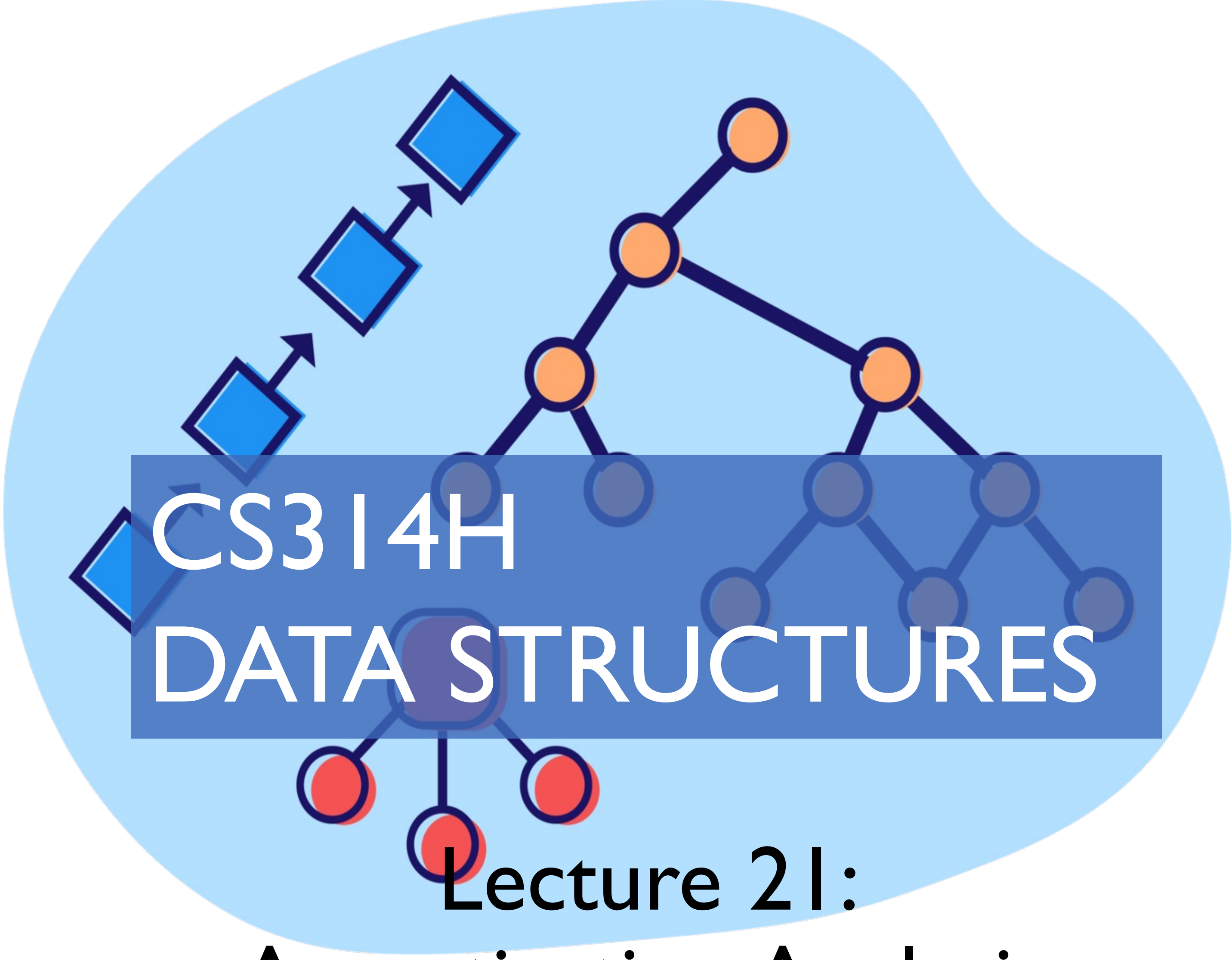

CS314H DATA STRUCTURES

Lecture 21: Amortization Analysis

Mikyung Han

Outline

1. Definition

Amortization Analysis

- a worst-case analysis of a a *sequence* of operations
- Used to obtain a tighter bound on the average cost per operation in the sequence
- Obtained by separately analyzing each operation in the sequence.

How is it different from average case analysis?

Average case analysis measures expected runtime over ALL possible inputs

- Need some *probability distribution on inputs (e.g., uniform random)*
- $E[T(n)] = \sum_{inputs\ I} P(I) \cdot T(I)$, where $T(I)$ is the time for input I.

"If I pick a random input,
how long does the algorithm typically take?"

