre. En mulig koding er:

0:	0	5:	1100
1:	1000	6:	1101
2:	1001	7:	1111
3:	1010	8:	11100
4:	1011	9:	11101

Midlere kodelengde blir

$$0.6 \cdot 1 \cdot 1 + \frac{0.4}{9} \cdot 4 \cdot 7 + \frac{0.4}{9} \cdot 5 \cdot 2 \quad = \quad 2.288\ldots$$

b)
For at kodingen fra (a) ikke lenger skal være optimal, må p bli så lav at 0 ikke lenger skal kodes med en bit. Dette vil skje når sifrene 1-9 ikke lenger danner et eget sub-tre i kodetreet.



Figur 1: Delvis oppbygd kodetre

Figur 1 viser situasjonen rett før sifrene 1-9 skal bli slått sammen til et eget tre. Tallene under hvert deltre angir total frekvens for dette deltreet. Dersom denne sammenslåingen skal være korrekt (T_1 og T_2 slås sammen) må

$$\begin{aligned} p &> \frac{5}{9}(1-p) \\ &\Downarrow \\ p &> \frac{5}{14} \approx 0.36 \end{aligned}$$

Så lenge dette er oppfylt vil koden være optimal.

c)
Definerer f til å være frekvensen til tegnet med minst frekvens. Under oppbygging av kodetreet vil først de to tegnene med lavest frekvens velges. Summen av disse nodenes frekvenser vil være $\geq 2f$. Siden enkelttegnet med høyest frekvens har en frekvens mindre enn $2f$, vil dette havne på samme nivå i kodetreet som tegnet med minst frekvens. Kodetreet vil derfor bli perfekt balansert, og alle tegn får en 8-bits kode. Huffmannkoding vil derfor ikke være mer effektiv enn vanlig fast 8-bits tegnkode i dette tilfellet.

Oppgave 3

a)
Forslaget til endring støttes fordi det sparer ett gjennomløp samtidig som svaret fremdeles blir korrekt ut i fra følgende begrunnelse:

Forandringen vil medføre at RELAX ikke vil bli utført når det bare er en node igjen i køen. Denne noden er den som har den lengste av alle korteste veier fra startnoden. Siden dette er tilfelle, kan heller ingen av de andre nodene nås raskere ved å gå via denne noden. Løsningen vil derfor fremdeles være korrekt. (Husk på at Dijkstra bare fungerer på grafer med positive vekter.)

Dersom en betrakter Dijkstra-algoritmen som syklister som sendes ut fra startnoden, vil dette tilsvare at den som kommer til den siste noden ikke trenger å sende ut nye syklister. Siden alle andre noder allerede er nådd, kan det umulig være kortere å gå via den siste.

b)
Bellmann-Fords algoritme tillater negative kantlengder (til forskjell fra Dijkstras algoritme). La s være kildenode og v en vilkårlig node i en graf med V kanter. Bellmann-Fords algoritme består av to faser:

Fase 1

- 1. gjennomløp finner den korteste vei fra s til v som består av maksimalt 1 kant.
- 2. gjennomløp finner den korteste vei fra s til v som består av maksimalt 2 kanter.
- $\vdots$
- $\mid V-1 \mid$. gjennomløp finner den korteste vei fra s til v som består av maksimalt $\mid V-1 \mid$ kanter.

Merk at alle kanter forsøkes på nytt som "skjøteelementer" for hvert gjennomløp. Dette garanterer at alle "max i-kants" veier blir vurdert, også den korteste av disse.

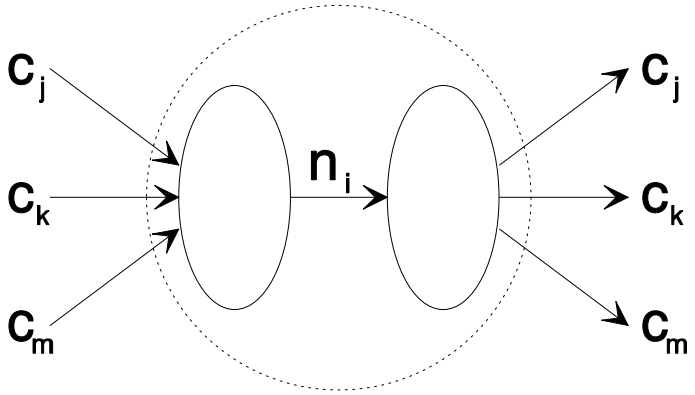
Fase 2

Fase 1 gir den korteste vei fra s til v som består av mindre enn V kanter. Det må derfor letes etter en vei som er enda kortere, men med flere kanter. I en korteste vei med V kanter eller mer vil minst én node inngå minst to ganger i den korteste veien. (Dersom V kanter skal forbinde noder i en liste trengs $V+1$ noder.) En node som inngår to ganger i en korteste vei, må være del av en løkke. Det må derfor være lønnsomt å gå i løkke på den korteste vei (s, v). For at så skal være tilfelle må løkken ha samlet negativ kostnad. Dersom en slik løkke eksisterer, vil korteste vei inneholde uendelig mange rundturer i løkken. Fase 2 begrenser seg derfor til å lete etter/finne slike løkker.

