Lecture 4
Amortized analysis

Spring 2023

Zhihua Jiang

Recall:

Hashing with Chaining:

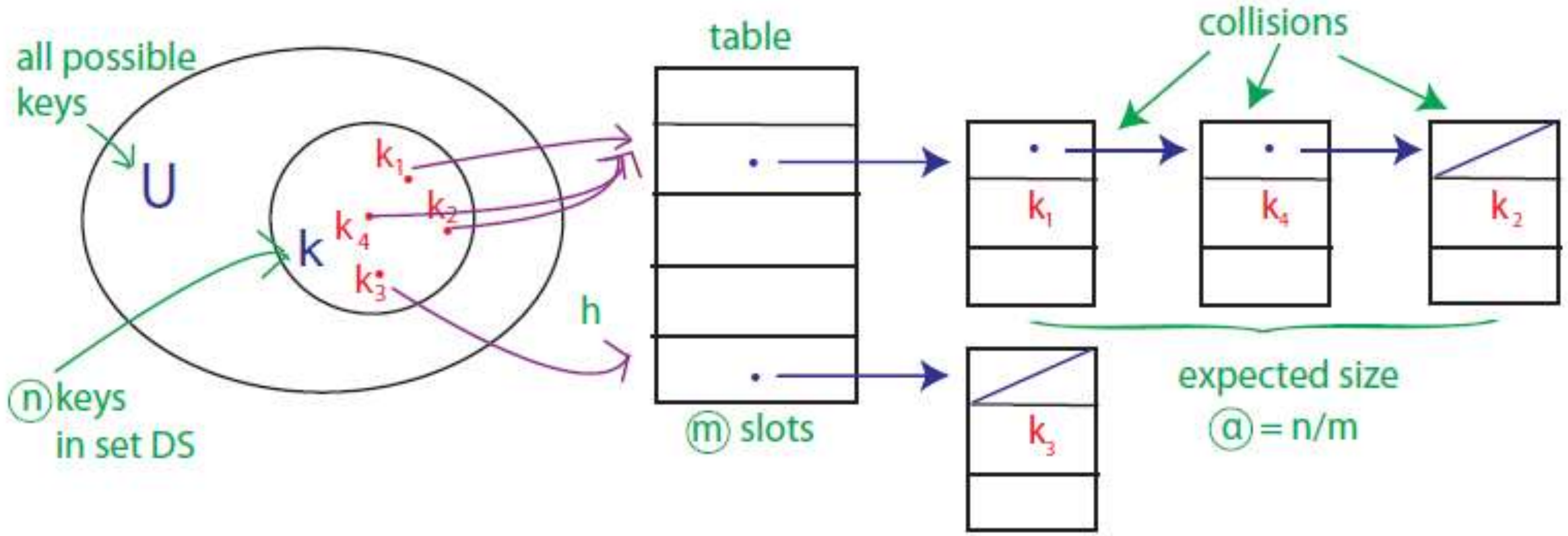


Figure 1: Hashing with Chaining

Expected cost (insert/delete/search): $\theta(\alpha)$, assuming simple uniform hashing *OR* universal hashing & hash function h takes $O(1)$ time.

How Large should Table be?

- want $m = \Theta(n)$ at all times
- don't know how large n will get at creation
- m too small $\implies$ slow; m too big $\implies$ wasteful

Idea:

Start small (constant) and grow (or shrink) as necessary.

Rehashing:

To grow or shrink table hash function must change (m, r)

$\implies$ must rebuild hash table from scratch

for item in old table: $\rightarrow$ for each slot, for item in slot

insert into new table

$\implies \Theta(n + m)$ time $= \Theta(n)$ if $m = \Theta(n)$

How fast to grow?

