

SIF8010 ALGORITMER OG DATASTRUKTURER

KONTINUASJONSEKSAMEN, 1999; LØSNINGSFORSLAG

OPPGAVE 1 (12%)

Anta at du skal lage et støtteprogram som umiddelbart skal varsle om at et ord blir skrevet feil under inntasting av vanlig tekst. Lovlige skrivemåter må nødvendigvis forhåndslagres i en ordliste for at dette skal være mulig.

a. Hvilken datastruktur vil du benytte for å lagre ordlisten?

Svar: (6%)
TRIES er det eneste riktige fra pensum.

b. Hvordan skal ditt program bruke ordlisten for å varsle feilstavelser i forhold til ordlisten?

Svar: (6%)

- Traversere TRIES'en for *hvert* nytt ord
⇒ Bruke “ett tegn” per navigerings-trinn.
- Varsle “ulovlig tegn” dersom løvnode (NIL) nås, dvs. lovlig ord ikke påtruffet.

OPPGAVE 2 (12%)

a. Hvordan finner Floyd–Warshalls algoritme ut om en graf G har sykler med negativ vekt?

Svar: (6%)
Det oppstår negative tall på diagonalen i D-matrisen. F–W må da stoppes.

b. Beskriv kort en algoritme som finner hvor mange sammenhengende subgrafer en urettet graf G består av. En (sub-)graf er sammenhengende dersom det er en kjede av kanter mellom alle par av noder i (sub-)grafen. (Merk: En graf G uten kanter kan klart ikke være sammenhengende.)

Svar: (6%)

- (1) Start farging ved en vilkårlig start-node. Traversér G så langt det går.
- (2) Finn, om mulig, ny ufarget node (dvs. ny “vilkårlig node”).
- (3) Hvis en ny start-node ble funnet, gå til punkt 1. Hvis ikke, avslutt.

Antallet sammenhengende subgrafer = antallet farger brukt.

OPPGAVE 3 (22%)

Vi definerer her et problem som vi kaller $P(x, n, k)$:

Det er gitt en tallfølge med n positive heltall: x_1, x_2, \dots, x_n . Denne tallfølgen skal deles inn i k påfølgende intervaller ($k < n + 1$) slik at den største intervallsummen blir minst mulig.

Eksempel. $P(x, 15, 3)$, $x = [1, 7, 3, 8, 10, 3, 1, 2, 1, 3, 6, 3, 5, 5, 3]$. Den beste inndelingen i intervaller er her:

$$[1, 7, 3, 8], [10, 3, 1, 2, 1, 3], [6, 3, 5, 5, 3]$$

De tre intervallsummene er henholdsvis 19, 20, og 22. Her er 22 den største intervallsummen, og samtidig den minste verdien vi kan få for den største intervallsummen i dette eksemplet.

a. Beskriv en effektiv algoritme, spesielt tilpasset problemet $P(x, n, 2)$. Du skal utføre så få addisjoner som mulig, samt finne algoritmens tidskompleksitet. Bruk Θ -notasjonen.

Svar: (5%)

Start i hver ende:

$$x = \begin{array}{cccccc} \boxed{x_1} & \boxed{x_2} & \boxed{x_3} & \cdots & \boxed{x_n} \\ i \Rightarrow & & & & \Leftarrow j \end{array}$$

Flytt i/j et hakk opp/ned for å øke den minste av de to summene (fra venstre og høyre) inntil i og j møtes. Kjøretiden blir $\Theta(n)$.

b. Beskriv en $O(n)$ -algoritme som avgjør om det er mulig å dele tallfølgen x_1, \dots, x_n inn i k intervaller slik at den største intervallsummen er mindre enn en gitt verdi s .

Svar: (5%)

```

sum ← 0
intervall ← 1
for i in [1 .. n]:
    sum ← sum + xi
    if sum > s:
        intervall ← intervall + 1
        sum ← xi
    end
end
return (intervall ≤ k)

```

c. Forklar hvordan algoritmen i b. kan utnyttes til å finne den beste løsningen av problemet $P(x, n, k)$. Angi kompleksiteten til den algoritmen du foreslår.

