Skriftlig Eksamen
Datastrukturer og Algoritmer (DM02)

Institut for Matematik og Datalogi
Odense Universitet

Onsdag den 18. juni 1997, 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. Specielt må man gerne begrunde en påstand med at henvise til, at det umiddelbart følger fra et resultat i lærebogen (hvis dette altså er sandt!). 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, i det omfang det er muligt, besvare de efterfølgende spørgsmål og blot antage, at man har en løsning til de foregående spørgsmål.

Opgave 1 (35%)

Vi ser på et enmandsspil på et bræt af følgende type:

17	2	100	87	33	14
1	2	3	1	1	1

Man starter i feltet længst til venstre, og man skal bevæge sig ud over den højre ende. Den øverste række tal angiver prisen ved at besøge et felt, mens den nederste række tal angiver, hvor mange felter man *maksimalt* må springe til højre (men man behøver altså ikke springe så langt). Man behøver ikke nødvendigvis besøge det sidste felt. Hvis det andetsidste felt f.eks. angiver, at man må springe 3, kan man springe ud over højrekanten derfra. Det gælder om at finde prisen på en billigste tur fra venstrefeltet ud over højrekanten.

I første omgang antager vi, at kun tallene 1, 2 og 3 optræder i spring-delen af felterne. Vi lader n betegne længden af brættet (ovenstående har altså længde 6).

Følgende PYTHON program løser dette problem:

```
class F:

  def __init__(self, cost, jump):
    self.cost = cost
    self.jump = jump

Game = [F(17,1),F(2,2),F(100,3),F(87,1),F(33,1),F(14,1)]

def Best(G, i):
  if i >= len(G):
    return 0
  else:
    Min = Best(G, i+1)
    for j in range(2, Game[i].jump + 1):
      if Best(G, i+j) < Min:
        Min = Best(G, i+j)
    return Game[i].cost + Min

print Best(Game, 0)
```

Spørgsmål a: Hvilket værdi udskrives af PYTHON programmet ovenfor? □

Spørgsmål b: Hvad er kompleksiteten af `Best` som funktion af n? Argumentér for dit svar. □

Spørgsmål c: Anvend dynamisk programmering til at opnå et hurtigere program. Angiv det nye program. □

Spørgsmål d: Hvad bliver kompleksiteten af programmet, der anvender dynamisk programmering? Argumentér for dit svar. □

Spørgsmål e: Antag nu, at der i spring-delen af felterne kan stå vilkårlige positive heltal i stedet for tallene 1, 2 eller 3. Forklar, hvordan man kan løse problemet vha. Dijkstra's algoritme. □

Opgave 2 (25%)

Et *palindrom* er en streng, der er ens forfra og bagfra. F.eks. er ordene *abba*, *dad*, *cc* og *enafdemderredmedfane* palindromer.

Ordet *datalogi* er ikke et palindrom, men det indeholder en delstreng af længde 3, *ata*, som er et palindrom.

Spørgsmål a: Skriv en PYTHON funktion `pal`, der tager to argumenter: en tekststreng og et heltal, der er mindst 2. `pal(t, k)` skal udskrive `yes`, hvis `t` indeholder en delstreng af længde `k`, der er et palindrom, og `no` ellers. □

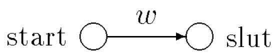
Spørgsmål b: Hvor mange sammenligninger mellem tegn i `t` foretager din PYTHON funktion, når den kaldes med `pal(t, k)`, og `t` har længde n? Argumentér for dit svar. □

Spørgsmål c: Skriv en PYTHON funktion `epal`, der tager en tekststreng som argument. `epal(t)` skal udskrive `yes`, hvis der eksisterer en delstreng i `t`, der er et palindrom, og som har længde mindst 2, og `no` ellers. Der skal anvendes så få sammenligninger mellem tegn i `t` som muligt. Argumentér for, at din løsning er korrekt. □

Opgave 3 (25%)

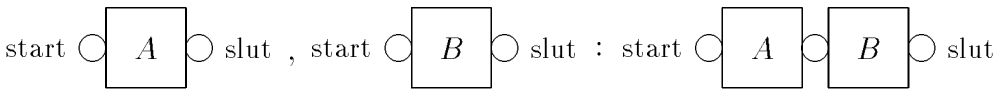
I denne opgave definerer vi en klasse af vægtede orienterede grafer, som vi kalder *ensrettede grafer*. Disse grafer har altid entydigt definerede start- og slutknuder.

