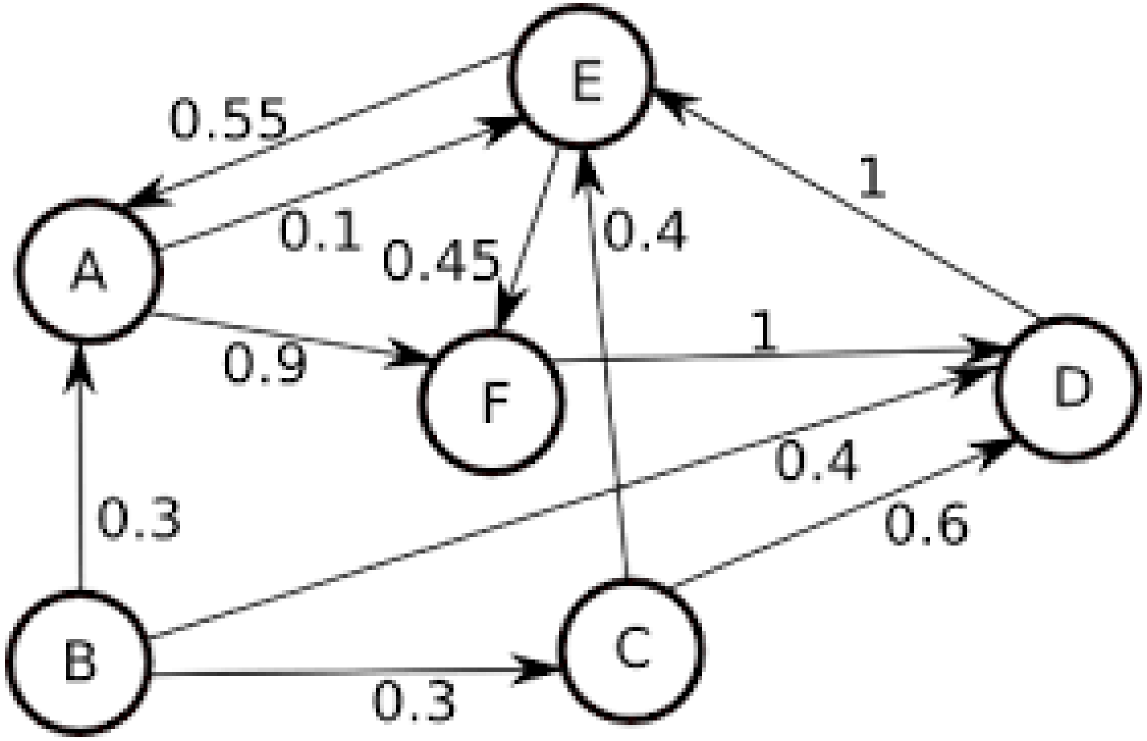
prob. 6

Floyd-Warshall
All Pairs Shortest-Path Algorithm

- Given a graph, the Floyd Warshall algorithm finds the shortest path between all pairs of nodes.



E
A
F
D
B
C
0.55
0.1
0.45
0.4
1
0.9
1
0.3
0.4
0.6
0.3

- DP solution uses a 2D array A to store path lengths between nodes.
- The starting point is the adjacency matrix A of the graph.
- A[i ,j]
- = 0 , if i = j
- = $w_{i\,j}$, if i and j are directly connected
- = ∞ , otherwise
- The DP algorithm is used to update the path lengths between all node pairs i and j.

(

DP approach

- The nodes between i and j are called intermediate nodes 1,2,.., k
- Using each intermediate node, the adjacency matrix A, is updated .
- $A^0 \rightarrow A^1 \rightarrow A^2 \rightarrow \ldots\ldots \rightarrow A^k$

- Let us say, using node 4, the shortest path between nodes i and j is $A^4[i,j]$.
- Now node 5 is used.
- Let $A^4[i, 5]$ be the shortest path from node i to node 5 , and let $A^4[5, j]$ be the shortest path from node 5 to node j. Then the distance between nodes i and j is updated using

- $A^5[i,j] = \min\{ A^4[i,j],\ A^4[i,5] + A^4[5,j] \}$

(

DP updation for each intermediate node

- In general, the shortest path between i and j is updated from $A^{k-1}[i, j]$,
- <u>if using node k, the shortest path from i to k, plus shortest path from k to j is found to be shorter than that</u>

- $A^{k}[i, j] = \min \{ A^{k-1}[i, j], \quad A^{k-1}[i, k] + A^{k-1}[k, j] \}$

- then shortest path either remains the same or gets updated.
- All the intermediate nodes are tested one by one , and path length gets updated

Flyod-Warshall Algo

- for k = 1 to n

```
for i = 1 to n
    for j = 1 to n
        A[i,j] = min { A[i,j] , D[i,k]+D[k,j] }
    endfor
endfor
```