Svar: (6%)

Kjører binærøket etter beste (dvs. den minste, brukbare) verdien for s i intervallet $[0, \sum_{i=0}^n x_i]$. Kompleksiteten blir

$$O(\lg \sum_{i=0}^n x_i) \cdot \Theta(n) = O(n \cdot \lg \sum_{i=0}^n x_i)$$

d. Beskriv en praktisk situasjon der problemet $P(x, n, k)$ dukker opp.

Svar: (6%)

NB: Viktig at det er snakk om en *sekvens*.

Eks: Man skal dele et sekvensielt arbeide (av typen avisrute) bestående av enheter med ulik arbeidsinnsats (f.eks. antall aviser som skal leveres til en adresse) i k deljobber (dvs. del-ruter) som skal være mest mulig like (største del minst mulig). Det må være mulig å hoppe inn i ruta på ethvert punkt.

OPPGAVE 4 (12%)

Vi har gitt m forskjellige sorterte lister, hver med n elementer. Det (de) største (= beste) elementene befinner seg først i listene.

a. Beskriv kort en effektiv algoritme & datastruktur for å velge ut de n største (= beste) elementene fra de m listene, i sortert orden. Anta at m kan være vilkårlig stor og at alle elementer som ikke blir valgt ut skal forkastes.

Svar: (6%)

Se også 6c.

Vi bruker en haug (heap/delvis ordnet tre). Listene settes inn i haugen med første element som nøkkel. For hvert element vi tar ut må vi sette inn lista med det neste elementet som nøkkel. Hvert element tar altså $O(\lg m)$ tid, og kjøretiden blir $O(n \cdot \lg m)$.

b. Ville du bruke metoden i a. også for “små” verdier av m ? Begrunn, og angi en eventuell alternativ metode.

Svar: (6%)

Hauger er mer komplekse, selvsagt, enn enkel lineær søking. For små m ($m \leq$ ca. 10) ville dette være best, selv om metoden blir $O(n \cdot m)$, siden man da slipper en god del overhead i forbindelse med bygging og oppdatering av haugen.

OPPGAVE 5 (17%)

Vi skal i denne oppgaven se på et praktisk problem som kan formuleres og løses ved hjelp av en dynamisk nettverksstruktur, en rettet graf $G = (V, E)$:

Anta at en mengde personer (f.eks. studenter på ekskursjon) til ulike tider låner ulike pengebeløp til/fra hverandre. En person kan framstilles som en node, og et lån mellom to personer som en kant med retning fra långiver til låntaker og med et tilknyttet beløp. Dette beløpet kan underveis skifte fortegn. G vil vokse etterhvert som nye lån opptas mellom nye personer. En node kan inngå både som långiver og låntaker i forhold til mange øvrige noder.

a. Når vil du opprett/fjerne en node/kant i denne modellen? (Forklar *presist*)

Svar: (4%)

Opprette node: Noden $\notin G$ og noden skal inngå i et lån.

Opprette kant: $(v_i, v_j) \notin E$ og v_j skal låne av v_i .

Fjerne node: Ingen kant knyttet til noden. (Dvs. alle fjernet.)

Fjerne kant: (v_i, v_j) fjernes hvis og bare hvis utestående lånebeløp, $b_{ij} = 0$.

b. Hvordan finner en ut hvor mye en person totalt har til gode/skylder?

Svar: (3%)

Person v_i har til gode:

$$\sum_{(v_i, v_j) \in E} b_{ij} - \sum_{(v_k, v_i) \in E} b_{ki}$$

(der v_i er konstant i begge summene), som kan bli negativt.

c. Hvordan vil du praktisk utføre et “sluttoppgjør” som svarer til at G reduseres til en tom graf?

Svar: (5%)

Alle noder gjør som i b.

De med negativt beløp legger dette i en “pott” (bank).

De med positivt beløp tar dette fra “potten”.

d. Dersom $|V|$ er “liten,” hvilken *statisk* datastruktur kunne passe for å løse oppgaven?

Angi hva du vil regne som “liten $|V|$.”)

Svar: (5%)

En halv to-dimensjonal tabell; långiver \times låntaker. (Trenger bare halvparten, idet vi bruker fortegn.)

OPPGAVE 6, ØVINGSRELATERTE OPPGAVER (25%)

Gjelder øving 1, sortering:

a. Er vektoren $A : [28, 18, 25, 6, 16, 23, 14, 7, 4, 2, 8]$ en haug (heap)?
(Begrunn svaret.)

