UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF2220 — Algoritmer og datastrukturer
Eksamensdag:	15. desember 2010
Tid for eksamen:	14.30 – 18.30
Oppgavesettet er på 8 sider.	
Vedlegg:	Ingen
Tillatte hjelpemidler:	Alle trykte eller skrevne

Kontroller at oppgavesettet er komplett før
du begynner å besvare spørsmålene.

Innhold

Generelle **råd**:

- Skriv **leselig** (uleselige svar er alltid feil . . .)
- Husk å **begrunne** svar.
- Hold kommentarene dine korte og presise. Hvis du bruker kjente datastrukturer som (list, set, map, binær-tre) trenger du ikke forklare hvordan de fungerer. Generelt: hvis du bruker abstrakte datatyper fra biblioteket kan du bruke disse uten å forklare hva de gjør.
- **Vekten** til en oppgave indikerer vanskelighetsgraden og tidsbruken du bør bruke på den oppgaven, ut i fra våre estimat. Dette kan være greit å benytte for å disponere tiden best mulig, dvs. ikke bruk mye tid på en oppgave med liten prosentsats (gå videre).
- Klassenavn og navn på interface er i oppgaveteksten angitt ved at vi bruker stor forbokstav. Dvs. `Set` kan angi interfacet `java.util.Set`, men ikke "set".

(Fortsettes på side 2.)

Oppgave 1 Binært søketre (vekt 15%)

1a Innsetting (vekt 5%)

Tegn treet du får ved å sette disse verdiene: 9, 1, 3, 10, 4, 6, 7 og 8, inn i et tomt binært søketre (i den **rekkefølgen**).

1b Logaritmisk tid (vekt 5%)

Er det mulig å finne elementer i binærtreet fra oppgave **1a** i logaritmisk (log_2) tid? Begrunn svaret.

1c Fjern elementer (vekt 5%)

Fjern verdiene 1 og 3 fra binærtreet. Tegn binærtreet etter at de to elementene er blitt fjernet.

Oppgave 2 Sortering (vekt 20%)

Se på metoden `sorted` gitt i Figur 1, og avgjør følgende.

2a Klassifisering (vekt 5%)

Er dette en **verdi** eller **sammenlignings** basert sorteringsalgoritme? Begrunn svaret.

2b Tidskompleksitet (vekt 5%)

Hva er worst case tidskompleksitet til denne metoden, gitt som $O(n, m)$ hvor:

1. n = `input.length`
2. m = største verdi i `input` array

2c Average case (vekt 5%)

Skiller average case tidskompleksitet seg fra worst case tidskompleksitet? Begrunn svaret.

2d Svakhet (vekt 5%)

At metoden ikke kan håndtere negative verdier i input arrayen er åpenbart en svakhet, kan du finne en annen svakhet ved denne måten å sortere tall på?

(Fortsettes på side 3.)

```
// assume positive input values

int[] sorted(int[] input){

    int[] sort = new int[input.length];

    int max = 0;

    for( int i = 0; i < input.length; i++ ){
        if(input[i] > max){
            max = input[i];
        }
    }

    int[] b = new int[max + 1]; // all values are 0

    for( int i = 0; i < input.length; i++ ){
        b[ input[i] ]++;
    }

    int counter = 0;

    for( int i = 0; i < b.length; i++ ){
        while( b[i] > 0 ){
            sort[counter] = i;
            b[i]--;
            counter++;
        }
    }

    return sort;
}
```

Figur 1: `sorted`

Oppgave 3 Huffmankoding (vekt 10%)

Anta at en input-fil har gitt deg følgende frekvenstabell:

