Skriftlig Eksamen
Geometriske Algoritmer (DM45)

Institut for Matematik og Datalogi
Syddansk Universitet, Odense

Onsdag den 19. januar 2000, kl. 9–13

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

Eksamenssættet består af 4 opgaver på 5 nummererede sider (1–5). 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.

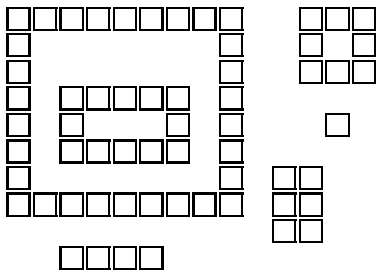
Bemærk, at hvis der er et spørgsmål i en opgave, man ikke kan besvare, må man gerne (så vidt det er muligt) besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Man behøver ikke specificere, hvordan simple geometriske konstant-tids beregninger (gymnasiepensum) foretages. Man må altså f.eks. gerne antage, at man har følgende funktioner til rådighed:

- Skærer 2 givne liniestykker hinanden?
- Beregn et evt. skæringspunkt for 2 givne liniestykker.
- Beregn en given linies vinkel med x-aksen.
- Ligger et givet punkt til højre for et givet orienteret liniestykke?

Opgave 1 (25%)

Antallet af rektangler i et "bit-map" skal bestemmes. Nedenfor ses et eksempel på et bit-map, hvor hvert lille kvadrat repræsenterer en pixel. Et typisk input er vist.



Vi repræsenterer pixels som koordinater.

Det er formentligt klart fra illustrationen, men for en sikkerheds skyld definerer vi nu, hvad et rektangel er i denne sammenhæng. Antag, at et rektangel med nederste venstre hjørne i (x, y) har højde $h \geq 1$ og bredde $b \geq 1$, hvor højde og bredde måles i antal pixels i siderne (det største rektangel i eksemplet har altså højde 8 og bredde 9). Rektanglet består da af alle pixels, der kan skrives som: $(x, y+i)$, $(x+b-1, y+i)$, $(x+j, y)$ eller $(x+j, y+h-1)$, hvor $i \in \{0, \ldots, h-1\}$ og $j \in \{0, \ldots, b-1\}$.

Input er en liste af (usorterede) pixels, der repræsenterer et antal af sådanne rektangler. Rektanglerne rører ikke hinanden, men nogle rektangler kan godt være indeholdt i andre.

Spørgsmål a: Design en $O(n \log n)$ plane sweep algoritme til at finde antallet af rektangler.

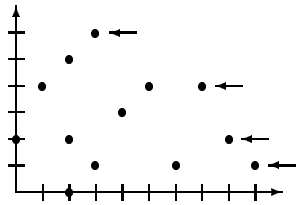
Skridtene i algoritmen skal forklares omhyggeligt. Det er ikke nødvendigt (eller ønskeligt), at der skrives et egentligt program.

Argumentér for, at tidskompleksiteten $O(n \log n)$ opnås. □

Opgave 2 (25%)

Et punkt i planen q *domineres af* et andet punkt p, hvis $p.x \geq q.x$ og $p.y \geq q.y$ (det antages, at der ikke findes punkter p og q, så $p.x = q.x$ og $p.y = q.y$).

Givet en endelig punktmængde S, så siges et punkt $p \in S$ at være *dominerende*, hvis det ikke domineres af noget andet punkt fra S. Vi lader $dom(S)$ betegne mængden af dominerende punkter i S. Nedenfor er $dom(S)$ udpeget af fire pile:



Nedenstående algoritme finder $dom(S)$. Vi antager følgende:

- S er sorteret lexikografisk. Dvs. at hvis $i < j$, så gælder:

$$S[i].x < S[j].x \vee (S[i].x = S[j].x \wedge S[i].y < S[j].y).$$

- D er i forvejen defineret til at være en liste af samme længde som S.

```
i, j = 0, 0
while i < len(S):
    while j > 0 and S[i].y >= D[j-1].y:
        j = j - 1
    D[j] = S[i]
    i, j = i + 1, j + 1
# Resultat i D[0], D[1], ..., D[j-1]
```

Spørgsmål a: Er det nødvendigt, at S er lexikografisk sorteret, eller er det tilstrækkeligt, at S er sorteret på x-koordinaten? Argumentér for dit svar. □

Spørgsmål b: Bevis, at algoritmen korrekt finder $dom(S)$.