endfor

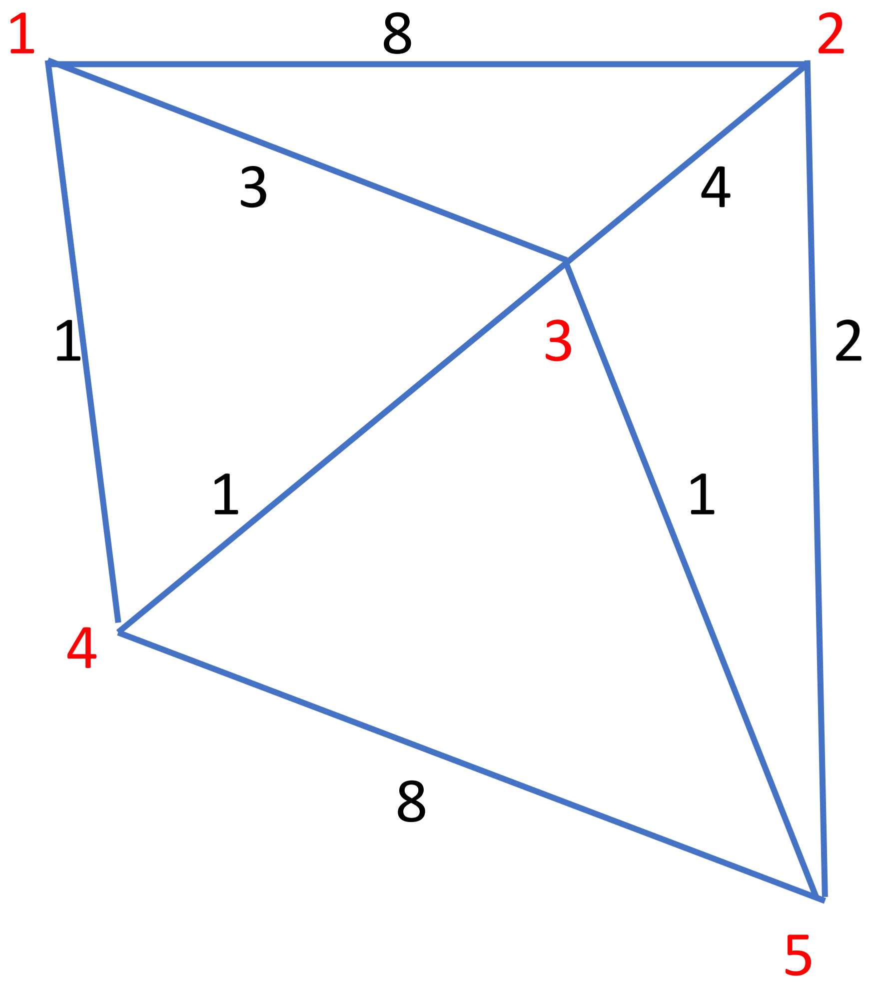
Complexity of Floyd-Warshall Algo

- Since there are 3 nested loops of size n, the complexity of the algorithm is $O(n^3)$.

steps

- We start with adjacency matrix of the graph A^0 ,
- then use node 1, and update it to A^1
- then use node 2, and update it to A^2
- . . . .

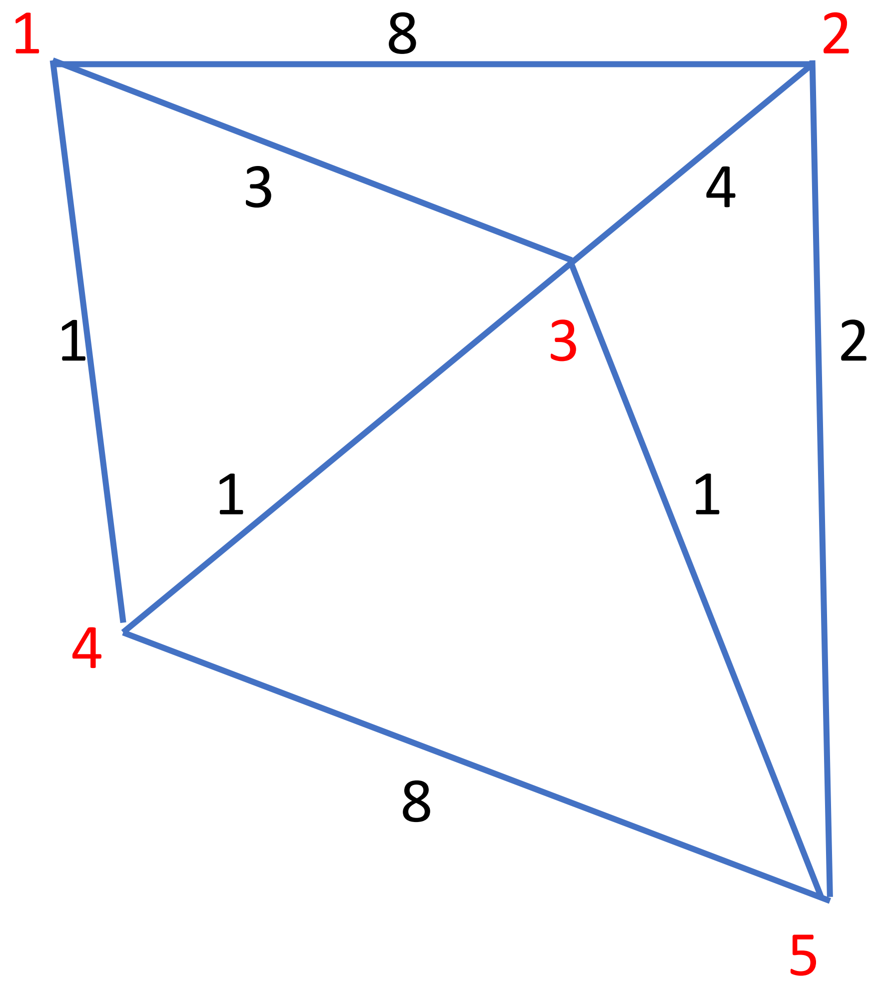
- Finally, use node n, and update it to A^n