- `a`: 9
- `b`: 2
- `c`: 4
- `d`: 2
- `e`: 1
- `f`: 1

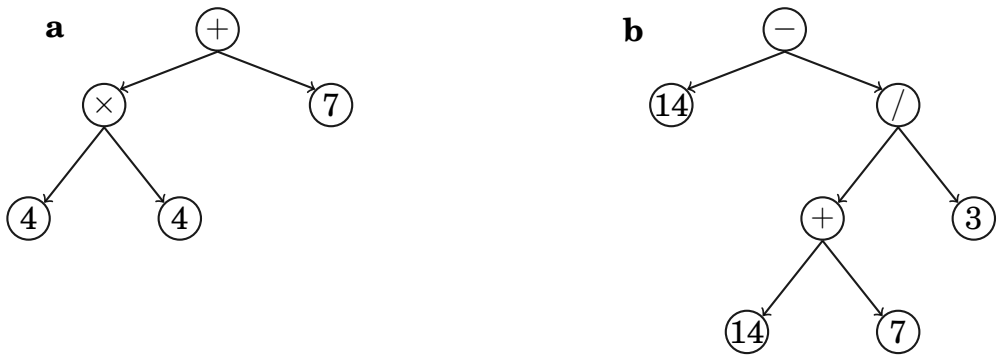
3a Huffmantre (vekt 10%)

Tegn et Huffmantre basert på frekvenstabellen.

(Fortsettes på side 4.)

Oppgave 4 Parser (vekt 25%)

Anta at en parser for binære regneoperasjoner genererer et tre, hvor hver løvnode representerer et tall, og alle andre noder representerer en binær regneoperasjon ($+, -, \times, /$). I Figur 2 kan du se to eksempler på slike trær. Hver node inneholder en token/delstreng av typen `String`, som enten er streng representasjonen av et tall, eller streng representasjonen av en binær operator. Binæroperasjonene er representert ved klassen `BinOp`, vist i Figur 3.



Figur 2: Parsede uttrykk

```
public class BinOp{

    Node root;

    public String toString(){ return root.str(); }
    public double eval(){  return root.eval();   }
    public String polish(){ return root.polish(); }

    class Node{

        String token;
        Node left, right;

        String str(){ /* TODO */ }
        double eval(){ /* TODO */ }
        String polish(){ /* TODO */ }

        boolean isLeaf(){ // utility function
            return left == null && right == null;
        }
    }
}
```

Figur 3: BinOp

(Fortsettes på side 5.)

4a String representasjon (vekt 5%)

De binære operatorene kan ikke anvendes i vilkårlig rekkefølge, så vi må passe på at `String` representasjonen vi lager sier noe om rekkefølgen operatorene skal anvendes. Feks. vil uttrykk **a** fra Figur 2 kunne misforstås dersom det representeres ved strengen "$4 \times 4 + 7$", siden det vil være uklart om vi mener: $(4 \times 4) + 7 = 23$, eller $4 \times (4 + 7) = 44$. Implementer `str` metoden i klassen `Node`, og bruk parenteser til å lage en entydig `String` representasjon. Uttrykk **a** fra Figur 2 kan feks. returnere denne strengen: "$((4 \times 4) + 7)$"

4b Evaluer (vekt 5%)

Implementer metoden `eval` i klassen `Node` som returnerer tallverdien (**double**) av et uttrykk.
Merk: I treet kan alle elementer som representerer tall konverteres ved hjelp av metoden: `Double.parseDouble(String s).`

4c Polsk notasjon (vekt 5%)

Det er ofte gunstigere å representere uttrykk helt uten parenteser, men det må da gjøres på en annen måte for at rekkefølgen på regneoperasjonene skal komme klart fram. Polsk notasjon[1] lager en entydig representasjon av binære uttrykk uten parenteser ved at en gir operandene/tallene etter en har gitt operatoren.

- **a**: $+ \times 4\ 4\ 7$
- **b**: $- 14 / + 14\ 7\ 3$

Implementer metoden `polish` i klassen `Node`, som returnerer den polske notasjonen for uttrykket.

4d Polsk kalkulator (vekt 10%)

Her ser du utregningen av uttrykk **b** fra Figur 2 i polsk notasjon.

- $- 14 / + 14\ 7\ 3$
- $- 14 / (+ 14\ 7)\ 3$
- $- 14 / 21\ 3$
- $- 14\ (/ 21\ 3)$
- $- 14\ 7$
- $(- 14\ 7)$
- 7

[1] Oppfunnet av den polske logikeren Jan Lukasiewicz

(Fortsettes på side 6.)

