

# Design and Analysis of Algorithms

## L29: Optimal Binary Search Dynamic Programming

Dr. Ram P Rustagi  
Sem IV (2020-Even)  
Dept of CSE, KSIT  
[rprustagi@ksit.edu.in](mailto:rprustagi@ksit.edu.in)

# Resources

- Text book 1: Levitin
  - Sec 8 . 2 , 8 . 3 , 8 . 4
- Text book 2: Horowitz
  - Sec 5 . 1 , 5 . 2 , 5 . 4 , 5 . 8 , 5 . 9
- RI: Introduction to Algorithms
  - Cormen et al.

# Binary Search

- Binary search tree
  - Key value of left child is smaller than parent
  - Key value of right child is greater than the parent
- Balanced binary search tree
  - Height :  $O(\log n)$
  - Example: Red Black tree, AVL Tree
  - For random input, average of binary search tree
    - $O(\log n)$
- Worst case height can be  $O(n)$ 
  - for a completely skewed binary tree

# Optimal Binary Search Tree

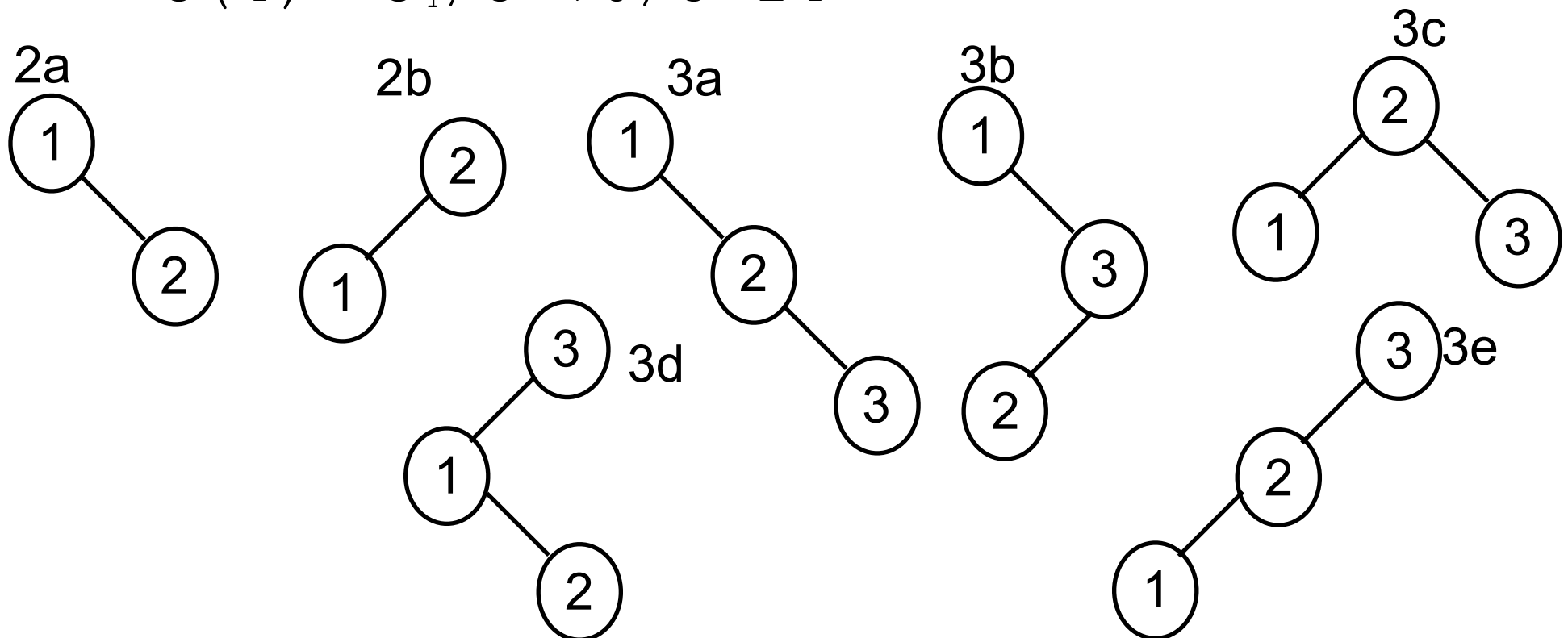
- Use case 1:
  - You need to translate a english document containing ( $n$  words) to Kannada.
  - You have a dictionary providing kannada translation for each english word.
  - Translation process:
    - Consider each word of english document, search in the english-kannada dictionary and use the same
    - Using a generic balanced binary search tree, average tranlation time would  $O(n \cdot \log_2 n)$  time.
  - If we know the frequency of occurrence of each word in english document, can we do better
    - How to optimally organize binary search tree?