- Adjacency matrix
- A^0

	1	2	3	4	5
1	0	8	3	1	∞
2	8	0	4	∞	2
3	3	4	0	1	1
4	1	∞	1	0	8
5	∞	2	1	8	0

- USING 1



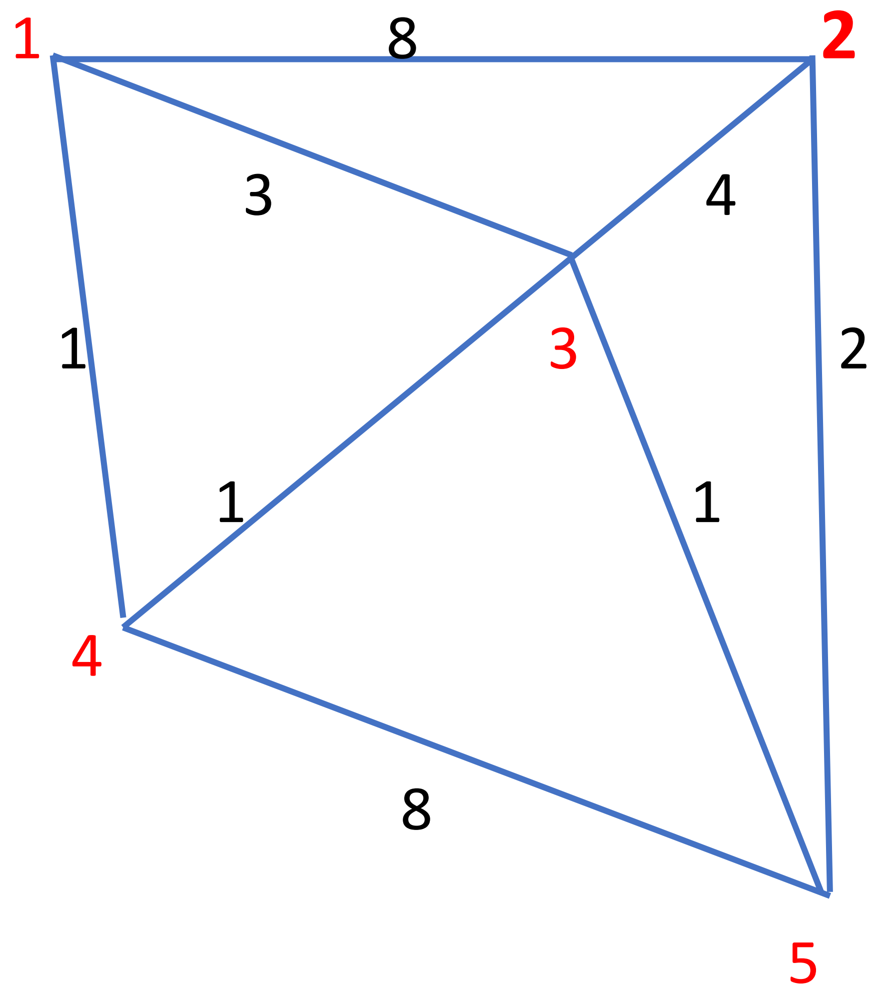
- A^0

	1	2	3	4	5
1	0	8	3	1	∞
2	8	0	4	∞	2
3	3	4	0	1	1
4	1	∞	1	0	8
5	∞	2	1	8	0

- Using node 1, A^1

	1	2	3	4	5
1	0	8	3	1	∞
2	8	0	4	9	2
3	3	4	0	1	1
4	1	9	1	0	8
5	∞	2	1	8	0

- Using node 2



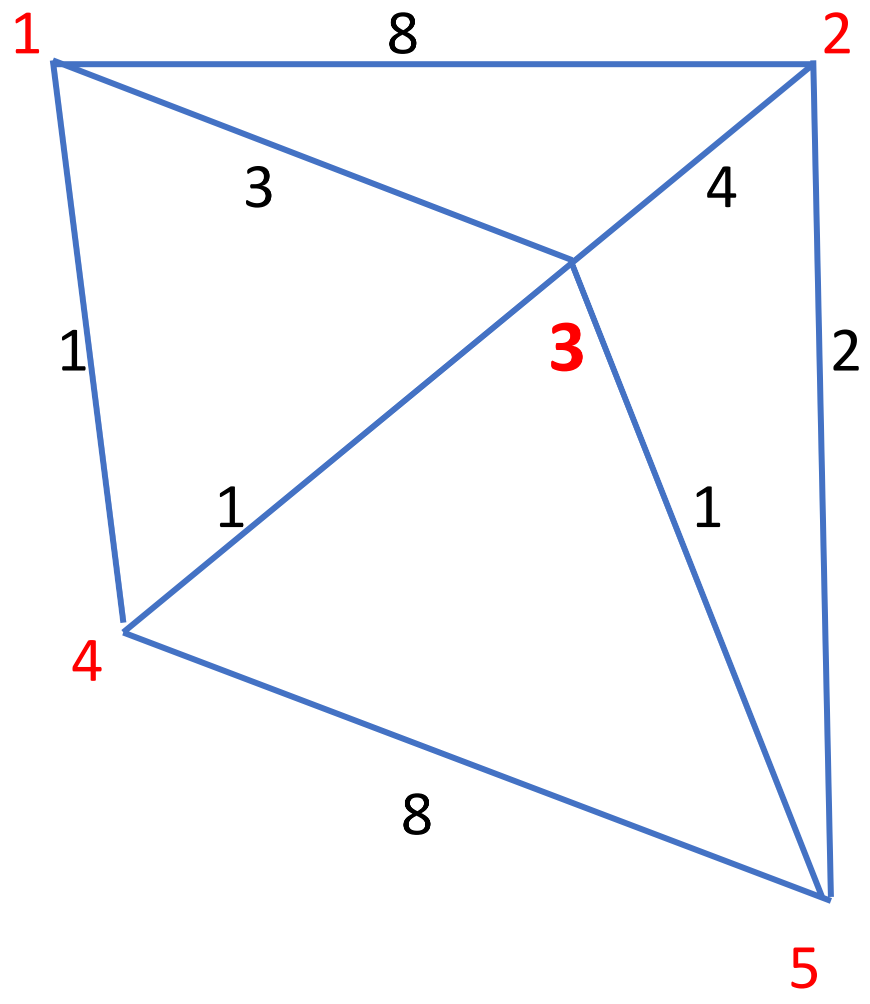
- A^1

	1	2	3	4	5
1	0	8	3	1	∞
2	8	0	4	9	2
3	3	4	0	1	1
4	1	9	1	0	8
5	∞	2	1	8	0

- Using node 2, A^2

	1	2	3	4	5
1	0	8	3	1	10
2	8	0	4	9	2
3	3	4	0	1	1
4	1	9	1	0	8
5	10	2	1	8	0

- Using node 3



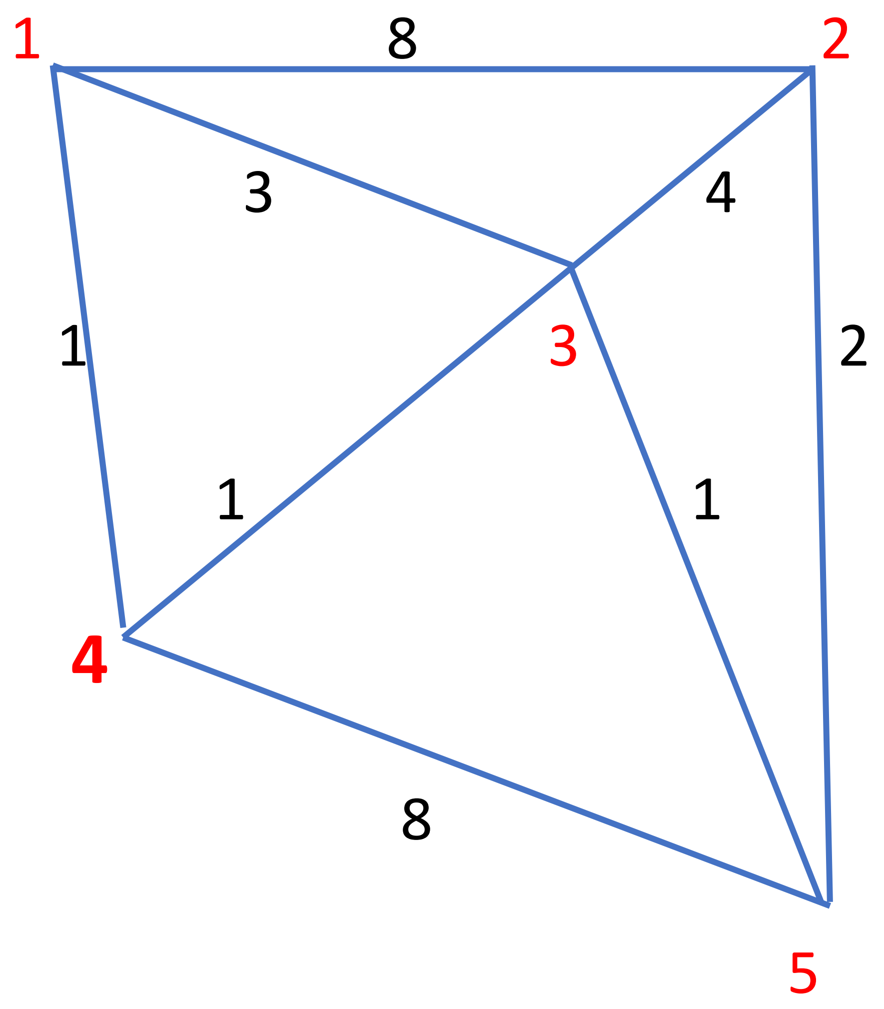
