

# Introduktion til algoritmik og datastrukturer

## Vejledende løsninger

IT-højskolen i København

16. januar 2001

Dette eksamenssæt består af 3 opgaver med i alt 13 delopgaver. De 13 delopgaver vægtes ens i bedømmelsen. Du har i alt 4 timer til din rådighed. Husk at angive sidetal, navn og studienummer på alle sider i din besvarelse. Eksamenssættet består af 8 nummererede sider.

CLR refererer til "Introduction to Algorithms" af Cormen, Leiserson og Rivest, 18. tryk, 1997.

I besvarelser, hvor der skal angives effektive algoritmer, lægges der i bedømmelsen vægt på den beskrevne løsnings asymptotiske tidskompleksitet. I spørgsmål, hvor der skal angives tidskompleksitet, skal denne udtrykkes i  $O$ -notation med mindst mulig vækstrate.

# Opgave 1

Denne opgave handler om  $O$ -bestemmelse, binær søgning og amortisering. I opgaven antages det, at tabeller med  $n$  indgange starter med indgang 0 og slutter med indgang  $n - 1$ .

a) Lad  $p1$  være en procedure, der har tidskompleksiteten  $O(n^2)$ , og  $p2$  en procedure, der har tidskompleksiteten  $O(n^3)$ . Lad  $p3$  være en procedure, der først kalder  $p1$  og dernæst kalder  $p2$  og ellers intet gør. Angiv tidskompleksiteten for  $p3$ .

Svar a)  $O(n^3)$

Betragt nedenstående procedurer:

F1( $n$ )

1.  $i \leftarrow 1$

F2( $n$ )

1. **while**  $n > 1$

2.       **do**  $n \leftarrow n - 1$

F3( $n$ )

1. **while**  $n > 1$

2.       **do**  $n \leftarrow \lfloor n/2 \rfloor$

F4( $n$ )

1. **while**  $n > 1$

2.       **do for**  $i \leftarrow 1$  **to**  $n$

3.               **do**  $j \leftarrow 1$

4.                $n \leftarrow n - 1$

b) Antag at ovenstående procedure kun bliver kaldt med et heltal  $n > 0$ . For hver af de fire ovenstående procedurer skal du angive tidskompleksiteten udtrykt i  $n$ .

Svar b)

F1:  $O(1)$ .

F2:  $O(n)$ .

F3:  $O(\lg n)$ .

F4:  $O(n^2)$ .

Lad  $S$  være en mængde af  $n$  forskellige heltal. Lad tallene fra  $S$  være repræsenteret i en tabel  $A$  med  $n$  indgange. Antag at tallene er repræsenteret sorteret i  $A$ , det vil sige at  $A[i] < A[j]$ , hvis  $0 \leq i < j \leq n - 1$ . Lad  $\text{COUNT}(x, y)$  være en procedure, der som argumenter har to heltal  $x$  og  $y$ , hvor  $x < y$ . Procedure  $\text{COUNT}(x, y)$  skal returnere antallet af tal fra  $S$ , der er både større end  $x$  og mindre end  $y$ . Eksempel: Hvis  $A = \langle 5, 10, 33, 49, 66, 75, 82, 95, 100 \rangle$ , så returnerer  $\text{COUNT}(47, 85)$  tallet 4.

c) Beskriv hvorledes  $\text{COUNT}(x, y)$  kan implementeres, så den har kompleksiteten  $O(\log n)$ .

Svar c) Vi kan lave en binær søgningen efter det mindste tal større end  $x$  og det største tal mindre end  $y$ . Dermed finder vi indices  $i$  og  $j$ , hvor  $0 \leq i \leq j \leq n$ . Antallet af tal mellem  $x$  og  $y$  er da  $j - i$ .

Lad  $B$  være en tabel bestående af  $n$  tal. Til at starte med er  $B$  initialiseret således, at alle tallene i tabellen er 0 eller 1. Altså  $B[i] = 0$  eller  $B[i] = 1$  for enhver indgang  $i$ . Betragt nu følgende procedure :

AMOR1( $i$ )

1. **if**  $0 \leq i \leq n - 1$  and  $B[i] = 1$
2.     **then**  $B[i] = 0$
3.         AMOR( $i + 1$ )

Eksempel: Lad  $B = \langle 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1 \rangle$ . Det vil sige at  $B[0] = 1$ ,  $B[1] = 0$ ,  $B[2] = 1$ ,  $B[3] = 1$ ,  $B[4] = 0$ . Efter et kald af AMOR1(2) vil  $B = \langle 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1 \rangle$ .

