Divide and Conquer

Karatsuba

$x = \underset{a}{56}\,\underset{b}{78}$ $\quad O(n^{\log_2 3})$

$y = \underset{c}{12}\,\underset{d}{34}$

$x = 10^{n/2}a + b,\ y = 10^{n/2}c + d$

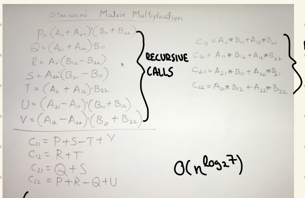
$x \cdot y = (10^{n/2}a + b)(10^{n/2}c + d)$

$x \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd$

1) Compute ac
2) Compute bd
3) Compute $(a+b)(c+d)$
4) ③ - ① - ② = $ad + bc$

Recursive Calls

Strassen's



By Abdul Bari ♥

The Potential Method

- Represents the prepaid work as potential which can be released to pay for future operations.
- Will perform n operations, starting with an initial data structure D_0.
- For each $i = 1, 2, \dots, n$, we let c_i be the actual cost of the ith operation and D_i be the data structure after applying the ith operation to data structure D_{i-1}.
- Potential function: Φ maps each data structure D_i to a real number $\Phi(D_i)$ which is the potential associated with data structure D_i.
- Amortized Cost $\hat{c}_i$ of the ith operation with respect to Φ:

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

- Total amortized cost:
  $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$
  $= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$
- If $\Phi(D_n) \geq \Phi(D_0) \longrightarrow \sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$
- $\Phi(D_i) \geq \Phi(D_0), \forall i$ to guarantee we pay in advance.
- Define $\Phi(D_0) = 0$, and show that $\Phi(D_i) \geq 0, \forall i$.
- $\Phi(D_i) - \Phi(D_{i-1}) > 0$?
  ↳ $\hat{c}_i$ is an overcharge to the ith operation. And the potential increases.
- $\Phi(D_i) - \Phi(D_{i-1}) < 0$?
  ↳ Decrease potential.

Stack Operations: PM

- Φ: number of objects in the stack.
- $\Phi(D_0) = 0$. Empty Stack
- D_i has nonnegative potencial because a stack cannot have a negative number of elements.
  $\Phi(D_i) \geq 0$
  $= \Phi(D_0)$
- If the ith operation is a PUSH and s is the total number of objects, then:
  $\Delta\Phi_i = \Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$
  Amortized Cost of Push:
  ↳ $\hat{c}_i = c_i + \Phi(D_i - \Phi(D_{i-1})$
  $= 1 + 1 = 2$
- If ith operation is MULTIPOP (S, K), which causes $K' = \min(K, S)$ objects to be popped off the stack.
  $\Phi(D_i) - \Phi(D_{i-1}) = (s - K') - s = K'$
  Amortized cost of MultiPop
  ↳ $\hat{c}_i = c_i + \Delta\Phi_i$
  $\hat{c}_i = K' - K' = 0$
- Amortized cost of the three operations is $O(1)$. Thus, for a sequence of n operations, the amortized cost is $O(n)$.
- Since $\Phi(D_i) \geq \Phi(D_0)$, the total amortized cost is an upper bound of the total actual cost.
- The worst-case of n operations is $O(n)$

Accounting Method

- Assign different charges to different operations, with some operations charged more or less than they actually cost. Amortized cost
- Credit: When an operation's amortized cost exceeds the actual cost.
- Different operations may have different amortized costs.
- Actual cost of ith operation: c_i
- Amortized cost of the ith operation: $\hat{c}_i$

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

- Total credit: $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$
  ↳ must be nonnegative at all times

Amortized Analysis

Aggregate Method

1) Calculate the total cost of all the n operations as $T(n)$
2) Calculate the average cost of each operation a $T(n)/n$

Aggregate method considers that each operation has the same cost (amortized).

Account Method

$\hat{c}_i = \$3.00 \begin{cases} \$1.00 & \text{for insert} \\ \$2.00 & \text{stored for later use} \end{cases}$

0 0 0 0 2 2 2 2 (1.00, 1.00)

0 0 0 0 0 0 0 0 2 2 2 2 ...