- A^2

	1	2	3	4	5
1	0	8	3	1	10
2	8	0	4	9	2
3	3	4	0	1	1
4	1	9	1	0	8
5	10	2	1	8	0

- Using node 3, A^3

	1	2	3	4	5
1	0	7	3	1	4
2	7	0	4	5	2
3	3	4	0	1	1
4	1	5	1	0	2
5	4	2	1	2	0

- node 4, node 4, node 4



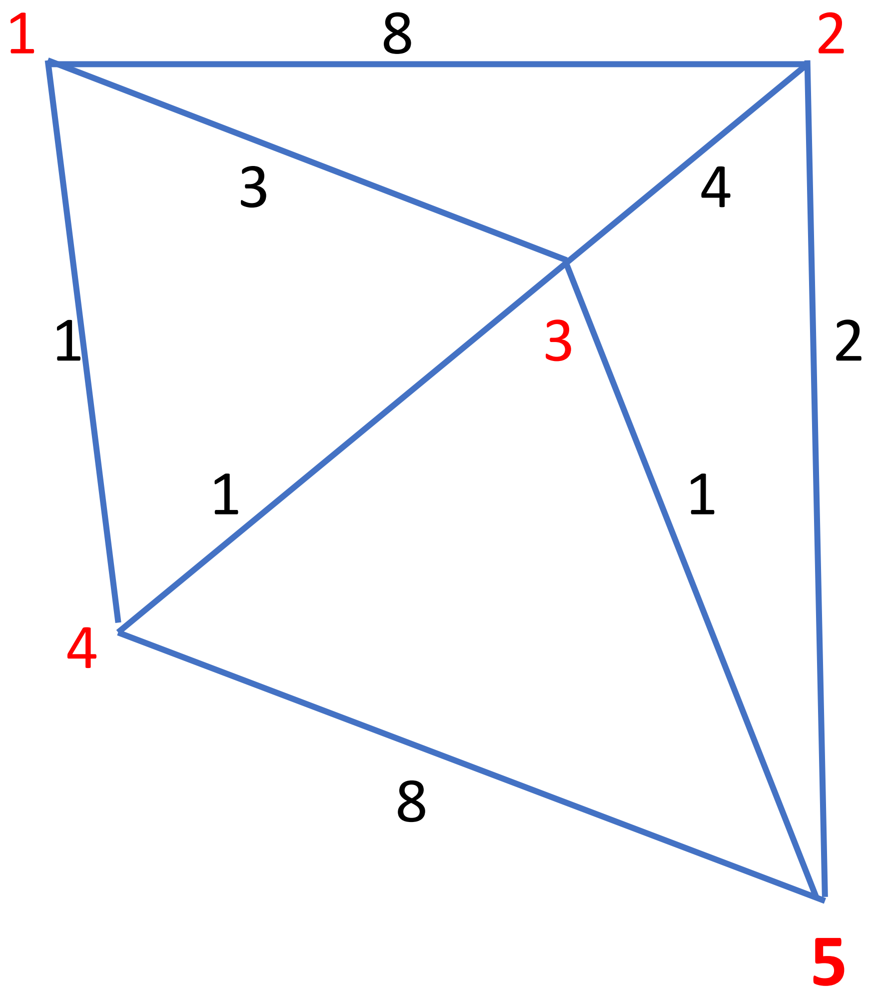
- old A^3

•	1	2	3	4	5
1	0	7	3	1	4
2	7	0	4	5	2
3	3	4	0	1	1
4	1	5	1	0	2
5	4	2	1	2	0

- Using node 4, A^4

•	1	2	3	4	5
1	0	6	2	1	3
2	6	0	4	5	2
3	2	4	0	1	1
4	1	5	1	0	2
5	3	2	1	2	0

- Finally using node 5



- old A^4

	1	2	3	4	5
1	0	6	2	1	3
2	6	0	4	5	2
3	2	4	0	1	1
4	1	5	1	0	2
5	3	2	1	2	0

- Using node 5, A^5

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

prob 7.

