

Loppu seuraa suoralla laskulla:

$$\begin{aligned}
 T_{\text{ave}}^{\text{TR}} &= \sum_{j=1}^n p_j (1 + \sum_{i=1}^n b'(i, j)) \\
 &= 1 + \sum_{1 \leq i < j \leq n} (p_j b'(i, j) + p_i b'(j, i)) \\
 &= 1 + \sum_{1 \leq i < j \leq n} (p_j b'(i, j) + p_i (1 - b'(i, j))) \\
 &\leq 1 + \sum_{1 \leq i < j \leq n} \left((p_j - p_i) \frac{p_i}{p_i + p_j} + p_i \right) \\
 &= 1 + 2 \sum_{1 \leq i < j \leq n} \frac{p_i p_j}{p_i + p_j} \\
 &= T_{\text{ave}}^{\text{MF}}
 \end{aligned}$$

eli väite $T_{\text{ave}}^{\text{TR}} \leq T_{\text{ave}}^{\text{MF}}$ pätee.

(Keskimääräisen tapauksen analyysi loppuu tältä erää tähän. Palaamme sanakirjaongelmaan vielä tasoitetun analyysin yhteydessä.)

2.4 Tasoitettu analyysi (amortized analysis)

Esimerkki Pino, jossa tavalliset Push- ja Pop sekä MultiPop(k) (aluksi pino tyhjä):

```
MultiPop( $k$ ):  
1.    $i := k$   
2.   while not Empty and  $i > 0$  do  
3.        $r :=$  Pop  
4.        $i := i - 1$   
       end while  
5.   return  $r$ 
```

Operaatioiden kustannukset (kertaluokka):

| | |
|-----------------|------------------------------------|
| Push | 1 |
| Pop | 1 |
| MultiPop(k) | $1 + \min\{k, \text{pinon koko}\}$ |

Suoritettaessa n operaatiota pino koko voi olla $\Theta(n)$, joten pahimmassa tapauksessa yksittäinenkin operaatio (nim. MultiPop(n)) voi viedä ajan $\Theta(n)$.

Kuitenkin **tasoitettu aikavaatimus** on $\Theta(1)$ /operaatio, sillä erityisesti jokaista MultiPop(k)-operaation sisällä suoritettavaa Pop-operaatiota kohti on suoritettu yksi Push, joten kokonais-Pop-määrä on $O(n)$.

Tehdään analyysi täsmällisemmin ensin **tilinpitomenetelmällä** ja sitten **potentiaalimenetelmällä**.

Analyysi kirjanpitomenetelmällä

Jokaiselle operaatiolle määritellään **tasoitettu kustannus**, joka on puhtaasti laskennallinen apukeino ja voidaan valita analyysin kannalta sopivalla tavalla.

Jokaisella operaatiolla on myös **todellinen kustannus** joka on sama (tai samaa kertaluokkaa tms.) kuin sen todellinen suoritus aika.

Kunkin operaation yhteydessä ajatellaan tehtäväksi seuraavat lisätoimet:

1. algoritmi saa "palkkioksi" suoritettavan operaation tasoitettun kustannuksen verran rahaa
2. algoritmi voi "tallettaa" osan "palkkiosta" tietorakenteeseen, ja "nostaa" tietorakenteesta aiempia talletuksia
3. askelista 1 ja 2 jääneellä "käteisellä" maksetaan operaation todellinen kustannus

Idea: jos aluksi tietorakenteessa ei ole talletuksia, ja jokaisen operaation todellinen kustannus kyetään maksamaan, niin

$$\sum_{\text{operaatiot}} (\text{tod. kustannus}) \leq \sum_{\text{operaatiot}} (\text{tas. kustannus}).$$

Esimerkki Pino-operaatiot kirjanpitomenetelmällä.
Valitaan seuraavat tasoitetut kustannukset ("tulot"):

| | |
|-------------|------------|
| Push | 2 yksikköä |
| Pop | 0 yksikköä |
| MultiPop(k) | 1 yksikkö |

Jokaisen pinossa olevan alkion yhteydessä pidetään yksi yksikkö rahaa.

