Lec24-examples-DP-flow

Sunday, November 5, 2023 9:05 PM

Optimal binary search tree

Problem: we are given sorted keys $k_1, \ldots, k_n$, and probabilities $p_1, \ldots, p_n$ that key i will be accessed at any point in time. Construct a binary search tree T that minimizes

$$C[T] = \sum_{i=1}^{n} p_i \, (\underbrace{\text{Depth}(T, k_i)}_{\text{distance from root to key } k_i} + 1)$$

expected cost to find a key in tree by going down.

Let $D(T, k) = \text{Depth}(T, k)$ for brevity.

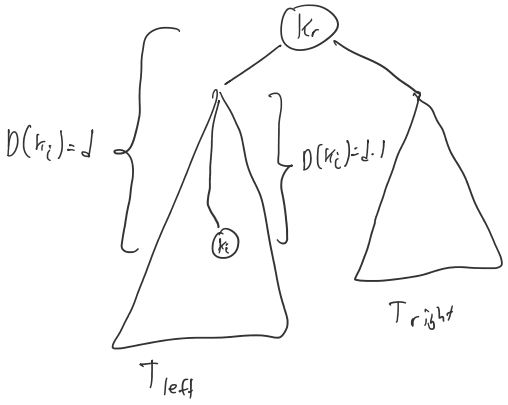
Let T be an optimal tree with root k_r.

$$C(T) = p_r + \underbrace{\sum_{a=1}^{r-1} p_a (D(T, k_a) + 1)}_{\text{left subtree}} + \underbrace{\sum_{a=r+1}^{n} p_a (D(T, k_a) + 1)}_{\text{right subtree}}$$

$$= p_r + \sum_{a=1}^{r-1} p_a + \sum_{a=1}^{r-1} p_a D(T, k_a) + \sum_{a=r+1}^{n} p_a + \sum_{a=r+1}^{n} p_a D(T, k_a)$$

$$= \sum_{a=1}^{n} p_a + \sum_{a=1}^{r-1} p_a (D(T_{left}, k_a) + 1) + \sum_{a=r+1}^{n} p_a (D(T_{right}, k_a) + 1)$$

$$= \sum_{a=1}^{n} p_a + C(T_{left}) + C(T_{right})$$

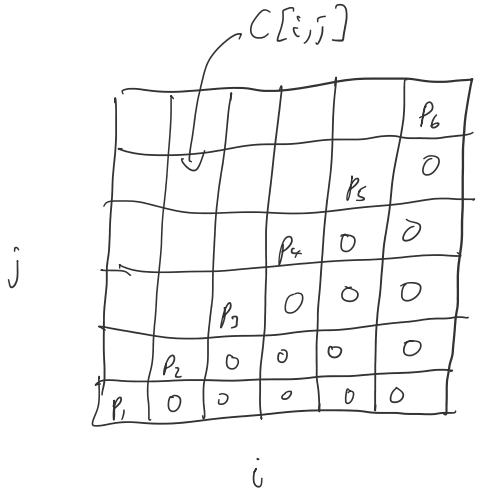


Recurrence: $C[i, j]$ = cost of optimal binary search tree on keys $k_i, \ldots, k_j$.

$$C[i,j] = \begin{cases} 0 & \text{if } j < i \quad \text{(tree is empty)} \\ p_i & \text{if } i = j \quad \text{(tree is single node)} \\ \left(\sum_{a=i}^{j} p_a\right) + \min_r \left\{ C[i,r-1] + C[r+1,j] \right\} \end{cases} \Big\} \text{ base cases}$$