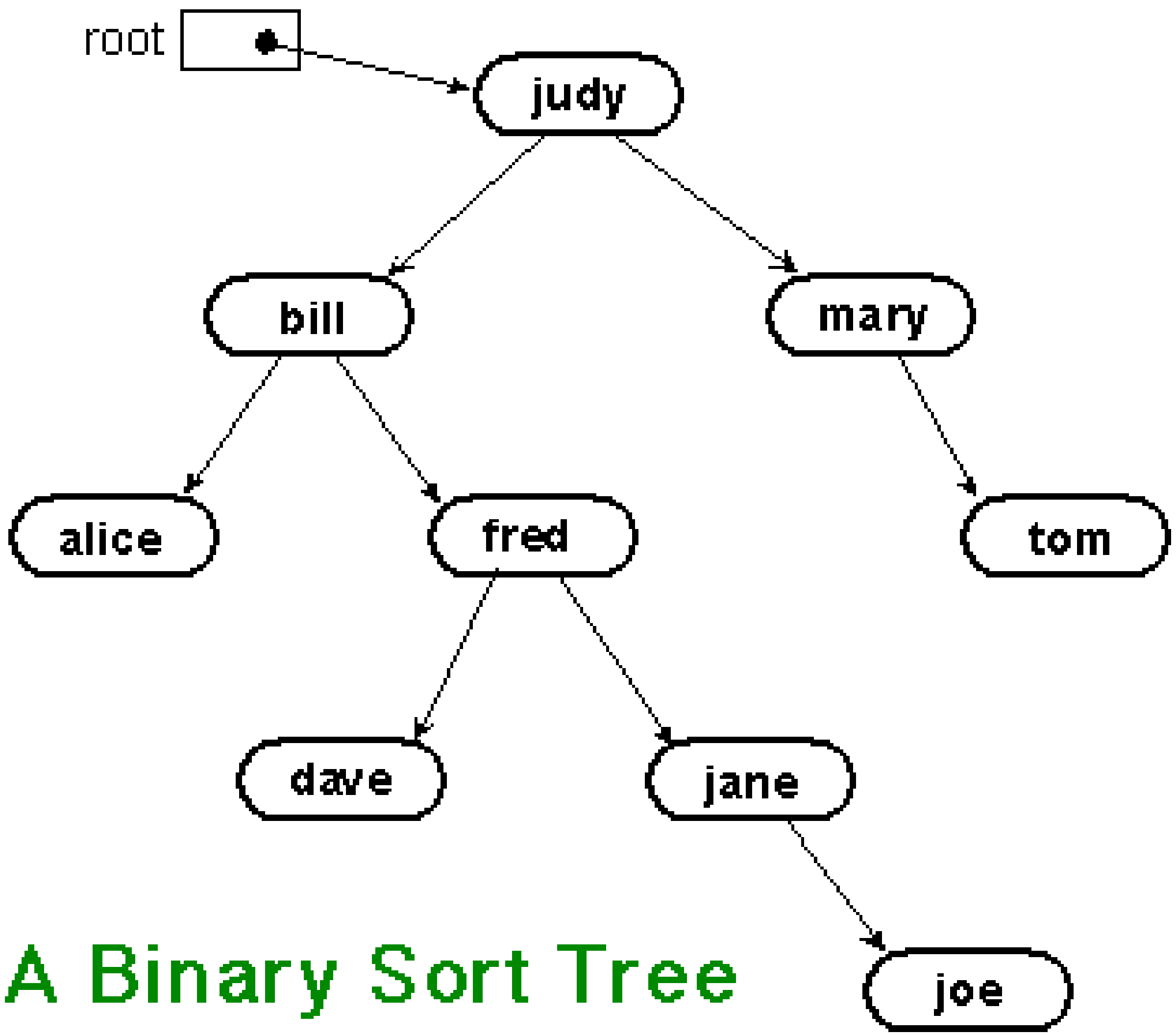
Optimal Binary Search Tree

- Suppose we are implementing a dictionary for words. The dictionary could be meaning of words, or reference to related information.
- Searching for a word in arbitrary placed words is O(n).
- A binary search tree reduces the time for individual search to O(log n).
- However, this will be true, if it is a balanced BST, otherwise it could be up to O(n).
- Note, the BST tree can be organized in number of ways.

- We need the BST tree that serves our purpose.
- What is the purpose of building BST?
- To search for words.
- If the frequently searched words are near the root, then it is okay.
- If those words are towards bottom of the tree, then overall search time is going to be very high.
- Is there a way to optimize it?

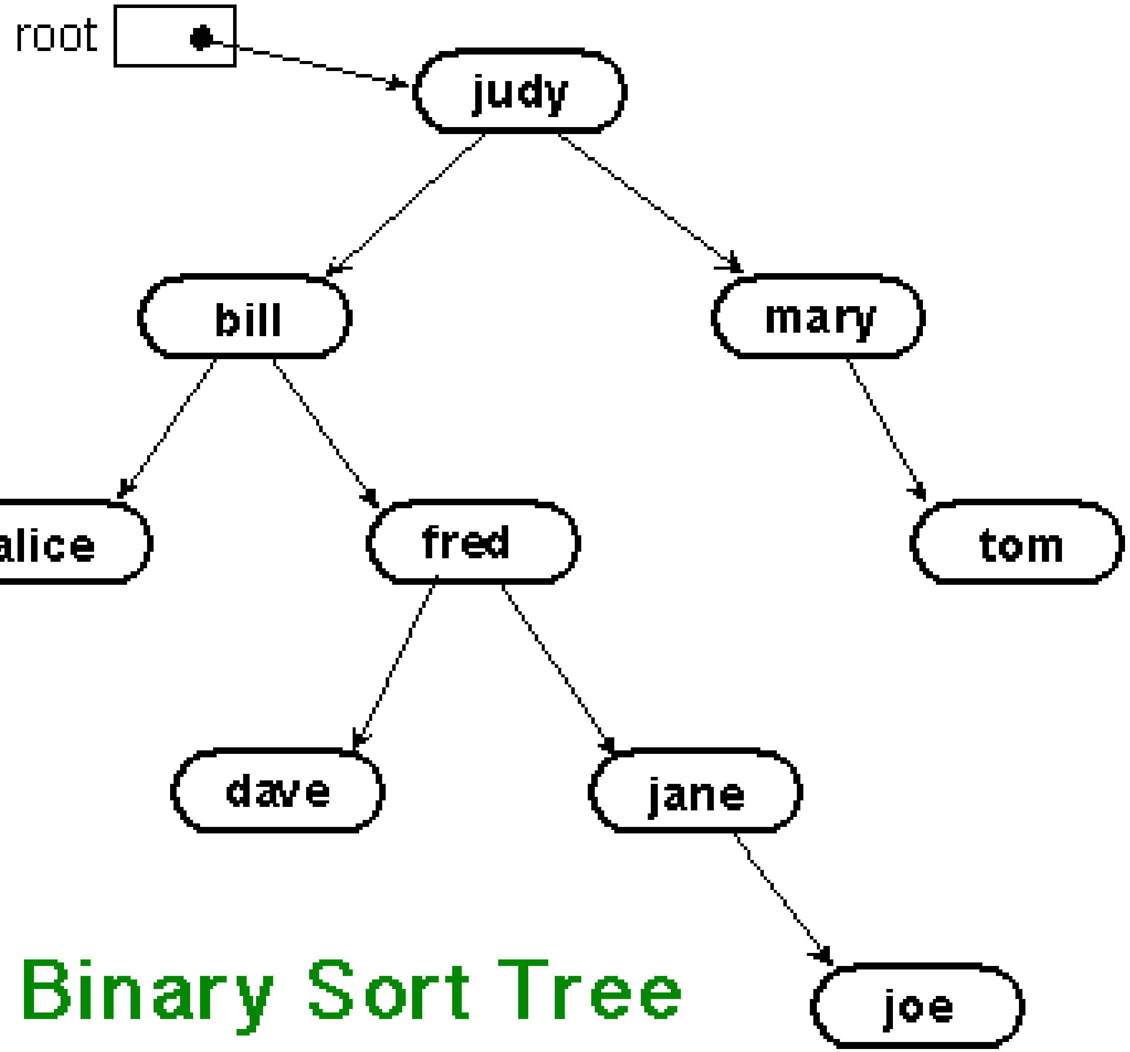
- Judy searched 100 times
- Joe searched 10 times

OKAY



- Judy searched 10 times
- Joe searched 100 times

Not Okay



A Binary Sort Tree

- We want to create such a BST tree
- where more frequently searched words appear towards the top

- We call it an OPTIMAL BST

- The OPTIMAL BST tree is one, where
- not only *alphabetical order*, but
- *frequency of search* also is taken into account.

