AMORTIZIRANA ANALIZA

- koristi se za ocjenu prosječnog vremena izvršavanja niza operacija nad nekom strukturom podataka, po operaciji
- u nizu n operacija može se dogoditi da je jedna, ili nekoliko njih, iznimno „skupa" (skupe), dok su sve ostale jeftine
- što je s prosjekom?
- amortizirana analiza garantira prosječno vrijeme u najgorem slučaju

$T(n)$ = ukupna cijena niza od n operacija $\quad n \in \mathbb{N}_0$

$T(n)/n$ = prosječna cijena po operaciji

Agregatna analiza

odnosi se na svaku operaciju, iako se u nizu od n operacija može nalaziti nekoliko tipova operacija

Operacije na stogu

PUSH (S, x) - stavlja x na stog S - $O(1)$

POP (S) - uzima element s vrha stoga S - $O(1)$

MULTIPOP (S, k) - uzima k elemenata s vrha stoga S. U koliko S sadrži manje od k elemenata, ova metoda prazni stog

```
MULTIPOP (S, k)
    while not STACK-EMPTY (S) and k > 0
        POP (S)
        k ← k - 1
    end while
```

PUSH(S, x) - 1

- POP(S) - 1
- MULTIPOP(S, k) - $\min(s, k)$ gdje je s broj elemenata na stogu S

Analiza vremena izvršavanja n operacija u nizu na stogu veličine n

- Vrijeme izvršavanja MULTIPOP operacije je $O(n)$ (u najgorem slučaju)
- n operacija $\Rightarrow$ $T(n) = O(n^2) \Rightarrow T(n)/n = O(n)$
- nije oštra ?
- Zašto

Agregatna analiza

- broj POP operacija (uključujući i MULTIPOP operacije) je u najgorem slučaju jednak broju PUSH operacija
- broj PUSH operacija na stogu veličine n je najviše n

Zaključak: $T(n) = O(n) \rightarrow T(n)/n = O(n)/n = O(1)$

$T(n)/n$ je amortizirano vrijeme izvršavanja „cijena"

Inkrementiranje binarnog brojača

- brojač je spremljen kao binarom broj koji se sastoji od k bitova u polju

$A[0, \ldots, k-1]$

vrijednost brojača:

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

Inicijalno $x = 0$

operacija: dodavanje $1 \pmod{2^k}$

Vrijednost	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Ukup cijena
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

```
INCREMENT (A)
  i ← 0
  while i < A.length and A[i] == 1
  |   A[i] ← 0
  |   i ← i+1
  end while
  if i < A.length
      A[i] ← 0
```

magnice vrijednosti od 0 do $2^k - 1$

Zadatak: Ocijeniti vrijeme izvršavanja u najgorem slučaju niza od n INCREMENT operacija.

+ jedna INCREMENT operacija : $O(k)$

$\Rightarrow T(n) = O(nk) \Rightarrow T(n)/n = O(k)$

Možemo li bolje? (nismo u svakom pozivu promijenili svaki od k bitova)

A[k]	A[k-1]	.....	A[3]	A[2]	A[1]	A[0]
			Svaka	Svaka	Svaka	Svaka
			osma	četvrta	druga	operacija
			8	4	2	1
			2^3	2^2	2^1	2^0

Ukupan broj okretanja bitova u n poziva:

$$T(n) = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \sum_{i=0}^{k-1} \frac{1}{2^i} < n \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = n \cdot \frac{1}{1-\frac{1}{2}}$$

$$= 2n = O(n)$$

$\Rightarrow T(n)/n = O(n)/n = O(1) \Rightarrow \boxed{T(n)/n = O(1)}$

Metoda kreditiranja

- Ideja: u nizu od n operacija svakoj pridružimo amortiziranu cijenu kojom ćemo „kreditirati" vrijeme izvršavanja budućih operacija
- Kako?

