

# Static Optimal Binary Search Tree

Build a binary search tree which gives a minimum search cost, assuming we know the frequencies  $p_i$  with which data  $k_i$  is accessed. The tree **cannot** be modified after it has been constructed.

Want to build a binary search tree with minimum expected search cost:

$$\text{Expected Cost} = \sum_{i=1}^n p_i \text{depth}(k_i)$$

$$\text{depth}(\text{root}) = 1 \Rightarrow \text{our assumption}$$

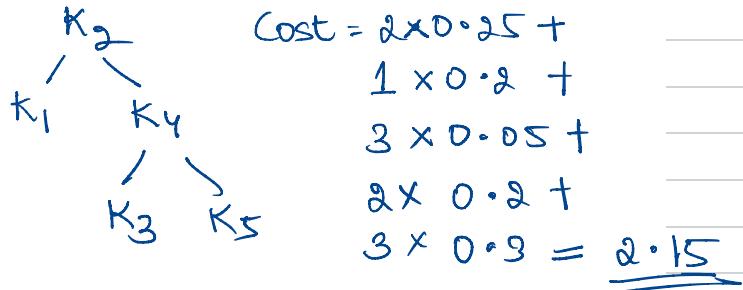
## Example

Consider 5 items

$$k_1 < k_2 < k_3 < k_4 < k_5$$

and their search probabilities

$$p_1 = 0.25, p_2 = 0.2, p_3 = 0.05, p_4 = 0.2, p_5 = 0.3.$$

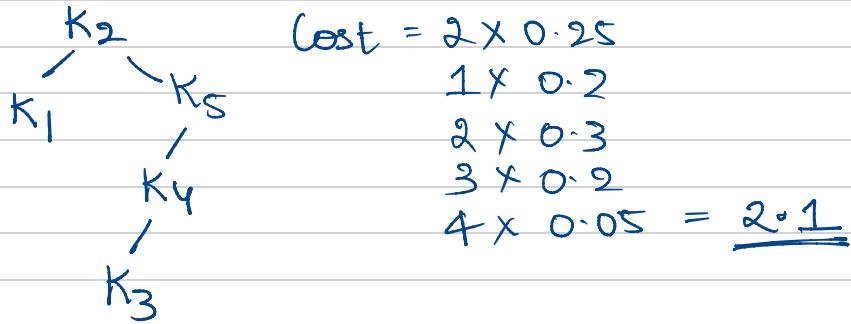


Since our tree structure is not unique, let us consider another possibility.

### Another possibility

$$k_1 < k_2 < k_3 < k_4 < k_5$$

$$p_1 = 0.25, p_2 = 0.2, p_3 = 0.05, p_4 = 0.2, p_5 = 0.3$$



### Optimal Substructure

Since my main structure is a **TREE** let us take our substructure to be a **SUBTREE**.

Let  $\text{OPT}[i, j]$  be the minimum cost of a tree made of  $k_i, k_{i+1}, \dots, k_{j-1}, k_j$ .

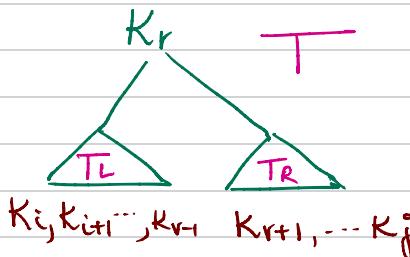
$$k_i \leq k_{i+1} \leq \dots \leq k_j$$

**NOTE : BST cannot have duplicates**

Subproblem is part of our i/p & our i/p is a tree so our subproblem will be a subtree

What is the root of  $\text{OPT}[i, j]$  subtree?

$$K_i \leq K_r \leq K_j$$



$r$  is the root

$$\text{cost}_T = \text{OPT}[i, j] = \underbrace{P_r}_{\text{root}} + \sum_{s=i}^{r-1} P_s \times \text{depth}_T(K_s)$$

$$+ \sum_{s=r+1}^j P_s \times \text{depth}_T(K_s)$$

$$\begin{aligned} \text{depth}_T(K_s) &= \text{depth}_{T_L}(K_s) + 1 \\ &= \text{depth}_{T_R}(K_s) + 1 \end{aligned}$$

$$\text{cost}_T = P_r + \sum P_s (1 + \text{depth}_{T_L}(K_s)) + \sum P_s (1 + \text{depth}_{T_R}(K_s))$$

$$= P_i + P_{i+1} + \dots + P_{r-1} + P_r + P_{r+1} + \dots + P_j$$

$$+ \boxed{\sum P_s \text{depth}_{T_L}(K_s)} + \boxed{\sum P_s \text{depth}_{T_R}(K_s)}$$

$$\text{OPT}[i, r-1]$$

$$\text{OPT}[r+1, j]$$

P.T.O

# Recurrence Relation

$$OPT[i, j] = P_i + \dots + P_j + \min_{i \leq r \leq j} [OPT[i, r-1] + OPT[r+1, j]]$$

