

Esimerkki otteluvoiton todennäköisyys

A ja B pelaavat sarjan pelejä. Sarjan voittaja on se, joka ensin voittaa n peliä.

Yksittäisessä pelissä A voittaa todennäköisyydellä p ja B todennäköisyydellä $q = 1 - p$. Ottelutulokset ovat toisistaan riippumattomia. Millä todennäköisyydellä A voittaa sarjan?

Olkoon $P(i, j)$ todennäköisyys että A voittaa sarjan, kun lähdetään liikkeelle tilanteesta jossa A on jo voittanut $n - i$ peliä ja B voittanut $n - j$ peliä. (Toisin sanoen A tarvitsee vielä i voittoa ja B j voittoa.)

Halutaan laskea $P(n, n)$. Ilmeisesti pätee palautuskaava

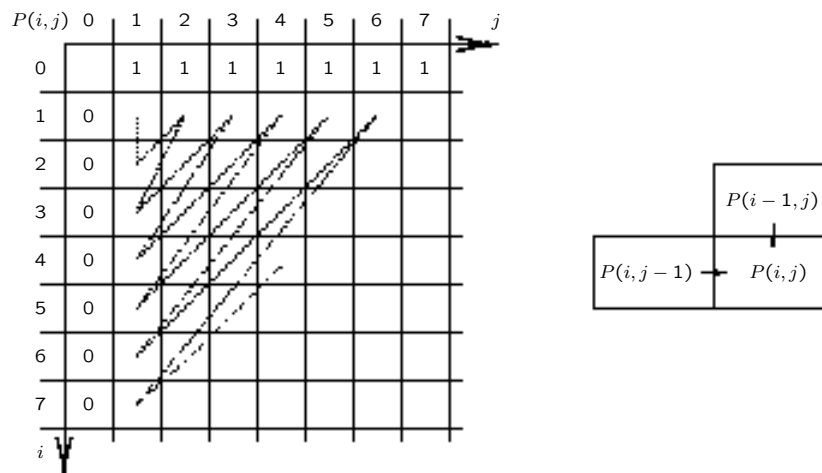
$$P(0, j) = 1 \quad j > 0$$

$$P(i, 0) = 0 \quad i > 0$$

$$P(i, j) = pP(i - 1, j) + qP(i, j - 1) \quad i > 0, j > 0$$

Siis arvot $P(i, j)$, missä $i + j = k$, voidaan laskea kun tunnetaan arvot $P(i', j')$, missä $i' + j' = k - 1$.

Taulukko P täytetään diagonaaleittain alkaen vasemmasta yläkulmasta.



Algoritmin ensimmäinen versio:

```

for  $k := 1$  to  $n$  do
     $P[0, k] := 1$ 
     $P[k, 0] := 0$ 
    for  $j := 1$  to  $n$  do
         $P[k - j, j] := pP[k - j - 1, j] + qP[k - j, j - 1]$ 
for  $k := n + 1$  to  $2n$  do
    for  $j := k - n$  to  $n$  do
         $P[k - j, j] := pP[k - j - 1, j] + qP[k - j, j - 1]$ 
return  $P[n, n]$ .

```

Tässä siis tarkastellaan diagonaalia $k = i + j$, $k = 1, 2, 3, \dots$. Kun $k > n$, ei enää tarvita koko diagonaalia (ainoastaan arvot $i, j \leq n$).

Muistinkäytön järkeistämiseksi merkitään ensin tilapäisesti diagonaalia k symbolilla D_k ; siis

$$D_k[j] = P[k - j, j].$$

Päivityssääntö

$$P[k - j, j] = pP[k - j - 1, j] + qP[k - j, j - 1]$$

tulee muotoon

$$D_{k-j}[j] = pD_{k-1}[j] + qD_{k-1}[j - 1].$$

Asiaa sen kummemmin ajattelematta voidaan todeta että riittää pitää kerrallaan muistissa vektorit D_{k-1} ja D_k , siis suunnilleen $2n$ eikä n^2 lukua.

