UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i	INF2220 — Algoritmer og datastrukturer
Eksamensdag:	13. desember 2011
Tid for eksamen:	14.30 – 18.30
Oppgavesettet er på 7 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).

Lykke til!
Ragnhild Kobro Runde og Ingrid Chieh Yu

(Fortsettes på side 2.)

Oppgave 1 Prioritetskø (vekt 10%)

En *max heap* er en binær heap hvor ordningskravet er slik at barn alltid er *mindre eller lik* sine foreldre.

Gitt verdiene $13, 24, 5, 6, 18, 57, 9, 12, 35, 8, 16, 3, 20, 35$ og 14

1a Innsetting (vekt 5%)

Vis array-representasjonen du får ved å sette inn verdiene en om gangen i en initielt tom max heap.

1b Lineær tid-oppbygging av heap (vekt 5%)

La B være array-representasjonen av et komplett binært tre med verdiene gitt over. Vis array-representasjonen av *max heap*en du får ved å bruke lineær tid *buildHeap*-algoritmen på B.

Oppgave 2 Kompleksitet (vekt 10%)

Hva er *worst-case* kompleksiteten av følgende implementasjoner:

2a For-løkker (vekt 3%)

```
for i = 1 to n {
  for j = i to n {
    for k = i to j {
       print (i, j, k);
    }
  }
}
```

2b Rekursjon (vekt 7%)

```
float foo(A) {
  n = A.length;
  if (n==1) {
    return A[0];
  }
  let A1,A1,A3,A4 be arrays of size n/2
  for (i=0; i <= (n/2)-1; i++) {
    for (j=0; j <= (n/2)-1; j++) {
      A1[i] = A[i];
      A2[i] = A[i+j];
      A3[i] = A[n/2+j];
      A4[i] = A[j];
    }
```

(Fortsettes på side 3.)

```
  }
  b1 = foo(A1);
  b2 = foo(A2);
  b3 = foo(A3);
  b4 = foo(A4);
}
```

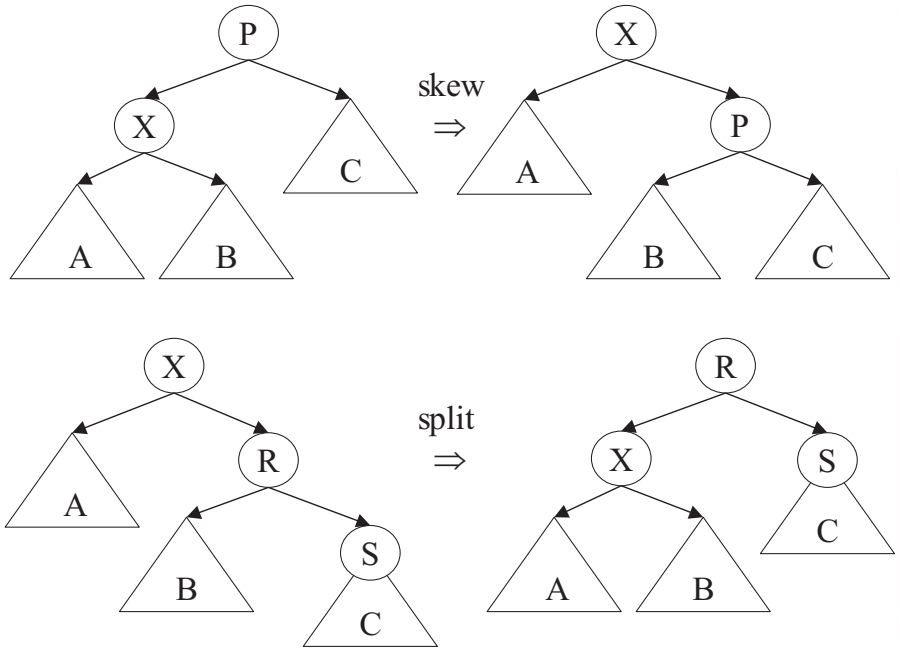
Oppgave 3 Binære Søketrær (vekt 15%)

Gitt følgende klasse for nodene i et binært søketre:

```
class BinNode {
    BinNode left, right;
    int element;
}
```

3a Balansering av trær (vekt 7.5%)

To populære metoder for å rebalansere et binært søketre er skew og split som vist på følgende figur:



1. Forklar hvorfor skew og split bevarer søketre-egenskapene.
2. Implementer skew og split uten hjelpemetoder. Begge metodene skal ta roten i det aktuelle subtreet (P for skew i eksempelet) som parameter, rotere som vist og returnere den nye roten (X for skew i eksempelet). Du kan anta at nodene tilsvarende X og P, R og S på figuren eksisterer (dvs at de ikke er null).