Fra venstre mot høyre, anvend en operator så fort du har to operander/tall. Anta at du blir gitt `String` representasjoner av binære uttrykk på polsk notasjon, slik at du kan dele uttrykkene inn i operander/tall og operatorer ved å kalle funksjonen `String.split`.

```
String[] tokens = polishExpression.split(" ");
```

Dette vil gi deg en array av delstrenger som er enten `String` representerte tall, eller operatorer. Anta videre at du har følgende hjelpemetoder tilgjengelig.

```
boolean isOp(String token){
    /* implementation left out */
}

String strApply(String op, String v1, String v2){
    /* implementation left out */
}

/* ---------------------- Example usage ---------------------- */

isOp("+") == true
isOp("*") == true
isOp("5") == false

strApply("+", "40", "1").equals("41") == true
strApply("*", "35", "2").equals("70") == true
strApply("/", "5", "2").equals("2.5") == true

/* ---------------------- Example usage ---------------------- */
```

Velg passende datastruktur og implementer metoden `polishEval` som tar imot uttrykk på polsk notasjon, og beregner tallverdien (**double**) av uttrykket.

```
double polishEval( String polishExpression ){
    /* TODO */
}
```

(Fortsettes på side 7.)

Oppgave 5 Diverse oppgaver (vekt 15%)

5a Topologisk sortering (vekt 7.5%)

Anta at du har en avhengighetsgraf med følgende noder: {Q, B, J, P, A, Z}. Grafen har 4 lovlige topologiske sorteringer, det finnes **ingen** annen sortering av nodene som tilfredstiller avhengighetene.

1. Q, A, B, J, Z, P
2. Q, B, A, J, Z, P
3. Q, A, B, Z, J, P
4. Q, B, A, Z, J, P

Tegn en graf som oppfyller kriteriene.

5b Boyer Moore (vekt 7.5%)

Beregn **good-suffix-shift** ut i fra nålen: **skjeskj**

Oppgave 6 Felles elementer (vekt 15%)

Anta at du har to bokstav arrays `char[] a1` og `char[] a2`, som begge inneholder 1-byte[2] bokstaver, der `a1.length == N` og `a2.length == M`.

6a Implementasjon (vekt 10%)

Implementer metoden med signaturen fra Figur 4, som returnerer en array av bokstaver som fins i både `a1` og `a2`. Rekkefølgen på bokstavene i arrayen som blir returnert er uten betydning.

```
// returns characters which are contained in both a1 and a2

char[] both(char[] a1, char[] a2);
```

Figur 4: Signature of `both` function

6b Kompleksitet (vekt 5)

Hva er worst case tidskompleksiteten på implementasjonen din? Begrunn svaret.

Lykke til!

[2] 1-byte bokstaver betyr at tallverdien til bokstavene vil ligge i intervallet 0...255

(Fortsettes på side 8.)

Metoder

Her er en liste av metoder dere kan finne nyttig under eksamen, de er alle tatt fra Java sitt standard bibliotek.

```
java.util.Set<E>

* boolean add(E e)                  // legg til element e
* boolean remove(Object o)          // fjern Object o
* boolean contains(Object o)        // er Object o med i Set?
* int size()                        // antall elementer


java.util.List<E>

* boolean add(E e)                  // legg til element e
* void add(E e, int index)          // legg til element e på index
* boolean remove(Object o)          // fjern Object o
* E remove(int index)               // fjern element på index
* E get(int index)                  // hent element på index
* int indexOf(Object o)             // finn index til o
* boolean contains(Object o)        // er Object o med i List?
* int size()                        // antall elementer


java.util.Map<K,V>

* boolean containsKey(Object o)    // er Object o en nøkkel?
* boolean containsValue(Object o) // er Object o en verdi?
* V put(K k, V v)                  // legg til nøkkel-verdi par
* V get(Object key)                // hent verdi fra nøkkel
* V remove(Object key)             // slett nøkkel-verdi par
* int size()                       // antall nøkkel-verdi par
* Set<K> keySet()                  // returner et Set av nøkler


java.util.PriorityQue<E>

* boolean add(E e)                  // sett element e inn i prioritetskø
* E peek()                          // hent element med høyest prioritet
* E poll()                          // samme som peek + slett element fra kø
* boolean contains(Object o)        // er Object o et element i PriorityQue?
* int size()                        // antall elementer i køen
```