**d)** Antag at  $B$  er initialiseret som beskrevet ovenfor, og  $B$  herefter alene opdateres via en række kald til AMOR1. Hvad er kompleksiteten af et sådan kald af AMOR1? Hvad er kompleksiteten af  $n$  sådanne kald af AMOR1? Begge svar skal begrundes.

**Svar d)** Et enkelt kald til Amor1 kan tage højst  $O(n)$  tid hvis alle indgange i  $B$  er 1.  $n$  kald til Amor1 kan tage højst  $O(n)$  tid da hver indgang i  $B$  højst ændres en gang.

Lad  $C$  være en tabel bestående af  $n$  tal. Til at starte med er  $C$  initialiseret således, at alle tallene i tabellen er 1. Altså  $C[i] = 1$  for enhver indgang  $i$ . Betragt nu følgende procedure:

AMOR2( $i, j$ )

1. **if**  $0 \leq i \leq n - 1$  and  $0 \leq j \leq n - 1$  and  $C[i] \geq C[j] > 0$
2.     **then for**  $k \leftarrow 1$  **to**  $C[j]$
3.         **do**  $C[i] \leftarrow C[i] + 1$
4.      $C[j] \leftarrow 0$

Eksempel: Antag at der har været en række kald af AMOR2 og  $C = \langle x, y, 0, z, 0, 0, w \rangle$ , hvor  $x, y, z, w > 0$ . Det vil sige, at  $C[0] = x$  og  $C[1] = y$ . Antag  $x > y$ . Efter et kald af AMOR2(0, 1) vil  $C = \langle v, 0, 0, z, 0, 0, w \rangle$ , hvor  $v = x + y$ . Det vil sige, at det mindste tal  $y$  er blevet lagt til det største tal  $x$ , hvilket har taget tiden  $O(y)$ .

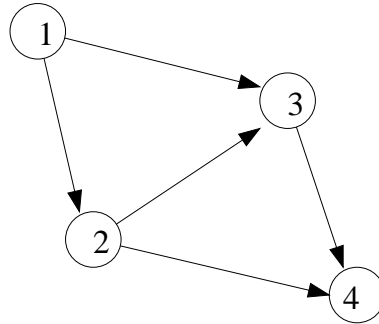
**e)** Antag at  $C$  er initialiseret som beskrevet ovenfor, og  $C$  herefter alene opdateres via en række kald til AMOR2. Hvad er kompleksiteten af et sådan kald af AMOR2? Hvad er kompleksiteten af  $n$  sådanne kald af AMOR2? Begge svar skal begrundes.

**Svar e)** Et enkelt kald til Amor2( $i, j$ ) tager højst  $O(C[j]) = O(n)$  tid. Betragt den  $i$ te indgang i  $C$  som en kasse med  $C[i]$  elementer. Amor2( $i, j$ ) kombinerer da kasse  $i$  og  $j$  ved at flytte indholdet af den mindste kasse  $i$  over i den største kasse  $j$ . Den nye kasse er nu mindst dobbelt så stor som den mindste kasse. Følgelig flyttes et element højst  $\lg n$  gange. Kompleksiteten af  $n$  kald til Amor2 er derfor højst  $O(n \lg n)$ .

## Opgave 2

Denne opgave handler om orienterede, vægtede grafer og algoritmer for at finde korteste veje i sådanne grafer.

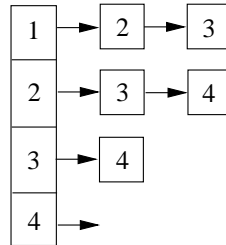
For alle delopgaver antages, at input-grafen er givet som en adjacency-list repræsentation, jvf. CLR side 465-466. Endvidere antages det at kanterne har positive heltalsvægte. I resten af opgaven lader vi  $n$  betegne antallet af knuder i input-grafen og  $m$  antallet af kanter.



Figur 1: Graf med vægte.

a) Angiv adjacency-list repræsentationen for grafen i figur 1.

Svar a) Se figur 2.



