CSCE 750
10/17/23

# Amortized Analysis

## Matrix chain mult (cont. first)

Recall $A_1, \ldots, A_n$ positive ints $p_0, \ldots, p_n$

$A_i$ is a $p_{i-1} \times p_i$ matrix

Multiply $A_1 \cdots A_n$ minimizing the number of scalar multiplications.

$$\underset{m \times n}{A} \cdot \underset{n \times s}{B}$$


takes $mns$ multiplications.

Notation: $A_{i..j} = A_i \cdots A_j$

$$\underset{p_0 \times p_j}{(A_1 \cdots A_j)}\underset{p_j \times p_n}{(A_{(j+1)} \cdots A_n)}$$

last mult takes $p_0 p_j p_n$ mults.

Generally: $$\underset{p_{i-1} \times p_k}{(A_i \cdots A_k)}\underset{p_k \times p_j}{(A_{(k+1)} \cdots A_j)}$$ $i < j$

Table $m[1..n, 1..n]$ such that,

for $1 \le i \le j \le n$, $m[i,j]$ = optimal # of mults to compute $A_{i..j}$

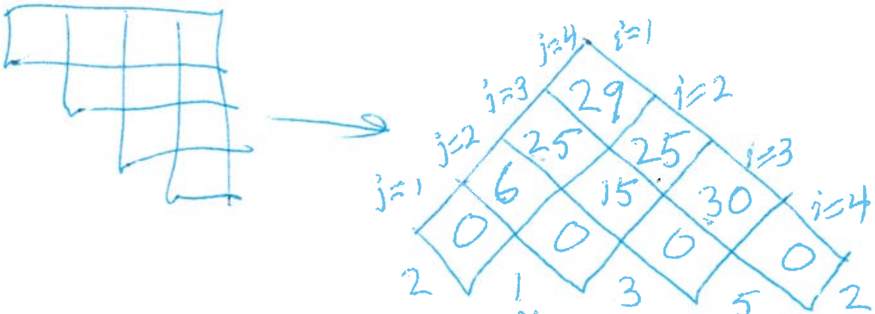
$i = j$: $m[i,j] = m[i,i] = 0$

$i < j$: $m[i,j] = \min\{m[i,k] + m[k+1,j] + p_{i-1}p_kp_j : i \le k \le j-1\}$

Fill m-table in order of increasing $j-i$ value

$$\underbrace{(A_i \cdots A_k)}_{m[i,k]}\underbrace{(A_{k+1} \cdots A_j)}_{m[k+1,j]}$$

Example: $\langle p_0, p_1, p_2, p_3, p_4 \rangle = \langle 2, 1, 3, 5, 2 \rangle$




k=1
25 + 0 + 2·1·2 = 29

k=2
6 + 30 + 2·3·2 = 48

k=3
25 + 0 + 2·5·2 = 45

15 + 0 + 2·1·5 = 25
vs
6 + 0 + 2·3·5 = 36

k=2
30 + 0 + 1·3·2 = 36
vs
k=3
15 + 0 + 1·5·2 = 25

Finding the optimal grouping:

For each $m[i,j]$, remember $(i<j)$ the value of k that gave you the min. Use $m[j,i]$ for this

$m[4,1] = 1$ $\qquad$ $A_1((A_2 A_3) A_4)$

$m[4,2] = 3$

Total run time: $\Theta(n^3)$.

---

Amortized analysis: used to determine an upper bound on the total time of a sequence of operations on a data structure.

Idea: Spread the cost of expensive ops ~~across~~ across the cheap ones, so that all ops have roughly the same cost.

Stack with multipop: Linked list support

push — Insert at front ($O(1)$ cost)

pop(k) — pop k items off the stack (or stop if stack is empty) ($O(k)$ cost)

Choose units so that

push ~~pop~~ — cost 1

multipop — cost k.

Start with an empty stack.

Seq of n operations takes time $O(n)$.

b/c the amortized cost of an op is $O(1)$. 

Potential Method. Let $D_0$ be the initial state of a fixed data struct. Let $D_i$ be the state (snapshot) of the struct after the $i$'th operation.

Suppose $c_i$ = cost of the $i$'th operation.

We will define the amortized cost $\hat{c}_i$ of the $i$'th operation such that

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

An upper bound on the $\sum_i \hat{c}_i$ then given an upper bound on $\sum_i c_i$.

Potential method: $\underset{\substack{\uparrow \\ \text{snapshot}}}{\Phi(D)} \in \mathbb{R}$ "potential of D" such that:

1) $\Phi(D_0) = 0$

2) $\Phi(D_i) \geq 0$ for all $i$.

Now define:

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

Then

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

$$= \left( \sum_{i=1}^{n} c_i \right) + \Phi(D_n) - \underbrace{\Phi(D_0)}_{=0}$$

$$= \sum_{i=1}^{n} c_i + \underbrace{\Phi(D_n)}_{\geq 0} \geq \sum_{i=1}^{n} c_i \quad (*)$$

Best: choose $\Phi$ so that $\hat{c}_i$ is small for all $i$

Stack with multipop:

$$\Phi(D) = \text{current size of the stack}$$

$$\hat{c}_i = \begin{cases} \underbrace{1}_{\text{cost}} + \underbrace{1}_{\substack{\text{change in} \\ \text{stack size}}} = 2 & c_i = \text{push} \\ k - k = 0 & c_i = \text{multipop}(k) \end{cases}$$

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

Choose $\Phi$ to be big before an expensive op, ~~so you~~ and small afterwards (to cover the cost).

Ex: Resizable array: Array of items, supporting

Lookup (cost 1)

$$\text{InsertAtEnd}\begin{cases}\text{cost } 1 & \text{if no resize}\\ \text{cost } \underline{n} & \text{if resize}\end{cases}$$

n = # number of items after Insertion