- Consider a case where we have only 3 elements
- keys = { 10, 12, 20 } with search freq. { 34, 8, 50 }
- There can be 5 following possible BSTs
- 10 12 20 10 20
- \ / \ / \ /
- 12 10 20 12 20 10
- \ / / \
- 20 10 12 12
- I II III IV V

- Cost for tree II: 8+ 68 + 100 = 176
- Cost for tree V: 50+ 68 + 24 = 142

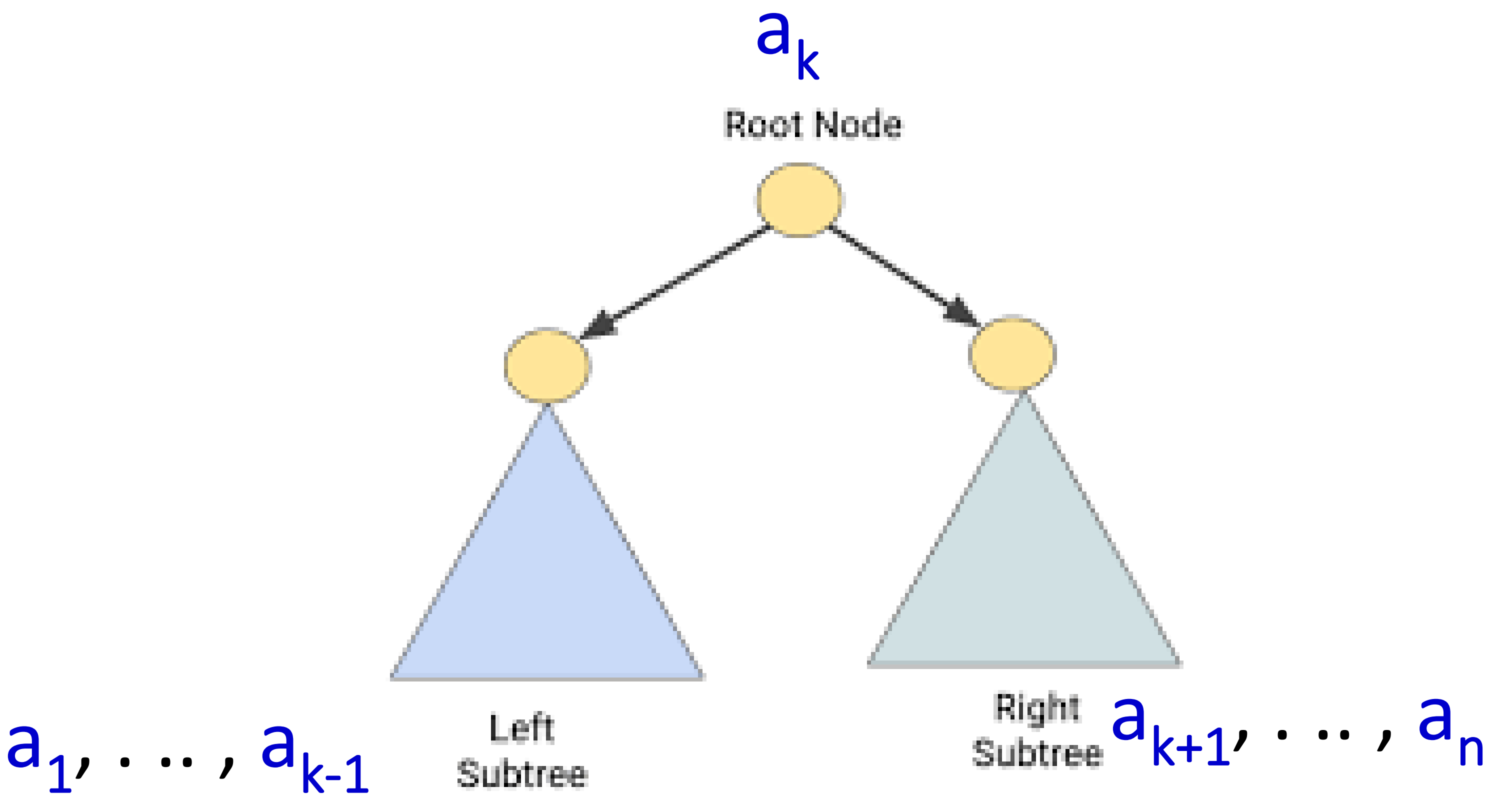
- A collection of words leads to a very large number of possible BSTs.
- Suppose we try to find cost of all possible BST trees,
- The time complexity of searching for best BST, may turn out to have exponential complexity.

$$\frac{{}^{2n}C_n}{n+1}$$

- So we use Dynamic Programming to solve this problem.

- Let a_1 , a_2, ... , a_n be the nodes of a BST, arranged in ascending order, and let p_1 , p_2, ... , p_n be the probabilities of searching these items.

- Suppose a_k is the root node of the BST,
- nodes a_1, ... , a_{k-1} are in the left subtree and
- a_{k+1}, ... , a_n are in the right subtree.


a_k
Root Node
$a_1, \ldots, a_{k-1}$
Left Subtree
Right Subtree
$a_{k+1}, \ldots, a_n$

- Let the average search time to process the left subtree is C[1. . . k-1] plus p_1 , p_2, ... , p_{k-1} to process the items in the root.

- By same logic, search time for right subtree is

  C[k+1…n] plus p_{k+1} , . . . , p_n .

- Cost of tree from node i to node j. k is the root node which splits the nodes in two parts. We need to figure out best value of k.
- $C[\, i, j\,] = \min \{C[i. . . k-1] + C[k+1. . .j]\} + \sum p_i$.
- $\quad\quad = \min \{C[i. . . k-1] + C[k+1. . .j]\} + p_i + . . . + p_j$.
- $C[\, i\,, j\,] = p_i$.
- $C[\, i, i–1] = 0$.