Todelliset kustannukset ("menot") on *todettu* aiemmin:

| | |
|-------------|---------------------------------------|
| Push | 1 |
| Pop | 1 |
| MultiPop(k) | $1 + \min \{ k, \text{pinon koko} \}$ |

Push: tulot 2 yksikköä; menot 1 yksikkö; talletetaan 1 yksikkö

Pop: tulot 0 yksikköä; menot 1 yksikkö; nostetaan 1 yksikkö

MultiPop(k): tulot 1 yksikkö; menot $1 + \min \{ k, \text{pinon koko} \}$ yksikköä; nostetaan $\min \{ k, \text{pinon koko} \}$ yksikköä

Selvästi aina

$$\text{menot} + \text{panot} \leq \text{tulot} + \text{nostot}$$

joten koska alkusaldo on nolla (eli pino aluksi tyhjä) ja loppusaldo on ei-negatiivinen,

$$\text{kokonaismenot} \leq \text{kokonaistulot}$$

eli koko operaatiojonolle

$$\text{todellinen aikavaatimus} \leq 2a + c$$

missä a , b ja c ovat Push-, Pop- ja MultiPop-operaatioiden lukumäärät. Erityisesti siis n operaatiota vie ajan $O(n)$.

Analyysi potentiaalimenetelmällä

Jokaiseen tietorakenteen tilaan D liitetään potentiaali $\Phi(D)$.

Olkoon D_i tietorakenteen tila kun on suoritettu i operaatiota. Operaation numero i tasoitettu kustannus on

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

missä c_i on operaation todellinen kustannus. Siis

$$\begin{aligned} \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n c_i + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0). \end{aligned}$$

Jos voidaan osoittaa $\Phi(D_i) \geq \Phi(D_0)$ kaikilla i , niin operaatiojonon tasoitettu kustannus on yläraja todelliselle kustannukselle.

Esimerkki Pino-operaatiot potentiaalimenetelmällä

Valitaan $\Phi(D) =$ pinon D koko.

Siis $\Phi(D_0) = 0$ ja $\Phi(D_i) \geq 0$ kaikilla i , joten

(tod. aikavaatimus) \leq (tas. aikavaatimus).

Tasoitettut kustannukset:

Push: $c_i = 1$, $\Phi(D_{i-1}) = m$, $\Phi(D_i) = m + 1$ jollain m ,
joten

$$\hat{c}_i = 1 + (m + 1) - m = 2.$$

Pop: $c_i = 1$, $\Phi(D_{i-1}) = m$, $\Phi(D_i) = m - 1$ jollain m ,
joten

$$\hat{c}_i = 1 + (m - 1) - m = 0.$$

MultiPop(k): $c_i = 1 + \min\{k, m\}$, $\Phi(D_{i-1}) = m$,
 $\Phi(D_i) = \max\{0, m - k\}$ jollain m , joten

$$\hat{c}_i = 1 + \min\{k, m\} + \max\{0, m - k\} - m = 1.$$

Siis

$$\text{(tas. aikavaatimus)} = 2a + c$$

missä a , b ja c ovat Push-, Pop- ja MultiPop-operaatioiden lukumäärät. Erityisesti siis n operaatiota vie ajan $O(n)$.

Sanakirjaongelma

Teemme tasoitetun analyysin potentiaalimenetelmällä (Sleator & Tarjan, CACM 1985)

Operaatiot ja perustoteutus listan L avulla samat kuin keskimääräisen tapauksen analyysissä. Operaatioiden kustannukset perustoteutuksessa:

| operaatio s_t | kustannus c_t |
|-----------------|--|
| access(z) | k missä $z = L[k]$ |
| insert(z) | $l + 1$ missä l on listan L pituus |
| delete(z) | k missä $z = L[k]$ |

Jos suoritetaan listan uudelleenjärjestelyjä, voi tästä tulla lisäkustannuksia.

Seuraavassa **vaihto** tarkoittaa kahden **peräkkäisen** alkion sijaintien vaihtamista keskenään. Kohtuullinen malli vaihtokustannuksille:

Ilmainen vaihto: juuri haetun tai lisätyn alkion siirtäminen kohti listan keulaa; kustannus 0

Maksulliset vaihdot: mikä tahansa muu vaihto; kustannus 1

Aiemmin käsitellyt algoritmit eivät tee maksullisia vaihtoja; esim. TR tekee yhden ja MF $k - 1$ ilmaista vaihtoa hakua kohti.

Seuraavassa A on mielivaltainen algoritmi ja s mielivaltainen operaatiojono.

Kun algoritmilla A suoritetaan operaatiojono s , merkitään

$C_A(s)$ kokonaiskustannus **lukuunottamatta** mahdollisia vaihtoja

$X_A(s)$ maksullisten vaihtojen lukumäärä

$F_A(s)$ ilmaisten vaihtojen lukumäärä

Siis algoritmin A kokonaiskustannus operaatiojonolla s on $C_A(s) + X_A(s)$.

