Amortized Analysis

Rully Soelaiman

Pendahuluan

- Dengan Amortized Analysis dapat ditunjukkan
- waktu rata-rata yang diperlukan untuk melakukan satu urutan operasi pada struktur data terhadap keseluruhan operasi yang dilakukan.
- Biaya rata-rata untuk satu operasi yang kemungkinan mahal akan menjadi lebih murah jika dilakukan pada suatu urutan operasi.
- Kinerja rata-rata tiap-tiap operasi pada keadaan terburuk (worst case).

Teknik Amortized Analysis

- Teknik yang umum digunakan adalah :
- Metode Aggregat, jika ditentukan suatu upper bound biaya total suatu urutan n operasi adalah T(n), maka biaya amortized per operasi adalah T(n)/n.
- Metode Accounting, tiap operasi mempunyai biaya amortized yang spesifik. Operasi yang dilakukan pada urutan awal dapat diberi biaya yang lebih mahal. Biaya tersebut selanjutnya akan menjadi "kredit prabayar" untuk nantinya digunakan membayar operasi-operasi yang berharga lebih kecil dari biaya sebenarnya.
- Metode Potensial, mempunyai karakteristik seperti pada metode Accounting. Tetapi pada metode potensial, kredit digunakan sebagai "Energi Potensial "pada struktur data.

Teknik Amortized Analysis

Operations:

op_1	op_2	op_3	op_4	op_5	op_6	op_7

Actual costs:

t_1 t_2 t_3 t_4 t_5 t_6 t_7

Amortized costs:

a_1 a_2 a_3 a_4 a_5 a_6 a_7

Yang perlu disyaratkan adalah $\sum t_i \leq \sum a_i$ pada semua urutan operasi, dimulai dari op_1.

Tidak disyaratkan $t_i \leq a_i$ untuk setiap i.

Metode Aggregat pada Operasi Stack

- Sebuah stack diimplementasikan dengan dua operasi fundamental dengan order O(1) yaitu:
  - **Push(S,x)**

```
1 top[S] ← top[S]+1
2 S[top[S]] ← x
```

  - **Pop(S)**

```
1 if Stack-Empty(S)
2  then error "Underflow"
3  else top[S] ← top[S]-1
4 return S[top[S]+1]
```

- Pada stack tersebut ditambahkan operasi **Multipop** :
  - **Multipop(S,k)**

```
1 while not Stack-Empty(S)and k ≠ 0
2   do Pop(S)
3      k ← k - 1
```

Metode Aggregat pada Operasi Stack

- Jika stack mempunyai s elemen, maka `multipop k` memiliki kompleksitas **O(min(s,k))** ; *bukan* **O(1)!**
- Dimisalkan terdapat suatu urutan n operasi pada stack yang meliputi **`push`**, **`pop`**, **dan `multipop`**, dan diawali dengan kondisi ***`empty stack`***.
- Worst-case time untuk suatu operasi **`multipop`** adalah **O(n),** sehingga worst-case untuk n operasi adalah **$O(n^2)$.**
- Dengan **metode aggregat**, dapat dihasilkan kompleksitas upper bound yang lebih baik berdasarkan fakta berikut:
  - Jumlah operasi pop paling banyak dilakukan sejumlah elemen yang telah dipush, yaitu O(n), sehingga total waktu yang diperlukan oleh seluruh operasi **`multipop`** juga O(n).
- Sehingga worst-case time untuk n operasi adalah **O(n).**
- Diperoleh amortized cost per operasi adalah **O(1).**

K-bit Binary Counter

- Akan diimplementasikan k-bit binary counter yang melakukan penghitungan naik mulai dari 0.
- Digunakan array ***A[0..k-1]*** bit, dengan ***length[A]= k***
- Untuk menambahkan 1 pada counter, digunakan prosedur berikut
  - **Increment (A)**

```
1 i ← 0
2 while i < length[A] and A[i] = 1
3  do A[i] ← 0
4     i ← i + 1
5 if i < length[A]
6  then A[i] ← 1
```

Binary Counter 8-bit

Counter Value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total Cost
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

Binary Counter k-bit

- Eksekusi tunggal pada prosedur **`Increment`** memiliki worst-case time complexity $\Theta(k)$, pada saat semua elemen array A bernilai 1.
- Sehingga pada suatu urutan n operasi **`Increment`** yang diinisialisasi dengan 0 memiliki worst-case time complexity $O(nk)$
- Dengan **metode aggregat**, dapat dihasilkan kompleksitas upper bound yang lebih baik berdasarkan fakta berikut:
  - Tidak semua bit berubah tiap kali prosedur **`Increment`** dipanggil!
- Sehingga worst-case time untuk n operasi adalah **O(n).**
- Diperoleh amortized cost per operasi adalah **O(1).**

