Høgskolen i Østfold

Løsningsforslag
EKSAMEN

Emnekode: ITF20006 000	Emne: Algoritmer og datastrukturer
Dato: 18. mai 2012	Eksamenstid: 09:00 til 13:00
Hjelpemidler: 8 A4-sider (4 ark) med egne notater	Faglærer: Gunnar Misund

Oppgavesettet består av 6 sider inklusive denne forsiden samt vedlegg. 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 hovedoppgaver, med tilsammen 13 deloppgaver. Alle oppgavene skal besvares. For hver hovedoppgave er det angitt hvor mye den teller ved sensur (deloppgavene i en hovedoppgave er likt vektet). Karakteren fastsettes dog på basis av en helhetsvurdering av besvarelsen.

RÅD:
SKRIV TYDELIG OG HARDT
DISPONER TIDA
IKKE LEVER BLANKE OPPGAVER
TA PAUSER
DON'T PANICK
:-)

Sensurdato: 11. juni 2012
Karakterene er tilgjengelige for studenter på studentweb senest 2 virkedager etter oppgitt sensurfrist. Følg instruksjoner gitt på: www.hiof.no/studentweb

Oppgave 1: 30%

A) Betrakt følgende matematiske funksjon:

$f(x) = 1/x^2 + 42 + 15\log_3(x)$

Hva er funksjonens orden i O-notasjon? Begrunn svaret.

O(logx)...O($\log_3$x) vil også godkjennes (det samme gjelder hvis n brukes i stedet for x. $15\log_3(x)$ er leddet med høyest orden, derfor stryker vi de andre leddene, og vi stryker også konstanten 15. Envher logaritmefunksjon, uansett base, er av samme orden.

B) Betrakt følgende javafunksjon:

```
int func (int n) {
   int i = 0;
     while(n > 0) {
          n = n/3;
          i++;
     }
   return i-1;
}
```

Hva er funksjonens orden i O-notasjon? Begrunn svaret.

O(logn)...O($\log_3$n) vil også godkjennes. N deles på 3, deretter deler vi igjen på 3, helt til vi står igjen med 0 (heltallsdivisjon)...og antallet divisjoner (minus 1) returneres...og dette er jo nettopp definisjonen av logaritmen med base 3. Og envher logaritmefunksjon, uansett base, er av samme orden.

C) Forklar hvordan du kan bruke en enkelt-lenket liste for å implementere en effektiv kø, dvs. der både det å ta ut og å sette inn et element er av konstant orden.

I tillegg til en referanse til første node i lista (head), oppretter vi en referanse til den siste noden i lista (tail). Vi setter inn et element (node) etter den siste i lista (tail.next = node; tail = node), og elementet som skal ut av køen er det første i lista (head = head.next...her gjør det seg typisk med en illustrasjon).

D) Du jobber som programmerer og har levert en implementasjon av en sorteringsalgoritme som er av kvadratisk orden, $O(N^2)$. Kunden klager, og forteller at algoritmen bruker 10 sekunder på et datasett som må kunne sorteres på maks. 5 sekunder. Du råder kunden til å bruke en datamaskin med minst dobbelt så rask

prosessor, for da vil sorteringene gå minst dobbelt så fort, uansett antallet elementer som skal sorteres.

Vil dette rådet løse kundens problem? Gi et begrunnet svar.

Tja. Det er riktig at sorteringene vil gå dobbelt så fort. Hvis kjøretida for denne algoritmen på den trege maskinen er T(N), vil kjøretida på en dobbelt så rask maskin være 0.5*T(N)...altså en forskjell på en konstant, men vi vet jo at etterhvert som N øker, vil en funksjon av kvadratisk orden ikke være brukbar i praksis. Dermed er vel dette et dårlig råd...du burde heller gitt kunden en loglineær funksjon :)

E) Studer javafunksjonen `List<int[]> foo(int n)` i vedlegget.

Hva gjør denne funksjonen?

Den generer alle mulig permutasjoner av tallene 0 til n-1. OBS: Selv om ikke vi har et return statement når p == n, vil rekursjonen stoppe fordi alle flaggene er satt til true...

F) *Hvor mange heltallstabeller vil det være i listen som returneres fra funksjonen* `List<int[]> foo(int n)` *i vedlegget? Begrunn svaret.*

N!, fordi dette nettopp er antallet permutasjoner av N elementer.

Oppgave 2: 20%

A) Studer javafunksjonen `void bar(char[] arr)` i vedlegget.

