UNIT 8
AMORTIZED ANALYSIS

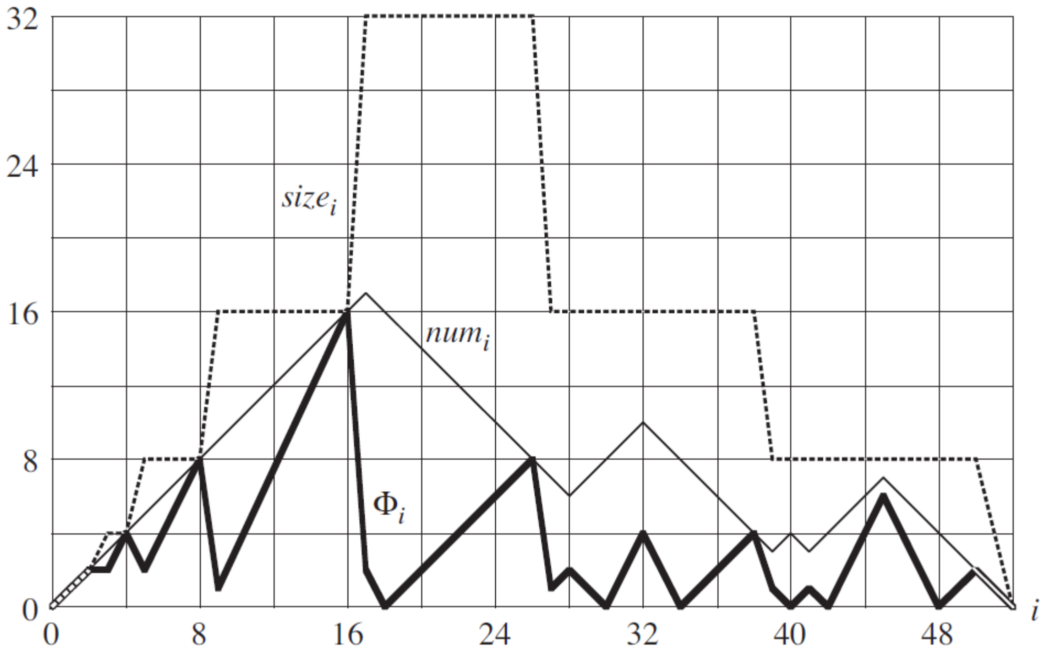
Iris Hui-Ru Jiang
Spring 2024

Department of Electrical Engineering
National Taiwan University

Outline

- Content:
  - Aggregate method
  - Accounting method
  - Potential method

- Reading:
  - Chapter 16



Amortized Analysis

- **Why?**
  - Find a tight bound of a sequence of data structure operations
- No probability involved, guarantees the average performance of each operation **in the worst case**
  - Amortized cost provides an upper bound of actual cost
- Three popular methods
  - Aggregate method
  - Accounting method
  - Potential method

Methods for Amortized Analysis

- Aggregate method
  - n operations take $T(n)$ time
  - Average cost of an operation is $T(n)/n$ time
- Accounting method
  - Charge each operation an amortized cost
  - Store the amount not used in “bank”
  - Use the stored amount for later operations
  - **Must guarantee nonnegative balance!!**
- Potential method
  - View “stored amount” as “potential energy”

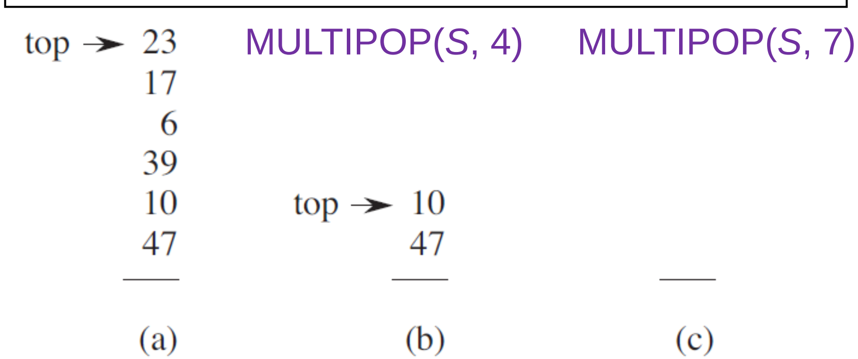
Aggregate Method: MULTIPOP

- n operations take $T(n)$ time $\Rightarrow$ average cost of an operation is $T(n)/n$ time
- Consider a sequence of n PUSH, POP, and MULTIPOP operations on an initially empty stack
  - Worst-case analysis: a MULTIPOP operation takes $O(n) \Rightarrow$ naïve analysis: $O(n^2)$, not tight!
  - Aggregate method: Any sequence of n PUSH, POP, MULTIPOP costs at most $O(n)$ time **(why?)** $\Rightarrow$ amortized cost of an operation: $O(n)/n=O(1)$

```
MULTIPOP(S, k)
while not Stack-Empty(S) and k > 0
    Pop(S)
    k = k-1
```

