Skriftlig Eksamen
Algoritmer og Datastrukturer (DM02)

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

Fredag den 2. januar 2004, kl. 9–13

Alle sædvanlige hjælpemidler (lærebøger, notater, osv.) samt brug af lommeregner er tilladt.

Eksamenssættet består af 4 opgaver på 4 nummererede sider (1–4).
Fuld besvarelse er besvarelse af alle 4 opgaver.
De enkelte opgavers vægt ved bedømmelsen er angivet i procent. Bemærk, at de enkelte spørgsmål i en opgave ikke nødvendigvis har samme vægt.

Der må gerne refereres til algoritmer og resultater fra lærebogen (enten den, som blev brugt i efteråret 2002, eller den, som blev brugt i efteråret 2003) inklusive øvelsesopgaverne. Henvisninger til andre bøger accepteres ikke som besvarelse af et spørgsmål.

Bemærk, at hvis der er et spørgsmål, man ikke kan besvare, må man gerne besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Husk at begrunde dine svar!

Opgave 1 (15%)

I denne opgave betragter vi problemet at evaluere et polynomium $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ i et givet punkt $x = x_0$.

F.eks. er $P(2) = 23$, hvis $P(x) = 2x^3 + x + 5$.

Lad A være et array af længde $n+1$, og lad $A[i] = a_i$, $0 \leq i \leq n$.
I eksemplet ovenfor er $A = \{5, 1, 0, 2\}$.

Følgende ineffektive metode beregner $P(x_0)$, når den kaldes med $\texttt{evalPoly}(A, x_0)$.

```
int evalPoly(int[] A, int x0) {
    int total = 0;
    for (int i=0; i<A.length; i++) {
        int y=1;
        for (int j=0; j<i; j++)
            y = y * x0;
        total = A[i] * y + total;
    }
    return total;
}
```

Spørgsmål a: Hvad er metodens køretid udtrykt i Θ-notation? □

Følgende mere effektive metode er opkaldt efter William G. Horner.

```
int horner(int[] A, int x0) {
    int total = A[A.length-1];
    for (int i=A.length-2; i>=0; i--)
        total = total * x0 + A[i];
    return total;
}
```

Spørgsmål b: Bevis, at i starten af for-løkken gælder $\texttt{total} = a_n x_0^{n-(\texttt{i}+1)} + a_{n-1}x_0^{n-(\texttt{i}+2)} + \ldots + a_{\texttt{i}+2}x_0 + a_{\texttt{i}+1}$, hvis metoden kaldes med $\texttt{horner}(A, x_0)$.
Husk, at $\texttt{A.length} = n + 1$. □

Spørgsmål c: Argumenter for, at metoden `horner` er korrekt, vha. resultatet fra spørgsmål b. □

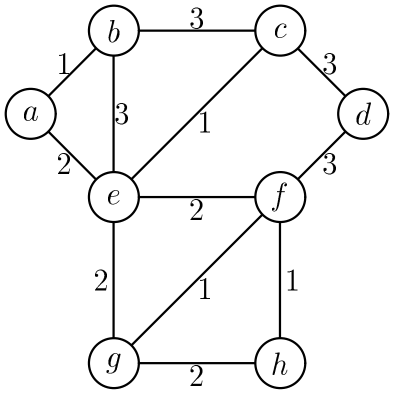
Opgave 2 (30%)

Denne opgave handler om at finde letteste udspændende træer og korteste veje i sammenhængende, ikke-orienterede grafer.

Lad n betegne antallet af knuder i grafen, og lad m være antallet af kanter. Bemærk, at $n \in O(m)$, da grafen er sammenhængende.

Spørgsmål a: Tegn et korteste-vej-træ for nedenstående graf. Træet skal have rod i knuden a. For hver knude skal den korteste vej fra a angives.

Forklar kort, hvordan du har fundet træet. $\square$



Spørgsmål b: Tegn et letteste udspændende træ for grafen fra spørgsmål a. Argumenter for, at dit træ er et letteste udspændende træ. $\square$

Spørgsmål c: Giv en simpel algoritme, som finder et letteste udspændende træ i tid $O(m)$, hvis alle kanter har samme vægt. $\square$

Spørgsmål d: Generelt har Prims algoritme køretid $O(m \log n)$. Antag nu, at alle kant-vægte er heltal mellem 1 og k. Forklar, hvordan man kan implementere Prims algoritme, så den får køretid $O(kn + m)$.