BASE CASES :

$$OPT[i, i] = P_i$$

$$OPT[i, i-1] = 0$$

OPTIMAL SOLUTION :

OPT [top right corner]

Filling up the table

array  $p = [p_1, p_2, \dots, p_n]$

set  $OPT[i, i-1] = 0$ , for  $1 \leq i \leq n$

set  $OPT[i, i] = p_i$ , for  $1 \leq i \leq n$

for( $k = 1; k < n; k++$ )

    for( $i = 1; i \leq n-k, i++$ )

$j = i + k;$

$O(n)$

$O(1)$

$O(1)$

$OPT[i, j] = p_i + \dots + p_j + \min_{r} (OPT[i, r-1] + OPT[r+1, j]);$   
 $(i \leq r \leq j)$

    return  $OPT[1, n]$ ;

Runtime Complexity-?  $\Rightarrow O(n^3)$

## Example

$n = 5$

(prob,value) =  $(0.1, 5), (0.3, 6), (0.9, 4), (0.3, 3), (0.1, 8)$

	0	1	2	3	4	5
1	0	0.1				
2		0	0.3			
3			0	0.9		
4				0	0.3	
5					0	0.1

$$\begin{aligned}
 & \text{if } r=1 : \text{OPT}[1,0] \\
 & \quad \text{OPT}[2,2] \\
 & \text{if } r=2 : \text{OPT}[1,1] \\
 & \quad \text{OPT}[3,2] \\
 \text{OPT}[1,0] &= 0 \\
 \text{OPT}[2,2] &= \frac{0.3}{0.3}
 \end{aligned}$$

Exercise :  $\text{OPT}[1,2]$

$$\text{OPT}[1,2] = 0.1 + 0.3 + \min_{1 \leq r \leq 2} [\text{OPT}[1,r-1] + \text{OPT}[r+1,2]]$$

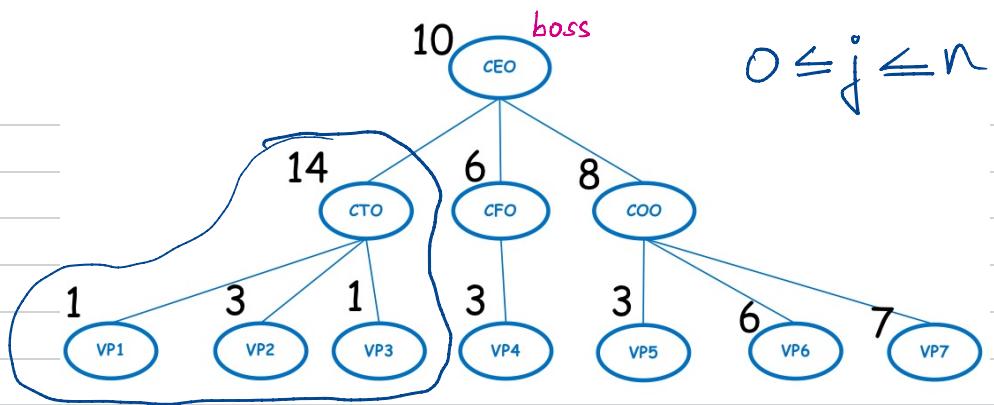
$$= 0.1 + 0.3 + \underline{0.1} = 0.5$$

because  $0.1 < 0.3$   
from the 2 sums  
we have completed.

$$\begin{aligned}
 \text{OPT}[1,1] &= 0.1 \\
 \text{OPT}[3,2] &= 0
 \end{aligned}$$

## Discussion Problem 1

Suppose you are organizing a company party. The corporation has a tree-like ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee  $j$  has a value  $v_j$  (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.



Shall we invite the boss?

$$\textcircled{1} \text{ No, } OPT = 36 = 14 + 6 + (3 + 6 + 7)$$

$$\textcircled{2} \text{ Yes, } OPT = 34 = 10 + 1 + 3 + 1 + 3 + 3 + 6 + 7$$

Let  $OPT[r]$  be the maximum fun value of a subtree rooted at  $r$ .

Choices:

$$\textcircled{1} \text{ } r \text{ is selected : } OPT[r] = V_r + \sum_g OPT[g]$$

$$\textcircled{2} \text{ } r \text{ is not selected : } OPT[r] = \sum_c OPT[c]$$

$g$  = grandchildren,  $c$  = children

Recurrence Relation

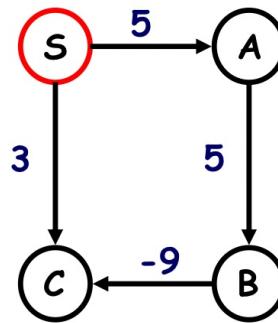
$$OPT[r] = \text{MAX} [V_r + \sum g, \sum c]$$

Base cases:

$$DPT[\text{NULL}] = 0$$

Runtime:  $O(n)$   $\Rightarrow$  This is so because when we start from the bottom of the tree we traverse upwards, we compute and visit each node only once.