Hva blir utskriften fra denne funksjonen med tabellen

```
D U H A R G O D H U D
```

som input?

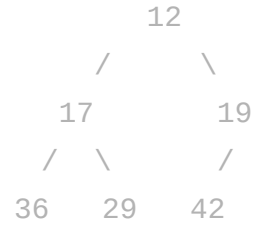
G!!HUD...her gjør det seg med en illustrasjon eller forklaring...

B) En stack kan implementeres ved hjelp av en tabell (array). Forklar hvordan dette kan gjøres, og drøft fordeler og ulemper med denne implementasjonsmåten.

Det mest nærliggende er å allokere en tabell av en viss størrelse, og ha en variabel f.eks. top som sier hvilken indeks som er «toppen» av stacken...og vedlikeholde denne ved hver push og pop. Når tabellen blir full, og vi skal legge inn et element til, så må vi allokere en større tabell og kopiere over. Dermed er push-operasjonen av lineær orden (worst case), og pop er av konstant orden. Dette er greit hvis f.eks. vi vet at stacken sjelden overgår en gitt størrelse. Men generelt sett, og særlig der antallet elementer varierer mye, ville det lønt seg med en listeimplementasjon.

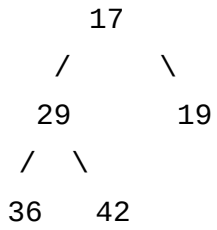
Oppgave 3: 30%

Gitt følgende MinHeap (binærheap der rota er noden med minst verdi):



A) *Foreta en «removeMin» operasjon, som tar ut den minste verdien i Minheap'en. Forklar framgangsmåten, og illustrer hvert trinn.*

Resultat under (hvert trinn skal illustreres!). Fremgangsmåten er delvis forklart i vedlegget...ta vare på verdien i rota, kopier verdien i den siste noden over i rota, slett den siste noden, og reparer heapen. Dvs start i rota, sjekk om rota er større enn sitt minste barn, i så tilfelle, bytt om disse verdien, gå ned til noden som er byttet om, sjekk barna etc



B) *Implementer metoden* `int deleteLast (TreeNode root)`*, som finner den siste noden i binærheapen med rota* `root`*, sletter denne fra treet, og returner verdien i noden som slettes. «Siste node» er den siste noden som vil bli besøkt i en level-order traversering. (Hint: Hvis vi tenker en level-order rekkefølge, er foreldren til den siste noden forløperen til den første noden som har ingen barn, eller noden som bare har ett barn).*

```
public static int deleteLast(TreeNode root) {

   int val = 0;
   if (root == null) {
       // Feilhåndtering av ett eller annet slag
       return val;
   }
   if (root.left == null) {
       // Dette vil si at vi har bare en node i treet
       // Dette er et spesialtilfelle som må håndteres
       // før denne metoden kalles
       // Feilhåndtering av ett eller annet slag
       return root.val;
   }
```

```
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.add(root);
    TreeNode prev = null;
    while (!q.isEmpty()){
        TreeNode curr = q.remove();

        if (curr.right == null && curr.left != null) {
            val = curr.left.val;
            curr.left = null;
            break;
        }

        if (curr.left == null) {
            val = prev.right.val;
            prev.right = null;
            break;
        }

        if (curr.left != null) {
            q.add(curr.left);
        }
        if (curr.right != null) {
            q.add(curr.right);
        }
        prev = curr;
    }


    return val;
}
```

C) *Implementer metoden* `int removeMin (TreeNode root)`, *som tar ut elementet med minst verdi i en MinHeap representert ved* `root`.

Husk at metoden i forrige oppgave kan brukes selv om du ikke fikk den til!

```
public static int removeMin(TreeNode root) {
   if (root == null) {
       // Feilhåndtering av ett eller annet slag
       return 0;
   };
   if (root.left == null) {
       // Dette er et spesialtilfelle som må håndteres før
       // vi kaller denne funksjonen
       // Feilhåndtering av ett eller annet slag
       return 0;
   }
   int val = root.val;
   root.val = deleteLast(root);
   TreeNode curr = root;
   while (curr.left != null){
       TreeNode smallest;
       if (curr.right== null) {
           smallest = curr.left;
       }
       else {
```

```
            smallest = curr.left.val < curr.right.val ?
                       curr.left : curr.right;
        }
        if (smallest > curr.val) {
            return val;
        }
        int tmp = curr.val;
        curr.val = smallest.val;
        smallest.val = tmp;
        curr = smallest;
    }
    return val;
}
```