Vink: anvend et array med plads til k hægtede lister. $\square$

Spørgsmål e: Hvad skal der gælde om k (udtrykt ved m og n), for at køretiden af algoritmen fra spørgsmål d er asymptotisk bedre end $\Theta(m \log n)$?
$\square$

Opgave 3 (30%)

Denne opgave går ud på at finde en længste voksende delfølge i en følge af positive heltal. En delfølge af en følge S af tal er som bekendt blot følgen S med nul eller flere tal slettet.

F.eks. er $\langle 1, 4, 7, 2 \rangle$ en delfølge af $\langle 1, 5, 4, 9, 4, 7, 2, 8 \rangle$.

En voksende følge er en følge $\langle n_1, n_2, \ldots, n_k \rangle$, hvor $n_1 < n_2 < \ldots < n_k$. Dvs. $\langle 1, 4, 9 \rangle$ er eksempelvis en voksende delfølge af $\langle 1, 5, 4, 9, 4, 7, 2, 8 \rangle$, og $\langle 1, 4, 7, 8 \rangle$ er en længste voksende delfølge af $\langle 1, 5, 4, 9, 4, 7, 2, 8 \rangle$.

Spørgsmål a: Find en længste voksende delfølge af $\langle 7, 2, 5, 3, 10, 8, 9, 4 \rangle$. □

Lad $S = \langle n_1, n_2, \ldots, n_k \rangle$ være en følge af positive heltal. Lad A være et array af længde $k + 1$, og antag at $A[0] = 0$ og $A[i] = n_i$, for $1 \leq i \leq k$.

Følgende rekursive metode beregner længden af en længste voksende delfølge af S, hvis den kaldes med `lvd(A,0,0)`.

```
int lvd(int[] A, int i, int j) {
    if (i == A.length) return 0;
    int max = lvd(A,i+1,j);
    if (A[i] > A[j]) {
        int m = lvd(A,i+1,i)+1;
        if (m > max) max = m;
    }
    return max;
}
```

Spørgsmål b: Forklar, hvordan `lvd` beregner sit resultat. □

Spørgsmål c: Vis ved et eksempel, at `lvd` nogle gange løser det samme delproblem flere gange. □

Spørgsmål d: Lav en ny version af `lvd` vha. dynamisk programmering, så delresultater kun beregnes én gang. Hvad er worst-case køretid og pladsforbrug for din algoritme? □

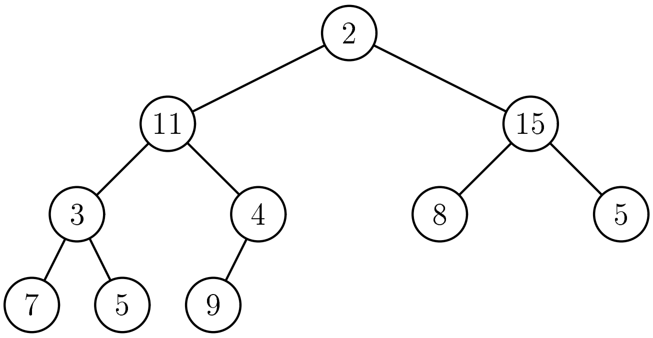
Opgave 4 (25%)

I denne opgave skal vi se på datastrukturen min-max-hob. Den svarer til datastrukturen hob, bortset fra at man kan udføre både EXTRACT-MIN og EXTRACT-MAX i tid $O(\log n)$, hvor n er antal elementer i min-max-hoben.

En min-max-hob har hob-struktur, dvs. det nederste lag af knuder er "fyldt op" fra venstre mod højre, og alle andre lag er helt fyldt op.

Ordningen af nøglerne i en min-max-hob er en smule anderledes end i en almindelig hob: i lag med lige nummer har hver knude v en nøgle, som er *mindre* end eller lig med alle nøgler i v's undertræ, og i lag med ulige nummer har hver knude v en nøgle, som er *større* end eller lig med alle nøgler i v's undertræ. (Husk, at v's undertræ er træet, som består af v og alle dens efterkommere.) Roden befinder sig i lag 0, rodens børn befinder sig i lag 1, osv.

Her er et eksempel på en min-max-hob med $n = 10$ nøgler.



Spørgsmål a: Hvordan finder man den største og den mindste nøgle i en min-max-hob? □

Spørgsmål b: Illustrer, hvordan man kan indsætte et element med nøgle 1 i min-max-hoben ovenfor ved at indsætte elementet nederst i min-max-hoben og "boble" det på plads (det skal selvfølgelig gøres en smule anderledes end i en almindelig hob). Vis de enkelte skridt. □

Spørgsmål c: Skriv pseudo-kode for EXTRACT-MIN og EXTRACT-MAX, som sletter henholdsvis det mindste og det største element i min-max-hoben. Begge algoritmer skal have køretid $O(\log n)$. □