Lause Kaikilla algoritmeilla A ja m operaation jonoilla s pätee

$$\begin{aligned} C_{MF}(s) &\leq 2C_A(s) + X_A(s) - F_A(s) - m \\ &\leq 2(C_A(s) + X_A(s)). \end{aligned}$$

Huomioita:

- pätee erityisesti jos A on valittu siten että se on optimaalinen juuri jonolle s
- siis vaikka jono s tunnettaisiin ennakoita ja tehtäisiin mielivaltaisia optimointeja, voitetaan yksinkertainen MF-heuristiikka korkeintaan kertoimella 2
- keskimääräisille vaatimuksille tästä seuraa

$$T_{\text{ave}}^{MF}(m) \leq 2T_{\text{ave}}^A(m)$$

millä tahansa jakaumalla

Todistus Kiinnitetään A ja $s = (s_1, \dots, s_m)$.

Olkoon L_t algoritmin MF lista ja L'_t algoritmin A lista, kun on suoritettu operaatiot (s_1, \dots, s_{t-1}) .

Kun L ja L' ovat kaksi listaa, joissa kummassakin on täsmälleen samat l alkioita, olkoon $\Phi(L, L')$ listojen L ja L' välisten **inversioiden** lukumäärä eli niiden alkioparien määrä, jotka ovat listoissa L ja L' eri järjestyksessä.

Valitaan potentiaaliksi $\Phi(L_t, L'_t)$ ja tarkastellaan MF-algoritmin tasoitettua kustannusta

$$\hat{c}_t = c_t + \Phi(L_t, L'_t) - \Phi(L_{t-1}, L'_{t-1})$$

missä c_t on operaation s_t todellinen kustannus MF-algoritmillä.

Aluksi listat ovat tyhjä joten $\Phi(L_0, L'_0) = 0$. Aina $\Phi(L_t, L'_t) \geq 0$, joten

$$T^{\text{MF}}(s) = \sum_{t=1}^m c_t \leq \sum_{t=1}^m \hat{c}_t$$

aiemmin esitetyn periaatteen mukaan.

Määritellään nyt algoritmin A operaatioon s_t liittyvät suureet

- a_t todellinen kustannus **lukuunottamatta** mahdollisia vaihtoja
- x_t maksullisten vaihtojen lukumäärä
- f_t ilmaisten vaihtojen lukumäärä

Osoitamme kaikille operaatioille

$$\hat{c}_t \leq 2a_t + x_t - f_t - 1 \quad (1)$$

mistä seuraa

$$\begin{aligned} T^{\text{MF}}(s) &\leq \sum_{t=1}^m \hat{c}_t \\ &\leq \sum_{t=1}^m (2a_t + x_t - f_t - 1) \\ &= 2C_A(s) + X_A(s) - F_A(s) - m \end{aligned}$$

eli väite.

Olkoon L_t'' algoritmin A lista kun on suoritettu operaatiot (s_1, \dots, s_n) **lukuunottamatta** operaatioon s_t mahdollisesti liittyviä vaihtoja. Kirjoitetaan

$$\hat{c}_t = c_t + \Phi(L_t, L_t'') - \Phi(L_{t-1}, L_{t-1}') + \Phi(L_t, L_t') - \Phi(L_t, L_t'').$$

Todistamme epäyhtälön (1) kahdessa osassa:

$$c_t + \Phi(L_t, L_t'') - \Phi(L_{t-1}, L_{t-1}') \leq 2a_t - 1 \quad (2)$$

$$\Phi(L_t, L_t') - \Phi(L_t, L_t'') \leq x_t - f_t. \quad (3)$$

Kohta (3) on helppo:

- maksullinen vaihto lisää korkeintaan yhden inversion
- maksuton vaihto poistaa yhden inversion, koska se siirtää listalla L_t'' alkiota kohti keulaa missä se jo on listalla L_t

Siis (3) pätee. Todistamme epäyhtälön (2) erikseen eri operaatiotyypeille.

Tapaus A: $s_t = \text{access}(z)$. Siis $L_t'' = L_{t-1}'$.

Olkoon $z = L_{t-1}[k] = L_{t-1}'[i]$. Siis $c_t = k$ ja $a_t = i$.

Olkoon y niiden alkioiden lukumäärä, jotka ovat ennen alkiota z listassa L_{t-1} mutta alkion z jälkeen listassa L_{t-1}' .