# Optimal Binary Search Tree

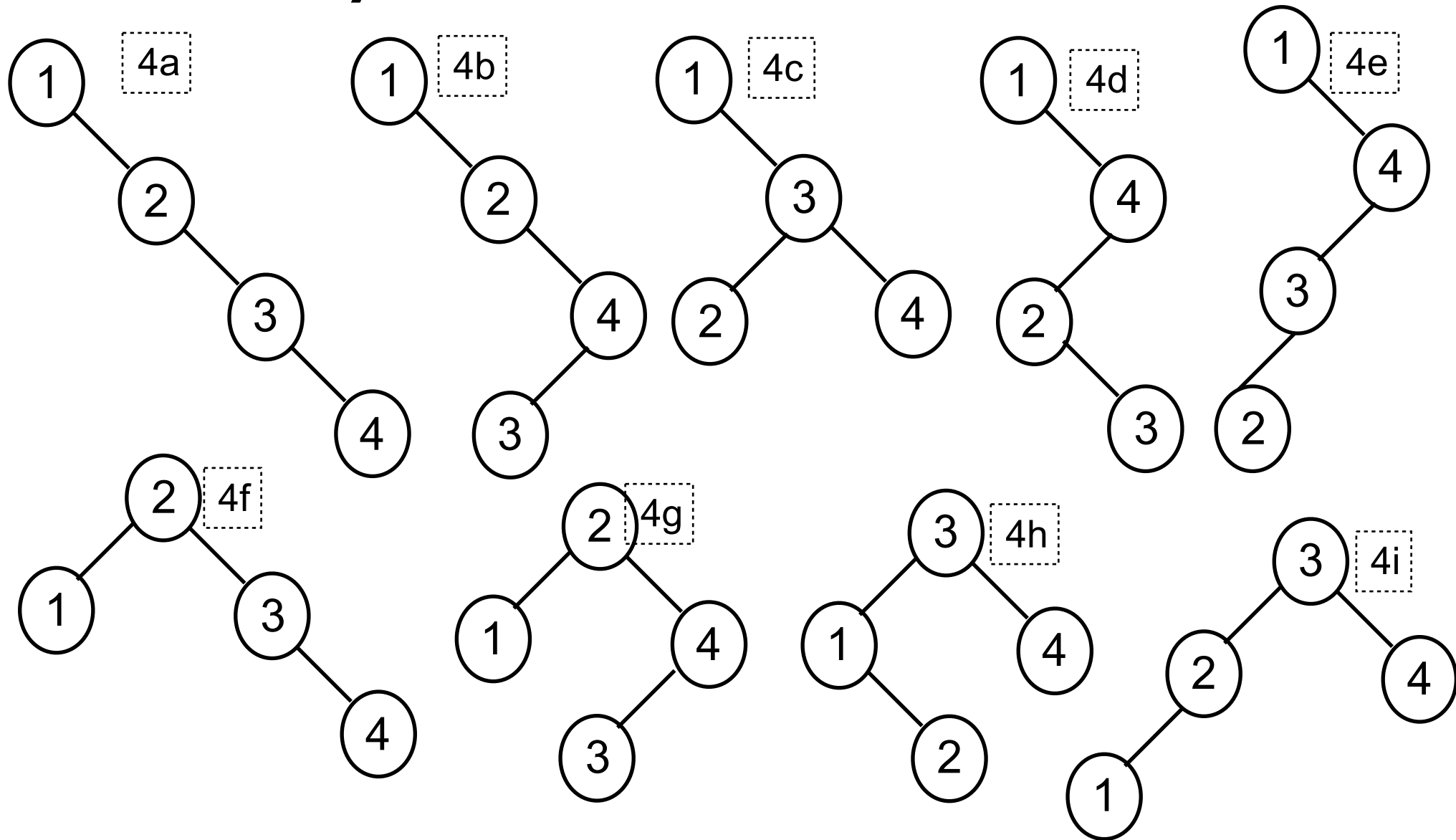
- Use case 2:
  - As an e-tailor, you are selling phones.
    - Total  $n$  types of brands/models etc.
  - Different customer will choose different brand/models
  - You organize the product details (price etc) in a binary search tree.
  - In general, each search  $O(n \cdot \log_2 n)$  takes time.
  - If we know the purchase frequency of each brand/model, can we improve upon the search time
    - How to optimally organize binary search tree?
- Objective: organize binary search tree in such a way to reduce average look up time.