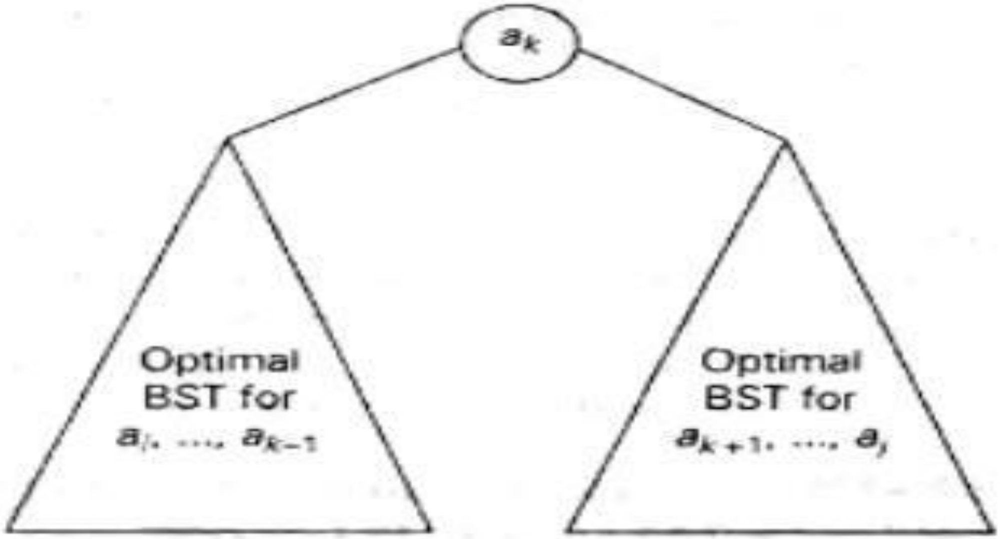
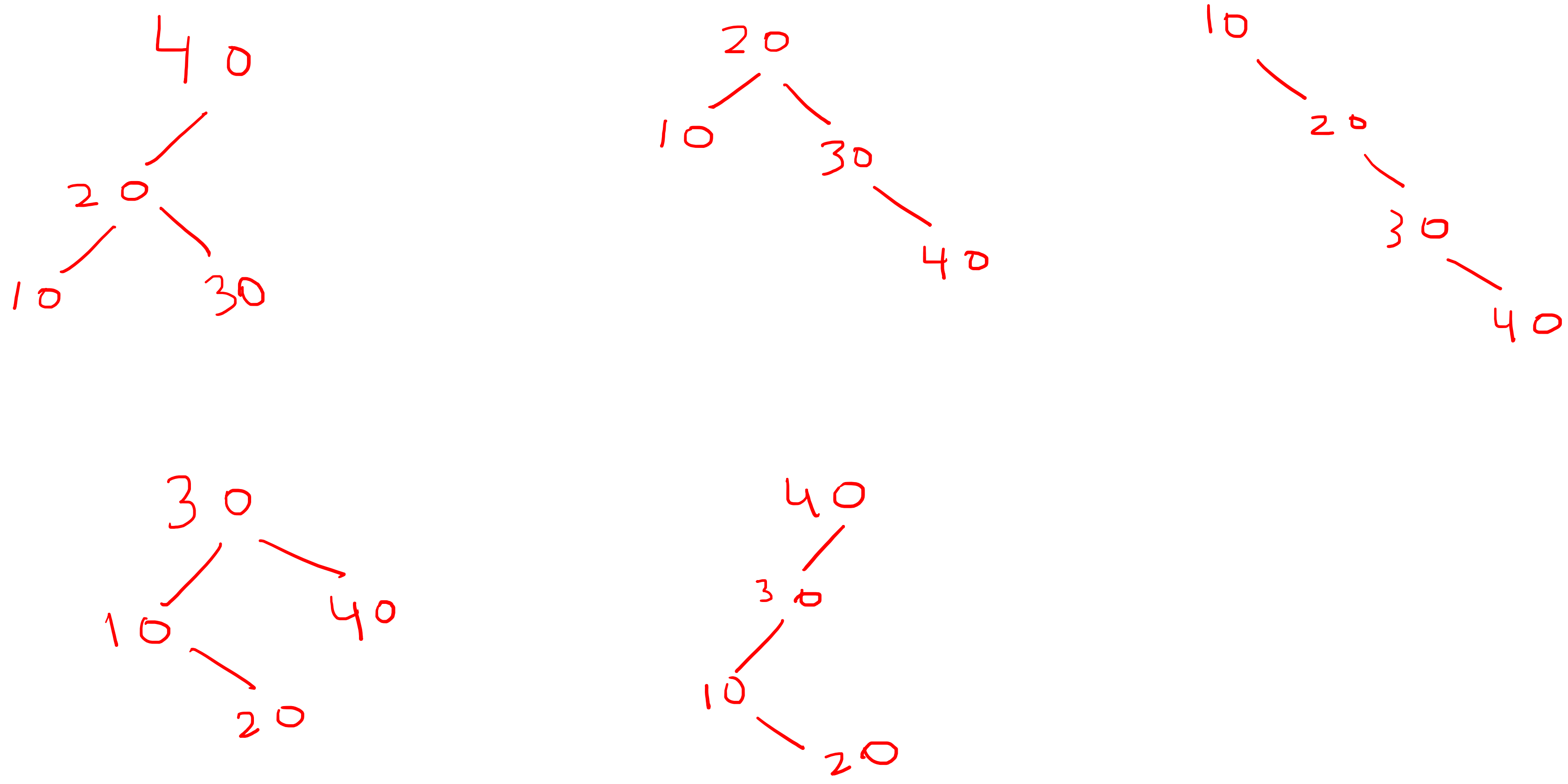


Fig: Binary search tree with root a_k and two optimal binary search subtrees and

- Example: Consider a 4 node tree with

	1	2	3	4
Keys →	10	20	30	40
Frequency →	4	2	6	3

- 14 trees are possible, we have to find min cost tree



- Brute force technique would involve examining each tree to find min cost tree.

- As n increases, possible number of BST will keep on growing.
- For n=6, the number is 132

- So Brute force technique will be cumbersome for bigger trees.

.

Dynamic Programming Approach for BST

.