Oppgave 4: 20%

Et av de enkleste systemene for koding av meldinger er å «bytte om» på tegnene, slik at teksten blir uforståelig. Vi skal nå se på et svært forenklet tilfelle, der vi har gitt en kodemelding som består av unike (ikke gjentatte) tegn, og en kodenøkkel i form av en tabell med tall som gir plasseringene av tegnene i de deschiffrerte meldingen. Her er et eksempel (vi ser på meldingene og nøkkelen som indekserte tabeller med første indeks lik 0):

```
Kodemelding:
G O R M E T A I L

Nøkkel:
6 8 0 1 2 7 5 3 4
Dette betyr at på indeks 0 skal vi hente tegnet fra indeks 6 i
meldingen, på indeks 1 skal vi ha tegnet fra indeks 8, etc.

Løsning:
A L G O R I T M E
```

A) *Imlementer metoden* `String crack (char[] message, int[] key)`, *som løser en gitt kodemelding basert på nøkkelen. Du kan anta at de to tabellene er av lik lengde, og at nøkkelen inneholder gyldige indekser (se vedlegg).*

```
public static String crack(char[] message, int[] key) {

   String s = «»;
   for (int i = 0; i < key.length; i++) {
       s += message[key[i]];
   }

   return s;
}
```

B) Hvis man ikke har tilgang til nøkkelen, kan man prøve seg fram ved å lage en rekke mer eller mindre tilfeldige nøkler, og så se om de gir et fornuftig resultat, som man da kan ta som utgangspunkt for å lage en komplett kodenøkkel.

Implementer metoden `void guess(char[] message)`. *Funksjonen skriver ut alle mulige strenger som inneholder de samme tegnene som meldingen, men i ulike rekkefølger (hvert tegn skal forekomme en og bare en gang). Du kan anta at* `message` *ikke inneholder duplikater (se vedlegg).*

Her blir mye gratis hvis man bruker foo :))

```
public static void guess(char[] message) {

   List<int[]> keys = foo(message.length);
   for (int[] key : keys) {
       System.out.println(crack(message, key));
   }

}
```

Vedlegg

Her følger kodebiter som dere skal jobbe i oppgavene. OBS: Hvis dere finner det hensiktsmessig, kan dere bruke funksjoner herfra i implementasjonsoppgavene, selv om dere ikke har klart å implementere de.

```
List<int[]> foo (int n) {
	// Driverfunksjon for fooRec
	// Skal behandles i 1E og 1F
	List<int[]> list = new ArrayList<int[]>();
	int[] comb = new int[n];
	boolean[] flag = new boolean[n];
	fooRec(comb, flag, 0, list);
	return list;
}


void fooRec(int[] comb, boolean[] flag, int p, List<int[]> list) {
	// Oppgave 1E og 1F
	if (p == comb.length) {
		int[] newcomb = comb.clone();
		list.add(newcomb);
	}

	for (int i = 0; i < comb.length; i++) {
		if (!flag[i]) {
			comb[p] = i;
			flag[i] = true;
			fooRec(comb, flag, p+1, list);
			flag[i] = false;
		}
	}
}


void bar (char[] arr) {
	// Oppgave 2A
	Stack stack = new Stack();
	int j =  arr.length-1;
	for (int i = 0; i <= j; i++) {
		if(arr[i] == arr[j]) {
		stack.push(arr[i]);
		} else {
			stack.push('!');
		}
		j--;
	}
	while(!stack.empty()) {
		System.out.print(stack.pop());
	}
}


class TreeNode {
	// Brukes i 3B og 3C
	int val;
	TreeNode left;
	TreeNode right;
}
```

```
int deleteLast (TreeNode root) {
      // Implementeres i 3B
      // Sletter den siste noden i minHeapen representert ved root
      // og returnerer verdien til denne
      // Den siste noden i en binærheap er
      // den siste i en leverorder rekkefølge
}


int removeMin (TreeNode root) {
      // Implementeres i 3C
      // Returnerer verdien i rota i minHeapen representert ved root
      // Verdien i den siste noden kopieres til rota,
      // og den siste noden slettes
      // Deretter repareres minHeapen
}


String crack (char[] message, int[] key) {
      // Implementeres i 4A
      // Returnerer en string med tegnene i message
      // i rekkefølgen gitt av key
}


void guess (char[] message) {
      // Implementeres i 4B
      // Skriver ut alle mulige kombinasjoner av tegnene i message
      // Anta at det ikke er duplikater i message
}
```

Slutt på oppgavesettet.