Høgskolen i Østfold

EKSAMEN
Løsningsforslag
–
med forbehold om bugs :-)

Emnekode: ITF20006 000	Emne: Algoritmer og datastrukturer
Dato: 20. mai 2011	Eksamenstid: 09:00 til 13:00
Hjelpemidler: 8 A4-sider (4 ark) med egne notater	Faglærer: Gunnar Misund

Oppgavesettet består av 4 sider inklusive denne forsiden. Kontroller at oppgaven er komplett før du begynner å besvare spørsmålene.

I implementasjonsoppgaver gir Java-kode best uttelling, men godt dokumentert pseudo-kode kan også brukes. Gjør dine egne forutsetninger hvis du føler behov for det.

Oppgavesettet består av 4 deler, med tilsammen 13 deloppgaver. Alle oppgavene skal besvares. For hver deloppgave er det angitt hvor mye oppgaven teller ved sensur. Karakteren fastsettes dog på basis av en helhetsvurdering av besvarelsen.

SKRIV TYDELIG :-)

Sensurdato: 16. juni 2011

Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist. Følg instruksjoner gitt på: www.hiof.no/studentweb

Oppgave 1: 20%

Her følger fem Java-funksjoner. For hver av de skal du forklare hva de gjør, samt angi funksjonenes orden i O-notasjon, inkludert begrunnelse. Hver algoritme teller 4%.

```
static void func1(int[] arr, int x) {
    for (int i = arr.length-1; i >= 0; i--) {
        if (arr[i] == x) {
            System.out.println(i);
        }
    }
}
```

Løkka går gjennom hele tabellen baklengs, og skriver ut indexen hvis den finner verdien x. Dette gir lineær orden O(N).

```
static void func2(int[] arr) {
    int s = arr.length / 2;
    for (int i = s; i < arr.length; i += 2) {
        System.out.println(arr[i]);
    }
}
```

Løkka går starter halveis i tabellen, går gjennom annet hvert element i siste halvdel og skriver ut innholdet. Dette gir lineær orden O(N).

```
static void func3(int[] arr, int i1, int i2) {
    int t = arr[i1];
    arr[i1] = arr[i2];
    arr[i2] = t;
}
```

Funksjonen bytter innholdet i element nr i1 og 12. Dette gir konstant orden, O(1).

```
static void func4(int[] arr) {
```

```
        for (int n = 0; n < arr.length - 1; n++) {
            int im = n;
            for (int m = n + 1; m < arr.length; m++) {
                if (arr[m] < arr[im]) {
                    im = m;
                }
            }
            func3(arr, im, n);
        }
    }
```

Dette er utvalgssortering. Første gang går den indre løkka n-1 ganger, neste gang går den n-2 ganger, etc. Byttingen er av konstant orden. Den indre løkka går et antall som tilsvarer summen av de N-1 første heltallene...og det vet vi gir kvadratis orden, O(N^2).

```
      static void func5(int[] arr, int f, int t, int k) {
        if (f < t) {
            int m = f + (t - f) / 2;
            if (k < arr[m]) {
                func5(arr, f, m, k);

            } else if (k > arr[m]) {
                func5(arr, m + 1, t, k);

            } else {
                System.out.println(m);
            }
        }
    }
```

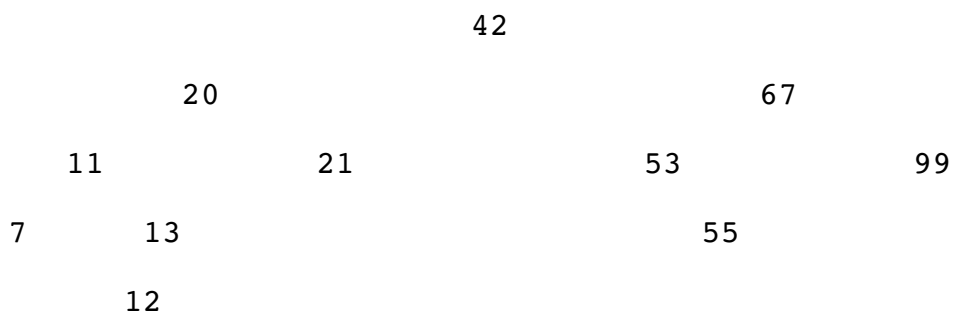
Dette er et rekursivt binærsøk. Størrelsen på input halveres hver gang, og dette gir logaritmisk orden, O(logN).

Oppgave 2: 30%

Her skal du jobbe med binære søketrær (hver deloppgave teller 10%).