# Binary Search Tree

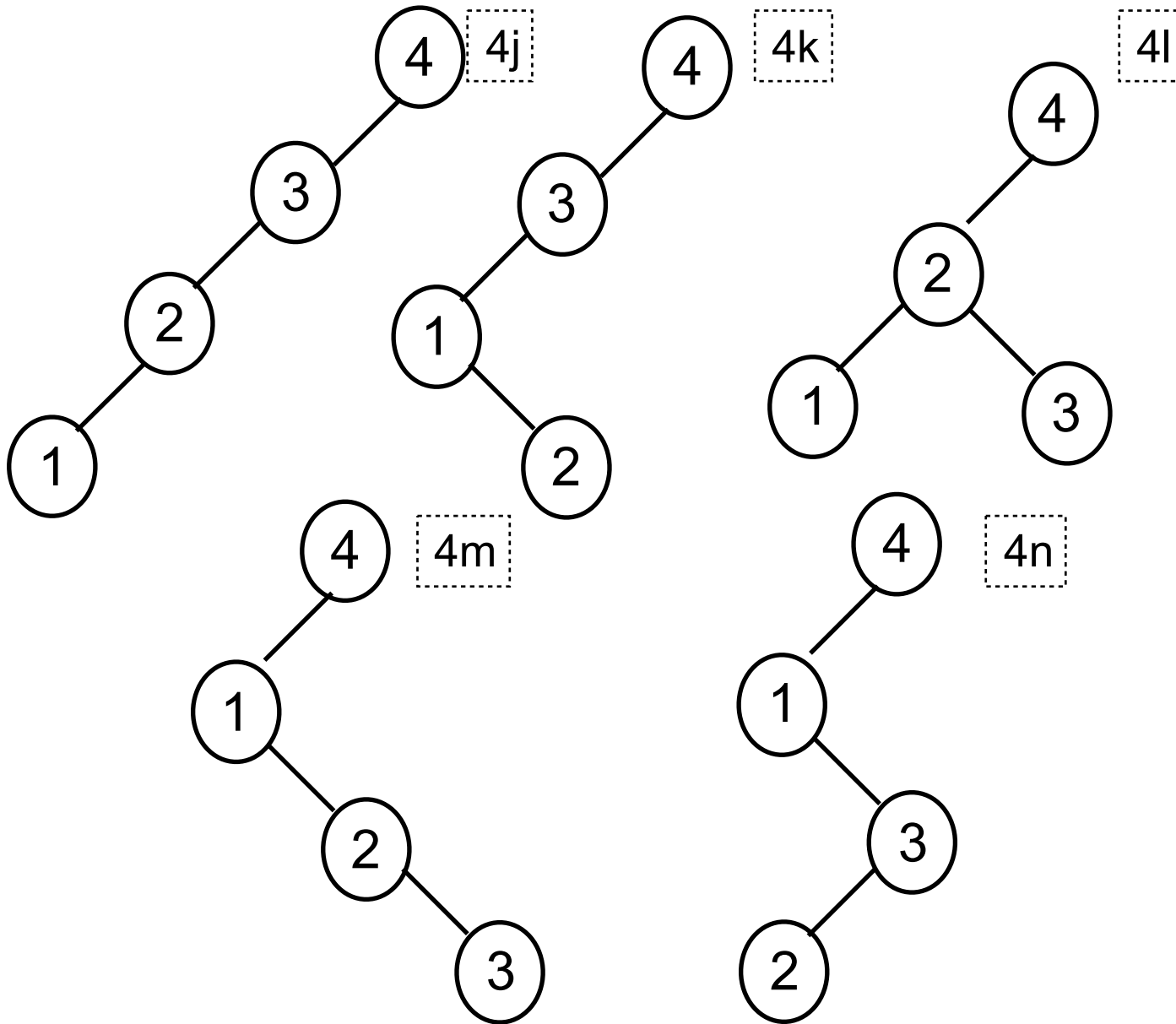
- Given  $n$  nodes, how many possible binary trees
  - Catalan number  $C(n) : \frac{2^n C_n}{(n+1)}$
  - $C(2) = \frac{4C_2}{3} = 6/3 = 2$
  - $C(3) = \frac{6C_3}{4} = 20/4 = 5$
  - $C(4) = \frac{8C_4}{5} = 70/5 = 14$



