Det matematisk-naturvitenskapelige fakultet
UNIVERSITETET I BERGEN
Eksamen i emnet I 120 - Algoritmer, datastrukturer og programmering
Mandag 21.Mai 2001, kl. 09-15.

Ingen hjelpemidler tillatt. Oppgavesettet består av 6 oppgaver. I vedlegget er du gitt dokumentasjon av utvalgte ADTer og klasser i JDSL, som du fritt kan bruke i dine besvarelser. Skriv ned alle forutsetninger du gjør som ikke står i oppgaveteksten. Lykke til!

Oppgave 1. Sekvenser (20%)

Oppgaven dreier seg om ADT RankedSequence (JDSL-dokumentasjon gitt i vedlegg) og vi antar at vi har et objekt S av en klasse som implementerer denne. Lengden av S er $n > 1$. Elementene som ligger i S er av type Integer og har verdier ≥ 0. For en $0 \leq i \leq n$ skriver vi S[i] for verdien av $i$te element i S, dvs for ((Integer)S.elemAtRank(i)).intValue(). Du kan bruke denne forkortelsen i din egen kode, men ellers skal du gi **fullstendig Java-kode**. Vi skal lage metode(r) **public void duplikat(RankedSequence S)** for å fjerne duplikater i S. F.eks. om S har innholdet $< 3, 6, 2, 3, 8, 3, 5, 5, 2 >$ skal dette omformes slik at ved metodekallets slutt har S innholdet $< 3, 6, 2, 8, 5 >$, dvs innbyrdes rekkefølge mellom første forekomst av hvert tall skal beholdes.

1.1 Anta vi har en garanti for at alle verdier i S er mellom 0 og 100, dvs $\forall i : 0 \leq i \leq n : 0 \leq S[i] < 100$. Programmer en duplikatfjernermetode som beskrevet over med verste-fall kjøretid O(n). Hvilken implementasjon antar du for RankedSequence?

1.2 Anta vi ikke har noen øvre grense for verdiene i S. Gi en O(n^2) duplikatfjernermetode for dette tilfellet.

1.3 Anta vi ikke har noen øvre grense for verdiene i S, men at S er sortert, dvs $S[0] \leq S[1] \leq ... \leq S[n-1]$. Gi en raskest mulig metode for duplikatfjerning i dette tilfellet, og angi kjøretiden. Hvilken implementasjon antar du for RankedSequence?

1.4 Her trenger du ikke gi Java-kode, kun en kort forklaring. Hva er kjøretiden til den raskeste algoritmen for det generelle tilfellet, dvs ingen øvre grense for verdiene i S, S er usortert, og vi tillater bruk av ekstra datastrukturer?

Oppgave 2. Ordbok/Dictionary (15%)

2.1 En ordbok/dictionary kan implementeres enten som a. sortert tabell, b. usortert tabell, eller c. binært søketre. Gi pseudokode for innsetting av nytt element i hvert av tilfellene a, b, og c.

2.2 Vi skal bruke et binært søketre som lagrer heltall. Vi starter med et tomt binært søketre, og bruker algoritme for innsetting og fjerning fra læreboka. Tegn for hver av instruksjonene under, i gitt rekkefølge, det resulterende binære søketre:
a. Insert(4) b. Insert(8) c. Insert(6) d. Insert(2) e. Insert(9) f. Insert(5) g. Remove(2) h. Remove(4)

2.3 Hva er verste-fall kjøretid for Insert og for Remove i et binært søketre med n elementer? Gi en forklaring for hvert av svarene.

Oppgave 3. Haug/Heap (15%)

3.1 En heap implementeres som en tabell/sekvens. Er sekvensen

$$< 3, 4, 6, 8, 9, 7, 10 >$$

en heap?

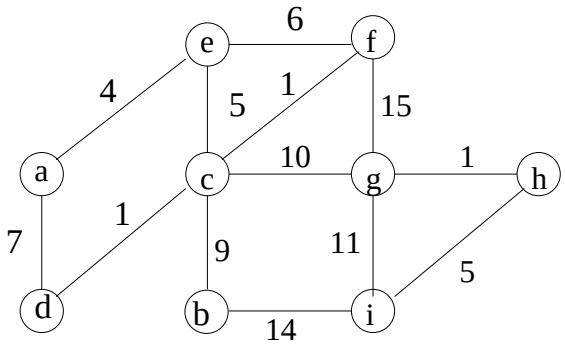
3.2 Vi skal bruke en heap som lagrer heltall. Vi starter med en tom heap, og bruker algoritme for innsetting og fjerning fra læreboka. Tegn for hver av instruksjonene under, i gitt rekkefølge, den resulterende heap'en:
a. Insert(4) b. Insert(8) c. Insert(6) d. Insert(2) e. Insert(9) f. Insert(5) g. RemoveMin h. RemoveMin

3.3 Hva er verste-fall kjøretid for Insert og for RemoveMin i en heap med n elementer, implementert vha en tabell? Gi en forklaring for hvert av svarene.