## The Shortest Path Problem



Dijkstra's greedy algorithm does not work on graphs with negative weights.

How can we use Dynamic Programming to find the shortest path? We need to somehow define ordered subproblems, otherwise we may get an exponential runtime.

In Dijkstra's we consider a node only once and this cannot be followed if the graph consists a **negative** edge. In this case we will have to consider a node more than once (maximum  $V-1$  times) to compute the shortest path. Hence we make use of DP.

## Intuition

Consider the path (with  $k$  edges)

$$v = w_0, w_1, \dots, w_{k-1}, w_k = u.$$

To have an optimal substructure the following path ( $k-1$  edges)

$$v = w_0, w_1, \dots, w_{k-1}$$

must be a shortest path to  $w_{k-1}$ .

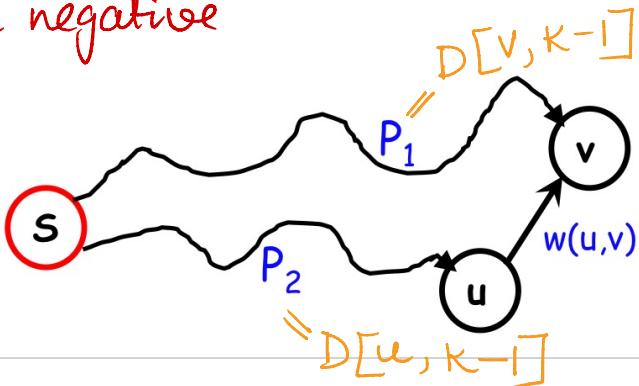
Thus, we will be counting the number of edges in the shortest path. This is how we order subproblems.

$D[v, k]$  denotes the length of the shortest path from  $s$  to  $v$  that uses at most  $k$  edges.

Here our path is represented by the number of edges.

# The Bellman-Ford Algorithm

$w$  could be negative



$$D[v, k] = \begin{cases} P_1 \\ P_2 + w(u, v) \end{cases}$$

$D[v, k]$  denotes the shortest path from  $s$  to  $v$  using at most  $k$  edges.

Case 1 : Path uses at most  $k-1$  edges.

Don't use adjacent vertex       $D[v, k] = D[v, k-1]$   
  ⇒ Don't update the table

Case 2 : Otherwise, use the adjacent vertex  
let  $u$  be an adjacent vertex.

$$D[v, k] = \min_u [D[u, k-1] + w(u, v)]$$

$$1 \leq k \leq V-1$$

## Recurrence Equation:

$$D[v, k] = \min_{\substack{O(1) \\ O(v)}} \left[ D[v, k-1], \min_u \left( D[u, k-1] + w(u, v) \right) \right]$$

## Base Cases

$$D[v, 0] = \infty$$

$$D[s, k] = 0$$

## Implementation

$D[v, k]$  denotes the length of the shortest path from  $s$  to  $v$  that uses at most  $k$  edges.

$$D[v, 0] = \text{INFINITY}; v \neq s$$

$$D[s, k] = 0; \text{ for all } k$$

V for  $k=1$  to  $V-1$ : V

V for each  $v$  in  $V$ :

E for each edge  $(u, v)$  in  $E$ :

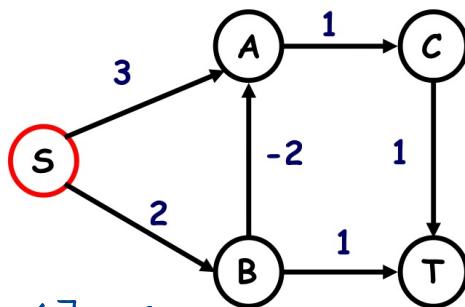
$$D[v, k] = \min(D[v, k-1], w(u, v) + D[u, k-1])$$

Since the total number of child edges cannot possibly exceed  $E$ , we take over worst case to be  $E$ .

Runtime - ?

$$\boxed{\mathcal{O}(V \cdot E)}$$

# Example



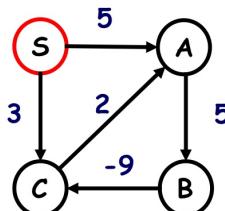
$$\kappa = 1 : D[A, 1] = 3 \\ D[B, 1] = 2$$

$$\kappa = 2 : D[A, 2] = \min(3, 2-2) = 0 \\ D[C, 2] = 4 \\ D[T, 2] = 3$$

$$\kappa = 3 : D[C, 3] = \min(4, 2-2+1) = 1 \\ D[T, 3] = \min(3, 3+1+1) = 3$$

$$\kappa = 4 : D[T, 4] = \min(3, 2-2+1+1) = 2$$

How would you apply the Bellman-Ford algorithm to find out if a graph has a **negative cycle**?

