Skriftlig Eksamen
Datastrukturer og Algoritmer (DM02)

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

Tirsdag den 12. januar 1999, 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%)

På et datalogisk institut skal der lægges fliser på badeværelsesgulvet. Der skal dannes en række datalogiske ord og begreber som f.eks. "dynamisk programmering". Desværre kan der ikke leveres enkelt-bogstavs-fliser. I stedet kan man købe fliser med forskellige kombinationer af bogstaver. Derfor har vi brug for at kunne afgøre, om en konkret tekst kan skrives vha. de flisetyper, der er til rådighed. Man kan købe lige så mange, det skal være, af de flisetyper, der føres.

Hvis man har følgende flisetyper:

AMI	DYN	EN	GO	HU	JASK	RAMME	RING	ROG	SK P	T	VA

så kan man f.eks. godt skrive

DYN	AMI	SK P	ROG	RAMME	RING

Bemærk, at en blank bare er et tegn på lige fod med alle andre tegn.

Nedenstående PYTHON program afgør generelt, om en tekst T kan skrives vha. flisetyperne L, hvor L er en liste af de strenge, man kan købe på fliser.

```python
def Makes(L, T, i, j):
    if i == j:
        return 1
    for f in L:
        if f == T[i:j]:
            return 1
    for k in range(i+1, j):
        if Makes(L, T, i, k) and Makes(L, T, k, j):
            return 1
    return 0

L = ["AMI","DYN","EN","GO","HU","JASK","RAMME",
    "RING","ROG","SK P","T","VA"]

T = "DYNAMISK PROGRAMMERING"

print Makes(L, T, 0, len(T))
```

Spørgsmål a: Forklar kort, hvordan programmet fungerer. □

Spørgsmål b: Vis, at man kan komme til at lave det samme kald adskillige gange under afviklingen af programmet (man behøver ikke bruge samme L og T som i eksemplet). □

Det viser sig, at udførelsestiden af programmet bliver eksponentiel i længden af strengen T.

Spørgsmål c: Skriv en ny og mere effektiv udgave af `Makes` vha. dynamisk programmering. □

Spørgsmål d: Hvad bliver tidskompleksiteten, når der anvendes dynamisk programmering? Argumentér for dit svar. □

Opgave 2 (25%)

Denne opgave handler om letteste udspændende skove. I kurset har vi specielt set på Kruskal's algoritme, der har en god kompleksitet på generelle grafer. Nu vil vi i stedet for generelle grafer se på specielle klasser af grafer. Spørgsmålene drejer sig om at konstruere algoritmer, der på disse specielle grafer, har endnu bedre kompleksiteter end Kruskal's algoritme. I hvert tilfælde skal den algoritme, man angiver, være så effektiv som muligt givet den begrænsning, der er lagt på input-graferne.

Alle graferne er ikke-orienterede, vægtede grafer. Graferne er ikke nødvendigvis sammenhængende, så den letteste udspændende skov består af et letteste udspændende træ for hver komponent i grafen.

Spørgsmål a: Alle kanter har vægt 1.

Beskriv en effektiv algoritme, og argumentér for algoritmens tidskompleksitet. □

Spørgsmål b: Alle kanter har vægt 1 eller 2.

Beskriv en effektiv algoritme, og argumentér for algoritmens tidskompleksitet. □

Spørgsmål c: Alle knuder har grad højst 2.

Beskriv en effektiv algoritme, og argumentér for algoritmens tidskompleksitet. □

Opgave 3 (25%)

I denne opgave har vi givet en liste af heltal A. Listen har længde n. Vi er interesseret i at beregne alle summer af dellister af længde m, hvor $1 < m < n$. Disse resultater gemmes i listen B i samme rækkefølge, som de optræder i A. Listen B antages at være lang nok. Følgende programstump laver denne beregning:

```
S, i = 0, 0
while i < m:
    S = S + A[i]
    i = i + 1
B[0] = S

# Piræ
while i < n: # I
    S = S + A[i] - A[i-m]
    i = i + 1
    B[i-m] = S
# Post
```

Hvis vi f.eks. har `A = [1,7,5,7,8,3,4]`, `n = 7` og `m = 3` så vil `B` komme til at indeholde `B = [13,19,20,18,15]`.