Binary Counter k-bit

- Untuk n operasi akan diperoleh jumlah perubahan bit pada tiap posisi indeks array sebagai berikut
  - A[0] berubah n kali
  - A[1] berubah $\lfloor n/2 \rfloor$ kali
  - A[2] berubah $\lfloor n/4 \rfloor$ kali ...dst.
- Secara umum, untuk i=0,1,.. $\lfloor \lg n \rfloor$, bit A[i] akan berubah $\lfloor n/2^i \rfloor$ kali untuk urutan n operasi
- Untuk i > $\lfloor \lg n \rfloor$, bit A[i] tidak pernah mengalami perubahan.
- Sehingga pada urutan n operasi didapatkan jumlah perubahan sebagai berikut

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$= 2n$$

Latihan 18.1-3

- Suatu urutan n operasi dilakukan pada suatu struktur data. Biaya operasi ke-i akan berharga i jika i merupakan pangkat mutlak dari 2, dan berharga 1 untuk keadaan lainnya. Gunakan Metode Aggregat untuk menentukan amortized cost per operasi.

Operasi ke	Biaya Operasi
1	$1(2^0)$
2	$2(2^1)$
3	1
4	$4(2^2)$
5	1
Total	9

Penyelesaian

$$Total = \sum_{i=0}^{\lfloor \lg n \rfloor} (2^i - 1) + \sum_{i=1}^{n} 1$$

Ingat Deret Geometris : $\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1}$

$$Total = 2^{\lfloor \lg n \rfloor + 1} - 1 - (\lfloor \lg n \rfloor + 1) + n$$

$$Total = 2^{\lfloor \lg n \rfloor + 1} + n - \lfloor \lg n \rfloor - 2$$

$$Total \leq 2^{\lg n + 1} + n - \lfloor \lg n \rfloor - 2$$

Ingat : $a^{\log_b n} = n^{\log_b a}$

$$Total \leq 3n - \lfloor \lg n \rfloor - 2$$

- Sehingga total cost untuk n operasi adalah **O(n).**
- Diperoleh amortized cost per operasi adalah **O(1).**

Metode Accounting

- Amortized Cost pada Metode Accounting terdiri atas actual cost dan kredit yang dapat dideposit atau ditarik.
- Untuk dapat menunjukkan bahwa biaya rata-rata per operasi pada worst-case adalah kecil, maka total amortized cost pada suatu urutan operasi harus merupakan upper bound dari total actual cost pada urutan tersebut.
- Total kredit yang berasosiasi dengan suatu struktur data harus selalu bernilai non negatif.

Metode Accounting pada Operasi Stack

- Assignment cost pada Operasi Stack

Operation	Actual Cost	Amortized Cost
`push`	1	2
`pop`	1	0
`multipop`	min(s,k)	0

- Sehingga dapat disimpulkan kompleksitas amortized cost ketiga operasi adalah O(1).
- Untuk n operasi stack, total amortized cost adalah O(n).

Metode Accounting pada Binary Counter

- Waktu eksekusi operasi Increment secara actual akan proporsional terhadap jumlah bit yang mengalami perubahan.
- Dengan Amortized Analysis, diperoleh Amortized Cost sebagai berikut
  - Set sebuah bit menjadi 1 $\rightarrow$ 2
  - Set sebuah bit menjadi 0 $\rightarrow$ 0
- Jumlah 1 pada counter tidak pernah negatif, sehingga jumlah kredit selalu nonnegatif.
- Untuk n operasi Increment, total amortized cost adalah O(n).

Penyelesaian 18.2-2

- Ditentukan amortized cost per operasi = 3.
- Maka nilai kredit setelah operasi ke-i adalah

$$Total_i = \sum_{j=1}^{i} 3 - CostActual_i$$

$$Total_i = 3i - (2^{\lfloor \lg i \rfloor + 1} + i - \lfloor \lg i \rfloor - 2)$$

- 1. Jika $i=2^k$ (dengan $k \geq 0$), maka
- $Total_i = 3i-(2^{k+1}+i-k-2) = 3i-(3i-k-2) = k+2$
- 2. Jika $i=2^k+j$ (dengan $k \geq 0$ dan $i \leq j < 2^k$)
- $Total_i = 3i-(2^{k+1}+i-k-2) = 3i-(2(i-j)+i-k-2) = 2j+k+2$
- Total Amortized Cost 3n merupakan upper bound dari total actual cost. Sehingga kompleksitas n operasi O(n). Amortized Cost per operasi O(1).