Operacije na stogu

Stvarna cijena		Amortizirane cijene	
PUSH	1	PUSH	2
POP	1	POP	0
MULTIPOP	$\min(k, S)$	MULTIPOP	0

Stvarna cijena i-te operacije: c_i

Amortizirana cijena i-te operacije: $\hat{c}_i$

Mora biti: $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$

kredit: $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$

Razmotrimo postavljanje tanjura na stog:
- Konobaru se plaća stavljanje tanjura na stog, uzimanje jednog i uzimanje k tanjura prema stvarnim cijenama.

Dovoljno je platiti konobaru svako stavljanje tanjura 2 €. Zašto

PUSH , PUSH , PUSH, POP, MULTIPOP (2) $n = 5$

↑ ↑ ↑ ↑ ↑

2$ 2$ 2$ 0$ 0$

$$\sum_{i=1}^{n} \hat{c}_i = 6$$

OPERACIJA	STVARNA	AMORT.	KREDIT (Σ_i)
PUSH	1$	2$	1$
PUSH	1$	2$	2$
PUSH	1$	2$	3$
POP	1$	0$	2$
MULTIPOP(2)	2$	0$	0$

kredit uvijek mora biti nenegativan

Na kraju $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i$

$$\sum_{i=1}^{j} \hat{c}_i \geq \sum_{i=1}^{j} c_i \quad \forall j \in \{1, \ldots, n\}$$

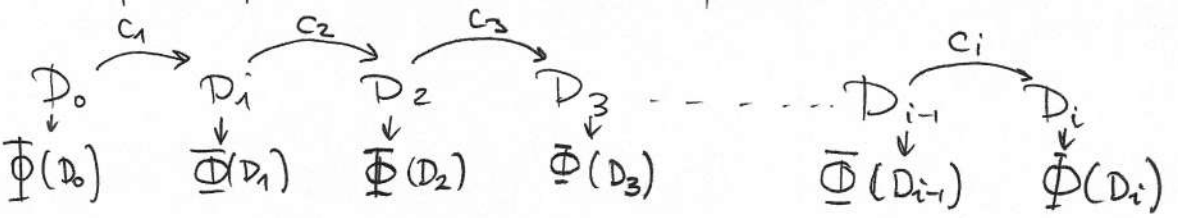
ne znamo koliko će biti operacije unaprijed !

$$\sum \hat{c}_i = O(n)$$

Metoda potencijala

- uvodi se pojam potencijala: koliko plaćamo buduće operacije

n operacije na strukturi podataka D



amortizirana cijena

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

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1}) \right)$$

$$= \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} \left(\Phi(D_i) - \Phi(D_{i-1}) \right)$$

$$\left(\cancel{\Phi(D_1)} - \Phi(D_0) + \Phi(D_2) - \cancel{\Phi(D_1)} + \right.$$

$$\left. \ldots \quad \Phi(D_n) - \cancel{\Phi(D_{n-1})} \right)$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

Ako je $\Phi(D_n) \geq \Phi(D_0)$ onda je $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$

Također je potrebno $\Phi(D_i) \geq \Phi(D_0) \quad \forall i \in [n]$

ji ne znamo koliko će se operacija izvršiti unaprijed

Operacije na stogu

$\Phi \leftarrow$ broj elemenata na stogu

$\Phi(D_0) = 0 \Rightarrow$ odatle slijedi $\Phi(D_i) \geq \Phi(D_0), \forall i \in [n]$ ✓

PUSH

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

$= 1 + (s+1 - s)$

$= 2$

MULTIPOP

$\Phi(D_{i-1}) = s \ (\geq 0) \quad k' = \min(k, s)$

$\Phi(D_i) = s - k' \ (\geq 0)$

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

$= k' + s - k' + s$

$= 0$

IP

$\Phi(D_{i-1}) = s \geq 0 \quad k' = \min(1, s)$

$\Phi(D_i) = s + k'$

isto $\quad \hat{c}_i = 0$

$\sum_{i=1}^{n} \hat{c}_i = O(n)$

✳ Inkrementiranje binarnog brojača (D.Z)