#iterations of **while**: $\min(s, k)$

We can pop each object at most once for each time we have pushed it onto the stack



Incrementing a Binary Counter

- Increment an initially zero k-bit binary counter

```
Increment(A)
i = 0
while i < A.length and A[i] == 1
    A[i] = 0        // flip 1 to 0
    i = i + 1
if i < A.length
    A[i] = 1
```

A.length = k

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

- 1 increment = 1 set + several resets

Aggregate Method: Incrementing a Binary Counter

- Worst case: an INCREMENT operation takes $O(k)$ time
  - $O(nk)$ for n INCREMENT operations $\Rightarrow$ not tight!
- The amortized cost: $O(1)$ time
  - $A[0]$, $A[1]$, $A[2]$, …: flips each time, every other time, every fourth time,... that INCREMENT is called
  - # Flips = $\sum_{i=0}^{\lfloor \lg n \rfloor} \lfloor \frac{n}{2^i} \rfloor < 2n \Rightarrow$ Amortized Cost = $O(n)/n = O(1)$

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

Accounting Method

- Stack operations (s: stack size):

	Actual cost	Amortized cost
PUSH	1	2
POP	1	0
MULTIPOP	$\min(s, k)$	0

  - For any sequence of *n* operations, **total actual cost ≤ total amortized cost** = *O*(*n*)
- Incrementing a binary counter
  - Charge an amortized cost of $2 to set a bit to 1
    - 1 → actual cost
    - 1 → credit
  - Don't charge anything to reset a bit to 0 (-$1 from credit)
  - For *n* increment operations, total credit = # of 1's in the counter ≥ 0
    - Total actual cost ≤ total amortized cost = *O*(*n*)

Must guarantee nonnegative balance!!

The Potential Method

- View the prepaid work as “potential” that can be released to pay for future operations
  - Difference: potential is associated with **the whole data structure**, not with specific items in the data structure
- The potential method:
  - D_0: initial data structure
    D_i: data structure after applying the i-th operation to D_{i-1}
    c_i: actual cost of the i-th operation
  - Define the potential function $\phi: D_i \rightarrow \mathbb{R}$
  - Amortized cost $\hat{c}_i$, $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_{i-1}))$$
$$= \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0)$$

  - Pick $\phi(D_n) \geq \phi(D_0)$ to make $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$
  - Often define $\phi(D_0) = 0$ and then show that $\phi(D_i) \geq 0, \forall i$

The Potential Method: Stack Operations

- Amortized cost of each operation = $O(1)$
- $\phi(D) = \#$ of objects in the stack D; $\phi(D_0) = 0$, $\phi(D_i) \geq 0$

- PUSH: $\widehat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + (s+1) - s = 2$

- POP: $\widehat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1}) = 1 + (-1) = 0$

- MULTIPOP(S, k): $k' = \min(s, k)$ objects are popped off

$$\begin{aligned}\widehat{c_i} &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &= k' - k' = 0\end{aligned}$$

Potential: Incrementing a Binary Counter

- Amortized cost of each operation = $O(1)$
- $\phi(D)$ = # of 1's in the counter D; let $b_i = \phi(D_i) \geq 0$
- Suppose the i-th increment operation resets t_i bits
  - $c_i \leq t_i + 1$ **01111** c_i **= 5,** t_i **= 4**
  - $b_i \leq b_{i-1} - t_i + 1$ **10000** b_{i-1} **= 4,** b_i **= 1**
- Amortized cost
  - $\widehat{c_i} = c_i + \phi(D_i) - \phi(D_{i-1})$
  - $\leq (t_i + 1) - t_i + 1 = 2$
- For $b_0 \leq k$, let $n = \Omega(k)$
  - $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \widehat{c_i} - \phi(D_n) + \phi(D_0)$
  - $\leq \sum_{i=1}^{n} 2 - b_n + b_0$
  - $= O(n)$

```
Increment(A)
i = 0
while i < A.length and A[i] == 1
    A[i] = 0        // flip 1 to 0
    i = i + 1
if i < A.length
    A[i] = 1
```