# Binary Search Tree : 4 nodes...



# Binary Search Tree : 4 nodes



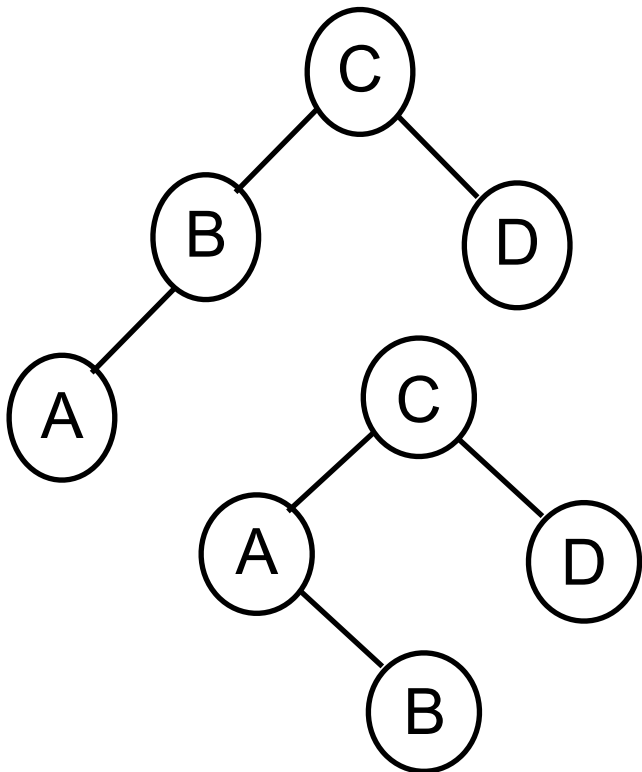


# Optimal Binary Search Tree

- Problem:
  - Given  $n$  keys  $a_1 \leq a_2 \leq \dots \leq a_n$ , with
  - respective probabilities of occurrences  $p_1, p_2, \dots, p_n$
  - Find a Binary Search Tree (BST) with
  - minimum average number of comparisons in successful search
- Brute force methods
  - Total number of BST:  $C(n) = \frac{2^n C_n}{(n+1)}$   
 $= \Omega(4^n / n^{1.5})$
  - Requires exponential number of searches
    - An impractical approach

# Example: BSTs

- Consider 4 keys A, B, C, D
  - with their probabilities as 0.1, 0.2, 0.4 and 0.3
- Compute the average number of comparisons for BSTs given below



- Average number of comparisons  
 $= 0.1 * 3 + 0.2 * 2 + 0.4 * 1 + 0.3 * 2$   
 $= 1.7$

- Average number of comparisons  
 $= 0.1 * 2 + 0.2 * 3 + 0.4 * 1 + 0.3 * 2$   
 $= 1.8$

# Finding Optimal BST

- for 4 nodes, possible BSTs : 14
  - Finding the optimal BST requires evaluation of 14 trees
  - When probability values changes, need recomputation to find a new BST
  - With increasing  $n$ , it becomes challenging
    - Requires exponential computing.
- Use of dynamic programming helps solve this issue in polynomial time.

