DAA IMPORTANT QUESTIONS

1. **Discuss Optimal Binary search trees with an example**.

Optimal Binary Search Tree:

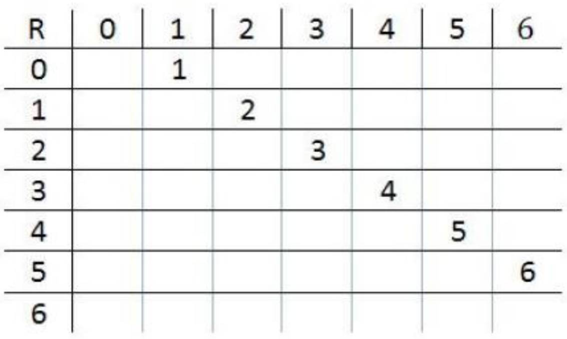
An **optimal binary search tree (Optimal BST)**, sometimes called a **weight-balanced binary tree**,[1] is a binary search tree which provides the smallest possible search time (or expected search time) for a given sequence of accesses (or access probabilities).

The cost of searching is a very important factor in various applications. The overall cost of searching a node should be less. The time required to search a node in BST is more than the balanced binary search tree as a balanced binary search tree contains a lesser number of levels than the BST. There is one way that can reduce the cost of a binary search tree is known as an **optimal binary search tree**.

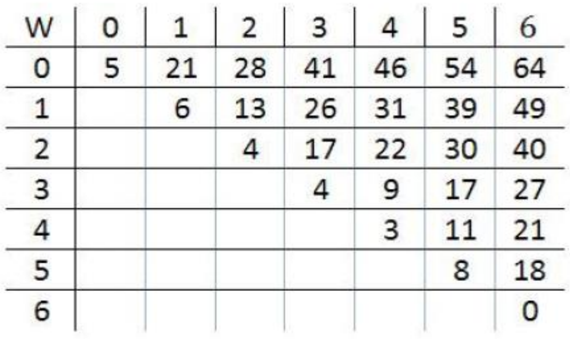
EXAMPLE:

Example: Find the optimal binary search tree for N = 6, having keys k1 … k6 and weights p1 = 10, p2 = 3, p3 = 9, p4 = 2, p5 = 0, p6 = 10; q0 = 5, q1 = 6, q2 = 4, q3= 4, q4 = 3, q5 = 8, q6 = 0. Construct an OBST

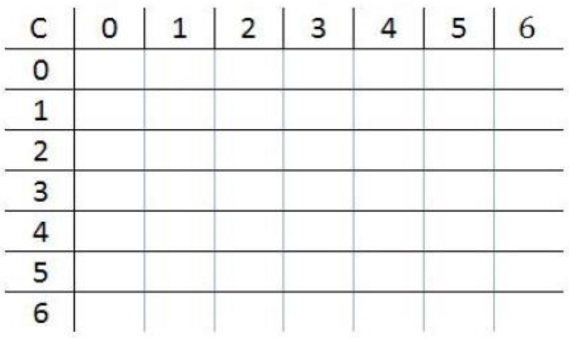
Initial array values:



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



W	0	1	2	3	4	5	6
0	5	21	28	41	46	54	64
1		6	13	26	31	39	49
2			4	17	22	30	40
3				4	9	17	27
4					3	11	21
5						8	18
6							0



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

The values of the weight matrix have been computed according to the formulas previously stated, as follows:

W (0, 0) = q0 = 5 W (0, 1) = q0 + q1 + p1 = 5 + 6 + 10 = 21

W (1, 1) = q1 = 6 W (0, 2) = W (0, 1) + q2 + p2 = 21 + 4 + 3 = 28

W (2, 2) = q2 = 4 W (0, 3) = W (0, 2) + q3 + p3 = 28 + 4 + 9 = 41

W (3, 3) = q3 = 4 W (0, 4) = W (0, 3) + q4 + p4 = 41 + 3 + 2 = 46

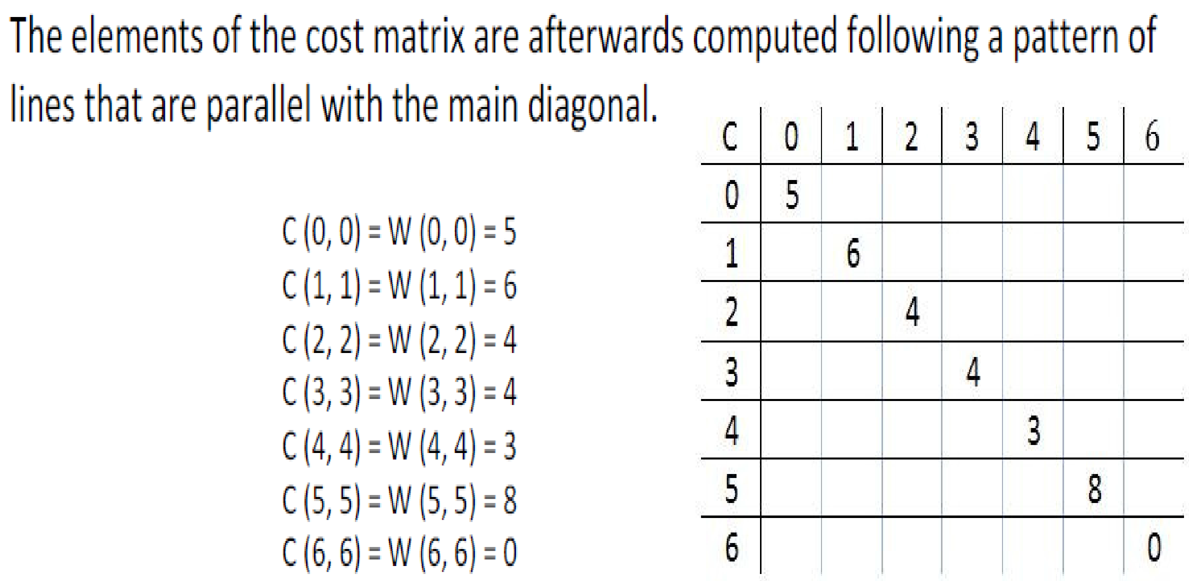
W (4, 4) = q4 = 3 W (0, 5) = W (0, 4) + q5 + p5 = 46 + 8 + 0 = 54

W (5, 5) = q5 = 8 W (0, 6) = W (0, 5) + q6 + p6 = 54 + 0 + 10 = 64

W (6, 6) = q6 = 0 W (1, 2) = W (1, 1) + q2 + p2 = 6 + 4 + 3 = 13

--- and so on --- until we reach:

W (5, 6) = q5 + q6 + p6 = 18



The elements of the cost matrix are afterwards computed following a pattern of lines that are parallel with the main diagonal.

C (0, 0) = W (0, 0) = 5
C (1, 1) = W (1, 1) = 6
C (2, 2) = W (2, 2) = 4
C (3, 3) = W (3, 3) = 4
C (4, 4) = W (4, 4) = 3
C (5, 5) = W (5, 5) = 8
C (6, 6) = W (6, 6) = 0

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

The elements of the cost matrix are afterwards computed following a pattern of lines that are parallel with the main diagonal.

C (0, 0) = W (0, 0) = 5
C (1, 1) = W (1, 1) = 6
C (2, 2) = W (2, 2) = 4
C (3, 3) = W (3, 3) = 4
C (4, 4) = W (4, 4) = 3
C (5, 5) = W (5, 5) = 8
C (6, 6) = W (6, 6) = 0

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

C	0	1	2	3	4	5	6
0	5	32					
1		6	23				
2			4	25			
3				4	16		
4					3	22	
5						8	26
6							0

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

C (0, 2) = W (0, 2) + min (C (0, 0) + C (**1**, 2), C (0, 1) + C (2, 2)) = 28 + min (**28**, 36) = 56
C (1, 3) = W (1, 3) + min (C (1, 1) + C (2, 3), C (1, 2) + C (**3**, 3)) = 26 + min (31, **27**) = 53
C (2, 4) = W (2, 4) + min (C (2, 2) + C (**3**, 4), C (2, 3) + C (4, 4)) = 22 + min (**20**, 28) = 42
C (3, 5) = W (3, 5) + min (C (3, 3) + C (4, 5), C (3, 4) + C (**5**, 5)) = 17 + min (26, **24**) = 41
C (4, 6) = W (4, 6) + min (C (4, 4) + C (5, 6), C (4, 5) + C (**6**, 6)) = 21 + min (29, **22**) = 43

C	0	1	2	3	4	5	6
0	5	32	56				
1		6	23	53			
2			4	25	42		
3				4	16	41	
4					3	22	43
5						8	26
6							0

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

Final array values:

C	0	1	2	3	4	5	6
0	5	32	56	98	118	151	188
1		6	23	53	70	103	140
2			4	25	42	75	108
3				4	16	41	68
4					3	22	43
5						8	26
6							0

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

The resulting optimal tree is shown in the bellow figure and has a weighted path length of 188.