Oppgave 4

a)
Rømningsproblemet kan omformuleres til et flyt-problem hvor ikke bare kantene har kapasiteter, men der det også er en begrensning på hvor stor flyt som kan gå gjennom hver enkelt node.

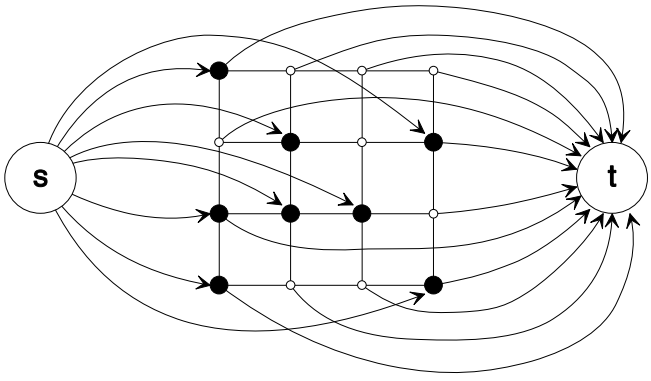
For å få problemet over på samme form som vanlige flyt-problem, må hver node i grafen omdannes som vist i figur 2. Den tidligere noden (stiplet sirkel) er delt i to, en node med alle inngående kanter fra den gamle noden, og en med alle utgående kanter. Mellom disse to nodene er det satt inn en kant med vekt n_i, som tilsvarer den totale gjennomstrømmningskapasitet i den gamle noden.



Figur2: Node med begrensning på total gjennomstrømning

Løsningen på rømningsproblemet blir da som følger:

Alle m startnoder som ikke ligger på randen av området tenkes å ha en kapasitet 1 fra en (virtuell) kildenode, samt kanter til alle fire nabonoder (også med kapasitet 1). Alle noder som ligger på randen av området får på samme måte en kant til et felles (virtuelt) sluk. De av disse som er startnoder har også en kant fra kilden.



Figur3: Flytnettverk for løsing av rømningsproblemet

Matrisen er dermed utvidet til et nettverk som vist i figur 3. De indre nodene som ikke er startnoder blir splittet i to som beskrevet over med en gjennomstrømningskapasitet på 1.

For å avgjøre om det finnes en rømningsvei for alle startnoder benyttes en maksimalflytalgoritme. Dersom en løsning finnes, vil enhver startnode finne en vei til sluket som ikke er i konflikt med en annen startnode. En løsning er derfor funnet om den totale flyten gjennom nettverket blir lik m. Rømningsveiene for startnodene finnes som de kantene som blir benyttet ved maksimum flyt.

b)
De neste flytforøkende veier som finnes blir: (D,5)(E,6),(F,2,B,4)
Den siste flytforøkende vei består i at flyten fra B til 2 kanselleres, mens det opprettes ny flyt fra B til 4, og fra F til 2. Nettoflyten vil derfor øke.

Oppgave 5

Antall utvalg av noder er oppgitt som n^k i oppgaveteksten. Dette er ment som en øvre grense, og dessuten lettere å regne med enn den eksakte verdien ($\left(\begin{array}{c} \text{n} \\ \text{k} \end{array} \right) \leq n^k$).

a)
Dersom K skal være med i NP må en gjettet løsning kunne verifiseres i polynomisk tid. Det kan åpenbart testes i $O(k^2)$ tid om et tilfeldig utvalg av k noder utgjør en klikk. Siden $k \leq n$ vil $O(k^2) \leq O(n^2)$. Testen kan derfor foretas i polynomisk tid, og problemet K er da pr. definisjon med i NP.

b)
Slik K er definert (G har n noder) har n ingen øvre grense, mens k har n som grense. A har derfor tidsforbruk som er $O(n^n \cdot n^2) = O(n^{n+2})$, dvs. minst eksponentielt tidsforbruk. Men at algoritmen A har eksponentielt tidsforbruk betyr *ikke* nødvendigvis at problemet K må være med i NPC. (A kan være mer "komplisert" enn K, og en "skyter spurv med kanon".) K kan derfor ikke klassifiseres ytterligere ut i fra opplysningene over.

Det er imidlertid bevist ved reduksjon til NPC at K $\in$ NPC.