Metode Potensial

- Teknik Amortized Analysis yang lain adalah
- **Metode Potensial**:
  - Tiap struktur data mempunyai asosiasi dengan suatu non-negative *potential.*
  - Ditentukan terlebih dahulu struktur yang digunakan pada n operasi. Urutan struktur data yang bersesuaian adalah: $D_0, D_1, \ldots, D_n$, dengan potential $\Phi(D_0), \Phi(D_1), \ldots, \Phi(D_n)$.
  - Jika c_i adalah actual cost untuk operasi i, maka *amortized cost* dinyatakan dengan

$$a_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$

  - Disyaratkan bahwa $\Phi(D_i) \geq \Phi(D_0)$, untuksemua i.

Metode Potensial

Syarat tersebut mengakibatkan kondisi

$$\sum_{i=1}^{n} a_i = \sum (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$
$$= (\sum c_i) + \Phi(D_n) - \Phi(D_0),$$

kondisi tersebut menjamin bahwa amortized cost selalu merupakan upper bound dari actual cost.

Metode Potensial pada Operasi Stack

- Didefinisikan fungsi potential Φ pada sebuah stack yang menyatakan jumlah elemen yang disimpannya.
- Pada awalnya, stack akan kosong sehingga $\Phi(D_0)=0$.
- Karena jumlah elemen pada stack tidak pernah negatif, maka $\Phi(D_i)\geq\Phi(D_0)=0$.
- Contoh penghitungan amortized cost untuk operasi push adalah
- $c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (s+1) - s = 2.$
- Untuk operasi **`multipop`**, misalkan k'=min(k,s)
- $c_i + \Phi(D_i) - \Phi(D_{i-1}) = k' - k' = 0$

Metode Potensial pada Binary Counter

- Didefinisikan fungsi potential b_i yang menyatakan banyaknya jumlah bit 1 setelah operasi Increment ke-i.
- Misalkan, pada operasi ke-i ada t_i bit yang direset. Actual cost maksimal untuk operasi tersebut adalah t_i+1. Jumlah bit 1 setelah operasi ke-i adalah $b_i \leq b_{i-1} - t_i + 1$, sehingga terjadi perbedaan potensial
- $\Phi(D_i)-\Phi(D_{i-1}) \leq b_{i-1} - t_i + 1 - b_{i-1}$
- $= 1 - t_i$
- Amortized cost untuk operasi Increment adalah
- $= ci + \Phi(D_i)-\Phi(D_{i-1})$
- $\leq t_i+1+ 1- t_i$
- $= 2$
- Sehingga kompleksitas n operasi O(n).

Penyelesaian 18.3-6

- Untuk mengimplementasikan sebuah queue dengan stack, maka digunakan dua buah stack dimana satu stack digunakan sebagai stack input (S_IN) dan stack lainnya sebagai stack output (S_OUT). Selanjutnya operasi pada queue dilakukan sebagai berikut :
- Pada ENQUEUE sebuah elemen, dikerjakan prosedur push pada stack input.
- PADA DEQUEUE sebuah elemen, dikerjakan prosedur pop satu elemen dari stack output; jika stack output kosong, maka terlebih dahulu dipindahkan seluruh elemen dari stack input ke stack output selanjutnya dikerjakan prosedur pop satu elemen dari stack output.

Penyelesaian 18.3-6

- Berikut pseudo code untuk operasi pada queue tersebut.
- **ENQUEUE(e,S_IN)**
- **push(S_IN,e)**

- **DEQUEUE(S_OUT)**
- **If (STACK-EMPTY(S_OUT))**
- **then while(!STACK-EMPTY(S_IN))**
- **push(S_OUT,pop(S_IN))**
- **if (STACK-EMPTY(S_OUT))**
- **then error "underflow"**
- **else return pop(S_OUT)**

Penyelesaian 18.3-6

- Untuk analisis amortized cost, didefinisikan fungsi potensial $\Phi(D_i)=2*elemen_i$, dengan $elemen_i$ adalah banyaknya elemen pada stack input setelah operasi ke i
- Amortized cost untuk operasi ENQUEUE :
- $= c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2 = 3$
- Amortized cost untuk operasi DEQUEUE yang tidak memerlukan pemindahan elemen dari stack input ke stack output :
- $= c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 0 = 1$
- Amortized cost untuk operasi DEQUEUE yang memerlukan pemindahan elemen dari stack input ke stack output :
- $= c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $= 2*elemen_{i-1}+1+0-2*elemen_{i-1} = 1$
- Sehingga diperoleh amortized cost operasi ENQUEUE dan DEQUEUE adalah O(1).

