Skriftlig Eksamen
Algoritmer og Datastrukturer (DM02)

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

Onsdag den 2. januar 2002, kl. 9–13

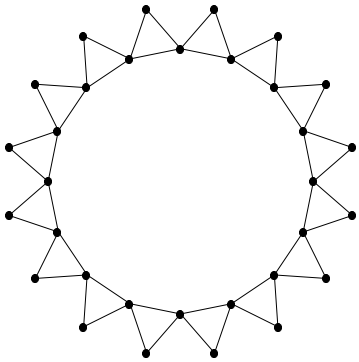
Alle sædvanlige hjælpemidler (lærebøger, notater, etc.) samt brug af lommeregner er tilladt.

Eksamenssættet består af 4 opgaver på 6 nummererede sider (1–6). Fuld besvarelse er besvarelse af alle 4 opgaver. De enkelte opgavers vægt ved bedømmelsen er angivet i procent. Der må gerne refereres til algoritmer og resultater fra lærebogen inklusive øvelsesopgaverne. Henvisninger til andre bøger (udover lærebogen) accepteres ikke som besvarelse af et spørgsmål.

Bemærk, at hvis der er et spørgsmål i en opgave, man ikke kan besvare, må man gerne (så vidt det er muligt) besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Opgave 1 (25%)

En *sol-graf* af grad $k \geq 3$ er en ikke-orienteret graf bestående af k trekanter sat sammen i en kreds. En sol-graf af grad 16 er illustreret nedenfor:



Spørgsmål a: Angiv, hvordan antallet af kanter, m, i en sol-graf afhænger af antallet af knuder, n. □

Vi antager nu, at kanterne er vægtede med ikke-negative tal. Vi er interesserede i at finde et letteste udspændende træ for en sol-graf, samt (uafhængigt heraf) korteste veje fra en given knude i grafen til alle andre knuder. De bedste algoritmer til dette, som kendes fra kurset, afvikles i tid $\Theta(n \log n + m)$ på generelle grafer. De næste spørgsmål drejer sig om at gøre det bedre på den særlige type grafer, som sol-grafer er.

Spørgsmål b: Forklar, hvordan man i tid $O(n)$ kan finde korteste veje fra en given knude til alle andre knuder i en sol-graf. □

Spørgsmål c: Forklar, hvordan man i tid $O(n)$ kan finde letteste udspændende træ i en sol-graf. □

Opgave 2 (25%)

I denne opgave ser vi på en anderledes metode til at måle, hvor ens to strenge er. Grundideén er at transformere de to strenge til de "matcher" ved at indsætte tegnet '?', der matcher et hvilket som helst tegn.

Mere formelt definerer vi, at

- to *tegn* matcher, hvis de er ens, eller hvis mindst ét af dem er '?'.
- to *strenge* matcher, hvis de er lige lange, og der for ethvert i gælder, at det i'te tegn i den ene streng matcher det i'te i den anden.

Omkostningen ved en given transformation defineres til at være det samlede antal '?'. Vi ønsker at finde den minimale omkostning ved at transformere to givne strenge til to matchende.

F.eks. kan "hundeejer" og "hakkejern" transformeres til

h	a	k	?	?	?	k	e	?	j	e	r	n
h	?	?	u	n	d	?	e	e	j	e	r	?

med en omkostning på 8.

Det er klart, at i en bedste løsning vil '?' aldrig optræde i samme position i de to matchende strenge.

Følgende rekursive metode finder den minimale omkostning for to strenge x og y, når den kaldes med mc(x, y, 0, 0):

```
public int mc(String x, String y, int i, int j)
{
    if (i ≥ x.length())
        return y.length()−j;
    else if (j ≥ y.length())
        return x.length()−i;
    else if (x.charAt(i) == y.charAt(j))
        return mc(x, y, i+1, j+1);
    else
        return Math.min(1 + mc(x, y, i, j+1), 1 + mc(x, y, i+1, j));
}
```

Spørgsmål a: Forklar, hvordan koden er relateret til problemstillingen; herunder hvorfor der adderes '1' nogle steder og ikke andre steder. □

Spørgsmål b: Vis (gerne ved et kort eksempel), at mc generelt under den rekursive beregning af svaret kaldes med samme argument flere gange. □

Spørgsmål c: Lav en ny version af mc med dynamisk programmering. □

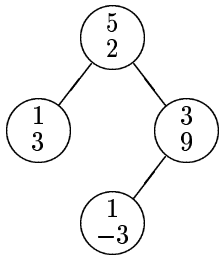
Spørgsmål d: Redegør for tabelstørrelse og kompleksitet i den udgave, hvor der anvendes dynamisk programmering. □

Opgave 3 (25%)

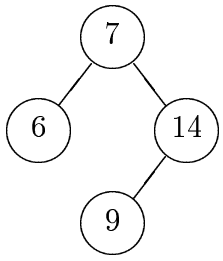
I denne opgave anvendes søgetræer til at indeholde heltalnøgler. Disse nøgler repræsenteres dog i søgetræet på en usædvanlig måde.