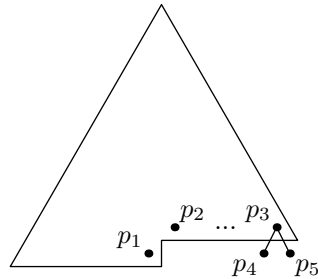
Svar: (2%)

Nei, 7 er større enn verdien i foreldre-noden.

b. På hvilke indekser kan et *unik*t minste element i en haug på 2039 elementer befinne seg?

Svar: (2%)

Den kan befinsse seg “nederst,” dvs. på det laveste nivået i haugen.



I figuren er p_1 siste element i haugen, dvs. element nummer 2039. “Nederst” vil her være det nederste nivået til venstre (samme nivå som p_1) eller samme nivå som p_2 og p_3 , med indeks større eller lik indeksen til p_2 . Elementet kan altså befinsse seg i intervallet $[index(p_2), 2039]$ og problemet blir å finne indeksen til p_2 . I en heapen er det “plass til” to barn for hvert av elementene fra og med p_2 til og med p_3 . Disse elementene vil havne etter p_1 og fylle plassen til og med p_5 . Indeksen til p_5 er $2^{11} - 1 = 2047$. Antallet punkter “til høyre” for p_1 er altså $2047 - 2039 = 8$. De vil ha $\frac{8}{2} = 4$ foreldrenoder. p_3 har index $2^{10} - 1 = 1023$, og følgelig blir indeksen til p_2 1020. Det unikt minste elementet kan altså ha en indeks i intervallet $[1020, 2039]$.

c. Gitt haugen $P : [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$. Her ser vi at element “1” befinsse seg på indeks #10. Anta at P er en prioritetskø, og at det utføres følgende serie av operasjoner på P :

- (1) Heap-Extract-Max(P)
- (2) Heap-Extract-Max(P)
- (3) Heap-Insert(P , 16)
- (4) Heap-Insert(P , 14)

Angi på hvilken indeks i P elementet “1” befinsse seg etter hver av de 4 operasjonene.

Svar: (4%)

Etter 1: 9, etter 2: 6, etter 3: 6, etter 4: 6

Gjelder øving 3, beste vei:

d. Hvilken én-til-alle-beste-vei-algoritme vil du velge i hvert av de 4 tilfellene nedenfor?

Svar: (8%)

- Tilfelle 1: G har kun positive vektorer og G syklisk
 \Rightarrow Dijkstra (B–F dårlig)
- Tilfelle 2: G har kun positive vektorer og G asyklisk
 \Rightarrow DAG shortest path (D og B–F dårlig)
- Tilfelle 3: G har minst én negativ vekt og G syklisk
 \Rightarrow Bellman–Ford
- Tilfelle 4: G har minst én negativ vekt og G asyklisk
 \Rightarrow DAG shortest path (B–F dårlig)

Gjelder øving 5, strengmatching:

e. Sammenlign tidskompleksiteten for de 3 klassiske algoritmene som inngikk i øvingen. Forklar kort grunnen til tidskompleksiteten for hver av algoritmene.

Svar: (3%)

Metode: Naïve string matcher

Tidskompleksitet: $\Theta((n - m + 1)m)$

Alle de m bokstavene i mønsteret P sjekkes for hvert av de $n - m + 1$ posisjonene P kan ha i forhold til T , der $n = length[T]$.

Svar: (3%)

Metode: KMP-Matcher

Tidskompleksitet: $O(m + n)$

COMPUTE-PREFIX-FUNCTION tar $O(m)$ tid, mens KMP-MATCHER tar $O(n)$ tid, og disse kjøres etter hverandre.

Svar: (3%)

Metode: Boyer-Moore-Matcher

Tidskompleksitet: $O((n - m + 1)m + |\Sigma|)$

COMPUTE-LAST-OCCURRENCE-FUNCTION tar $O(m + |\Sigma|)$ tid, COMPUTE-GOOF-SUFFIX-FUNCTION tar $O(m)$ tid, og algoritmen bruker $O(m)$ tid for hvert skift (som KMP-Matcher).