# Optimal BST: DP Approach

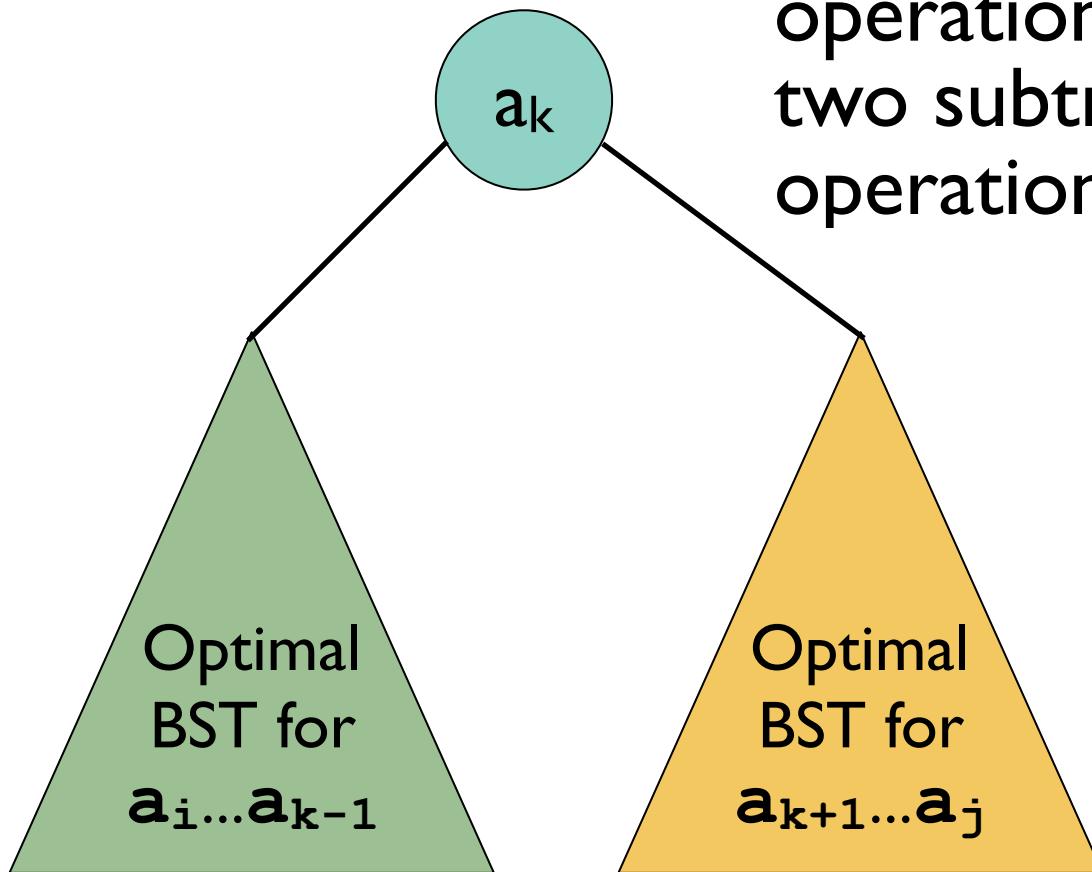
- Given  $n$  keys:  $a_1 \leq a_2 \leq \dots \leq a_n$ , with
  - respective prob. of occurrences  $p_1, p_2, \dots, p_n$
- Let  $C(i, j)$  denote the smallest number of comparisons in a successful search for BST  $T_{i,j}$ .
  - Tree  $T_{i,j}$  consists of keys  $a_i \leq a_{i+1} \leq \dots \leq a_j$ , where
    - $i, j$  are some integer indices  $1 \leq i \leq j \leq n$ .
- Thus, desired answer for our  $n$  keys would be  $C(1, n)$
- Dynamic Programming approach:
  - Find smaller instances corresponding to  $C(i, j)$
  - with the aim to solve  $C(i, n)$

# Optimal BST: DP Approach

- Solving  $C(i, j)$  for  $T_i^j$ ,  $a_i \leq a_{i+1} \leq \dots \leq a_j$ ,  $1 \leq i \leq j \leq n$
- Derive a recurrence for  $C(i, j)$ .
  - Need to find the root  $a_k$  ( $i \leq k \leq j$ ) for  $T_i^j$ ,
  - Consider all possible ways of choosing root  $a_k$ 
    - $a_k$  could be any node between  $a_i$  and  $a_j$
  - To find an optimal BST with root  $a_k$ ,
    - use principle of optimality
    - Left subtree will have keys  $a_i \leq \dots \leq a_{k-1}$  arranged optimally
    - Right subtree will have keys  $a_{k+1} \leq \dots \leq a_j$  arranged optimally.

# DP for Optimal BST

- Trees  $T_i^{k-1}$ , and  $T_{k+1}^j$ , are 1 level below the root node  $a_k$ .
- Comparison with  $a_k$  require 1 operation, comparisons of keys in two subtrees need to count this operation of comparison at root  $a_k$



# DP for Optimal BST

- Recurrence for BST using DP

$$\begin{aligned}
 C(i, j) &= \min_{i \leq k \leq j} \left\{ p_k \cdot 1 + \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \in T_i^{k-1} + 1) \right. \\
 &\quad \left. + \sum_{s=k+1}^j p_s \cdot (\text{level of } a_s \in T_{k+1}^j + 1) \right\} \\
 &= \min_{i \leq k \leq j} \left\{ p_k + \sum_{s=i}^{k-1} p_s \cdot \text{level of } a_s \in T_i^{k-1} + \sum_{s=i}^{k-1} p_s \right. \\
 &\quad \left. + \sum_{s=k+1}^j p_s \cdot \text{level of } a_s \in T_{k+1}^j + \sum_{s=k+1}^j p_s \right\} \\
 &= \min_{i \leq k \leq j} \left\{ \sum_{s=i}^j p_s + \sum_{s=i}^{k-1} p_s \cdot \text{level of } a_s \in T_i^{k-1} + \sum_{s=k+1}^j p_s \cdot \text{level of } a_s \in T_{k+1}^j \right\} \\
 &= \sum_{s=i}^j p_s + \min_{i \leq k \leq j} \left\{ C(i, k-1) + C(k+1, j) \right\} \quad (1)
 \end{aligned}$$