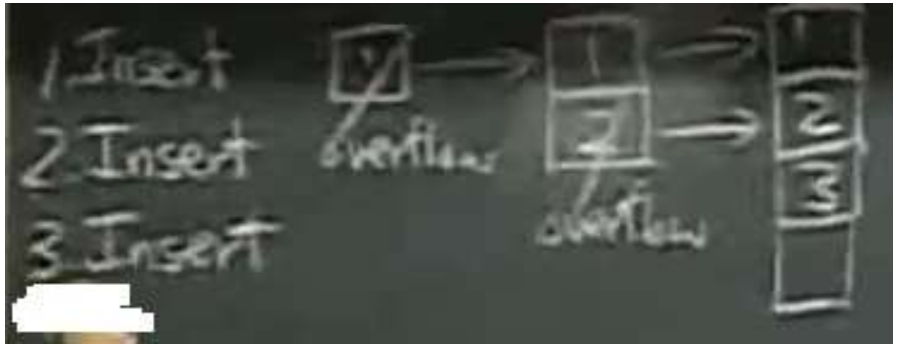
When n reaches m, say

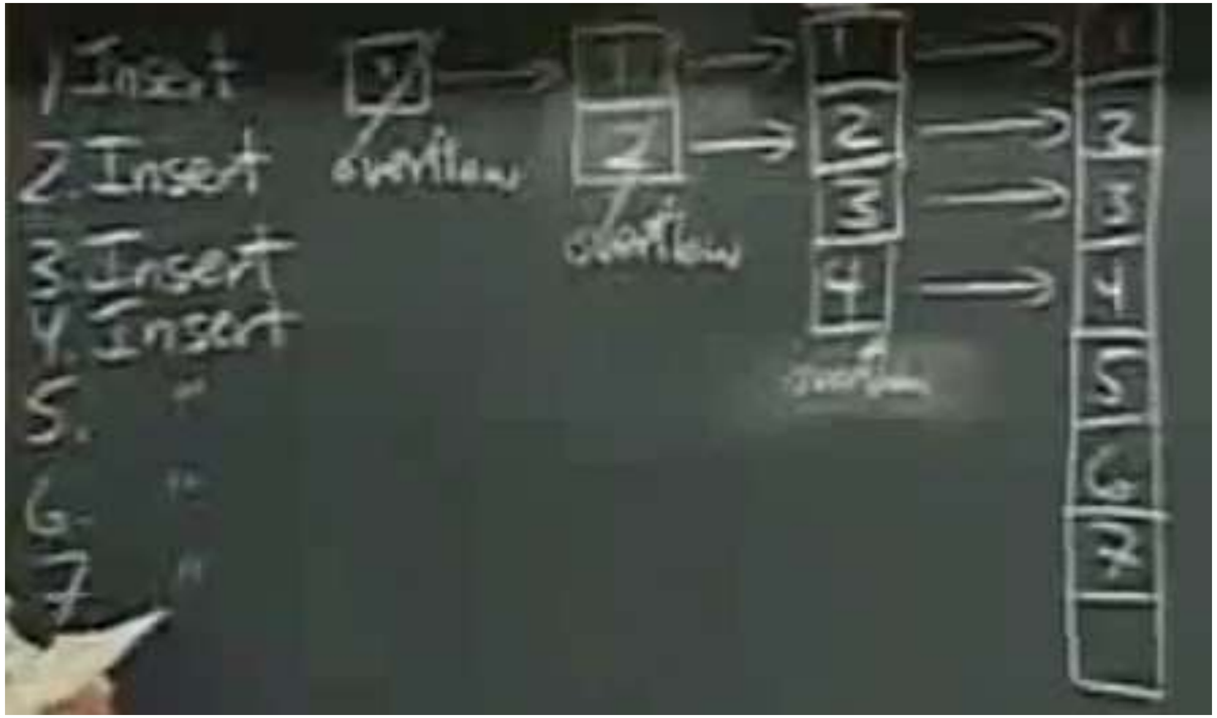
- $m+=1$?
  $\implies$ rebuild every step
  $\implies$ n inserts cost $\Theta(1+2+\cdots+n)=\Theta(n^2)$

- $m*=2$? $m=\Theta(n)$ still $(r+=1)$
  $\implies$ rebuild at insertion 2^i
  $\implies$ n inserts cost $\Theta(1+2+4+8+\cdots+n)$ where n is really the next power of 2
  $=\Theta(n)$

- a few inserts cost linear time, but $\Theta(1)$ "on average".