$\Phi(D_i) = b_i \leftarrow$ broj jedinica nakon i-te operacije

$\Phi(D_0) = 0 \qquad \Phi(D_i) \geq \Phi(D_0) \ \forall i$

$\Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i$

Dinamičke tablice (vektori)

- motivacija: dinamički vektor (STL) vector <--->
  - amortizirano vrijeme izvršavanja n operacija koje se sastoji od
    * INSERT
      - ako tablica/vektor ima barem jednu slobodnu lokaciju ($T.num < T.size$) onda je $O(1)$
      - ako je tablica puna ($T.num = T.size$) alociraj (2 puta) veću, prepiši prvu i ubaci novi element
    * DELETE:
      - obriši element iz tablice stvarajući jednu novu slobodnu memorijsku lokaciju
      - ako je broj elemenata u tablici značajno manji od veličine tablice, zamijeni postojeću tablicu manjom.

- faktor ispunjenosti (load factor)

$$\alpha(T) = \begin{cases} 1, & T.num = 0 \\ \frac{T.num}{T.size}, & T.num \neq 0 \end{cases}$$

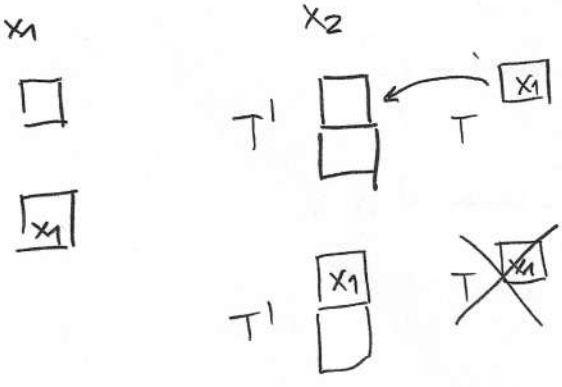
- želimo postići $\alpha(T) = \Theta(1)$

Proširenje tablice

- postižemo operacijom TABLE-INSERT (T, x)
- inicijalno, T.num = T.size = 0

```
TABLE-INSERT (T, x)
    if T.size = 0
        alociraj tablicu T veličine 1
        T.size = 1
    end if
    if T.num = T.size                          } spanzija
        alociraj tablicu T' veličine 2·T.size
        prepiši sve elemente iz T u T'
        dealociraj T
        T ← T'        (C pointeri)
        T.size ← 2·T.size
    end if
    ubaci x u T
    T.num ← T.num + 1
```

inicijalno



itd.

Stvarna cijena:

Zašto razjasniti na ploči

$$c_i = \begin{cases} i, & i-1 = 2^k \text{ za neki } k \\ 1, & \text{inače} \end{cases}$$

Agregatna analiza:

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j < n + \frac{2^{\lg n+1}-1}{2-1} < n + \frac{2n-1}{1}$$

$$= 3n-1 < 3n$$

$$\Rightarrow \sum_{i=1}^{n} c_i < 3n$$

$$\Rightarrow T(n)/n = O(1)$$

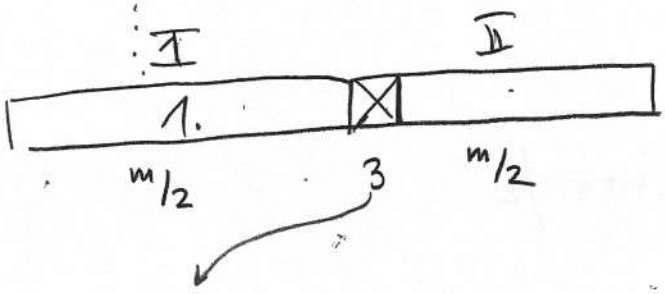
Metoda kreditiranja

STVARNA

TABLE-INSERT	1	
TABLE INSERT	1	
TABLE INSERT	3	(EXP)
TABLE INSERT	1	
TABLE INSERT	5	(EXP)