Hver knude indeholder et par bestående af et basistal (øverst) og et forskydningstal (nederst). Fortolkningen er, at et forskydningstal i en knude u skal lægges til samtlige basistal i u's undertræ.

Træet



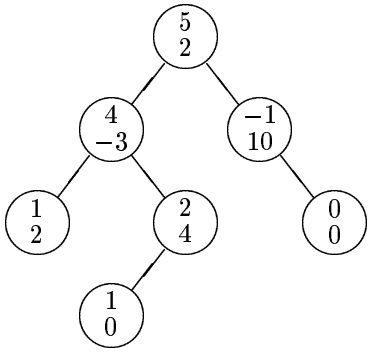
svarer altså til det sædvanlige træ



i det $7 = 5 + (2)$, $6 = 1 + (2 + 3)$, $14 = 3 + (2 + 9)$ og $9 = 1 + (2 + 9 - 3)$.

En knudes nøgle findes altså ikke eksplicit i dens knude, men kan beregnes som summen af knudens basistal og alle forskydningstal på vejen fra roden til knuden.

Spørgsmål a: Hvilket normalt søgetræ svarer nedenstående træ til?



□

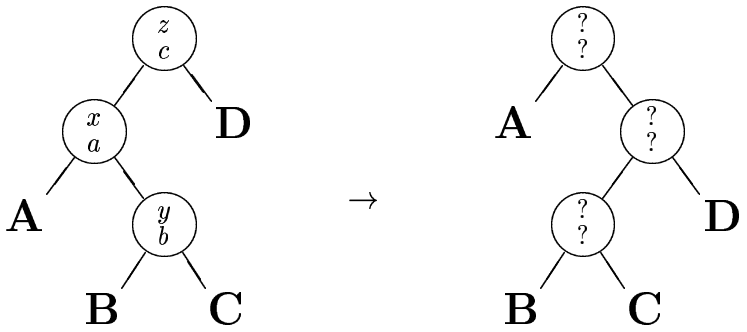
Spørgsmål b: Forklar, hvordan følgende metoder kan implementeres med de angivne kompleksiteter:

Operation	Kompleksitet
$search(k)$	$O(h)$
$insert(k)$	$O(h)$
$add(c)$	$O(1)$

Her angiver h træet højde, *search* søger efter og returnerer i positivt fald en reference til en knude, *insert* indsætter en nøgle k, mens *add* lægger konstanten c til alle nøgler i træet. □

For at sikre sig at $h \in O(\log n)$, hvor n er antallet af elementer i strukturen, må man omarrangere træet ind i mellem. Der er dog et problem med operationer, der flytter rundt på knuderne, for de ændrer stien ovenover andre knuder, som dermed får ændret deres samling af forskydningstal. I følgende spørgsmål ser vi på én sådan operation.

Spørgsmål c: Nedenfor ses en højrerotation. A, B, C og D er undertræer, x, y og z er basistal, og a, b og c er forskydningstal. Denne operation kan foretages overalt i træet. Knuden med z og c kan altså være en vilkårlig knude og ikke nødvendigvis roden af hele træet.



Vis, hvordan værdier kan fyldes ind i knuderne efter rotationen, så træet repræsenterer samme mængde nøgler efter operationen som før. □

Opgave 4 (25%)

Vi ser på beregningen af *halesummerne* for et array A. Dvs. for enhver hale (også kaldet postfix) ønsker vi at beregne summen af halens elementer og gemme disse i et array H.

Mere præcist antager vi følgende præbetingelse:

$$\text{A er et ikke-tomt array af heltal indiceret fra 0 til } n-1$$

Længden af A er altså $n \geq 1$.

Vi ønsker følgende postbetingelse:

$$\forall k \in \{0, \ldots, n-1\}\colon\ H[k] = \sum_{j=k}^{n-1} A[j]$$

Spørgsmål a: Angiv det ønskede H, hvis A indeholder 1, 2, 3, 5, 4, 3. □

Følgende lille kodeudsnit beregner H udfra A:

```
int[] H = new int[A.length];
H[A.length−1] = A[A.length−1];
i = A.length − 2;

while (i ≥ 0)
{
    H[i] = H[i+1] + A[i];
    i = i − 1;
}
```

Vi skal nu se på korrektheden af dette med udgangspunkt i invarianten I, der som sædvanligt er tilknyttet while-løkken.

$$I : (\forall k \in \{i+1, \ldots, n-1\}\colon\ H[k] = \sum_{j=k}^{n-1} A[j]) \wedge (i \geq -1)$$

Det må uden videre benyttes, at variablen i er et heltal.

Spørgsmål b: Vis, at I er en invariant for while-løkken. □

Spørgsmål c: Vis, at while-løkken terminerer. □

Spørgsmål d: Argumentér for, at programmet er korrekt. □