Lähemmin tarkastelemalla havaitaan, että yksi vektori D riittää: kun vektoriin D lasketaan arvot $D_k(j)$ lähtien **suurista** j , uudet arvot voidaan kirjoittaa vanhojen päälle, koska arvon $D_k(j)$ laskemiseen ei tarvita arvoja $D_{k-1}(j')$ missä $j' > j$. Saadaan viimeistelty algoritmi

```
for  $k := 1$  to  $n$  do
     $P[0, k] := 1$ 
     $P[k, 0] := 0$ 
    for  $j := n$  downto  $1$  do
         $D[j] := pD[j] + qD[j - 1]$ 
for  $k := n + 1$  to  $2n$  do
    for  $j := n$  downto  $k - n$  do
         $D[j] := pD[j] + qD[j - 1]$ 
return  $D[n]$ .
```

Edellisissä esimerkeissä taulukointia käytettiin suoraviivaisesti rekursiivisen funktion laskemiseen. Mielenkiintoisempia ovat erilaiset [optimointisovellukset](#), joihin nimi "dynaaminen ohjelmointi" viittaa.

Esimerkki matriisitulon laskujärjestys.

On laskettavana matriisitulo $A_1 \times \dots \times A_n$ missä A_i on $p_i \times q_i$ -reaalilukumatriisi. Siis vaaditaan $q_i = p_{i+1}$, mutta muuten dimensiot voivat olla mielivaltaisia.

Kun kerrotaan (naiivilla menetelmällä) $r \times s$ -matriisi $s \times t$ -matriisilla, suoritetaan rst kertolaskua ja tulos on $r \times t$ -matriisi.

Tarkastellaan esimerkkinä tulon ABC laskemista kun A on 10×100 -matriisi, B on 100×5 -matriisi ja C on 5×50 -matriisi, Matriisitulo on liitännäinen, joten laskun tulos ei riipu siitä lasketaanko $(AB)C$ vai $A(BC)$. Laskenta-aika sen sijaan riippuu:

$$(AB)C : 10 \cdot 100 \cdot 5 + 10 \cdot 5 \cdot 50 = 7500$$

$$A(BC) : 100 \cdot 5 \cdot 50 + 10 \cdot 100 \cdot 50 = 75000$$

Seuraavassa esitetään taulukointiin perustuva menetelmä optimaaliset laskujärjestyksen löytämiseksi.

Olkoon optimaalisessa järjestyksessä koko tulon viimeisenä laskettavaksi tuleva kertolasku $(A_1 \dots A_k) \cdot (A_{k+1} \dots A_n)$. Siis kaikkiaan tarvittavien kertolaskujen lukumäärä on

$$x + y + p_1 q_k q_n$$

missä x ja y ovat tulojen $A_1 \dots A_k$ ja $A_{k+1} \dots A_n$ laskemiseen tarvittavien kertolaskujen määrät.

Ongelmana on, että mikään ilmeinen tapa valita viimeisen operaation indeksi k jollain heuristiikalla suoraan lukuja (p_i, q_i) tarkastelemalla ei tunnu toimivan. (Kokeile!)

Toisaalta kaikkia mahdollisia laskujärjestyksiä ei voi yksitellen tutkia, koska niitä on yhtä monta kuin n -lehtisiä täydellisiä binaaripuita eli $\frac{1}{n} \binom{2n-2}{n-1}$ (Catalanin luvut; HT).

Olkoon $m(i, j)$ pienin määrä kertolaskuja matriisitulon $A_i A_{i+1} \dots A_j$ laskemiseksi. Halutaan siis tietää $m(1, n)$. Hieman yllättäen ratkaisuna ongelmaan on laskea $m(i, j)$ kaikille $1 \leq i, j \leq n$.

