Design and Analysis of Algorithms CS218M

Amortized Complexity

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Worst Case Complexity versus Amortized Complexity

Stack

- Push(S,x) and Pop(S). Each operation is $O(1)$.

Worst Case Complexity versus Amortized Complexity

Stack

- Push(S,x) and Pop(S). Each operation is $O(1)$.
- Aggregate cost of n operations $O(n)$. Hence Amortized cost per operation is $O(n)/n \; = \; O(1)$.

Worst Case Complexity versus Amortized Complexity

Stack

- Push(S,x) and Pop(S). Each operation is $O(1)$.
- Aggregate cost of n operations $O(n)$. Hence Amortized cost per operation is $O(n)/n \; = \; O(1)$.
- Multipop(S,k) pops k elements and leaves stack empty if $k > |S|$.

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

Worst Case Complexity versus Amortized Complexity

Stack

- Push(S,x) and Pop(S). Each operation is $O(1)$.
- Aggregate cost of n operations $O(n)$. Hence Amortized cost per operation is $O(n)/n = O(1)$.
- Multipop(S,k) pops k elements and leaves stack empty if $k > |S|$.

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

- Worst case complexity of Multipop(S,k) is $O(min(s, k))$. Hence, Aggregate cost of n operations is $O(n^2)$. Pessimistic.

Worst Case Complexity versus Amortized Complexity

Stack

- Push(S,x) and Pop(S). Each operation is $O(1)$.
- Aggregate cost of n operations $O(n)$. Hence Amortized cost per operation is $O(n)/n = O(1)$.
- Multipop(S,k) pops k elements and leaves stack empty if $k > |S|$.

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k > 0
    POP(S)
    k = k - 1
```

- Worst case complexity of Multipop(S,k) is $O(min(s, k))$. Hence, Aggregate cost of n operations is $O(n^2)$. Pessimistic.
- Before you delete k elements you must do k push, each of $O(1)$. Hence Aggregate cost of k push and one Muplipop(S,k) is $O(k)$ and amortized cost per operation is $O(1)$.

Accouting for Low Complexity Operations: Credit and Potential

- Idea: For each low complexity operation, retain some credit to be averaged with high complexity operation.

Accouting for Low Complexity Operations: Credit and Potential

- Idea: For each low complexity operation, retain some credit to be averaged with high complexity operation.
- given a sequence of n operations, let c_i be the actual cost of op_i and let $\hat{c}_i$ be the amortized cost of op_i.

Accouting for Low Complexity Operations: Credit and Potential

- Idea: For each low complexity operation, retain some credit to be averaged with high complexity operation.
- given a sequence of n operations, let c_i be the actual cost of op_i and let $\hat{c}_i$ be the amortized cost of op_i.
- Let, $CREDIT = \Sigma_{i=1}^{n} \hat{c}_i - \Sigma_{i=1}^{n} c_i$. We require that $CREDIT \geq 0$.

Accouting for Low Complexity Operations: Credit and Potential

- Idea: For each low complexity operation, retain some credit to be averaged with high complexity operation.
- given a sequence of n operations, let c_i be the actual cost of op_i and let $\hat{c}_i$ be the amortized cost of op_i.
- Let, $CREDIT = \Sigma_{i=1}^{n} \hat{c}_i - \Sigma_{i=1}^{n} c_i$. We require that $CREDIT \geq 0$.
- Potential of a datatype configuration D is denoted $\Phi(D)$. It gives CREDIT retained by reaching D. It can define Credit

Potential

- Let op_i take datatype from config D_{i-1} to D_i.
  Hence $\hat{c}_i \; = \; c_i \; + \; \Phi(D_i) - \Phi(D_{i-1})$..
  Hence, $\Sigma_{i=1}^{n} \; \hat{c}_i \; = \; (\Sigma_{i=1}^{n} \; c_i) \; + \; \Phi(D_n) - \Phi(D_0)$
- If invariantly$\Phi(D_i) - \Phi(D_{i-1}) \geq 0$ then aggregate $\Sigma_{i=1}^{n} \; \hat{c}_i$ is upperbound on $Sigma_{i=1}^{n} \; c_i$.
- Potential function determines amortized cost $\hat{c}_i$ of each operation.
- We typically define $\Phi(D_0) = 0$ and require that $\Phi(D_i) \geq 0$.

Stack

Let $\Phi(S) = |S|$ the number of elements in the stack.

Stack

Let $\Phi(S) = |S|$ the number of elements in the stack.

- Hence $D_0 = 0$ and $D_i \geq 0$ for all D_i.

Let $\Phi(S) = |S|$ the number of elements in the stack.

- Hence $D_0 = 0$ and $D_i \geq 0$ for all D_i.
- Let $Op_i = Push(S, x)$: We have $\Delta\Phi \ = \ 1$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi(D) \ = \ 1 + 1$.
- Let $Op_i = Pop(S$: We have $\Delta\Phi \ = \ -1$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi(D) \ = \ 1 + (-1)$.
- Let $Op_i = Multipop(s, k)$: We have $\Delta\Phi \ = \ -min(s, k)$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi \ = \ min(s, k) + (-min(s, k)) \ = \ 0$.

Let $\Phi(S) = |S|$ the number of elements in the stack.

- Hence $D_0 = 0$ and $D_i \geq 0$ for all D_i.
- Let $Op_i = Push(S, x)$: We have $\Delta\Phi \ = \ 1$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi(D) \ = \ 1 + 1$.
- Let $Op_i = Pop(S$: We have $\Delta\Phi \ = \ -1$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi(D) \ = \ 1 + (-1)$.
- Let $Op_i = Multipop(s, k)$: We have $\Delta\Phi \ = \ -min(s, k)$.
  Hence, $\hat{c}_i \ = \ c_i + \Delta\Phi \ = \ min(s, k) + (-min(s, k)) \ = \ 0$.
- Hence amortized cost of each stack operation is $O(1)$.

Binary Counter

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 of k bits

represented as bit-array $A[0..k-1]$.