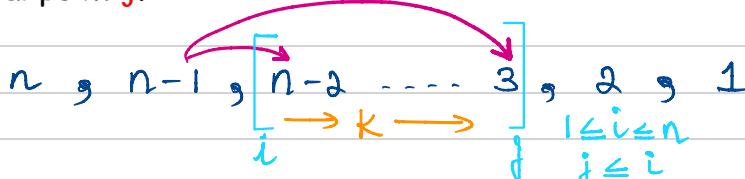


$$C-A-B-C \\ 2+5+(-9) = 2$$

→ We add an extra loop to our traversal  
 → After the last loop if the table entry changes, there is a **negative cycle**  
 → return no solution exists.

## Discussion Problem 2

There are  $n$  trading posts along a river numbered  $n, n-1 \dots, 1$ . At any of the posts you can rent a canoe to be returned at any other post downstream. (It is impossible to paddle against the river). For each possible departure point  $i$  and each possible arrival point  $j < i$ , the cost of a rental is  $C[i, j]$ . However, it can happen that the cost of renting from  $i$  to  $j$  is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post  $k$  between  $i$  and  $j$  and continue your journey in a second (and, maybe, third, fourth ...) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point  $i$  to each possible arrival point  $j$ .



Let  $\text{OPT}[i, j]$  be the minimum rental cost going from  $i$  to  $j$ .

$$\text{OPT}[i, j] = \min_{j \leq k \leq 1} [C[i, k] + \text{OPT}[k, j]]$$

$$\text{OPT}[i, i] = 0$$

$$\text{Runtime : } O(n^2 \times n) = O(n^3)$$

P-T. O

## Variant B

$$i = n, j = 1$$

Use an 1 D array

Let  $\text{OPT}[i]$  be the minimum rental cost from  $j$  to  $i$ .

$$\text{OPT}[i] = \min_{1 \leq k \leq i} [c[i, k] + \text{OPT}[k]]$$

$$\text{OPT}[1] = 0 \Rightarrow \text{Basecase}$$

$$\text{Runtime} = O(n \times n) = \underline{\underline{O(n^2)}}$$

## CHAIN MATRIX MULTIPLICATION - Abdul Bari

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$\rightarrow 2 \times 3 \quad 3 \times 2$$

must definitely satisfy to do CMM

$$\text{Dimension} = 2 \times 2$$

$$\text{Total no. of multiplications} = 2 \times 3 \times 2 = \underline{\underline{12}}$$

## Example : (With example formula)

Let solution =  $C[1,3]$   $A_1$   $A_2$   $A_3$

$2 \times 3$	$3 \times 4$	$4 \times 2$
$d_0$	$d_1$	$d_1$
$d_1$	$d_2$	$d_2$
$d_3$	$d_3$	$d_3$

$$\begin{array}{c} (A_1 \times A_2) \times A_3 \\ \hline \begin{matrix} 2 & 3 & 3 & 4 \\ \hline C[1,2] = 24 \end{matrix} \quad \begin{matrix} 4 & 2 \\ \hline 0 \quad C[3,3] \end{matrix} \end{array}$$

$$\begin{array}{c} A_1 \times (A_2 \times A_3) \\ \hline \begin{matrix} 2 & 3 & 3 & 4 & 4 & 2 \\ \hline C[1][0] \end{matrix} \quad \begin{matrix} C[2,3] = 24 \\ 3 \times 4 \times 2 = 24 \end{matrix} \end{array}$$

$$\Rightarrow 2 \times 4 \quad 4 \times 2$$

$$2 \times 4 \times 2 = 16$$

$$\text{do } d_2 \quad d_3$$

$$\therefore \text{Total} = 24 + 0 + 16 = 40$$

$$C[i, k] + C[k, j] + d_0 \times d_1 \times d_2$$

$$d_{i+1} \quad d_{k+1} \quad d_j$$

Observe the  
reduced equation  
carefully

$$\Rightarrow 2 \times 3 \quad 3 \times 2$$

$$2 \times 3 \times 2 = 12$$

$$\text{do } d_1 \quad d_3$$

$$\therefore \text{Total} = 0 + 24 + 12 = 36$$

$\Leftrightarrow$  CHEAPER

$$C[i, k] + C[k, j] + d_0 \times d_1 \times d_2$$

$$d_{i+1} \quad d_{k+1} \quad d_j$$

**FORMULA:**  $C[i, j] = \min_{i \leq k \leq j} [C[i, k] + C[k+1, j] + d_{i+1} \times d_k \times d_j]$