Amortized Analysis pada tabel dinamis

- Pada kasus berikut akan dibahas aplikasi amortized analysis pada persoalan ekspansi dan kontraksi sebuah tabel secara dinamis.
- Dengan amortized analysis, akan ditunjukkan bahwa amortized cost untuk penyisipan dan penghapusan item pada tabel adalah O(1), walaupun actual cost untuk operasi tersebut dapat lebih besar jika akan menyebabkan terjadinya ekspansi atau kontraksi.
- Selanjutnya, akan ditunjukkan bahwa ukuran ruang yang tidak terpakai pada tabel tidak akan pernah melebihi suatu faktor tertentu terhadap total ukuran ruang.
- Diasumsikan bahwa tabel dinamis memiliki dua operasi utama yaitu TABLE-INSERT dan TABLE-DELETE.

Amortized Analysis pada tabel dinamis

- TABLE-INSERT menyisipkan sebuah item pada sebuah slot.
- TABLE-DELETE menghapuskan sebuah item pada table dan membebaskan slotnya.
- Load factor $\alpha(T)$ dari tabel nonempty menyatakan jumlah item yang disimpan pada tabel dibagi dengan ukuran tabel. Dalam kasus berikut $\alpha(T)$ yang ditentukan adalah 1.
- Jika seluruh slot telah terisi atau $\alpha(T)$ adalah 1, maka secara heuristik tabel baru akan diekspansi sebanyak dua kali jumlah slot pada tabel lama.

Amortized Analysis pada tabel dinamis

- Berikut pseudo code untuk prosedur TABLE-INSERT.

```
TABLE-INSERT(T,x)
1 if size[T]=0
2  then allocate table[T] with 1 slot
3       size[T]← 1
4 if num[T] = size[T]
5  then allocate new-table with 2*size[T] slots
6       insert all items in table[T] into new-table
7       free table[T]
8       table[T] ← new-table
9       size[T] ← 2*size[T]
10 insert x into table[T]
11 num[T] ← num[T] + 1
```

Amortized Analysis pada tabel dinamis

- Biaya operasi ke-i jika ruang masih tersedia pada tabel, $c_i=1$.
- Jika tabel penuh, maka akan terjadi ekspansi, sehingga biaya operasi $c_i=i$.
- Kompleksitas worst-case satu operasi adalah O(n), sehingga untuk n operasi adalah $O(n^2)$. Dengan amortized analysis akan ditentukan upper bound yang lebih baik.
- Kondisi : Operasi ke-i akan menyebabkan terjadinya ekspansi bila i-1 merupakan pangkat mutlak dari 2.
- Sehingga biaya total untuk n operasi

$$\sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j$$

Amortized Analysis pada tabel dinamis

- Dengan Metode Aggregat, diperoleh amortized cost sebagai berikut :

$$\begin{aligned}\sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n\end{aligned}$$

- Sehingga untuk n operasi, kompleksitas worst-case O(n).
- Amortized cost per operasi O(1).