- Let us now take up the DP approach.
- We need 2 matrices, the COST matrix (based on Frequency)
- and the BST order matrix (based on Key values)

	1	2	3	4
Keys →	10	20	30	40
Frequency →	4	2	6	3

- We create a COST matrix C[,] using dynamic programming
- C[i , i] = p_i .
- C[i, i – 1] = 0 .
- Each C[i, j] is computed for all possible values of k and taking the minimum
- C[i,j]   = min {C[i. . . k-1] + C[k+1. . .j]} + p_i+ . . + p_j .

- For our example
- $C[i\ , i\ ] = p_i$.
- $C[i, i - 1] = 0$.

	0	1	2	3	4	COST
1	0	4				
2		0	2			
3			0	6		
4				0	3	
5					0	

	1	2	3	4
Keys →	10	20	30	40
Frequency →	4	2	6	3

COST

	0	1	2	3	4
1	0	4			
2		0	2		
3			0	6	
4				0	3
5					0

BST

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

- $C[i,j] = \min \{C[i...k-1] + C[k+1...j]\} + p_i + ... + p_j$.
- calculate C[1,2] , k has 2 values
- k =1, k=2
- $C[i,j] = \min \{C[1,0] + C[2,2] , C[1,1] + C[3,2] \} + p_1 + p_2$.
- $= \min\{ 2,4\} + 4 + 2 = 8$
- Min cost is found for k = 1,
- Update this information on the corresponding BST matrix

	0	1	2	3	4	COST
1	0	4	8			
2		0	2			
3			0	6		
4				0	3	
5					0	

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

- C[i,j] = min {C[i. . . k-1] + C[k+1. . .j]} + p_i+ . . + p_j .
- calculate C[1,2] , k has 2 values
- k =1, k=2
- C[i,j]= min {C[1,0] + C[2,2] , C[1,1] +C[3,2] } + p_1+ p_2 .
- = min{ 2,4} + 4 + 2 = 8
- Min cost is found for k = 1,
- Update this information on the corresponding BST matrix

	0	1	2	3	4	**COST**
1	0	4	8			
2		0	2	10		
3			0	6		
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1			
2		0	2	3		
3			0	3		
4				0	4	
5					0	

- $C[i,j] = \min\{C[i...k-1] + C[k+1..j]\} + p_i + p_j$.
- calculate **C[2,3]** ,
- k has 2 values
- k =2, k=3
- $C[i,j] = \min\{ C[2,1] + C[3,3] , C[2,2] + C[4,3] \}$
- $+ p_2 + p_3$.
- $= \min\{0+6, 2+0\} + 2 + 6$
- $= 2+8$
- $= 10$

- The min. value is obtained for k =3
- so update BST matrix

	0	1	2	3	4	**COST**
1	0	4	8			
2		0	2	10		
3			0	6	12	
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1			
2		0	2	3		
3			0	3	3	
4				0	4	
5					0	

- C[i,j]= min {C[i. . . k-1] + C[k+1. . .j]} + p_i+ . . + p_j .
- calculate C[3,4] , k has 2 values
- k =3, k=4
- C[i,j]=min { C[3,2] + C[4,4] ,
  C[3,3] +C[5,4] } + p_3+ p_4 .
- = min{ 0+3, 6+0} + 6 +3.
- = 3+9
- = 12
- The min. value is obtained for k = 3

	0	1	2	3	4	**COST**
1	0	4	8	20		
2		0	2	10		
3			0	6	12	
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1	3		
2		0	2	3		
3			0	3	3	
4				0	4	
5					0	

- Calculate C[1,3] , k has 3 values
- k =1, k=2, k = 3
- =min { C[1,0] + C[2,3] ,
- C[1,1] + C[3,3],
- C[1,2] +C[4,3] } + p_1+ p_2 + p_3..
- = min{ 0+10,
- 4+6,
- 8+0 } + 4 + 2 + 6.
- = 8 + 12
- = 20

	0	1	2	3	4	**COST**
1	0	4	8	20		
2		0	2	10	16	
3			0	6	12	
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1	3		
2		0	2	3	3	
3			0	3	3	
4				0	4	
5					0	

- calculate C[2,4] , k has 3 values
- k=2, k = 3, k =4
- =min { C[2,1] + C[3,4] ,
- C[2,2] + C[4,4],
- C[3,4] +C[5,4] } + p_2 + p_3 + p_4
- = min{ 0+12,
- 2+3,
- 12+0 } + 2 + 6 +3.
- = 5 + 11 = 16.

- Min value is obtained for k=3

	0	1	2	3	4	**COST**
1	0	4	8	20	26	
2		0	2	10	**16**	
3			0	6	12	
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1	3	3	
2		0	2	3	3	
3			0	3	3	
4				0	4	
5					0	

- calculate C[1,4] , k has 4 values
- k = 1, k=2, k = 3, k =4
- C =min { C[1,0] + C[2,4] ,
- C[1,1] + C[3,4],
- C[1,2] +C[4,4],
- C[1,3]+C[5,4] }

$+ p_1 + p_2 + p_3 + p_4$

- C= min { 0+16, 4+12, 8+3, 20+0} +4+2+6+3.
- = 11 + 15 = 26.
- Again best value obtained for k = 3

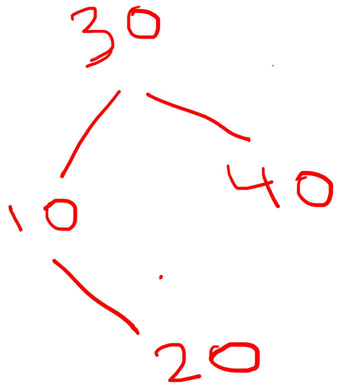
	0	1	2	3	4	**COST**
1	0	4	8	20	26	
2		0	2	10	**16**	
3			0	6	12	
4				0	3	
5					0	

	0	1	2	3	4	**BST**
1	0	1	1	3	3	
2		0	2	3	3	
3			0	3	3	
4				0	4	
5					0	

	1	2	3	4
Keys →	10	20	30	40
Frequency →	4	2	6	3

- We have now solved the problem using DP approach.
- To draw the final optimal BST, we make use of the BST matrix.

- Now to work out the structure of optimal BST.
- Note BST(1,4) is 3, so key 3 is the root   (value 30)
- key 4 is greater than key 3, so forms right of 30
- R(1,2) is 1, so key 1 forms the root of left subtree of key 3.
- key 2 will be right child of key 1



- Let us verify the cost of optimal BST.
- 6*1 + 4*2 + 3*2 + 2*3
- = 6+8+6+6 = 26