```
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
```

represented as bit-array $A[0..k-1]$.

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
```

- Worst case complexity of *INC* is $\theta(k)$.

Binary Counter of k bits

represented as bit-array $A[0..k-1]$.

```
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
```

- Worst case complexity of *INC* is $\theta(k)$.
- Amortized worst case complexity of *INC* is ???.

Amortized Complexity: Counter

Let $\Phi(D)$ be number of trailing 1 in config D. Let $t_i = \Phi(D_{i-1}$.

- $c_i = t_i + 1$, Also $\Delta\Phi \ = \ -t_i$.
- Hence $\hat{c}_i \ = \ c_i + \Delta\Phi \ = \ 1$. Hence $O(1)$.

Dynamic Table

- TABLE-INSERT
- TABLE-DELETE

Dynamic Table

- TABLE-INSERT
- TABLE-DELETE
- Table Expansion:
  $T.size$ the size of table.
  $T.num$ Number of occupying elements in table.

Dynamic Table

- TABLE-INSERT
- TABLE-DELETE
- Table Expansion:
  $T.size$ the size of table.
  $T.num$ Number of occupying elements in table.
- If $T.size = T.num$ then before insert, allocate double sized table and copy.

Dynamic Table Insert

```
TABLE-INSERT(T, x)
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.num == T.size
    allocate new-table with 2 · T.size slots
    insert all items in T.table into new-table
    free T.table
    T.table = new-table
    T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Costs

INSERT-TABLE

Costs

INSERT-TABLE

- If $T.num < T.size$ then $c_i = 1$.

Costs

INSERT-TABLE

- If $T.num < T.size$ then $c_i = 1$.
- If $T.num = T.size$ then $c_i = num_i$

Potential Function

Becomes 0 every-time the table is doubled.

Potential Function

Becomes 0 every-time the table is doubled.

- $\Phi(T) = 2 \cdot T.num - T.size$.

Potential Function

Becomes 0 every-time the table is doubled.

- $\Phi(T) \; = \; 2 \cdot T.num \; - \; T.size$.
- Initially, $T.num = T.size = 0$. Hence $\Phi_0 = 0$.

Potential Function

Becomes 0 every-time the table is doubled.

- $\Phi(T) \ = \ 2 \cdot T.num \ - \ T.size$.
- Initially, $T.num = T.size = 0$. Hence $\Phi_0 = 0$.
- Assume occupancy α is at least 0.5. Hence, $\Phi_i \geq 0$.

Potential Function

Becomes 0 every-time the table is doubled.

- $\Phi(T) \; = \; 2 \cdot T.num \; - \; T.size$.
- Initially, $T.num = T.size = 0$. Hence $\Phi_0 = 0$.
- Assume occupancy α is at least 0.5. Hence, $\Phi_i \geq 0$.
- Case 1: $T.num < T.size$. Then,

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\
&= 3 \,.
\end{aligned}
$$

Potential Function

Becomes 0 every-time the table is doubled.

- $\Phi(T) \;=\; 2 \cdot T.num \;-\; T.size$.
- Initially, $T.num = T.size = 0$. Hence $\Phi_0 = 0$.
- Assume occupancy α is at least 0.5. Hence, $\Phi_i \geq 0$.
- Case 1: $T.num < T.size$. Then,

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\
&= 3\,.
\end{aligned}
$$

- Case 2: $T.num = T.size$. Then,

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\
&= num_i + 2 - (num_i - 1) \\
&= 3\,.
\end{aligned}
$$

TABLE-DELETE

- Let $\alpha = T.num/T.size$ be the load factor.
- Double the table on INSERT when $\alpha_{i-1} = 1$.

TABLE-DELETE

- Let $\alpha = T.num/T.size$ be the load factor.
- Double the table on INSERT when $\alpha_{i-1} = 1$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/2$

TABLE-DELETE

- Let $\alpha = T.num/T.size$ be the load factor.
- Double the table on INSERT when $\alpha_{i-1} = 1$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/2$
- Aggregate complexity of n operations becomes $O(n^2)$.

TABLE-DELETE

- Let $\alpha = T.num/T.size$ be the load factor.
- Double the table on INSERT when $\alpha_{i-1} = 1$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/2$
- Aggregate complexity of n operations becomes $O(n^2)$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/4$

TABLE-DELETE

- Let $\alpha = T.num/T.size$ be the load factor.
- Double the table on INSERT when $\alpha_{i-1} = 1$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/2$
- Aggregate complexity of n operations becomes $O(n^2)$.
- Halve the table on DELETE when $\alpha_{i-1} = 1/4$

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