Beviset kan være, men behøver ikke være, baseret på invariantteknikker. □

Spørgsmål c: Hvad er algoritmens tidskompleksitet? Argumentér for dit svar. □

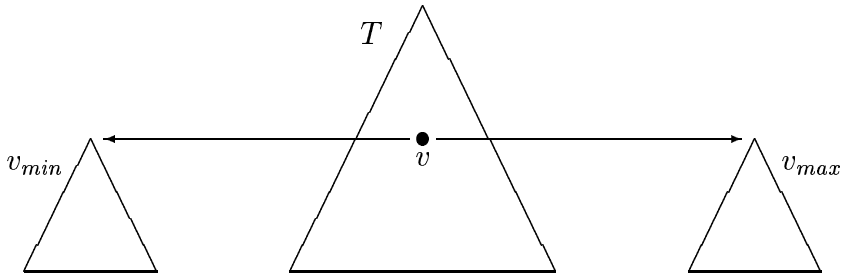
Spørgsmål d: Antag, at S ikke er sorteret. Design en $O(n \log n)$ del-og-hersk algoritme, der finder $dom(S)$.

I den basale del af algoritmen skal problemet opdeles i to cirka lige store halvdele, der løses rekursivt, hvorefter svarene kombineres til det samlede svar. □

Opgave 3 (25%)

I denne opgave antager vi, at data er i generel position (specialtilfælde kan altså ignoreres).

Vi beskriver en datastruktur, der er lavet udfra n punkter i planen. Først laves der et balanceret søgetræ T indeholdende alle punkterne ordnet efter x-koordinat. Til hver eneste knude v i dette træ associerer vi to prioritetssøgetræer, v_{min} og v_{max}. De indeholder begge en kopi af alle punkter i v's undertræ. Prioritetssøgetræet v_{min} er ordnet som et søgetræ på punkternes y-koordinat og som en hob på x-koordinaterne med mindste x-koordinater øverst. For v_{max} er den eneste forskel, at største x-koordinater er øverst. Dele af konstruktionen illustreres nedenfor.



Spørgsmål a: Forklar, hvordan man kan rapportere alle punkter i området $[x : x'] \times [y : y']$.

Vink: Brug de associerede træer til venstre- og højrebarn af v_{split}, hvor v_{split} er den knude i T, hvor søgning efter x og x' deles. □

Spørgsmål b: Gør rede for pladsforbrug, konstruktionstid og forespørgselstid. □

Opgave 4 (25%)

I denne opgave ser vi på n mus placeret under x-aksen og n oste placeret over x-aksen. Vi antager, at den samlede mængde af mus og oste er i generel position; f.eks. er der altså ikke tre ko-lineære elementer fra den samlede mængde, eller to mus (henholdsvis oste) med samme x-koordinat.

Vi vil gerne tildele præcis én ost til hver mus på en sådan måde, at musene med garanti ikke støder sammen, når de løber i en lige linie hen til deres ost. Vi kræver derfor følgende om tildelingen: for alle par af to mus, må de to liniestykker, der går fra de to mus til deres respektive tildelte oste, ikke skære hinanden.

Mere præcist er input en (usorteret) liste af mus og en (usorteret) liste af oste. En mus (henholdsvis ost) er et objekt (eller record) med et felt, `match`, hvori en reference til den tildelte ost (henholdsvis mus) skal skrives.

Spørgsmål a: En første idé til en algoritme kunne være at tildele den mus med i'te mindste x-koordinat den ost med i'te mindste x-koordinat. Vis, at denne algoritme ikke løser problemet. □

Spørgsmål b: Vis, at man ikke kan løse problemet hurtigere (asymptotisk), end man kan sortere n heltal. □

Det vides, at man givet et konveks hylster kan slette et punkt og retablere det konvekse hylster for de resterende punkter i amortiseret tid $O(\log n)$. Det kan gøres på en sådan måde, at kanter, som ikke fjernes fra den ordnede konveks-hylster-kantliste, ikke berøres. Eventuelle referencer til disse vil altså stadig være korrekte efter retableringen.

Spørgsmål c: Vis, f.eks. ved at udnytte ovenstående, at en tildeling kan laves i tid $O(n \log n)$. □