Edellä esitetyn perusteella pätee

$$\begin{aligned} m(i, j) &= 0 \quad \text{jos } j \leq i \\ m(i, j) &= \min_{1 \leq k \leq j-1} (m(i, k) + m(k + 1, j) + p_i q_k q_j) \quad \text{muuten.} \end{aligned}$$

Olkoon vielä $s(i, j)$ se indeksi k joka minimoi arvon $m(i, k) + m(k + 1, j) + p_i q_k q_j$. Siis optimaalisessa laskujärjestyksessä lasketaan

$$A_i \dots A_j = (A_1 \dots A_{s(i,j)}) \cdot (A_{s(i,j)+1} \dots A_j).$$

Jos s on annettu taulukossa, niin seuraava rekursiivinen funktio laskee tulon $A_1 \dots A_n$ suorittamalla optimaaliset $m(1, n)$ kertolaskua:

```
LaskeTulo(i,j):
  if  $i = j$  then return  $A_i$ 
  else
     $X := \text{LaskeTulo}(i, s[i, j])$ 
     $Y := \text{LaskeTulo}(s[i, j] + 1, j)$ 
    return  $XY$ 
```

Taulukot m ja s lasketaan siis seuraavasti:

```
for  $j := 1$  to  $n$  do  $m[j, j] := 0$   
for  $t := 1$  to  $n - 1$  do  
    % tarkastellaan tuloja  $A_i \dots A_{i+t}$   
    for  $i := 1$  to  $n - t$  do  
         $j := i + t$   
         $m[i, j] := +\infty$   
        for  $k := i$  to  $j - 1$  do  
             $r := m[i, k] + m[k + 1, j] +$   
                 $p[i]p[k + 1]p[j + 1]$   
            if  $r < m[i, j]$  then  
                 $m[i, j] := r$   
                 $s[i, j] := k$   
            end if  
        end for  
    end for  
end for
```

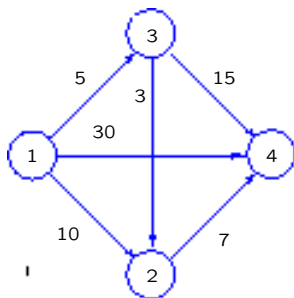
Aikavaativuus on ilmeisesti $\Theta(n^3)$.

Floydin-Warshallin algoritmi: esimerkkinä verkko-ongelmasta tarkastelemme *kaikkien* lyhimpien polkujen löytämistä.

On annettu suunnattu painotettu verkko $G = (V, E, \ell)$ missä $\ell: E \rightarrow \mathbf{R}^+$. Merkintöjen yksinkertaistamiseksi oletetaan $V = \{1, \dots, n\}$ ja laajennetaan ℓ siten, että $\ell(i, i) = 0$ kaikilla i ja muuten $\ell(i, j) = +\infty$ jos $(i, j) \notin E$.

Tehtävänä on määrittää $n \times n$ -taulukko D missä $D[i, j]$ on lyhimmän polun pituus solmusta i solmuun j . (Polun pituus on tässä kaarten painojen summa.)

Ongelma ratkeaa taulukoimalla välivaiheita D_k , $k = 0, \dots, n$, missä $D_k[i, j]$ on lyhimmän sellaisen polun pituus, joka päätepisteitä lukuunottamatta ei käy missään solmuista $k + 1, \dots, n$. Siis $D_0[i, j] = \ell(i, j)$ ja $D_n = D$.



$$D[1, 4] = 5 + 3 + 7 = 15$$

$$D_0[1, 4] = 30 \quad D_1[1, 4] = 30$$

$$D_2[1, 4] = 17 \quad D_3[1, 4] = 15$$

Jos ei päde $D_k[i, j] = D_{k-1}[i, j]$, niin lyhin polku solmusta i solmuun j vain solmuja $1, \dots, k$ käyttäen tosiaan käy solmussa k . Tällöin tämä lyhin polku tietysti muodostuu kahdesta osasta:

1. lyhin polku solmusta i solmuun k vain solmuja $1, \dots, k - 1$ käyttäen ja
2. lyhin polku solmusta k solmuun j vain solmuja $1, \dots, k - 1$ käyttäen.

Saadaan siis palautuskaava

$$D_0[i, j] = \ell(i, j)$$

$$D_k[i, j] = \min \{ D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j] \}.$$

(Toinen tärkeä sovellus samasta ideasta on verkon transitiivisen sulkeuman laskeminen. Tällöin halutaan $D[i, j] = 1$ jos solmusta i ylipäänsä on polku solmuun j , ja $D[i, j] = 0$ muuten. Tällöin kaavaksi tulee

$$D_0[i, j] = 1 \quad \text{jos } (i, j) \in E$$

$$D_0[i, j] = 0 \quad \text{muuten}$$

$$D_k[i, j] = \max \{ D_{k-1}[i, j], D_{k-1}[i, k] \cdot D_{k-1}[k, j] \}.$$

Tästä saatavaa algoritmia nimitetään [Warshallin algoritmiksi](#).)