A) *Bygg opp et binært søketre fra denne sekvensen:*

`42 20 11 21 13 67 53 99 7 12 55`



Implementer funksjonen `void insert (Node n, Node root)`*, som setter inn noden* `n` *i det binære søketreet med rot i* `root`*. Se vedlegg for definisjonen av* `class Node`*.*

```
void insert(Node n, Node root) {
    if (root == null) return;
    if (n.val <= root.val) {
        if (root.left == null) root.left = n;
        else  insert (n, root.left);
    }
    else {
        if (root.right == null)  root.right = n;
        else insert (n, root.right);
    }
}
```

B) *Gjør grundig rede for hvordan man sletter en node fra et binært søketre. Ta deretter utgangspunkt i treet som er bygget opp i A).*

Man starter ved å rekursivt (eller iterativt) finne noden man skal slette. Det er tre tilfeller:

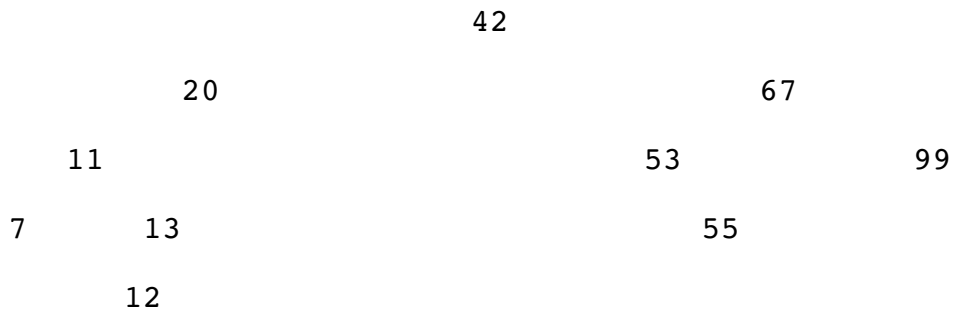
1) Noden har ikke barn: Man setter foreldren sin referanse til denne noden til null («klipper» av bladnoden).

2) Noden har ett barn: Man setter foreldren sin referanse til å peke på barnet («bypass»)

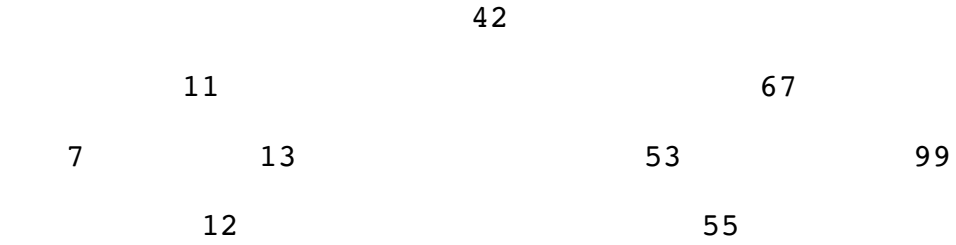
3) Noden har to barn: Man finner enten noden med den største verdien i venstre subtre, eller den med minst verdi i høyre subtre. Bytt verdiene i noden som skal slettes med verdien i en av disse nodene. Bruk tilfelle 1) eller 2) og slett noden som man har hentet den nye verdien fra.

Slett først noden 21, deretter 20, og til slutt 42. Illustrer hvordan du går fram.

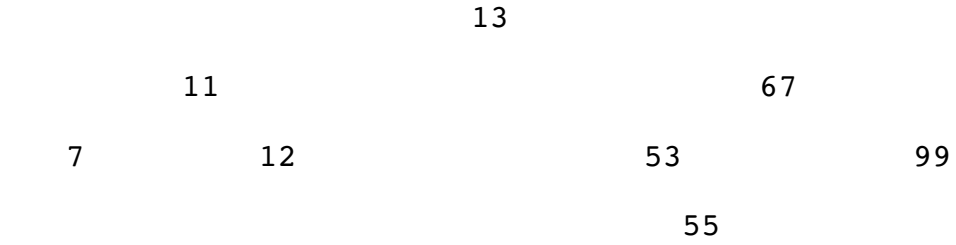
Slette 21: Klippe av noden.



Slette 20: Sette 42 sitt venstre barn til 11 (bypass).



Slette 42: Bytte verdi med f.eks. største i venstre subtre, 13, og deretter fjerne 13 ved bypass..



C) *Implementer metoden* `void deleteSmallest (Node root)`*, som fjerner noden med minst verdi i det binære søketreet med rot i* `root`*.*

```
  static void deleteSmallest(Node root) {
    if (root == null) return;
    if (root.left == null) {
        root = root.right;
        return;
    }
    Node curr = root;
    while(curr.left.left != null) {
        curr = curr.left;
    }
    curr.left = curr.left.right;
}
```

Oppgave 3: 20%

Denne oppgaven dreier seg om standard køer (FiFo: First in, First out), slike man f.eks. finner foran kassa i kantina (deloppgavene teller likt).

1. *Forklar hvordan du kan bruke en enkeltlenket liste (hver node har en referanse til neste node) til å implementere en FiFo kø. Angi i pseudokode hvordan man setter inn og tar ut et element i køen.*

   Vi lager en klasse FiFo klasse som i hovedsak inneholder en referanse til til første element i køen, first, som også er første element i lista, og en referanse til siste element i køen, som også er siste node i lista.