Figur 2: Adjacency-liste repræsentation af figur 1

b) Lad  $s$  og  $t$  være to knuder i en orienteret graf  $G$ . Angiv en effektiv algoritme for at finde den korteste vej fra en given knude  $s$  til en given knude  $t$  i  $G$ . Angiv også tidskompleksiteten for algoritmen, under antagelse af at  $m$  er  $O(n)$ .

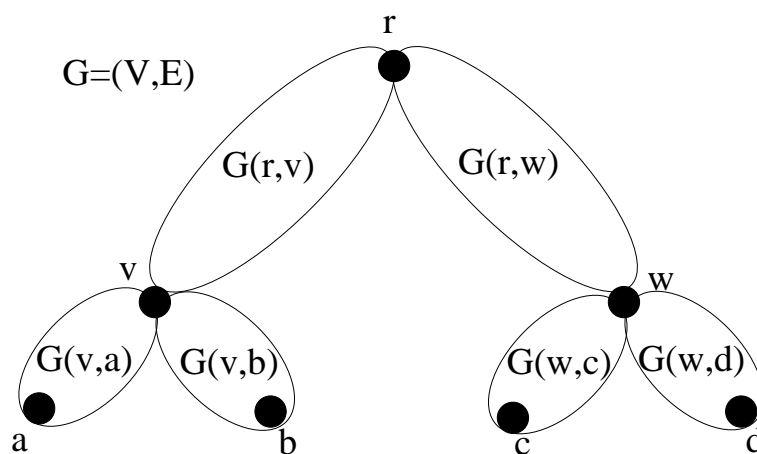
Svar b) Vi kan anvende Dijkstras algoritme. Tidskompleksiteten er  $O(m \lg n) = O(n \lg n)$ .

c) Beskriv en algoritme, der finder længden af den korteste vej fra  $s$  til  $t$ , der går via en given knude  $w$ . Det vil sige finder længden af den korteste vej fra  $s$  til  $t$ , hvor  $w$  ligger på vejen. Angiv tidskompleksiteten for din løsning.

**Svar c)** Find den korteste vej fra  $s$  til  $w$  og fra  $w$  til  $t$  og læg længderne sammen. Dette kan gøres med to kald til Dijkstra og tager derfor  $O(n \lg n)$ .

Betragt nu et rodfæstet, binært træ  $T$  med roden  $r$ . For en kant  $(u, v)$  i træet  $T$  er  $v$  barn til  $u$ . Til hver kant  $(u, v)$ , i træet  $T$ , er der associeret en orienteret graf  $G(u, v)$ . Disse grafer udgør komponenterne i en samlet stor graf  $G = (V, E)$  med  $n$  knuder, som vi kalder en komponent graf. I hver delgraf  $G(u, v)$  er der to specielle knuder  $s(u), t(v) \in V$ , der kan være fælles med de øvrige delgrafer. Vi antager, at der altid er en orienteret vej fra  $s(u)$  til  $t(v)$  i  $G(u, v)$ . Der gælder for hvert par af kanter  $(u, v)$  og  $(v, w)$  i  $T$ , at knuden  $t(v)$  er lig med knuden  $s(w)$  for henholdsvis  $G(u, v)$  og  $G(v, w)$ . Ingen øvrige knuder i delgraferne kan være fælles med andre delgrafer.

Antag at man i adjacency-list repræsentation for  $G$  også har information om, i hvilken delgraf en grafkant ligger.



Figur 3: Eksempel på komponentgraf

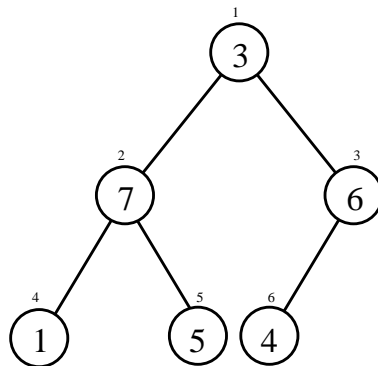
**d)** Antag at alle delgraferne  $G(u, v)$  har højst  $O(\log n)$  knuder og kanter. Beskriv en algoritme, der finder den korteste vej fra  $s(r)$  til alle øvrige knuder i  $G$  i tid  $O(n \log \log n)$ .