AMORTIZIRANA

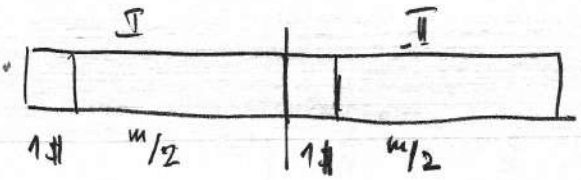
TABLE - INSERT	3
TABLE - INSER	3
TAB - - -	3



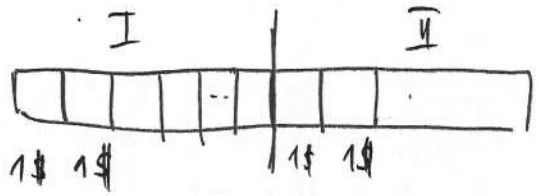
3 $ - 1 $ za ubacivanje

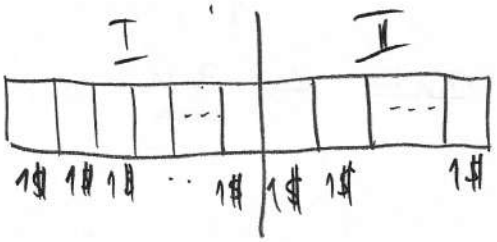
1 $ kredita za prepisivanje prilikom sljedeće ekspanzije

1 $ kredita za neki od postojećih $m/2$ elemenata unutar I

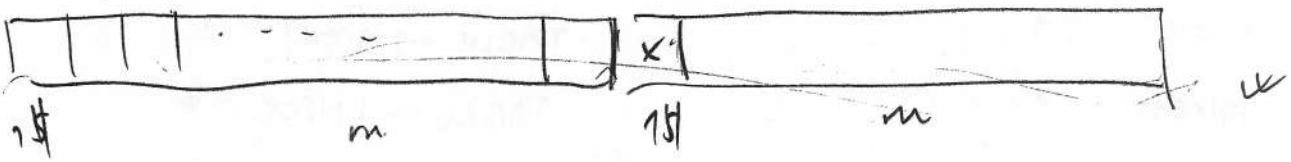


1$ ubacivanja





neko m/2 ubacivanja
kreditiramo sam cijelu tablicu
od m elemenata (bloka I + II)
što mogu potrošiti na
prepisivanje



$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \qquad \sum_{i=1}^{n} \hat{c}_i = 3n$$

Metoda potencijala : iskazati pomoću T.num i T.size

$$\Phi(T) = 2 \cdot T.num - T.size$$

- nakon ekspanzije $T.num = T.size/2$

$$\Phi(T) = 2 \cdot \frac{T.size}{2} - T.size = 0$$

- odmah prije ekspanzije $T.num = T.size$

$$\Phi(T) = 2 \cdot T.size - T.size = T.size$$

inicijalno, $num_0 = 0$ $size_0 = 0$, $\Phi(T_0) = 0$

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

$num_i = T_i.num$
$size_i = T_i.size$

- nakon ekspanzije $size_i = size_{i-1}$

$\hat{c}_i = c_i + \Phi(T_i) - \Phi(T_{i-1})$ ↘ n_i, s_i

$= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1})$

$= 1 + (2 \cdot num_i - size_i) - (2 \cdot (num_i - 1) - size_i)$

$= 1 + 2 = 3$ //

- neposredno prije ekspanzije $size_i = 2\, size_{i-1}$

$size_{i-1} = num_{i-1} = num_i - 1$

$\rightarrow size_i = 2(num_i - 1)$

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

$= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1})$

$= n_i + (2 \cdot n_i - 2(n_i - 1)) - (2(n_i - 1) - (n_i - 1))$

$= n_i + (2n_i - 2n_i + 2) - (2n_i - 2 - n_i + 1)$

$= 2 + 2 - 1 = 3$ //