1. Insert
2. Insert
overflow
1
2


1. Insert
2. Insert
3. Insert
overflow
overflow
2
3


1. Insert
2. Insert
3. Insert
4. Insert
5.
6.
7.
overflow
overflow
overflow
2
3
4
5
6
7

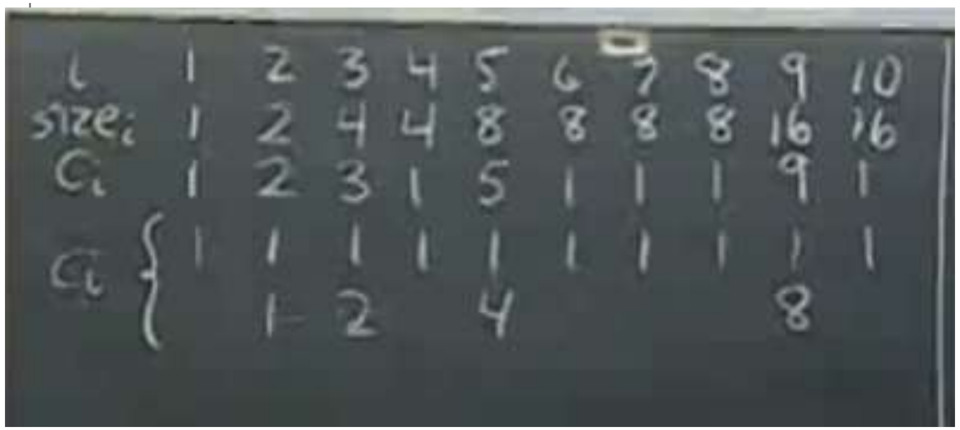
<u>Analysis</u>

Seq. of n Insert operations

Worst-case cost of 1 Insert = $\Theta(n)$ → worst-case cost of n Inserts = $n\ \Theta(n) = \Theta(n^2)$

Wrong! n Inserts take $\Theta(n)$ time!

Let c_i = cost of i^{th} insertion = $\begin{cases} i & \text{if } i\text{-1 is power of 2} \\ 1 & \text{otherwise} \end{cases}$



c_i = cost of insertion + cost of copy

$$\text{cost of } n \text{ Inserts} = \sum_{i=1}^{n} C_i = n + \sum_{j=0}^{\lfloor \log(n-1) \rfloor} 2^j \le 3n = \Theta(n)$$

$$a_n = a_1 \cdot q^{(n-1)}$$

$$S_n = \frac{a_1(1-q^n)}{1-q}\ (q \neq 1)$$

$$2^{\lfloor \log(n-1) \rfloor + 1} \le 2^{\log n + 1} = 2n$$

Thus, average cost per Insert = $\Theta(n)/n = \Theta(1)$

Amortized analysis

Analyze a seq. of operations to show that average cost per operation is small, even though one operation may be expensive.

No probability – average performance in worst case

Types of amortized arguments

- Aggregate (just saw)
- Accounting
- Potential

The last two are more precise – allocate specific amortized cost to each operation

Accounting method

- Charge i^{th} operation a fictitious amortized cost $\hat{C}_i$ ($1 pays for 1 unit of work)

- Fee is consumed to perform operation
- Unused amount stored in "bank" for use by later operations
- Bank balance must not go negative

Must have $\sum_{i=1}^{n} C_i \le \sum_{i=1}^{n} \hat{C}_i, \forall n$

Dynamic table:

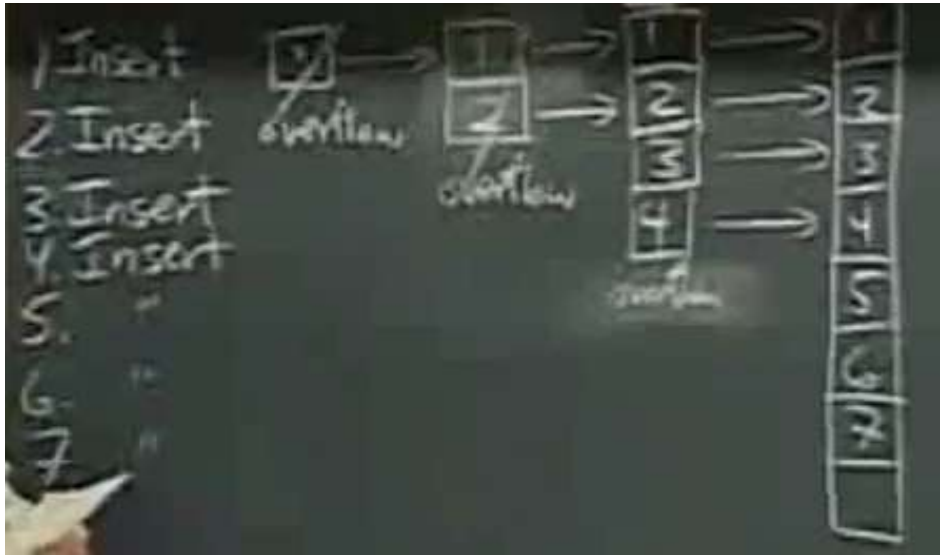
- Charge $\hat{C}_i = \$3$ for i^{th} Insert