Siis $k - y - 1$ alkiota on ennen alkiota z kummassakin listassa.

Alkion z siirtäminen listan L_{t-1} kärkeen purkaa y inversiota mutta luo $k - y - 1$ uutta, joten

$$\begin{aligned}c_t + \Phi(L_t, L_t'') - \Phi(L_{t-1}, L_{t-1}') &= k + (k - y - 1) - y \\ &= 2(k - y) - 1.\end{aligned}$$

Nyt $k - y - 1 \leq i - 1$, koska alkion z edeltä listassa L_{t-1}' pitää löytyä ainakin $k - y - 1$ alkiota. Siis

$$2(k - y) - 1 \leq 2i - 1 = 2a_t - 1.$$

Tapaus B: $s_t = \text{insert}(z)$.

Oletetaan toteutuksesta, että jos alkio z on jo listassa, toimitaan kuten tapauksessa $\text{access}(z)$.

Muuten MF lisää alkion listan loppuun ja välittömästi vaihtaa sen listan keulaan.

Myös algoritmi A lisää alkion listan loppuun ja tekee sitten mahdollisesti vaihtoja.

Jos alkio on jo listassa, analyysi palautuu tapaukseen A. Muuten olkoon listassa l alkioita, joten

$$c_t = a_t = l + 1.$$

Kun MF siirtää alkion z listansa keulaan, syntyy l inversiota, joten

$$\Phi(L_t, L_t'') - \Phi(L_{t-1}, L_{t-1}') = l.$$

Siis

$$\begin{aligned} c_t + \Phi(L_t, L_t'') - \Phi(L_{t-1}, L_{t-1}') &= l + 1 + l \\ &= 2(l + 1) - 1 \\ &= 2a_t - 1. \end{aligned}$$

Tapaus C: $s_t = \text{delete}(z)$.

Olkoon $z = L_{t-1}[k] = L'_{t-1}[i]$. Siis $c_t = k$ ja $a_t = i$.

Olkoon y niiden alkioiden lukumäärä, jotka ovat ennen alkiota z listassa L_{t-1} mutta alkion z jälkeen listassa L'_{t-1} . Siis $k - y - 1$ alkiota on ennen alkiota z kummassakin listassa.

Alkion z poistaminen purkaa ainakin y inversiota. Uusia ei tietenkään synny, joten

$$\Phi(L_t, L''_t) - \Phi(L_{t-1}, L'_{t-1}) \leq -y.$$

Nyt $k - y - 1 \leq i - 1$, koska alkion z edeltä listassa L'_{t-1} pitää löytyä ainakin $k - y - 1$ alkiota. Siis

$$c_t + \Phi(L_t, L''_t) - \Phi(L_{t-1}, L'_{t-1}) \leq k - y \leq i = a_t \leq 2a_t - 1.$$

□

Vastaava tulos **ei** päde heuristiikoille TR ja FC: On olemassa sellaiset m operaation jonot n alkiolle, että

$$T^{\text{MF}}(s) = \Theta(m) \quad \text{ja} \quad T^{\text{TR}}(s) = \Theta(nm)$$

ja

$$T^{\text{MF}}(s') = \Theta(m) \quad \text{ja} \quad T^{\text{FC}}(s') = \Theta(nm).$$

2.5 Tilavaativuuden analysointi

Yleensä tarkastellaan [työtilaa](#), siis tilavaativuutta poislukien syötteen ja tulosteen vaatima tila:

$$S(x) = \text{työtilan tarve syötteellä } x$$

Vastaavasti $S_{\max}(n)$ ja $S_{\text{ave}}(n)$.

Jos algoritmi varaa tilaa dynaamisesti, tilavaativuus on ilmeisesti suurin kerralla varattuna oleva muistin määrä.

Eryteisesti rekursiivisilla proseduureilla tilavaativuus on [aktivaatitietuepinon](#) maksimikoko.

Proseduurin

```
 $P(X, n)$ :  
  var  $v_1[n], v_2[n], \dots, v_m[n]$   
  begin  
    ...  
     $P(X_1, n_1)$   
    ...  
     $P(X_2, n_2)$   
    ...  
     $P(X_k, n_k)$   
    ...  
  end
```

tilavaativuudelle S pätee

$$S(X, n) = \Theta\left(1 + \sum_{i=1}^m |v_i[n]| + \max_{1 \leq i \leq k} S(X_i, n_i)\right).$$

Jos muuttujat $v_i[n]$ vievät vakiotilan, tilavaativuus on Θ (rekursion maksimisyvyys).