# DP for Optimal BST

$$C(i, j) = \sum_{s=i}^j p_s + \min_{i \leq k \leq j} \left\{ C(i, k-1) + C(k+1, j) \right\} \quad (1)$$

- **Recurrence for BST using DP:**
  - $C(i, i-1) = 0$  since no left subtree for root  $a_i$ ,
  - $C(j, j+1) = 0$  since no right subtree for root  $a_j$
  - $C(i, i) = p_i * 1 = p_i$  since tree has only one key  $a_i$
- **Example: computation for  $C(2, 4)$  using eqn (1)**

$$\begin{aligned} C(2, 4) &= \sum_{2 \leq s \leq 4} p_s + \min \{ C(2, 1) + C(3, 4), \\ &\quad C(2, 2) + C(4, 4), \\ &\quad C(2, 3) + C(5, 4) \} \\ &= \sum_{2 \leq s \leq 4} p_s + \min \{ 0 + C(3, 4), \quad p_2 + p_4, C(2, 3) + 0 \} \end{aligned}$$



# DP for Optimal BST

$$\begin{aligned}
 C(2, 4) &= \sum_{2 \leq s \leq 4} p_s + \min \{ C(2, 1) + C(3, 4), \\
 &\quad C(2, 2) + C(4, 4), \\
 &\quad C(2, 3) + C(5, 4) \} \\
 &= \sum_{2 \leq s \leq 4} p_s + \min \{ 0 + C(3, 4), p_2 + p_4, C(2, 3) + 0 \}
 \end{aligned}$$

	0	1	2	3	4	5
1						
2		0	$p_2$	$C(2,3)$	<b><math>C(2,4)</math></b>	
3			0	$p_3$	$C(3,4)$	
4				0	$p_4$	
5					0	

# DP for Optimal BST

$$C(i, j) = \sum_{s=i}^j p_s + \min_{i \leq k \leq j} \left\{ C(i, k-1) + C(k+1, j) \right\} \quad (1)$$

• Contribution for  $C(i, j)$  is from

$C(i, i-1) + C(i+1, j)$

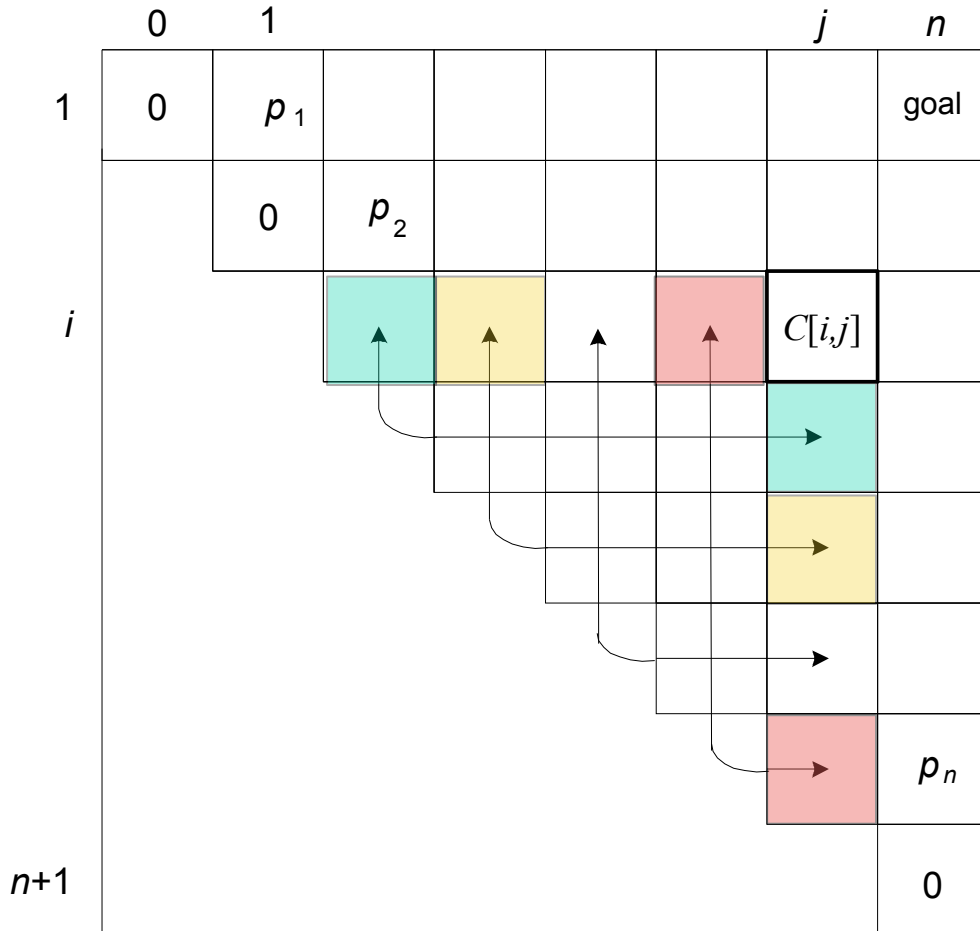
$C(i, i) + C(i+2, j)$

$C(i, i+1) + C(i+3, j)$

:

:

$C(i, j-1) + C(j+1, j)$



# Exercise: Optimal BST

- Consider 4 keys : A, B, C, D with their prob. as  
 -  $p_A=0.1$ ,  $p_B=0.2$ ,  $p_C=0.4$ ,  $p_D=0.3$ ,

$$C(1,4) = \sum_{s=1}^4 p_s + \min_{1 \leq k \leq 4} \left\{ C(1, k-1) + C(k+1, 4) \right\}$$

	0	1	2	3	4
1	0	0.1			
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

DAA/Dynamic Programming

	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

RPR/

19

# Exercise: Optimal BST

$$C(1, 2) = \sum_{1 \leq s \leq 2} p_s + \min\{C(1, 0) + C(2, 2), C(1, 1) + C(3, 2)\} \\ = 0.3 + \min\{0 + 0.2, 0.1 + 0\} = 0.4, \text{ optimal } k=2$$

$$C(2, 3) = \sum_{2 \leq s \leq 3} p_s + \min\{C(2, 1) + C(3, 3), C(2, 2) + C(4, 3)\} \\ = 0.6 + \min\{0 + 0.4, 0.2 + 0\} = 0.8, \text{ optimal } k=3$$

$$C(3, 4) = \sum_{3 \leq s \leq 4} p_s + \min\{C(3, 2) + C(4, 4), C(3, 3) + C(5, 4)\} \\ = 0.7 + \min\{0 + 0.3, 0.4 + 0\} = 1.0, \text{ optimal } k=3$$

	0	1	2	3	4
1	0	0.1	0.4		
2		0	0.2	0.8	
3			0	0.4	1.0
4				0	0.3
5					0

DAA/Dynamic Programming

	0	1	2	3	4
1		1	2		
2			2	3	
3				3	3
4					4
5					

RPR/

20

# Exercise: Optimal BST

$$C(1, 3) = \sum_{1 \leq s \leq 3} p_s + \min \{ C(1, 0) + C(2, 3), \\ C(1, 1) + C(3, 3), C(1, 2) + C(4, 3) \} \\ = 0.7 + \min \{ 0 + 0.8, 0.1 + 0.4, 0.4 + 0 \} = 1.1, \text{ opt } k=3$$

$$C(2, 4) = \sum_{2 \leq s \leq 4} p_s + \min \{ C(2, 1) + C(3, 4), \\ C(2, 2) + C(4, 4), C(2, 3) + C(5, 4) \} \\ = 0.9 + \min \{ 0 + 1.0, 0.2 + 0.3, 0.8 + 0 \} = 1.4, \text{ opt } k=3$$

	0	1	2	3	4
1	0	0.1	0.4	1.1	
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

DAA/Dynamic Programming

	0	1	2	3	4
1		1	2	3	
2			2	3	3
3				3	3
4					4
5					

RPR/

21

# Exercise: Optimal BST

$$\begin{aligned}
 C(1, 4) &= \sum_{1 \leq s \leq 4} p_s + \min \{ C(1, 0) + C(2, 4), \\
 &\quad C(1, 1) + C(3, 4), \\
 &\quad C(1, 2) + C(4, 4), \\
 &\quad C(1, 3) + C(5, 4) \} \\
 &= 1.0 + \min \{ 0 + 1.4, 0.1 + 1.0, 0.4 + 0.3, 1.1 + 0 \} \\
 &= 1.7, \text{ optimal } k=3
 \end{aligned}$$