(Fortsettes på side 4.)

3b ABBA-trær (vekt 7.5%)

Et ABBA-tre er et rød-svart tre hvor kun høyre-barn kan være røde. Hver node i treet inneholder i tillegg informasjon om sitt nivå slik at:

a. Alle bladnoder har nivå = 1.

b. Alle svarte noder har nivå en mindre enn sin forelder.

c. Alle røde noder har nivå likt som sin forelder.

Utvid BinNode-klassen over med de nødvendige attributtene og skriv en metode som sjekker om et rød-svart tre er et gyldig ABBA-tre i henhold til de gitte kravene. Du kan anta at treet er et gyldig rød-svart tre. Lag gjerne hjelpemetoder hvis du finner det mest hensiktsmessig.

Oppgave 4 Binærtre (vekt 10%)

Et komplett binærtre med N elementer er lagret i en array, i posisjon 1 til N. Hvor stor må arrayen være for

1. et binærtre med to ekstra nivåer?

2. et binærtre med N noder hvor den dypeste noden er på dybde $2\ log\ N$?

3. en binærtre med den dypeste noden på dybde $4.1\ log\ N$?

Oppgave 5 Grafer (vekt 13%)

5a Lengste vei i en graf (vekt 5%)

Skriv en metode `longestPath` slik at den, gitt en vektet rettet asyklisk graf $G = (V, E)$, returnerer vekten av den lengste veien i grafen i tid $\mathcal{O}(|V| + |E|)$. (Grafen kan inneholde kanter med negative vekter.) Begrunn hvorfor denne algoritmen tilfredstiller tidskravet.

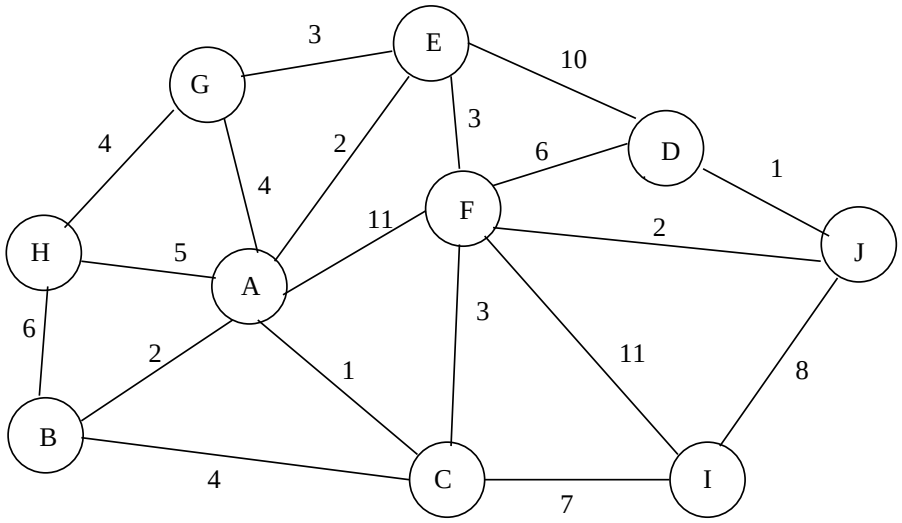
Hva må algoritmen gjøre for å returnere den korteste veien?

Du kan skrive metoden i pseudokode, men den må være detaljert. Det må komme klart fram hva initialiseringen består av og hva hovedløkkene gjør.

5b Kruskal og Prim (vekt 5%)

Gitt følgende graf:

(Fortsettes på side 5.)



1. Finn et minimum spenntre for grafen ved hjelp av *Kruskal*s og *Prim*s algoritmer. Vis tydelig resultatet etter hvert trinn av algoritmene ved å liste kantene i den rekkefølgen de er valgt.
2. Er dette treet unikt?
3. Når kan vi garantere at treet er unikt?