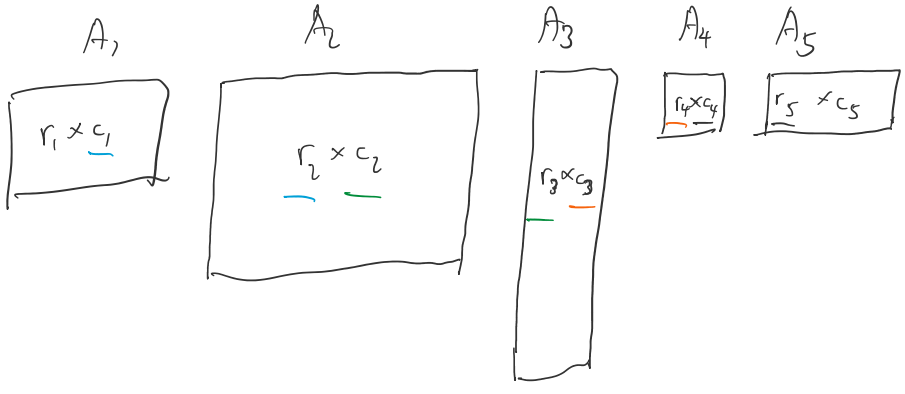
↑ minimum over possible choices of root

Want: $C[1,n]$. Can fill $C[i,j]$ matrix in increasing order of $j-i$.



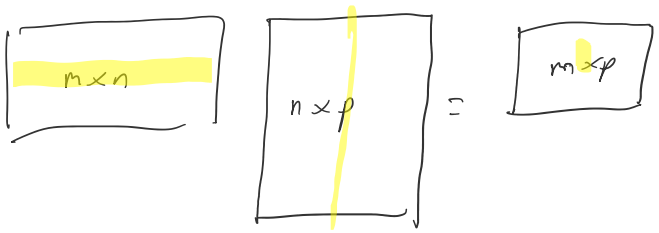
Matrix-chain multiplication

Want to multiply matrices $A_1, A_2, \ldots, A_n$:



$r_1 \times c_5$ answer

Cost of multiplication:

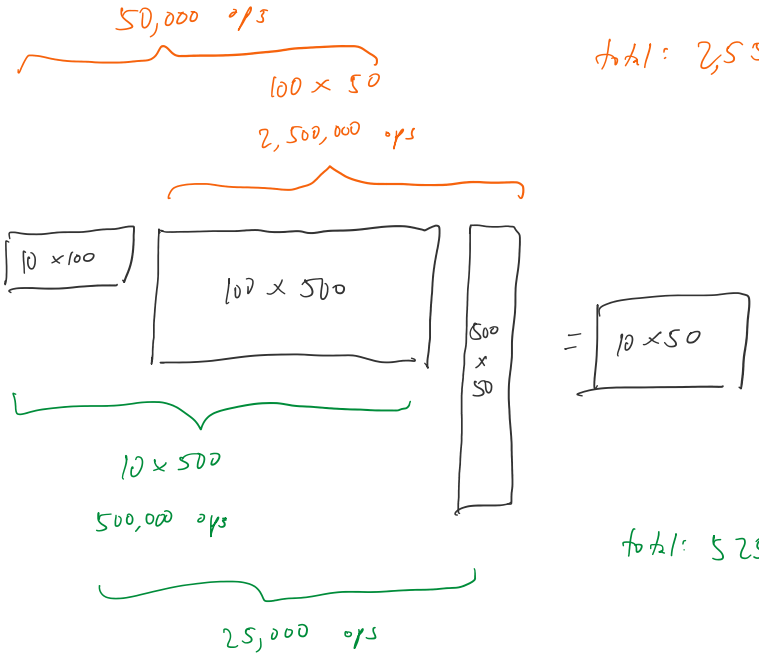


$m \times n \times p$ multiplications

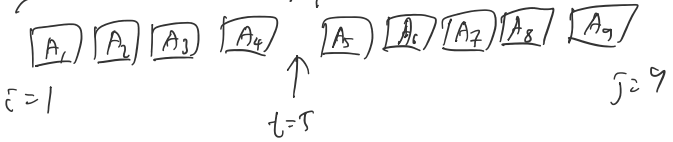
But $(A_1 A_2)((A_3 A_4)(A_5 A_6)) = (A_1 (A_2 A_3))(A_4 (A_5 A_6)) = \cdots$

All multiplication orders give same results, but take different amounts of time.

Ex.