Computing the node positions in the tree:

- The root of the optimal tree is R(0, 6) = k3;

- The root of the left subtree is R(0, 2) = k1;

- The root of the right subtree is R(3, 6) = k6;

- The root of the right subtree of k1 is R(1, 2) = k2

- The root of the left subtree of k6 is R(3, 5) = k5

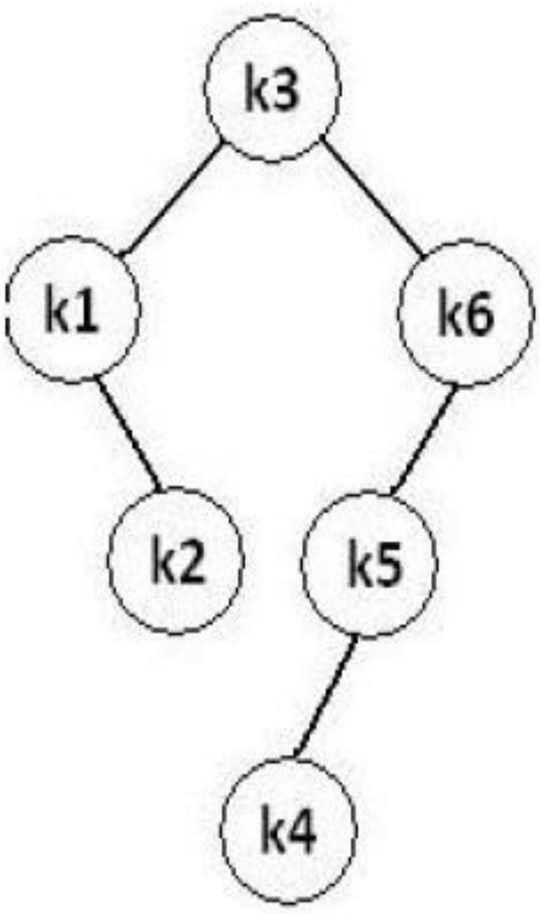
- The root of the left subtree of k5 is R(3, 4) = k4

Thus, the optimal binary search tree obtained will have the following structure:

Disadvantages:

1. Constructed for a finite set of elements with the weights known.

2. Time Complexity for calculating the matrices and constructing OBST is high , though the worst time complexity for search is O(log n)

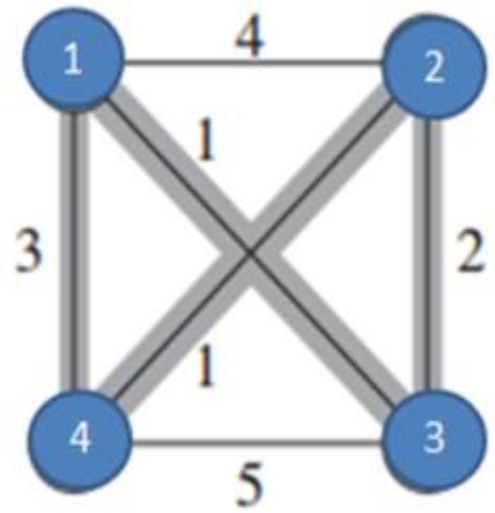


2. **Construct an Optimal Binary Search tree for the given data.**

   Refer previous example

3. **Apply dynamic programming to find optimal tour of travelling sales person problem**