Amortization analysis measures average runtime per operation over a worst-case sequence of operations

- No probability needed
- Does NOT depend on the input
- Used when
  - Some operations are cheap (insert to an unsorted array)
  - Other operation is expensive but less often (insert to a full array that needs resizing)

"What is per-operation cost over a sequence of operations when an expensive operation occurs only occasionally?"

Resizable Array Example

- Add operation in ArrayList
  - If full, double the size of the array and copy existing elements over
- Array is doubled in 2nd, 3rd, and 5th step, copying 1, 2, and 4 elements respectively
- Time complexity of inserting n items?
  - Worst case insert one item is O(n), thus inserting n items is $O(n^2)$ … is it?
- Tighter bound when considering doubling happens "once in a while"

```
           +--+
Insert 11  |11|
           +--+
           +--+--+
Insert 12  |11|12|
           +--+--+
           +--+--+--+--+
Insert 13  |11|12|13|  |
           +--+--+--+--+
           +--+--+--+--+
Insert 14  |11|12|13|14|
           +--+--+--+--+
           +--+--+--+--+--+--+--+--+
Insert 15  |11|12|13|14|15|  |  |  |
           +--+--+--+--+--+--+--+--+
```

Amortization analysis considers sequence of n operations!

Outline

1. Motivation
2. 3 Methods
    1. Aggregate method
    2. Banking method
    3. Potential method

3 methods in amortization analysis

- Aggregate method: the total running time for a sequence of operations is analyzed.
- Accounting (banker's) method:
- Potential (physicist's) method

Aggregate method for resizable array

- Let c_i be the cost of i^{th} insertion
- Let d_i be the cost of doubling
  - Cost of copying existing elements
- S_i be the array size
- $c_i = d_i + 1$

```
            +--+
Insert 11   |11|
            +--+
            +--+--+
Insert 12   |11|12|
            +--+--+
            +--+--+--+--+
Insert 13   |11|12|13|  |
            +--+--+--+--+
            +--+--+--+--+
Insert 14   |11|12|13|14|
            +--+--+--+--+
            +--+--+--+--+--+--+--+--+
Insert 15   |11|12|13|14|15|  |  |  |
            +--+--+--+--+--+--+--+--+
```

i	1	2	3	4	5	6	7	8	9	10
s_i	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

$$d_i = \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ 0 & \text{otherwise} \end{cases}$$

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is a power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

$$T(n) = \sum_1^n c_i = \sum_1^n d_i + n = O(n)$$

3 methods in amortization analysis

- Aggregate method
- Accounting (banker's) method: "pay" for each operation in a way that "charges" money to inexpensive operation to later "pay" for the expensive operation

Accounting method

- find a *payment* charged to each individual operation such that the total cost for seq of n operation $\leq$ sum of the n payments
- Let c_i be the actual cost, c'_i be the payment for i^{th} insertion
- Balance $b_i = c'_i - c_i$
- Balance must stay non-negative for each step: $b_i \geq 0$ for all i
- $\Sigma_{1 \leq i \leq n} \; c_i \;\leq\; \Sigma_{1 \leq i \leq n} \; c'_i$

How much should we charge for each operation?

Sum of the lowest possible charges will be the bound for the time complexity

Accounting method for resizeable array

- $\Sigma_{1 \le i \le n} \; c_i \le \Sigma_{1 \le i \le n} \; c'_i$, $b_i \ge 0$
- Assume it costs 1 unit to insert and 1 unit to copy

How much should we charge for each operation?

Accounting method

i	1	2	3	4	5	6	7	8	9	10
s_i	1	2	4	4	8	8	8	8	16	16
c_i	1	2	3	1	5	1	1	1	9	1

```
           +--+
Insert 11  |11|
           +--+
           +--+--+
Insert 12  |11|12|
           +--+--+
           +--+--+--+--+
Insert 13  |11|12|13|  |
           +--+--+--+--+
           +--+--+--+--+
Insert 14  |11|12|13|14|
           +--+--+--+--+
           +--+--+--+--+--+--+--+--+
Insert 15  |11|12|13|14|15|  |  |  |
           +--+--+--+--+--+--+--+--+
```

- Charge 1 for each insert
- Charge 2 for each insert
- Charge 3 for each insert

What is the minimum charge needed per operation?

Intuition: Why charge 3 unit per insert?