A.length = k

Summary: Amortized Analysis

- Why amortized analysis?
  - Find a tight bound of a **sequence** of data structure operations
- Aggregate method
  - *n* operations take *T*(*n*) time
  - Average cost of an operation is *T*(*n*)/*n* time
- Accounting method
  - Charge each operation an amortized cost
  - Store the amount not used in “bank”
  - Use the stored amount for later operations
  - **Must guarantee nonnegative balance!!**
  - $\hat{c}_i = c_i + credit(stored)$

Amortized cost ≥ actual cost
Credit is stored for each operation

How to define a “tight” amortized cost to maintain nonnegative balance?

- Potential method
  - View “stored amount” as “potential energy”
  - $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$

Amortized cost ≥ actual cost
Potential is stored for all processed operations

How to define a “tight” potential function to maintain nonnegative potential?

Dynamic Table Expansion

- Insertion only for the time being
- Goal: Try to make table as small as possible
- Idea: Allocate more memory when needed
  1. Initialize table size *m* = 1
  2. Insert elements until the # of elements = *m*
  3. Generate a new table of size 2*m*
  4. Copy old elements into a new table
  5. Insert the new element
  6. Goto Step 2
- Actual cost: c_i = i-th insertion

$$c_i = \begin{cases} i & if\ i-1 = 2^k\ for\ some\ k \geq -, copy\ i-1\ old, insert\ new \\ 1 & otherwise\ (insert\ new) \end{cases}$$

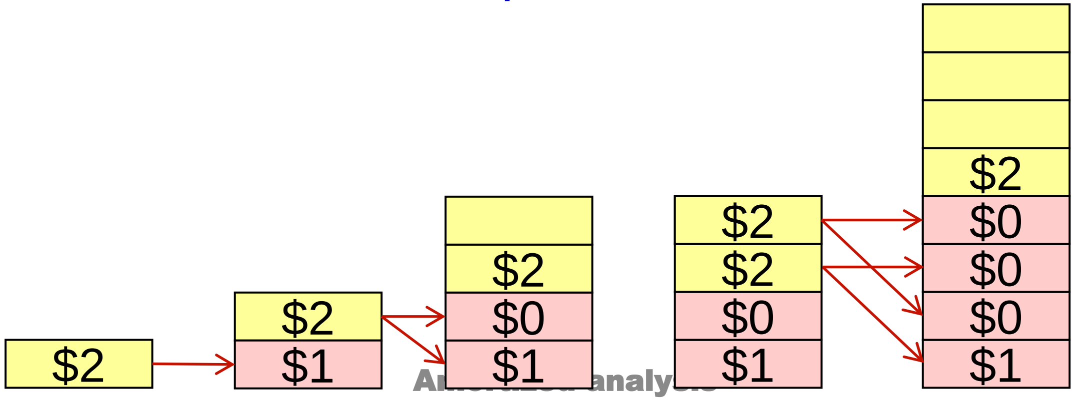
- One insertion can be costly, but in total?
  – Worst-case cost of an insertion = $O(n) \Rightarrow$ total time for n insertions = $O(n^2)$ Not tight!!

Expansion: Aggregate and Accounting Analyses

- **Aggregate analysis**: amortized cost of an operation < 3

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \leq n + 2n \leq 3n$$

- **Accounting analysis**: amortized cost of an operation < 3
  - Charge each operation \$3 (amortized cost): \$1 for immediate insertion and store \$2
  - When table doubles, \$1 for a re-inserting item and \$1 for re-inserting another old item
  - Total runtime = # of dollars spent ≤ # of dollars entered table = 3*n*



Expansion: Potential Analyses

- *num*[*T*]: # of elements in *T*; *size*[*T*]: size of the table *T*
- $\phi(T)$= 2 *num*[*T*] - *size*[*T*]
  - Right before expansion: $\phi(T)$ = *num*[*T*]
  - Right after expansion: $\phi(T)$ = 0
- $\phi_0 = 0; \phi_i \geq 0$
- Table is always at least half full: *num*[*T*] ≥ *size*[*T*]/2 ⇒ $\phi(T) \geq 0$
- If i-th operation does not trigger an expansion ($size_i = size_{i-1}$):