Travelling sales person problem



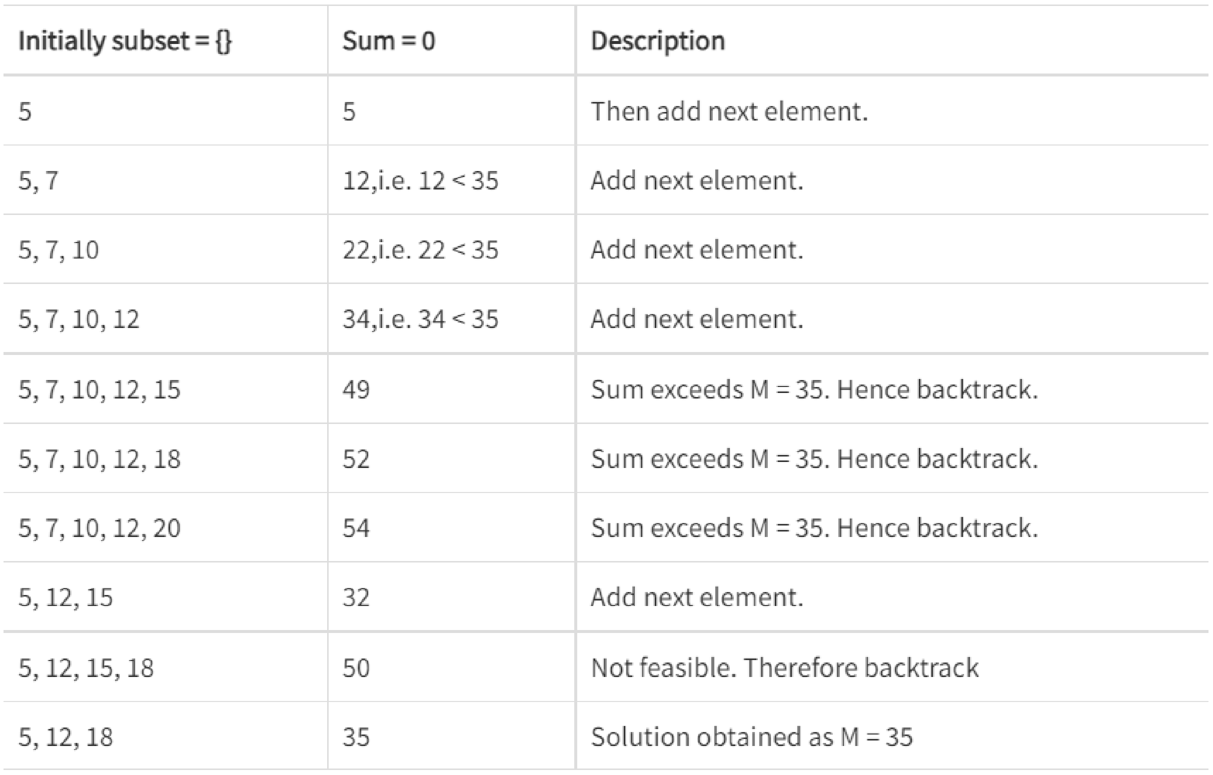
$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 4 & 1 & 3 \\ 2 & 4 & 0 & 2 & 1 \\ 3 & 1 & 2 & 0 & 5 \\ 4 & 3 & 1 & 5 & 0 \end{array}$$

- cost(2, ∅)=d(2,1)=4
- cost(3, ∅)=d(3,1)=1
- cost(4, ∅)=d(4,1)=3
- cost(2, {3})=dist(2, 3)+cost(3, ∅)=2+1=3
- cost(2, {4})= dist(2, 4)+cost(4, ∅)=1+3=4
- cost(3, {2})= dist(3, 2)+cost(2, ∅)=2+4=6
- cost(3, {4})=8
- cost(4, {2})=5
- cost(4, {3})=6