**Svar d)** For hver komponent  $G(u, v)$  beregner vi korteste veje fra  $s(u)$  til alle andre knuder i  $G(u, v)$ . Komponenterne beregnes oppefra og ned så korteste veje i  $G$  beregnes korrekt. Antallet af komponenter er  $O(n/\log n)$  og antallet af knuder i hver komponent er  $O(\log n)$ . Følgelig er tidskompleksiteten  $O(\frac{n}{\log n}(\log n \cdot \log \log n)) = O(n \log \log n)$ .

## Opgave 3

a) Illustrer udførelsen af  $\text{HEAPIFY}(A, 1)$  for hoben  $A$  i figur 4. Anvend stilen fra figur 7.2 i CLR side 143.

Svar a) Først ombyttes 7 med 3. Dernæst ombyttes 3 med 5 og algoritmen stopper.



Figur 4: Hoben  $A$ , der skal udføres  $\text{HEAPIFY}$  på.

I det lille land Algostan er det blevet besluttet, at der skal laves et register-system kaldet **Amandos**. I den første version af systemet skal registret kun kunne håndtere følgende tre operationer:

$\text{INIT}(n)$ : Initialiserer registret, så det kan håndtere et maksimum på  $n$  personer i registret.

$\text{INDSÆT}(CPRnr, Indkomst)$ : Indsætter en person med cpr-nummer  $CPRnr$  i registret. Personen skal tilknyttes en oplysning om årlige indkomst, der er på kr.  $Indkomst$ . Det antages, at registret ikke allerede indeholder en person med cpr-nummer  $CPRnr$  i forvejen ved et sådant kald.

$\text{SLET LAVESTE INDKOMST}()$ : Returnerer CPR-nummeret for en person, der har den laveste indkomst, samt sletter denne person fra registret.

b) Beskriv hvordan første version af **Amandos** kan laves, så  $\text{INIT}(n)$  tager tiden  $O(n)$ , og så de to andre operationer tager tiden  $O(\log n)$ , hvor  $n$  er det maksimale antal af personer, der kan være i registret.

**Svar b)** Vi kan anvende et rød-sort træ  $T$  ordnet over CPR-nr. Lad  $s$  betegne det maksimale antal elementer og  $r$  være antallet af elementer i  $T$ .

INIT( $n$ ): Sæt  $s \leftarrow n$  og  $r \leftarrow 0$

INDSÆT( $CPRnr, Indkomst$ ): Hvis  $r + 1 > s$  udskriv fejlmeddelelse. Ellers sæt ind i  $T$  og sæt  $r \leftarrow r + 1$ .

SLET LAVESTE INDKOMST(): Finder det mindste element og sletter det. Sæt  $r \leftarrow r - 1$ .

Alle operationerne tager  $O(\log n)$  tid.

Næste version af systemet skal kunne lidt mere, nemlig understøtte operationen:

SØG( $CPRnr$ ): Afgør om en person med cpr-nummer  $CPRnr$  er i registret. Hvis der er en person med dette cpr-nummer, skal personens årlige indkomst returneres.

**c)** Beskriv hvordan registret kan udvides til at understøtte ovennævnte SØG operation i tiden  $O(\log n)$ , således at tiderne fra b) bibeholdes.

**Svar c)** SØG( $CPRnr$ ): Udfør en søgning i  $T$  med TREE-SEARCH som defineret i CLR.

I den sidste version af systemet vil man også gerne have følgende operation

INDKOMSTMELLEM( $a, b$ ): Returnerer true, hvis der findes en person i registret med en årlig indkomst imellem kr.  $a$  og kr  $b$ . D.v.s. med en indkomst  $x$ , hvor  $a \leq x \leq b$ . I modsat fald returneres false.

**d)** Beskriv hvordan registret kan udvides til også at understøtte operationen INDKOMSTMELLEM i tiden  $O(\log n)$ , således at tiderne fra c) bibeholdes.

**Svar d)** INDKOMSTMELLEM( $a, b$ ): Beregn følgende

$a_u \leftarrow \text{TREE-SUCCESSOR}(a)$

$b_l \leftarrow \text{TREE-PREDECESSOR}(b)$

Returner true hvis  $a \leq a_u \leq b$  eller  $a \leq b_l \leq b$ . Ellers returner FALSE