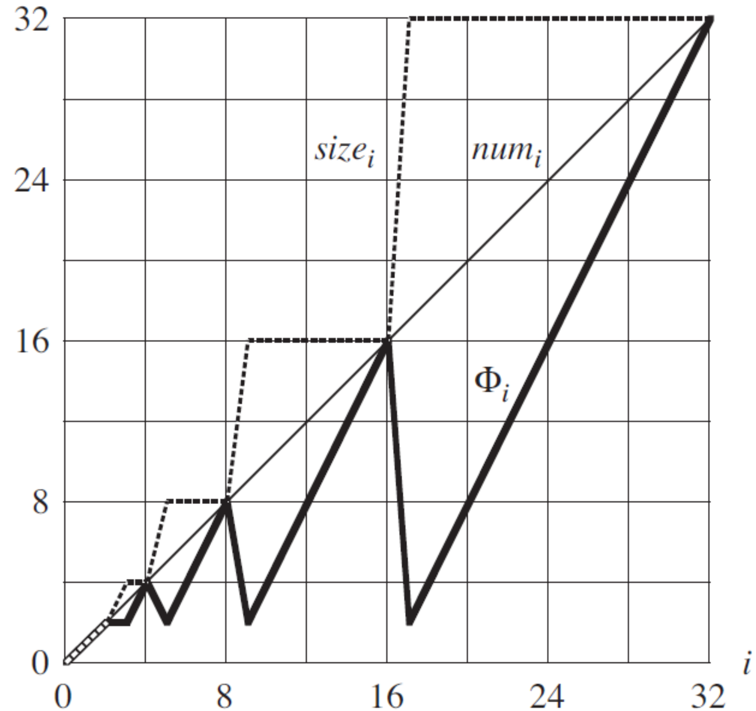
$$\begin{aligned}\widehat{c_i} &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (2\, num_i - size_i) - (2\, num_{i-1} - size_{i-1}) \\ &= 1 + (2\, num_i - size_i) - (2\, (num_i - 1) - size_i) \\ &= 3\end{aligned}$$

- If i-th operation triggers an expansion ($size_i /2 = size_{i-1} = num_i - 1$):

$$\begin{aligned}\widehat{c_i} &= c_i + \phi_i - \phi_{i-1} \\ &= num_i + (2\, num_i - size_i) - (2\, num_{i-1} - size_{i-1}) \\ &= num_i + (2\, num_i - (2\, num_i - 2)) - (2\, (num_i - 1) - (num_i - 1)) \\ &= 3\end{aligned}$$

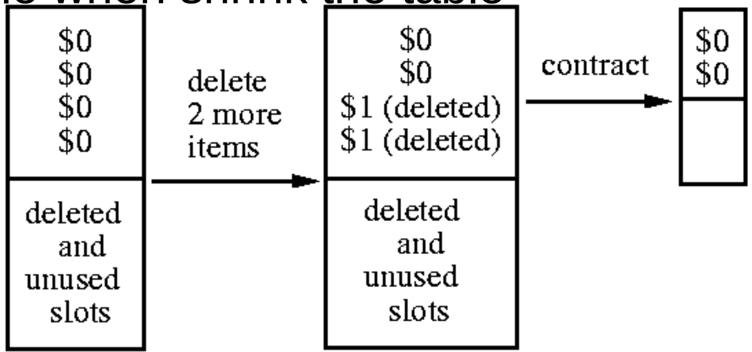
Effect of Expansion

- $\phi(T)$= 2 *num*[*T*] - *size*[*T*]
  - Right before expansion: $\phi(T)$= *num*[*T*]
  - Right after expansion: $\phi(T)$= 0
- $\phi_0 = 0; \phi_i \geq 0$
- Table is always at least half full: *num*[*T*] ≥ *size*[*T*]/2
  $\Rightarrow \phi(T) \geq 0$



Expansion and Contraction: Accounting Analysis

- Bad idea: Double the table when overflow (as before), halve it when table < full/2
  - Cause thrashing when repeatedly halve and double it if repeatedly insert and delete 2 items
  - $n = 2^k$: insert $n/2$ items and then *I D D I I D D …* $\Rightarrow$ amortized cost of an operation = $\Theta(n)$
- Better idea: Double the table when overflow and halve it when table < full/4
- Accounting analysis
  - Charge \$3 for each insertion (as before)
  - Charge \$2 for deletion:
    - Store extra \$1 in emptied slot; use later to pay to copy remaining items to a new table when shrink the table