Oppgave 4. Grafalgoritmer (20%)

I deloppgavene under skal du simulere forskjellige grafalgoritmer på grafen G gitt i figuren, alltid med utgangspunkt i node a. Vi sier at en node *oppdages* idet en algoritme første gang treffer på noden. Dersom det finnes et valg, så anvend alfabetisk ordning, f.eks. betyr dette at naboene til en node hentes fram i alfabetisk rekkefølge f.eks. ved følgende nabolister:

-a: d, e	-d: a, c	-g: c, f, h, i
-b: c, i	-e: a, c, f	-h: g, i
-c: b, d, e, f, g	-f: c, e, g	-i: b, g, h



Vær nøye med å overholde den alfabetiske ordningen, og dobbeltsjekk dine svar, da et helt korrekt svar vil gi vesentlig flere poeng.

4.1 I hvilken rekkefølge vil DFS(a), dvs dybde-først-søk fra a, oppdage nodene i G?

4.2 I hvilken rekkefølge vil BFS(a), dvs bredde-først-søk fra a, oppdage nodene i G?

4.3 Vi sier at node x *forlates* idet det rekursive kallet DFS(x) terminerer. I hvilken rekkefølge vil DFS(a) forlate nodene i G?

4.4 Du skal nå simulere Dijkstra(a), dvs Dijkstra's algoritme for korteste stier med startnode a, på grafen G, og svare på to spørsmål. I hvilken rekkefølge vil Dijkstra(a) trekke noder inn i 'skyen' av noder som har fått korteste sti fastslått? Hva er de suksessive verdiene som blir tildelt hver node i distansetabellen? Angi disse verdiene for hver enkelt node, f.eks. for node d er svaret: $\infty, 7$.

4.5 Du skal nå simulere Prim(a), dvs Prim-Jarnik's algoritme for minimum utspennende tre på grafen G hvor vi antar node a blir vilkårlig valgt i første steg, og svare på to spørsmål. Hva blir det resulterende minste utspennende treet? Tegn dette treet. I hvilken rekkefølge tildelte Prim(a) kantene til dette treet? Sett f.eks. et nummer på hver kant i treet som reflekterer når de ble lagt til treet.

Oppgave 5. Kompleksitetsanalyse (15%)

For hver av metodene `f1, f2, f3, f4, f5`, bestem verste-fall kjøretid i stor-O (big-Oh) notasjon for kallene `f1(a,N), f2(a,N), f3(N), f4(a,N), f5(a,N)` som en funksjon av argumentet N. For enkelhets skyld antar vi N er et positivt heltall som er en potens av 2. Den øvre grensen du gir skal være så tett som mulig, og du skal droppe uvesentlige konstanter, f.eks. for lineær kjøretid er O(N) et bedre svar enn O(2N+logN), selv om begge strengt tatt er korrekte. NB: Metodene utfører ikke nødvendigvis en fornuftig beregning, det er kun kjøretiden som skal analyseres. **Gi en begrunnelse for hvert svar.**

```
void f1(int[] a, int n) {
   int k,j;
   k = 1;
   while (k < n) {
      for (j = 0; j < n*n; j++) {
         a[j] = a[j]/k;
         k = 2*k;
      }
   }
}
```

```
void f2(int[] a, int n) {
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         for (int k = 0; k < j; k++) {
            a[i] = a[i]+j*k;
         }
      }
   }
}
```

```
int f3(int n) {
   if (n < 2) return n;
   return n/2+f3(n-1);
}
```

```
void f4(int[] a, int n) {
   for (int i = 0; i < n; i++) {
      a[i] = f3(i);
   }
}
```

```
int f5(int[] a, int n) {
   if (n < 2) return n;
   for (int i = 0; i < n; i++) {
      a[i] = a[i]+i;
   }
   return f5(a, n/2)+f5(a, n/2);
}
```

Oppgave 6. Køer (15%)

Oppgaven dreier seg om ADT Queue (JDSL-dokumentasjon gitt i vedlegg). Vi skal lage metoder for å sjekke om to køer er like. En kø S er *lik* en kø Q hvis enten:

- både `S.isEmpty()==true` og `Q.isEmpty()==true`, eller
- både `S.isEmpty()==false` og `Q.isEmpty()==false` samtidig som `S.front()==Q.front()` og køen som resulterer etter `S.dequeue()` er *lik* køen som resulterer etter `Q.dequeue()`.

Vi antar at vi har en implementasjon av ADT Queue, dvs en `class Ko implemements Queue`. Du skal lage en ny klasse som arver fra `class Ko` og har en ny metode `public boolean eqQueue(Queue Q)` som returnerer true hvis og bare hvis køen metoden kalles fra er *lik* argumentet Q. Du skal bruke kall til metodene fra `interface Queue` og gi **fullstendig Java-kode**, med bruk av JDSL.

6.1 Du skal implementere `eqQueue` slik at metodekallet returnerer såsnart det er oppdaget om køene er like, dvs køene kan bli endret.

6.2 Du skal nå implementere `eqQueue` slik at når et kall til denne metoden returnerer er begge køene gjenoppbygget, uavhengig av om køene var like.

Jan Arne Telle Chunming Rong