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

	cost ($\hat{c}_i$)	balance (\$)	cost ($\hat{c}_i$)	balance (\$)
a	3	2 = 3-1	2	1 = 2-1
a b	3	3 = 2+3-1-1	2	1 = 1+2-1-1
a b c	3	3 = 3+3-2-1	2	0 = 1+2-2-1
a b c d	3	5 = 3+3-1	2	1 = 0+2-1
a b c d e	3	3 = 5+3-4-1	2	-2 = 1+2-4-1
a b c d e f	3	5 = 3+3-1	2	
a b c d e f g		7 = 5+3-1	2	
a b c d e f g h		9 = 7+3-1	2	
		3 = 9+3-8-1	2	

Potential Method

$\Phi: \{D_i\} \to \mathbb{R}$
- $\Phi(D_0) = 0$
- $\Phi(D_i) \geq 0, \forall i$

$\hat{c}_i = c_i + \Delta(\Phi_i)$
$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Bank Account → Potencial Energy

i-th operation transform $D_{i-1} \to D_i$

$\hat{c}_i = c_i + \Delta(\Phi_i)$
$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $\Delta\Phi_i > 0 \Rightarrow \hat{c}_i > c_i$
- $\Delta\Phi_i < 0 \Rightarrow \hat{c}_i < c_i$

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

Problem

$\Phi(D_i) = 2i - \text{size}$

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
$\hat{c}_i = c_i + (2i - \text{size}) - (2(i-1) - \text{size})$

Case 1: // Insertion

$\hat{c}_i = 1 + (2i - \text{size}) - (2(i-1) - \text{size})$
$\hat{c}_i = 1 + \cancel{2i} - \cancel{\text{size}} - \cancel{2i} + 2 + \cancel{\text{size}}$
$\hat{c}_i = 3$

Case 2: // Expansion

$\hat{c}_i = \underbrace{(i+1)}_{\text{copies + insert}} + (2i - \overset{\text{size}}{2i}) - (2(i-1) - \overset{\text{size}}{i})$

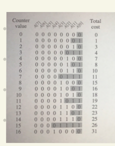
$\hat{c}_i = i + 1 + 2i - 2i - 2i + 2 + i$
$\hat{c}_i = 3$

Aggregation: Incrementing a Binary Counter

- Array that counts upwards from 0
- $A.\text{length} = K$
- LSB $= A[0]$, MSB $= A[K-1]$
- $x \sum_{i=0}^{K-1} A[i] \cdot 2^i$.

$\boxed{\text{Increment}(A)}$

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



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

$\sum_{i=0}^{K-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$
$= n \left(\frac{1}{1 - 1/2} \right)$
$= 2n$

- The worst-case time for a sequence of n Increment operations is therefore $O(n)$
- The average cost of each operation, and therefore the amortized cost per operation, is $O(n)/n = O(1)$

Accounting Method: Incrementing a Counter

- \$1.00 to flip a bit
- Amortized cost of \$2.00 to set a bit to 1
- Actual cost to flip a bit: 1
- Credit after setting a bit: \$1.00. We can use it to set it back to 0.
- At any point in time, every 1 has a dollar of credit on it. Thus, the amortized cost to reset a bit is 0.
- Cost of resetting the bits within the while loop is paid for by the dollars on the bits that are reset.
- "Increment" procedure sets at most one bit, and therefore the amortized cost of an Increment is at most 2.
- Total 1s never become negative, and thus the amount of credit stays nonnegative at all times.
- Thus, for n Increment operations, the total amortized cost is $O(n)$, which bounds the total actual cost.

Incrementing a Binary Counter: PM

- Φ of the counter after the ith Increment is b_i, the total number of 1's after the ith operation.
- ith Increment resets t_i bits.
- Actual cost of the operation is at most $t_i + 1$
- If $b_i = 0$, then the ith operation resets all K bits, and so $b_{i-1} = t_i = K$.
- If $b_i > 0$, then $b_i = b_{i-1} - t_i + 1$.
- In either case, $b_i \leq b_{i-1} - t_i + 1$
- $\Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1}$
  $= 1 - t_i$
- Amortized cost:
  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
  $\leq (t_i + 1) + (1 - t_i)$
  $= 2$
- If the counter starts at 0, $\Phi(D_0) = 0$. Since $\Phi(D_i) \geq 0, \forall i$, the total amortized cost of a sequence of n Increment operations is an upper bound on the total actual total cost, and so the worst-case cost of n Increment operations is $O(n)$.
- If $b_0 \geq 0$:
  - After n Increment Operations it has b_n 1s. where $0 \leq b_0$ and $b_n \leq K$.
  - $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0)$
  - $\hat{c}_i \leq n, \forall 1 \leq i \leq n$. $\Phi(D_0) = b_0$ and $\Phi(D_n) = b_n$
  - Total actual cost of n Increment operations
    $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} 2 - b_n + b_0$
    $= 2n - b_n + b_0$
  - $b_0 \leq K$ as long as $K = O(n) \to$ Actual cost is $O(n)$
  - If we execute at least $n = \Omega(K)$ Increment operations the total actual cost is $O(n)$, no matter b_0.