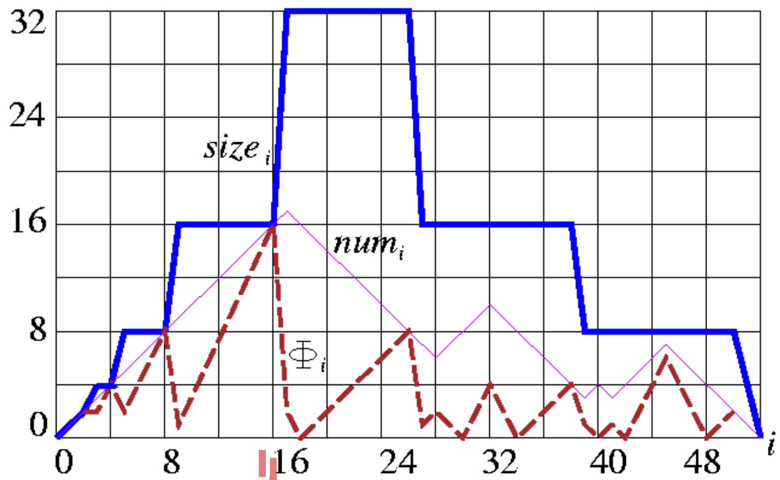
Assume we halve the table when size <= 1/4

Potential Analysis: Expansion and Contraction

- Load factor: $\alpha(T) = num[T]/\text{size}[T]$ if $num[T] > 0$; $\alpha(T) = 1$ if $num[T] = 0$
- Define the potential function $\phi(T)$:

$$\Phi(T) = \begin{cases} 2num[T] - size[T] & \text{if } \alpha(T) \geq 1/2, \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2. \end{cases}$$

  - $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$, $\phi_0 = 0$, $\phi_i \geq 0$
  - $\alpha = 1 \Rightarrow \phi(T) = num[T]$: sufficient potential for expansion
  - $\alpha = 1/2 \Rightarrow \phi(T) = 0$
  - $\alpha = 1/4 \Rightarrow \phi(T) = num[T]$: sufficient potential for contraction



$$\Phi(T) = \begin{cases} 2num[T] - size[T] & \text{if } \alpha(T) \geq 1/2, \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2. \end{cases}$$

Potential Analysis: Insertion

- i-th operation is an insertion: $num_i = num_{i-1} + 1$
  - $\alpha_{i-1} \geq 1/2$: $\hat{c}_i \leq 3$ (as before)
  - $\alpha_{i-1} < 1/2$, $\alpha_i < 1/2$:

$$\begin{aligned} \hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i - 1)) \\ &= 0 \end{aligned}$$

  - $\alpha_{i-1} < 1/2$, $\alpha_i \geq 1/2$:

$$\begin{aligned} \hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (2\, num_i - size_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (2\, (num_{i-1} + 1) - \mathrm{size}_{i-1}) - (size_{i-1}/2 - num_{i-1}) \\ &= 3\, num_{i-1} - 3size_{i-1}/2 + 3 \\ &= 3\, \alpha_{i-1}\, size_{i-1} - 3size_{i-1}/2 + 3 \\ &< 3size_{i-1}/2 - 3size_{i-1}/2 + 3 \\ &= 3 \end{aligned}$$

$$\Phi(T) = \begin{cases} 2num[T] - size[T] & \text{if } \alpha(T) \geq 1/2, \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2. \end{cases}$$

Potential Analysis: Deletion

- i-th operation is a deletion: $num_i = num_{i-1} - 1$
  - $\alpha_{i-1} < 1/2$, $\alpha_i \geq 1/4$: no contraction $\Rightarrow size_i = size_{i-1}$

    $$\begin{aligned} \hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i + 1)) \\ &= 2 \end{aligned}$$

  - $\alpha_{i-1} < 1/2$, $\alpha_i < 1/4$: with contraction $\Rightarrow size_i/2 = size_{i-1}/4 = num_i + 1$

    $$\begin{aligned} \hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= (num_i + 1) + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= (num_i + 1) + ((num_i + 1) - num_i) - ((2num_i + 2) - (num_i + 1)) \\ &= 1 \end{aligned}$$

  - $\alpha_{i-1} \geq 1/2$: $\hat{c}_i \leq$ some constant