

02-613 Week 11

Algorithms and Advanced Data Structures

Aidan Jan

November 5, 2025

Dynamic Programming (Continued)

Optimal Binary Search Tree

Given a sorted list of keys k_1, k_2, \dots, k_n , and the probabilities p_1, p_2, \dots, p_n that key i will be accessed, construct a binary search tree T that minimizes

$$\text{cost}(T) = \sum_{i=1}^n p_i (\text{Depth}(T, K_i) + 1)$$

Let $r \in [i]$ be the root of T .

To solve this problem, we can first break up this equation to account for the two subproblems: either go left or go right.

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

We can then extract the +1's.

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

From here, we can condense the sums of the probabilities (first 3 terms), and recognize that the two terms summing depth terms are our subproblems. Now, we can write a recurrence relation:

$$= \sum_{i=1}^n p_i + C(T_{\text{left}}) + C(T_{\text{right}})$$

Let C be defined as

$$C[i, j] = \begin{cases} 0 & i > j \\ p_i & i = j \\ \sum_{l=1}^n p_l + C(1, r-1) + C(r+1, j) & i < r < j \end{cases}$$

Basically, $C[i, j]$ represents the optimal for the tree containing numbers of indices i to j . Therefore, our optimal solution for the full tree would be $C[1, n]$.

- The first case (0) is to ensure that $i < j$.
- The second case (p_i) is when only one index is selected.
- The third case (the sum) is the general case.

To solve this, we can fill in a $n \times n$ grid, where i iterates from $1 \dots n$ across columns and j iterates from $n \dots 1$ across rows.

- As a consequence of the recurrence relation, all numbers below the minor diagonal are zeros (because $i > j$).
- Additionally, numbers on the minor diagonal are the values of p_i .
- We want to solve for $(1, n)$, which would be the top left corner.
- We iterate from the diagonal outwards, since the base case is the probability values on the diagonal.
- This takes $O(n^3)$ to run. There are n^2 entries to fill, and each entry takes $O(n)$ since it is a sum of a subset of the next diagonal.

Now, we have a table. How do we recover the tree?

- We create another table r for recovery. Let

$$r[i, j] = \arg \min \{ \sum x + C[i, r - 1] + C[r + 1, n] \}$$

- Since we use argmin, we get an indexing on all the values in the half above the minor diagonal.

Matrix Multiplication

Suppose we have a series of n matrices to multiply, A_1, A_2, \dots, A_n , where the shapes are different. For example, A_1 may have size $r_1 \times c_1$, A_2 has size $r_2 \times c_2$, etc.

- We can multiply two matrices as long as they are next to each other, since matrix multiplication is associative. For example, if $n = 3$, we can either do $(A_1 \times A_2) \times A_3$, or $A_1 \times (A_2 \times A_3)$
- We want to find the optimal number of multiplications to calculate the final result.

We can imagine this problem as two subproblems. Pick a multiplication in the middle of the list, and let the last operation be $(A_1 \cdots A_j) \times (A_{j+1} \cdots A_n)$. We can optimize the number of multiplications of this using this as the recurrence relation.

$$\text{OPT}(i, k) = \min_{i < j < k} r_i \times c_n \times c_j + \text{cost} \left(\prod_{a=i}^j A_a \right) + \text{cost} \left(\prod_{b=j+1}^k A_b \right)$$

We can use the base cases $\text{OPT}(i, i) = 0$ and $\text{OPT}(i, i + 1) = r_i \times c_j \times r_j$