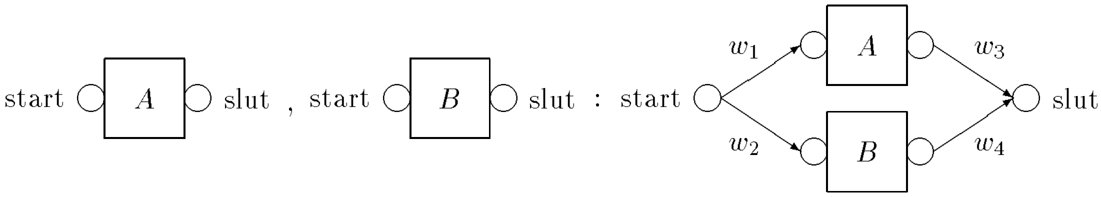
Den mindste ensrettede graf har formen:



Her er vægten w et eller andet positivt heltal.

For at lave større ensrettede grafer bruges følgende to regler, der begge tager to ensrettede grafer A og B og sætter dem sammen til én. Den første regel sætter A og B sammen ved at smelte startknuden for B sammen med slutknuden for A. Den anden regel laver en ny knude med kanter til startknuderne for A og B, og en ny knude, som slutknuderne for A og B sættes til at pege på.





Her er w_1, w_2, w_3 og w_4 igen vilkårlige positive heltal. Ovenstående definerer ensrettede grafer i den forstand, at kun grafer, der kan konstrueres udelukkende ved brug af ovenstående regler, kaldes ensrettede.

Vi lader n betegne antallet af knuder i en ensrettet graf og m antallet af kanter. Man kan bevise, at $m \in O(n)$.

Spørgsmål a: Forklar, hvordan man kan finde de korteste veje fra startknuden til alle andre knuder i en ensrettet graf i tid $O(n)$. Alle knuder har et felt, hvor man kan skrive, hvad den korteste vej til den pågældende knude er. (Dijkstra's algoritme er *ikke* hurtig nok.) $\square$

Spørgsmål b: Hvor mange kanter kan højst pege på en tilfældigt valgt knude i en ensrettet graf? Argumentér for dit svar. $\square$

Spørgsmål c: Forklar, hvordan man kan finde vægten af et letteste udspændende træ i tid $O(n)$. Man må gerne antage, at der er felter i knuderne, man kan skrive i. (Kruskal's og Prim's algoritmer er *ikke* hurtige nok.) $\square$

Opgave 4 (15%)

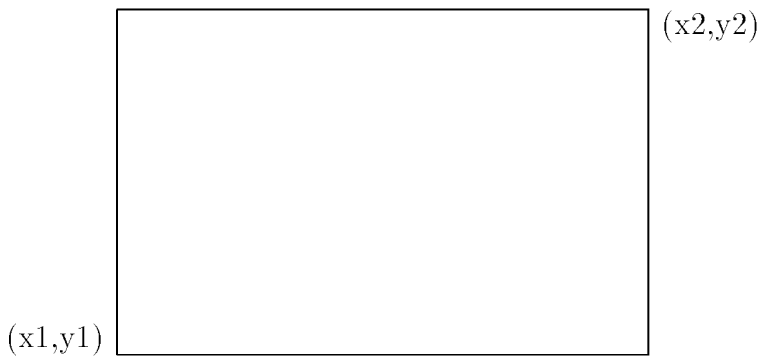
Vi vil gerne lave en datastruktur Points, der kan håndtere punkter (x, y) i planen. I første omgang vil vi gerne have operationerne: *Points*, *insert*, *delete* og *ismember*, hvor

`T = Points()`	laver en ny struktur
`T.insert(x,y)`	indsætter et nyt punkt (x,y) i strukturen
`T.delete(x,y)`	sletter punktet (x,y) fra strukturen
`T.ismember(x,y)`	afgør om punktet (x,y) findes i strukturen

`T = Points()` skal kunne udføres i konstant tid, mens resten af operationerne skal være amortiseret $O(\log n)$, hvor n er antallet af punkter i strukturen på det givne tidspunkt.

Spørgsmål a: Forklar, hvordan en struktur Points kan realiseres, så vi opnår de krævede kompleksiteter. □

Spørgsmål b: Nu vil vi tilføje en operation *contains*, der tager to punkter som argumenter. Kaldet `T.contains(x1,y1,x2,y2)` skal returnere alle de punkter i strukturen, der ligger i det rektangel, de to punkter definerer:



Argumenterne opfylder altid, at x1 < x2 og y1 < y2. Forklar, hvordan *contains* kan implementeres, uden at kompleksiterne på de andre operationer forværres. □