Institut for Matematik og Datalogi
Syddansk Universitet, Odense

26. oktober 2003
finie02@imada.sdu.dk

DM02 – Opgaver fra ugeseddel 7

Øvelsesopgaver 21/10, 22/10 og 23/10

1. Cormen et al. 15.2-3

   Brug substitutionsmetoden til at vise at løsningen til rekursionsligningen (15.11) er $\Omega(2^n)$

   $$P(n) = \begin{cases} \Theta(1) & , n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & , n \geq 2 \end{cases}$$

   Antager $P(n) \geq c \cdot 2^n$

   $$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &\geq \sum_{k=1}^{n-1} c2^k \cdot c2^{n-k} \\ &= c^2 \sum_{k=1}^{n-1} 2^k 2^{n-k} \\ &= c^2 \sum_{k=1}^{n-1} 2^{n-k+k} \\ &= c^2 \sum_{k=1}^{n-1} 2^n \\ &= c^2(n-1)2^n \\ &\geq c2^n \end{aligned}$$

   Hvor sidste ulighed gælder f.eks. ved valg af $c = 2$, for $n \geq 2$.

2. Cormen et al. 15.2-3
   Vis at en fuld parentessætning i et udtryk med n elementer har netop $n - 1$ par parenteser.

   Bevis ved induktion:
   Basis $n = 2$
   Et udtryk med to elementer, $(A \cdot B)$, har netop én parentes.
   $n - 1 = 2 - 1 = 1$ OK

   Induktion $n \geq 2$
   Vi ser på et udtryk med n elementer.

   $$A_1 \cdot A_2 \cdot \cdots \cdot A_n$$

For at sætte parenteser i/udregne dette udtryk, kan vi starte med at bestemme at den sidste udregning skal være mellem plads k og $k+1$.

$$(A_1 \cdot A_2 \cdot \cdots \cdot A_k \cdot A_{k+1} \cdot \cdots \cdot A_n)$$

Nu har vi delt udtrykket i to deludtryk: A_1 til A_k og A_{k+1} til A_n, som hver skal parentiseres/udregnes.

$$((A_1 \cdot A_2 \cdot \cdots \cdot A_k) \cdot (A_{k+1} \cdot \cdots \cdot A_n))$$

Ifølge induktionsantagelsen skal vi i hvert deludtryk bruge et antal parenteser svarende til antallet af elementer i udtrykket minus én. Hertil skal vi lægge den parentes som vi satte yderst, for lave beregningen mellem de to deludtryk.
Antal parenteser i alt:

$$(k-1) + (n-k-1) + 1 = n - 2 + 1 = n - 1$$

Vi ser at resultatet passer med vores antagelse, og kan konkludere at en fuld parentessætning for et udtryk med n elementer kræver netop $n-1$ parenteser.

3. 15.3-1
(udeladt med vilje)

4. 15.3-3
Overvej en variation af matrixmultiplikationsproblemet hvor målet er at sætte parenteser i en følge af matricer således at antallet af skalarmultiplikationer maksimeres.

   Har dette problem en optimal delstruktur?

   Svaret er Ja.
   Hvis vi ser på et udtryk som er parentiseret således at det bruger maksimalt antal af skalar multiplikationer (herefter: ”max mult”), kan vi nemt se at hvis stedet hvor de yderste parenteser er sat (f.eks. så opdelingen sker mellem A_k og A_{k+1}) giver max mult, er det en forudsætning at så er det en forudsætning at de indre parenteser har max mult.
   Ved modstrid: Hvis de indre parenteser ikke har max mult, er det muligt at ændre parentessætningen så de får max mult, og derved vil den yderste beregning give et højere tal, og dermed kunne den ikke allerede være max mult.

   Altså har problemet maksimalt delstruktur, fordi hvert delproblem skal optimeres for at vi får den optimale samlede løsning.

5. 15.4-1
(udeladt med vilje)

6. 15.4-2
(udeladt med vilje)

7. 15.4-3
Lav en ”memoized”version af LCS-Length som kører i $O(mn)$ tid.

   Den iterative udgave af LCS-Length er på side 353 i Cormen.
   ”Memoized version”er den rekursive algoritme, hvor vi gemmer delresultater i tabellen c. Hvis et delresultat skal bruges igen, returnerer vi bare værdien fra tabellen.
   Se f.eks. Memoized-Matrix-Chain s. 348.

```
Memoized-LCS-Length (X,Y)
  m <- Length[X]
  n <- Length[Y]

  for i <- 1 to m
    for j <- 1 to n
      c[i,j] <- Infinity

  for i <- 1 to m
    c[i,0] <- 0
  for j <- 1 to n
    c[0,j] <- 0

  LCS-Length-Memo(X,Y,c,m,n)
  return c[m,n]

LCS-Length-Memo(X,Y,c,i,j)
  if c[i,j] < Infinity
    return c[i,j]
  else if X[i] == Y[j]
    c[i,j] <- LCS-Length-Memo(X,Y,c,i-1,j-1) + 1
  else
    c[i,j] <- Max{
     LCS-Length-Memo(X,Y,c,i,j-1),
     LCS-Length-Memo(X,Y,c,i-1,j)
     }
  return c[i,j]
```

8. 15-6 Moving on a checkerboard
   Vi har et n x n skakbræt, hvor vi skal gå med en brik fra bunden til toppen af brættet. Fra et felt må vi kun gå opad på brættet, enten til feltet lige ovenover, eller til et af felterne som er på rækken ovenover og et skridt til højre eller til venstre. Dog er det ikke tilladt at gå 'ud af brættet'.
   Hver gang vi går fra et felt x til et felt y kan vi indkassere $p(x, y)$ dollars.

   Konstruer en algoritme som beregner ruten brikken skal gå fra bunden af brættet til toppen af brættet så vi indkasserer så mange dollars som muligt.

   Algoritme
   Felterne gennemløbes fra top til bund, række for række.
   For hvert felt x gemmes to værdier:

   - Højeste gevinst det er muligt at opnå når man er gået til dette felt.
     Beregnes ud fra de tre felter y,v,w i foregående række som $maxgevinst = maxp(y, x), p(v, x), p(w, x)$
   - Hvilket felt man kom fra for at opnå den højeste gevinst.
     Beregnes som $frafelt = y|p(y, x) = maxgevinst$

   Herefter kan startfeltet for ruten aflæses som feltet den maksimale gevinst i nederste række, og ruten der skal følges gennem brættet, kan findes ved at gå fra startfeltet til $frafelt$ og derfra notere felterne der gennemløbes ved hele tiden at gå videre til $frafelt$.

Køretid
Brætttet gennemløbes én række af gangen, og for hver felt laves $\Theta(3) = \Theta(1)$ beregninger af $p(x, y)$. Herefter gennemløbes den sidste række for at finde startfeltet i $O(n)$, og vi finder ruten op igennem brættet i $\Theta(n)$.
Hermed får vi køretid:
antal felter · antal beregninger + find startfelt + find rute

$$n^2 \cdot \Theta(1) \cdot \Theta(p) + O(n) + \Theta(n) \in \Theta(n^2) \cdot \Theta(p)$$

I tilfældet hvor $p(x, y)$ kan findes i konstant tid har vi altså en $\Theta(n^2)$ køretid.