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

DAA/Dynamic Programming

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					

RPR/

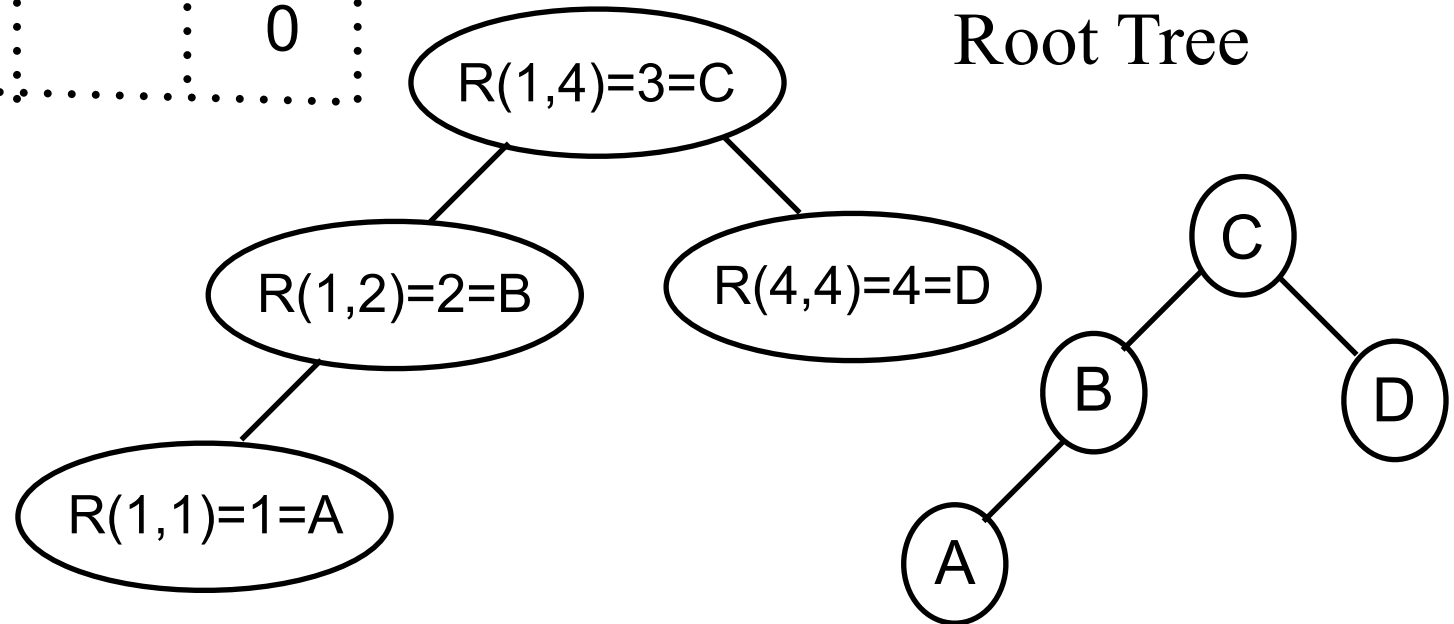
22

# Ex: Optimal BST Construction

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4

Root Tree



# Algorithm

**ALGORITHM** *OptimalBST*( $P[1..n]$ )

//Finds an optimal binary search tree by dynamic programming

//Input: An array  $P[1..n]$  of search probabilities for a sorted list of  $n$  keys

//Output: Average number of comparisons in successful searches in the

// optimal BST and table  $R$  of subtrees' roots in the optimal BST

**for**  $i \leftarrow 1$  **to**  $n$  **do**

$C[i, i - 1] \leftarrow 0$

$C[i, i] \leftarrow P[i]$

$R[i, i] \leftarrow i$

$C[n + 1, n] \leftarrow 0$

**for**  $d \leftarrow 1$  **to**  $n - 1$  **do** //diagonal count

**for**  $i \leftarrow 1$  **to**  $n - d$  **do**

$j \leftarrow i + d$

$minval \leftarrow \infty$

**for**  $k \leftarrow i$  **to**  $j$  **do**

**if**  $C[i, k - 1] + C[k + 1, j] < minval$

$minval \leftarrow C[i, k - 1] + C[k + 1, j]; kmin \leftarrow k$

$R[i, j] \leftarrow kmin$

$sum \leftarrow P[i];$  **for**  $s \leftarrow i + 1$  **to**  $j$  **do**  $sum \leftarrow sum + P[s]$

$C[i, j] \leftarrow minval + sum$

**return**  $C[1, n], R$



# Time Efficiency: Optimal BST

- From general analysis of algo,
  - 3 nested loops, each running  $n$  times
- Thus time efficiency:  $O(n^3)$ 
  - Space Efficiency:  $O(n^2)$
- Time Efficiency: Accounting time smartly.
  - Entries in `root` ( $2^{\text{nd}}$ ) table are always non-decreasing
    - Along each row and column
    - Value of `root` table entry  $R[i, j]$  is limited to the range  $R[i, j-1], \dots, R[i+1, j]$
    - This reduces the time complexity to  $O(n^2)$

# Summary

- Binary search tree
- Optimal binary search tree
- Dynamic programming for BST
- Algo: DP for BST
- Evaluation of  $C(i, j)$  and Tree construction