5c Spenntre (vekt 3%)

Gitt en sammenhengende urettet graf G med unike vekter på kantene og la S være et minimalt spenntre av G. Er det slik at kanten med den nest minste vekten $e \in G$ må være en kant i S? Forklar.

Oppgave 6 Rebusløp (vekt 15%)

Universitetet i Uqbar skal arrangere rebusløp for de nye studentene sine. Tilsammen er det N lag som alle skal innom de samme M postene, hvor $N \leq M$. Universitetet ønsker et oppsett hvor alle lagene starter samtidig, og at det er max ett lag om gangen på hver post. For enkelhets skyld antar vi at alle lagene bruker like lang tid på hver post (og mellom postene).
To lag skal heller ikke ha samme rekkefølge på to av postene (dersom lag 1 skal gå direkte fra post 1 til post 2, skal ingen andre lag kunne gjøre det samme på noe punkt i rebusløpet).
Gitt klassen `RebusLop` under, hvor konstruktøren setter `N` og `M` til å være henholdsvis antall lag og antall poster. I tillegg initialiseres arrayen `oppsett`. Du skal nå implementere metoden `lagOppsett` som fyller `oppsett`-arrayen med et komplett oppsett for rebusløpet i henhold til reglene over - dersom et slikt eksisterer. For $N = M = 4$ kan arrayen for eksempel bli seende slik ut:

(Lag 0)	0	1	2	3
(Lag 1)	1	3	0	2
(Lag 2)	2	0	3	1
(Lag 3)	3	2	1	0

(Fortsettes på side 6.)

Radene i arrayen representerer altså i hvilken rekkefølge de enkelte lagene kommer til de ulike postene.
Bestem selv eventuelle parametre til metoden `lagOppsett`. Når et mulig oppsett er ferdig, skal metoden kalle `skrivUt`. Innmaten i denne skal du imidlertid *ikke* implementere. Utvid gjerne klassen `RebusLop` med flere variable hvis du finner det hensiktsmessig, og angi eventuelt hvordan disse skal initialiseres i konstruktøren. Det er også tillatt å lage flere hjelpemetoder hvis du ønsker det.

```
class RebusLop {
    int N, M;
    int[][] oppsett;

    // Eventuelle andre variable

    RebusLop(int antallLag, int antallPoster) {
        N = antallLag;
        M = antallPoster;
        oppsett = new int[N][M];
        // Eventuell initialisering av andre variable
    }

    void lagOppsett(...) {
        // TODO
    }

    void skrivUt() {
        // Innhold ikke tatt med, skriver ut det ferdige oppsettet.
    }
}
```

Oppgave 7 Tekstalgoritmer (vekt 12%)

7a Boyer Moore (vekt 4%)

Beregn *good suffix shift* til *nålen*: `a b c a b c a c a b`

7b Boyer Moore (vekt 4%)

En mismatch er funnet under leting etter nålen `a b c a b c a c a b` som angitt nedenfor:

```
Høystakk: a a c a b c a a b c a b c a b c a c a b
     Nål: a b c a b c a c a b
```

Hvor i høystakken vil *Boyer Moore*-algoritmen lete etter neste match? (dvs. hvor langt vil Boyer More skifte nålen etter denne mismatch?)

7c Huffmankoding (vekt 4%)

Lag en Huffmankode for strengen `nonconcurrence`. Vis dekodingstabellen.

(Fortsettes på side 7.)

Oppgave 8 Det Nederlandske Flagget (vekt 15%)

Spillet "Det nederlandske flagget" består av et antall røde, hvite og blå brikker på rekke. Målet med spillet er å bytte om på brikkene slik at alle de røde brikkene ligger til venstre, alle de blå brikkene til høyre, og alle de hvite brikkene i midten. De eneste lovlige operasjonene er å undersøke en brikkes farge og å bytte om på to brikker.

Anta at utgangsstillingen er representert i en char-array hvor de eneste mulige array-verdiene er 'R', 'H' og 'B' (det er altså ingen ledige plasser i arrayen). Lag en metode som sorterer arrayen i henhold til kriteriene over med så få ombyttinger som mulig.

Hint: Tenk på arrayen som bestående av fire (dynamisk store) deler — ferdigsorterte røde, ferdigsorterte hvite, usorterte, og ferdigsorterte blå. Initielt vil da alle brikkene tilhøre den usorterte delen, og de tre andre delene være tomme.