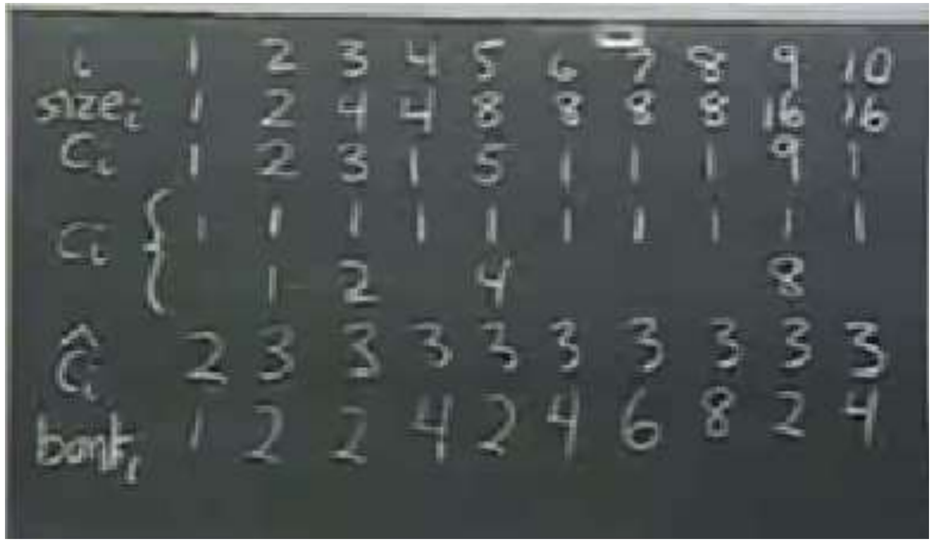
\$1 pays for immediate Insert; \$2 stored for table doubling

- When table doubles

\$1 moves recent item; \$1 moves old items

$$\sum_{i=1}^{n} \hat{C}_i = 3n, \sum_{i=1}^{n} C_i \le 3n \Rightarrow \sum_{i=1}^{n} C_i \le \sum_{i=1}^{n} \hat{C}_i$$





Potential method

"Bank account" viewed as potential energy of dynamic set

Framework:

- Start with data structure D_0
- Operation i transforms $D_{i-1} \rightarrow D_i$

Cost of operation i is C_i

- Define potential function

$$\Phi : \{D_i\} \rightarrow R, \Phi(D_0) = 0 \text{ and } \Phi(D_i) \geq 0, \forall i$$

- Amortized cost $\hat{C}_i$ w.r.t. Φ is $\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1})$

Potential difference $\Delta\Phi_i = \Phi(D_i) - \Phi(D_{i-1})$

If $\Delta\Phi_i > 0$, then $\hat{C}_i > C_i$, Operation *i* stores work in data structure for later.

If $\Delta\Phi_i < 0$, then $\hat{C}_i < C_i$, data structure delivers up stored work to help pay for operation *i*.

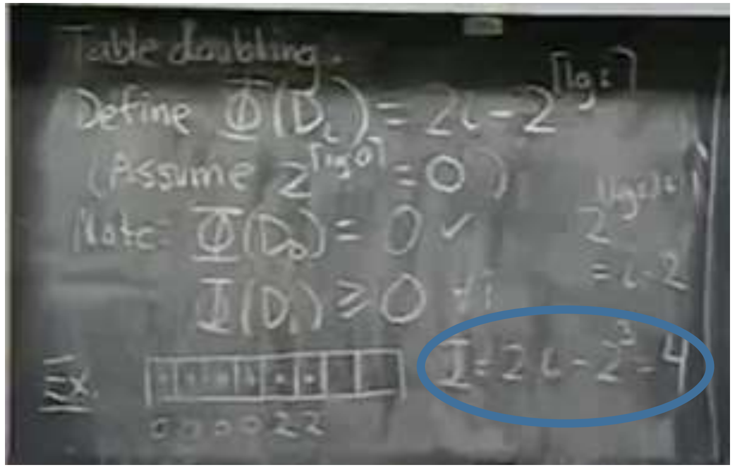
Total amortized cost of *n* operations is