I denne opgave skal det vises, at den anden `while`-løkke virker korrekt. Man kan antage, at der gælder følgende præ-betingelse til den anden `while`-løkke:

$$i = m \ \wedge \ S = A[0] + \cdots + A[m-1] \ \wedge \ B[0] = S$$

Lad I betegne prædikatet:

$$\begin{aligned} & i \leq n \\ \wedge \quad & S = A[i-m] + \cdots + A[i-1] \\ \wedge \quad & \forall j \in \{0, \ldots, i-m\}\colon\ B[j] = A[j] + \cdots + A[j+m-1] \end{aligned}$$

Spørgsmål a: Vis, at I er en invariant for den anden `while`-løkke. □

Spørgsmål b: Vis, at den anden `while`-løkke terminerer. □

Postbetingelsen er:

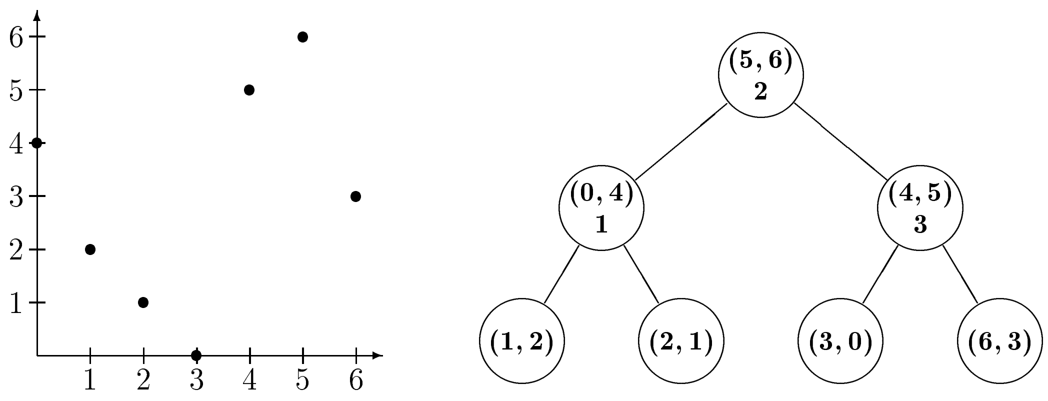
$$\forall j \in \{0, \ldots, n-m\}\colon\ B[j] = A[j] + \cdots + A[j+m-1]$$

Spørgsmål c: Vis, at algoritmen er korrekt. □

Opgave 4 (25%)

Denne opgave handler om punkter i planen og forskellige former for søgning efter disse punkter. Vi definerer en søgestruktur, som vi kalder et *prioritetssøgetræ* (forkortet PS-træ).

Nedenfor ses et eksempel. Til venstre er der et koordinatsystem indeholdende syv punkter. Til højre er det prioritetssøgetræ, punkterne opbevares i.



En knude i et PS-træ har felterne: `left`, `right`, `x`, `y` og `xlm`, hvor "xlm" står for "x left max". Dvs. en reference til et venstre undertræ (`left`), et højre undertræ (`right`), et punkt (`x` og `y`), samt en ekstra x-værdi (`xlm`), der skal indeholde den største x-koordinat i venstre undertræ. Da bladene ikke har et venstre undertræ, har deres `xlm`-felt ikke nogen (interessant) værdi. På tegningen er de derfor udeladt.

For nemheds skyld antager vi, at ingen punkter har samme x- eller y-koordinat.

Nedenfor forklarer vi, hvordan et PS-træ for n punkter kan konstrueres rekursivt. Husk, at $\lceil \frac{n}{2} \rceil$ betegner det mindste heltal, der er større end eller lig med $\frac{n}{2}$.

- Hvis $n = 0$, er PS-træet tomt.
- Hvis $n = 1$, består PS-træet af en enkelt knude indeholdende det ene punkt.
- Hvis $n > 1$, anbringes det punkt, der har størst y-koordinat, i roden. De resterende $n - 1$ punkter deles i en venstrehalvdel bestående af de $\lceil \frac{n-1}{2} \rceil$ punkter med mindst x-koordinat og en højre halvdel bestående af resten af punkterne.

  Rodens venstrebarn er et PS-træ, der konstrueres udfra punkterne i venstre halvdel, og rodens højrebarn er et PS-træ, der konstrueres udfra punkterne i højre halvdel.

  Endelig anbringes den største x-koordinat i venstre undertræ også i roden i feltet `xlm`.

Det kan antages, at man har en funktion `IsLeaf` til rådighed, som afgør, om en knude er et blad eller ej.

Spørgsmål a: Skriv en PYTHON-funktion, der i tid $O(\log n)$ afgør, om et punkt (x, y) findes i et givet PS-træ T. □

Spørgsmål b: Forklar, hvordan man givet et tal b kan finde (og udskrive) alle de punkter, der har y-koordinat større end eller lig med b.

Hvis der er k sådanne punkter, skal resultatet findes i tid $O(k)$. □

Spørgsmål c: Givet to tal a_1 og a_2, så $a_1 < a_2$. Hvor i træet finder man punkter (x, y), så $a_1 \leq x \leq a_2$?

Igen antager vi, at k er størrelsen af output. Hvordan udskriver man alle disse punkter i tid $O(\log n + k)$? □

Spørgsmål d: Givet tre tal a_1, a_2 og b, så $a_1 < a_2$. Hvor i træet finder man punkter (x, y), så $a_1 \leq x \leq a_2$ og $y \geq b$?

Igen antager vi, at k er størrelsen af output. Hvordan udskriver man alle disse punkter i tid $O(\log n + k)$? □