Let m be the m^{th} element inserting

- One unit is paid toward inserting m
- One unit is paid toward copying m
- One unit is paid toward copying one previously inserted element

We can further optimize by charging 1 unit for first insert and charging 3 units for all subsequent inserts

Amortized time complexity of inserting n elements $\leq 3n$, thus $O(n)$

3 methods in amortization analysis

- Aggregate method
- Accounting (banker's) method
- Potential (physicist's) method: defines potential function where potential increases or decreases with each successive operation (but cannot be negative)

Potential method

- Potential method associates credit/potential with the entire data structure
  - Similar concept as “payment” in banker’s method
- The data structure represents prepaid work as "potential energy" that can later be released to pay for future operations.

Potential method: defines potential for entire data structure

- Let c_i be the __________ cost
- D_i be the __________ of the data structure after i^{th} operation
- ϕ_i (D_i): _____________ that maps D_i to a real number
  - Represents potential of Di after i^{th} operation
  - Typically,
  - Need to ensure
- Amortized cost a_i = actual cost of + change in potential by i^{th} operation

- $\sum_0^{n-1} a_i = \sum_0^{n-1} c_i + \phi_n(D_n)$

Potential method: defines potential for entire data structure

- Let c_i be the actual cost
- D_i be the state of the data structure after i^{th} operation
- ϕ_i (D_i): Potential function that maps D_i to a real number
  - Represents potential of Di after i^{th} operation
  - Typically $\phi_0 (D_0) = 0$
  - Need to ensure $\phi_n (D_n) \geq 0$
- Amortized cost a_i = actual cost of + change in potential by i^{th} operation
  - $a_i = c_i + \{\phi_i (D_i) - \phi_{i-1} (D_{i-1})\}$
- $\sum_1^n a_i = c_1 + \ldots c_{n-1} + \phi_n (D_n) - \phi_0 (D_0) = \sum_1^n c_i + \phi_n (D_n)$

Potential method provides a formal proof that amortized cost of the upper bound for the actual cost

In potential method,
figuring out the right potential function is the key

- Array resizing example
  - Initially potential ϕ_0 is 0
  - Good times: When there is enough capacity (N<M), insert 1
  - Bad times: When full (N==M), double the capacity, copy M elements and insert 1.
- How much credit/potential do we need to save up during good times so that can use it during the bad times?

We need to devise a potential function
where credit increases during good times
- just enough to cover M potential energy drop during bad times

Potential method for resizable array

- ϕ_i (D_i): 2N – M
  - N is the number of elements and M is the capacity of the array
- $a_i = c_i + \{\phi_i(D_i) - \phi_{i-1}(D_{i-1})\}$
- **Case 1: no copying needed (N < M)**
  - What is c_i?
  - After i^{th} operation, N -> N +1, no change in M
  - $a_i = c_i + \{2(N+1) - M\} - \{2(N) - M\} = ?$
- **Case 2: copying is needed (N == M, copying N elements)**
  - What is c_i?
  - After i^{th} operation, N -> N +1, M: N -> 2N
  - $a_i = c_i + \{2(N+1) - 2M\} - \{2(N) - M\}$
    $= c_i + \{2(N+1) - 2N\} - \{2(N) - N\}$

Potential method for resizable array

- ϕ_i (D_i):
  - N is the number of elements and M is the capacity of the array
- $a_i = c_i + \{\phi_i (D_i) - \phi_{i-1} (D_{i-1})\}$
- Phase 1: no copying needed (<)
  - What is c_i?
  - After i^{th} operation,
  - $a_i =$
- Phase 2: copying is needed (==)
  - What is c_i?
  - After i^{th} operation,
  - $a_i =$

Practice!

Ex 1: Stack with a multipop

- push(x) //Push x onto the stack
- x = pop() //Pop x from the top of the stack
- multipop(k) Pop the k topmost elements off of the stack.
  - If the stack contains fewer than k items, pop the entire stack.
- Assume NO resizing is needed.

What is the worst-case cost of each operation of stack?

Naïve approach

Costs:

- Push(): O(1)
- pop(): O(1)
- multipop(k): O(min(n,k)), where n is the number of items on the stack
- For a sequence of m operations, the maximum height of the stack is m, thus the worst-case cost for any single operation is O(m)

What is the amortized cost of each operation of stack?

Ex 2: Even more dynamic array

Let c be the capacity of the array.
Let n be the number of actual list elements.

- initialize(): create an empty list with capacity 2. ($c = 2$ and $n = 0$)
- append(): if $c = n$ then grow(). Now insert the element into the table.
- pop(): if $n = c/4$ and $c \geq 4$ then shrink(). Now erase the element from the table

Where
- grow() increases the capacity of the array from c to $2c$,
- shrink() decreases the capacity of the array from c to $c/2$.

What is the amortized cost of append, pop, and initialize?