   Det ble bedt om pseudokode, men her er eksakt kode:

```
static void insert(Node first, Node last, Node n) {
    if (first == null) {
        first = last = n;
    } else {
        last.next = n;
        last = n;
    }
}
```

```
    static Node remove(Node first, Node last) {
      if (first == null) return null;

      Node rem = first;
      if (first == last) first = last = null;
      else first = first.next;

      return rem;
    }
```

2. *Anta at vi har en situasjon der vi vet at køen aldri vil overstige et visst antall elementer. Forklar hvordan vi kan bruke en tabell for å lage en* effektiv *implementasjon av en slik kø. Angi i pseudokode hvordan man setter inn og tar ut et element i denne køen.*

   Vi bruker en såkalt sirkulær tabell, der den neste plassen etter index length-1 er element nr 0. Vi har to integer variable first og last, som peker på elementet som tilsvarer henholdsvis første og siste i køen. Sette inn i køen tilsvarer å legg inn verdien i indeks nr last+1, og øke last med en. Ta ut er å ta vare på og returnere verdien i indeks first, og øke first med 1.

   Det ble bedt om pseudokode, men her følger eksakt Javakode:

```
public class ArrQueue {

    int[] arr;

    int first, last;

    int size;

    public ArrQueue(int n) {

        arr = new int[n];

        size = 0;

        first = last = 0;

    }
```

```
    public int deQueue() {
        if (size == 0) {
            return -1;
        }
        int f = arr[first];

        if (first == arr.length - 1) {
            first = 0;
        } else {
            first++;
        }
        size--;
        return f;
    }
    public void enQueue(int i) {
        if (size == 0) {
            arr[0] = i;
            first = last = 0;
            size++;
            return;
        }
        if (last == arr.length - 1) {
            last = 0;
        } else {
            last++;
        }
        arr[last] = i;
        size++;
    }
}
```

Oppgave 4: 30%

Nettverket du skal jobbe med i denne oppgaven er gitt som en liste av noder, der hver node har en liste av sine naboer, og avstanden til de:

```
A: B-8 D-12 F-42 C-14 (Node A har som nabo B, med avstand 8, etc.)
B: D-3
C: E-7
D: A-12 F-20
E: F-1
F: D-20 E-1
```

A) Tegn dette nettverket. Angi nodenes navn, retningen på kantene, samt avstandene.

Trivielt :)

B) Angi pseudokoden for en såkalt bredde-først traversering av et nettverk. Foreta en slik traversering av nettverket med start i A, og angi rekkefølgen på nodene som denne traverseringen resulterer i.

Bredde-først

Input: Node Start

1. Oppretter en kø Kø

2. Marker at Start er besøkt, og legger den inn i Kø

3. Så lenge Kø ikke er tom

    3.1 Ta ut første node F og skriv den ut

    3.2 For alle naboer N til F som ikke er besøkt

3.2.1 Marker N som besøkt og legg den inn i Kø

Rekkefølge: A B D F C E

C) Finn de korteste veiene fra node A til alle de andre nodene ved hjelp av Dijkstras algoritme. Lag en oversiktlig tabell som viser hvert trinn i algoritmen. Angi til slutt de korteste veiene fra A til de andre nodene.

Ferdigbehandlede noder er streket under. Oppdaterte noder (der veien er blir kortere enn før) er uthevet med fet skrift.

	a	b	c	d	e	f	Kommentar
Init	a:0	-:∞	-:∞	-:∞	-:∞	-:∞	Avstanden fra a til a settes til 0. De andre avstandene settes til uendelig, og deres "via"-noden settes til "-"
1	a:0	**a:8**	**a:14**	**a:12**		**a:42**	Vi velger noden med kortest avstandsverdi, dvs. a. Dernest oppdaterer vi avstanden til a ("via a") i alle nabonodene som ikke er ferdigbehandlet, dvs,. b, c, d og f. Noden a er etter dette ferdig behandlet, og vi markerer med å sette strek under den.
2		a:8		**b:11**			Vi velger node b (kortest vei). Vi ser at det er kortere å gå til B sin nabo D via B enn direkte fra A, og oppdaterer.
3				b:11		**d:31**	Vi velger node d (korteste vei til nå), og oppdaterer naboen F (A er allerde ferdig). Denne sin avstand blir avstanden til D (11) pluss avstanden D-F (20), som blir 31, som er mindre enn tidligere avstand 42.
4			a:14		**c:21**		Vi tar ut node c, og oppdaterer nabeoen E
5					c:21	**e:22**	Neste node ut (kortest avstand), er E. Avstanden fra A til naboen F blir avstanden fra A-E (21) pluss avstanden E-F (1), totalt 22, som er kortere enn 31, og vi oppdaterer F.
6						e:22	Det er nå kun en node som ikke er ferdigbehandlet, b, og den har jo da ingen naboer som ikke er ferdigbehandlet, og vi er framme!

Følgende korteste veier er funnet:

- a: 0
- b: 18(via a)
- c: 14 (via a)
- d: 11 (via b)
- e: 21 (via c)
- f: 22 (via e og c)

Vedlegg

```
public class Node {
	public int val;
	public Node left;
	public Node right;
```

```
}
```

Slutt på oppgavesettet.