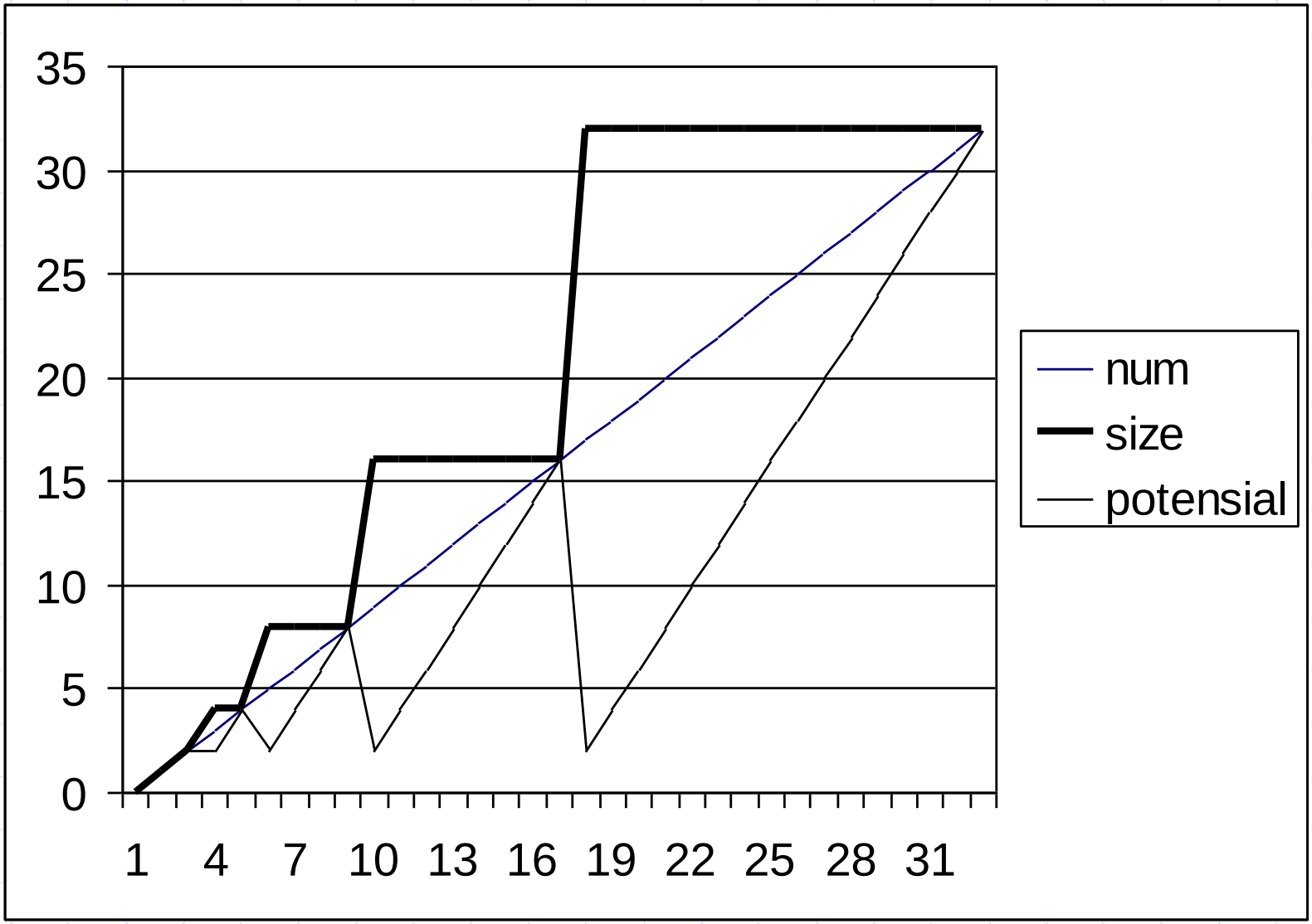
Amortized Analysis pada tabel dinamis

- Dengan Metode Accounting, satu operasi TABLE-INSERT memiliki amortized cost =3.
- Tiap item akan membayar pada 3 kejadian penyisipan yaitu : untuk penyisipan item tersebut pada tabel, untuk pemindahan item tersebut ketika tabel diekspansi, untuk pemindahan item lain yang sudah pernah dipindahkan sebelumnya satu kali.
- Sehingga untuk n operasi, kompleksitas worst-case O(n).
- Amortized cost per operasi O(1).

Amortized Analysis pada tabel dinamis

- Untuk metode potensial, didefinisikan fungsi potensial $\Phi(T)=2*num[T] - size[T]$. Nilai awal potensial tersebut adalah 0. Karena tabel berisi sekurang-kurangnya separuh dari ukurannya, $num[T] \geq size[T]/2$, maka $\Phi(T)$ selalu nonnegative.
- num_i menyatakan jumlah item yang tersimpan pada tabel setelah operasi ke-i. $size_i$ menyatakan ukuran tabel setelah operasi ke-i.
- Jika pada operasi ke-i tidak terjadi ekspansi, maka $size_i=size_{i-1}$. Amortized cost untuk operasi tersebut :
- $= c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $= 1 + (2.num_i - size_i) - (2.num_{i-1} - size_{i-1}) = 3$
- Jika pada operasi ke-i terjadi ekspansi, maka $size_i/2=size_{i-1}= num_i -1$
- Amortized cost untuk operasi tersebut :
- $= c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $= num_i + (2.num_i - size_i) - (2.num_{i-1} - size_{i-1}) = 3$
- Sehingga amortized cost operasi TABLE-INSERT adalah O(1).

Efek Urutan Operasi Pada Parameter Tabel



Next lecture

- NP-Completeness