Lecture 8 : AA (22-08-24)

Stack operations

Push(S, x) : Insert x into Stack S → O(1)

Pop(S) : Delete the top element from → O(1)

Multipop(S, K): Delete the top 'k' element.

Multipop (S, K) : worst case → O(k)

```
while NOT Stack-Empty(S) and K>0
    POP(S)
    K = K-1
```

n: maximum number of elements in the stack

What is the worst-case time for any stack operation? → O(n)

Worst-case analysis

For any sequence of 'n' stack operation

$\boxed{\text{total worst case time required in } O(n^2)}$ → Not tight bound

observation: we can pop an element atmost once after it is pushed into the stack

Therefore, total number of POP( ) operations including the call from MultiPOP( ) is atmost the number of Push( ) operations.

⇓

Basically number(POP) ≤ number(PUSH) provided stack is not empty

Claim 1: Any sequence of n stack operations taken $\boxed{O(n)}$ total time in the $\boxed{\text{worst-case}}$

Claim 2: The $\boxed{\text{average cost}}$ of any stack operation is $O(1)$.

Any sequence of n-operations takes $O(n)$ total time in the worst-case.

Therefore, on average in the worst case, any stack operation takes $\boxed{O(1)}$ time.

$$\frac{O(n)}{n} = O(1)$$

Amortized Analysis
- Aggregrate method
- Accounting method
- Potential method

Key points:

1. Amotized analysis is different from average-case analysis
2. In Amotized analysis, we show that the average cost of an operation is small if we average over a sequence of operations even though a single operation ove the sequence may be expensive.

Aggregate method:

→ We show that for all 'n', a sequence of n operations takes worst-case time $T(n)$ in total.

→ Thus, the average cost of the amortized cost of the operation is $\frac{T(n)}{n}$.

Incrementing a binary counter
(K-bit counter)

$A[0, K-1]$:

$A[0] \rightarrow$ LSB

$A[K-1] \rightarrow$ MSB

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

$\downarrow$

value of the counter

counter value	A[3]	A[2]	A[1]	A[0]
0 $\rightarrow$	0	0	0	0
1 $\rightarrow$	0	0	0	1
2 $\rightarrow$	0	0	1	0
3 $\rightarrow$	0	0	1	1
4 $\rightarrow$	0	1	0	0
5 $\rightarrow$	0	1	0	1
6 $\rightarrow$	0	1	1	0
7 $\rightarrow$	0	1	1	1
8 $\rightarrow$	1	0	0	0
9 $\rightarrow$	1	0	0	1
10 $\rightarrow$	1	0	1	0
11 $\rightarrow$	1	0	1	1
12 $\rightarrow$	1	1	0	0
13 $\rightarrow$	1	1	0	1
14 $\rightarrow$	1	1	1	0
15 $\rightarrow$	1	1	1	1

Increment(A)

```
i = 0
while i < A.length and A[i] == 1
        A[i] = 0
        i = i + 1
if i < A.length
        A[i] = 1
```

Cost of increment = Number of bits flipped

Claim 1: In the worst-case an increment takes $\Theta(K)$ time.

Claim 2: over any sequence of n increments, total worst-case time taken by a K-bit counter in ~~$\Theta(nk)$~~ $\rightarrow < 2n \in O(n)$

$A[0] \rightarrow n$
$A[1] \rightarrow n/2$
$A[2] \rightarrow n/4$
$\vdots$
$A[i] \rightarrow n/2^i$

Total number of bits-flipped over any sequence of n-increment in

$$T(n) = \sum_{i=0}^{K-1} n/2^i \leq n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i}$$
$$= 2n$$

$$\frac{a}{1-r}$$

$$= \frac{1}{1-\frac{1}{2}} = \frac{1}{\frac{1}{2}} = 2$$

Q. Suppose we perform a sequence of $\boxed{n\text{-operations}}$ on a data structure in which cost of the $\boxed{i\text{-th operation}}$ is defined as follows:

$$\boxed{\begin{aligned} c_i &= i \text{ if } i \text{ is exact power of } 2 \\ &= 1 \text{ otherwise} \end{aligned}}$$

what is the amortized cost for operation?

Solution:

$$T(n) = \boxed{\sum_{i=1}^{n} c_i}$$

Calculate the exact value of this expression

$$< n + \sum_{K=0}^{\log n} 2^K$$
$$= 3n - 1$$
$$T(n) \in O(n)$$

Accounting method:

→ In this method, we assign different charges (cost/credit) to different operations. Some operations are charged more than the actual cost and some operations are charged less than the actual cost.

→ Amount charged to an operation is called amortized cost of the operation

→ Charge/credit = Amortized cost − Actual cost

Goal: "The amortized cost of a sequence of n operations is an upper bound on the actual cost"

$\hat{c}_i$ = amortized cost for operation 'i'

c_i = actual cost for operation

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

$$\Rightarrow \underbrace{\sum \hat{c}_i - \sum_{i=1}^{n} c_i}_{} \geq 0$$

$\boxed{\text{Total Credit} \geq 0}$: ← Goal

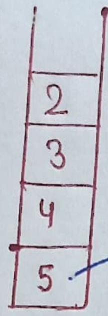
If Total credit is closer to zero we would be getting tight bound

Stack operation

Operation	Actual cost (c_i)	Amortized cost ($\hat{c}_i$)
Push(S, x)	1	2
Pop (S)	1	0
Multipop(S,k)	min(n,k)	0

no of elements present in the stack

Observation: An element must be pushed into the stack before performing the pop() operation



2 units → 1 unit to pay for the push( )
→ 1 unit to be used for pop()

$$\hat{T}(n) = \sum_{i=1}^{n} \hat{c}_i = 2n = O(n)$$

Incrementing a binary counter

operation	Amortized Cost ($\hat{c}_i$)
set a bit $0 \to 1$	2 units
reset a bit $1 \to 0$	0 units

2 units → 1 unit for set in the current operation (flip)
→ 1 unit for reset in future

[number of resets operations $\leq$ number of set operations]

Total amortized cost [$\hat{T}(n)$] = 2n $\in O(n)$

from the previous observation.