DP recurrence: $OPT(i,j) = OPT(i,t-1) + OPT(t,j)$ + multiplying the two matrices

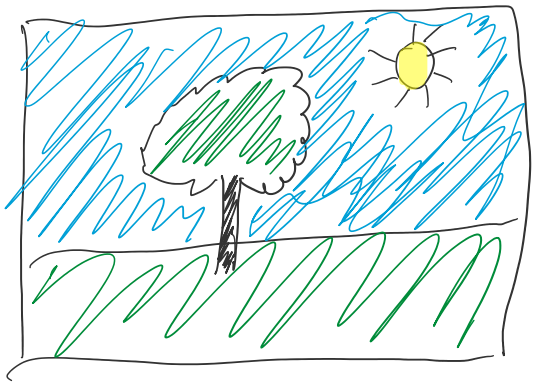


$$OPT(i,j) = \min_{i \le t \le j} \{ OPT(i,t-1) + OPT(t,j) + r_i c_{t-1} c_j \}$$

Base case: $OPT(i, i+1) = mul(A_i, A_{i+1}) = r_i c_i c_{i+1}$.

Image segmentation:

Image segmentation:



Given an image, what is foreground & what is background?

e.g. a tree against a blue sky.

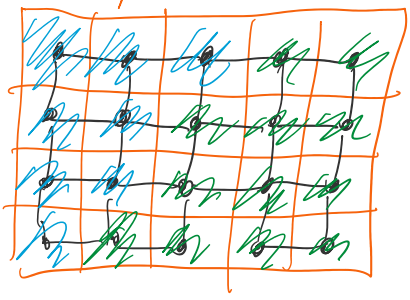
Idea: Use color. If p_i is blue, likely to be background.
If p_i is green, likely to be foreground.

More formally, have likelihoods
a_i = likelihood pixel is in foreground
b_i = likelihood pixel is in background

Caveat: If p_i is a foreground pixel, then nearby pixels are also likely foreground.

Idea: Encode image by undirected graph $G(V, E)$
- V: pixels
- E: edges between adj neighbors

excerpt from image as graph



sky — tree

penalty p_{ij} for putting i in foreground and j in background or vice versa.

Image segmentation problem: Partition the set of pixels into two sets A & B to maximize

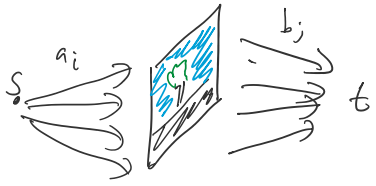
$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ i,j \text{ sep.}}} p_{ij}.$$

Minimum Cut problem: Partition the vertices of a directed graph

Minimum Cut problem: Partition the vertices of a directed graph into 2 sets $A \dot{+} B$ with $s \in A$, $t \in B$ to minimize weight of edges crossing from A to B

Differences: max vs. min
source & sink
objective function $q(A, B)$
undirected vs. directed.

Reduction: Make all edges into two directed edges with capacity p_{ij}.
Add source s with edges pointing to every pixel with capacity a_i
Add sink t with edges pointing from every pixel with capacity b_j.



Want to maximize $q(A,B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$

Notice: Let $Q = \sum_{i \in V} (a_i + b_i) = \left(\sum_{i \in A} a_i + \sum_{j \in B} b_j\right) + \left(\sum_{i \in A} b_i + \sum_{j \in B} a_j\right)$.

maxing this = minning this

Same as minimizing $q'(A,B) = \sum_{i \in A} b_i + \sum_{j \in B} a_i + \sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$

Capacity of cut (A, B):

- $\sum_{i \in A} b_i$: cutting edges (i, t) $\forall i \in A$ (cost of putting i in foreground)
- $\sum_{j \in B} a_i$: cutting edges (s, i) $\forall i \in B$ (cost of putting i in background)
- $\sum_{\substack{(i,j) \in E \\ i,j \text{ sep}}} p_{ij}$: cuts b/t edges in image

Thus, finding the minimum cut in this network flow is exactly optimizing the foreground-background segmentation.