Lopulta algoritmi saadaan seuraavaan muotoon:

```
 $D := \ell$   
for  $k := 1$  to  $n$  do  
  for  $i := 1$  to  $n$  do  
    for  $j := 1$  to  $n$  do  
       $D[i, j] := \min \{ D[i, j], D[i, k] + D[k, j] \}$ 
```

Tässä on otettu huomioon, että $D_k[i, k] = D_{k-1}[i, k]$ ja $D_k[k, j] = D_{k-1}[k, j]$, joten selvittää yhdellä taulukolla D . Aikavaativuus on $\Theta(n^3)$ ja tilavaativuus $\Theta(n^2)$.

Jos halutaan myös itse lyhimmät polut eikä pelkkiä niiden pituuksia, voidaan algoritmiin esim. liittää taulukko $S[i, j]$ joka kertoo, mihin solmusta i pitää mennä, että päästään solmuun j .

Ahneet algoritmit (greedy algorithms)

Idea: Muodostetaan jono osaratkaisuja S_0, S_1, \dots, S_m missä S_m on koko ongelman ratkaisu, S_0 on jokin triviaali lähtöaskel, ja S_i saadaan osaratkaisusta S_{i-1} tekemällä siihen jokin hyvältä näyttävä ("lokaalisti optimaalinen") lisäys.

Ongelman rakenteesta riippuu, johtavatko lokaalisti optimaaliset askelet aina koko ongelman optimaaliseen ratkaisuun.

Esimerkki rahanvaihto: annettu euromäärä esitettävä pienimmällä määrällä seteleitä ja kolikoita. Esim.

$$83 \text{ euroa} = (50 + 20 + 10 + 2 + 1) \text{ euroa.}$$

Olkoon $x < 100$ rahamäärä ja $A = \{50, 20, 10, 5, 2, 1\}$. Ahne ratkaisu:

1. $S := \emptyset$ (monijoukko)
2. Toista kunnes $\sum_{a \in S} a = x$: aseta $S := S \cup \{z\}$ missä $z \in A$ on suurin jolla $\sum_{a \in S} a + z \leq x$.

Jos käytössä olevien rahojen arvot on valittu sopivasti (niinkuin ne yleensä ovat), algoritmi antaa optimaalisen vastauksen. Tämä ei kuitenkaan ole itsestään selvää. Esim. jos puuttuisi 10 ja 5 euron rahat, saataisiin $60 = 50 + 2 + 2 + 2 + 2 + 2$ eikä $60 = 20 + 20 + 20$.

Dijkstran algoritmi: esimerkki ahneesta polunetsimisestä verkossa

On annettu suunnattu painotettu verkko $G = (V, E, \ell)$ kuten Floydin-Warshallin algoritmossa, mutta nyt on lisäksi annettu lähde $v_0 \in V$ ja halutaan tietää ainoastaan ne lyhimmät polut joiden alkusolmu on v_0 .

Olkoon siis $\delta(v)$ lyhimmän polun pituus solmusta v_0 solmuun v .

Ratkaisun perusajatus on pitää yllä sellaista joukkoa $S \subseteq V$, että oikea arvo $\delta(v)$ tiedetään kaikille $v \in S$. Aluksi $S = \{v_0\}$, ja kun on saavutettu $S = V$ ratkaisu on valmis. Joukkoa S laajennetaan ahneesti lisäämällä siihen aina sellainen alkio joka näyttäisi olevan mahdollisimman lähellä lähdettä v_0 .

Laskennan aikana pidetään yllä taulukkoa D , jonka tulkinta on seuraava:

- jos $v \in S$ niin $D[v] = \delta(v)$
- muuten $D[v]$ on lyhimmän sellaisen polun pituus solmusta v_0 solmuun v , joka päätepisteitään lukuunottamatta sisältää vain joukon S solmuja.

Siis kun lopulta $S = V$, taulukko D sisältää halutun tuloksen.