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

Table doubling:

Define $\Phi(D_i) = 2i - 2^{\lceil \log i \rceil}$ (Assume $2^{\lceil \log 0 \rceil} = 0$)

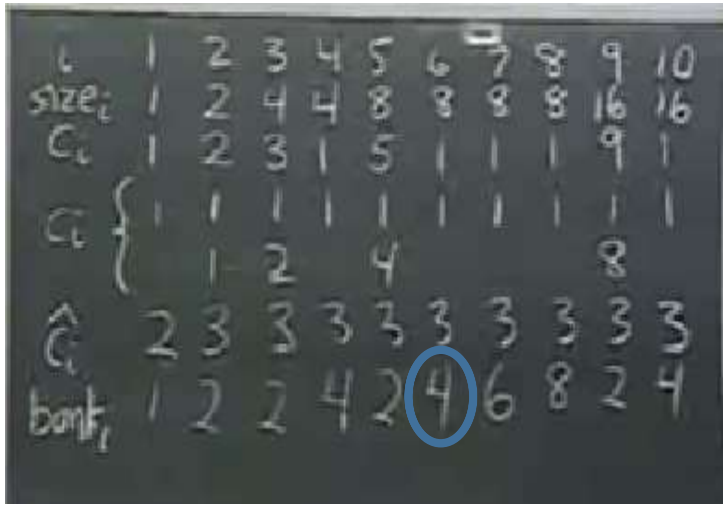
Note $\Phi(D_0) = 0, \Phi(D_i) \geq 0$, because $2^{\lceil \log i \rceil} \leq 2^{\log i + 1} = 2i$



Ex.

$$\Phi(D_6) = 2 \times 6 - 2^{\lceil \log 6 \rceil} = 4$$

$$bank_6 = 4$$



Amortized cost of i^{th} Insert:

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

$$= \begin{cases} i \text{ if } i\text{-1 is exact power of 2} \\ 1 \text{ otherwise} \end{cases} + (2i - 2^{\lceil \log i \rceil}) - (2(i-1) - 2^{\lceil \log(i-1) \rceil})$$

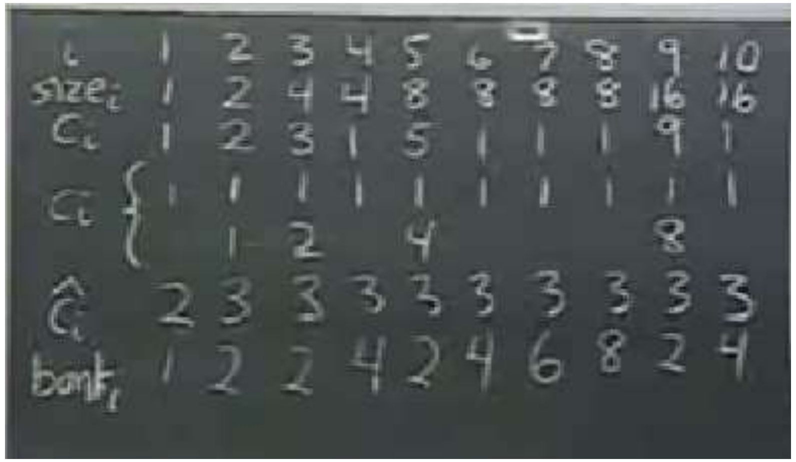
$$= \begin{Bmatrix} i \text{ if} \ldots \\ 1 \ldots \end{Bmatrix} + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log(i-1) \rceil}$$

Case 1: i-1 is exact power of 2

$$\hat{C}_i = i + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log(i-1) \rceil} = i + 2 - 2(i-1) + (i-1) = 3$$

Case 2: i-1 is <u>not</u> exact power of 2

$$\hat{C}_i = 1 + 2 - 2^{\lceil \log i \rceil} + 2^{\lceil \log(i-1) \rceil} = 3 \text{ (i.e., } 2^{\lceil \log i \rceil} = 2^{\lceil \log(i-1) \rceil})$$



n Inserts cost $\Theta(n)$ in worst case

Conclusions

- Amortized costs provide a clean abstraction for data structure performance.
- Any methods can be used, but each has situations where it is arguably simplest or most precise.
- Different potential functions or accounting costs may yield different bounds.