cost(i ,S)=min{dist(i,K)+cost(K,S-{K})
K ∈S

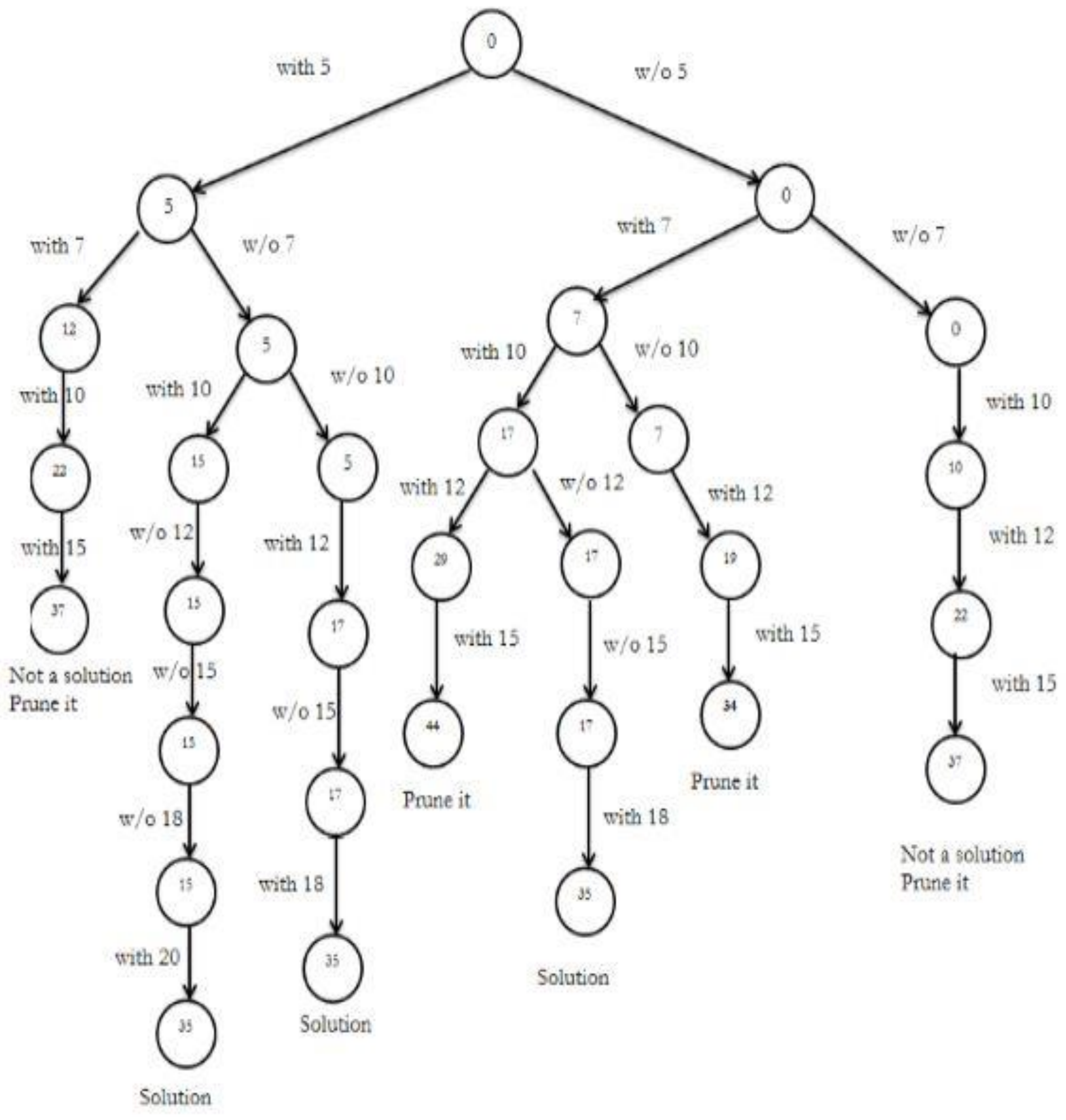
- cost(2, {3,4})=min{dist(2,3)+cost(3,4),dist(2,4)+cost(4,3)}=min{10,7}=7
- cost(3, {2,4})= min{dist(3,2)+cost(2,4),dist(3,4)+cost(4,2)}=6
- cost(4, {2,3})= min{dist(4,2)+cost(2,3),dist(4,3)+cost(3,2)}=4
- cost(1, {2,3,4})=min{dist(1,2)+cost(2,{3,4}),dist(1,3)+cost(3,{2,4},dist(1,4)+cost(4,{2,3})} =min{11,7,7}=7
- So the TSP tour is 1->3->2->4->1 or 1->4>2->3->1
- cost of the tour is 7

4. **Develop a backtracking algorithm for sum of subsets problem. Apply it for the given set s= {10, 7, 5, 18, 12, 20, 15} with the sum Value 35.**



Initially subset = {}	Sum = 0	Description
5	5	Then add next element.
5, 7	12,i.e. 12 < 35	Add next element.
5, 7, 10	22,i.e. 22 < 35	Add next element.
5, 7, 10, 12	34,i.e. 34 < 35	Add next element.
5, 7, 10, 12, 15	49	Sum exceeds M = 35. Hence backtrack.
5, 7, 10, 12, 18	52	Sum exceeds M = 35. Hence backtrack.
5, 7, 10, 12, 20	54	Sum exceeds M = 35. Hence backtrack.
5, 12, 15	32	Add next element.
5, 12, 15, 18	50	Not feasible. Therefore backtrack
5, 12, 18	35	Solution obtained as M = 35

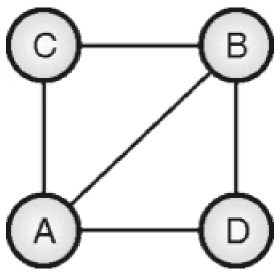
The state space tree is shown as below in figure 8. (5, 7, 10, 12, 15, 18, 20)



5.Explain Hamiltonian cycles problem with an example.

- In an undirected Graph, Hamiltonian path is a path that visits each vertex exactly once.
- Hamiltonian cycle or circuit is a Hamiltonian path that ends at the starting vertex.
- A graph having Hamiltonian cycles is called Hamiltonian Graph.
- we can find the Hamiltonian Circuits using Backtracking approach.

Example:



Solution:
We can start with any random vertex. Let us start with vertex A. Neighbors of A are {B, C, D}. The inclusion of B does not lead to a complete solution. So explore it as shown in Figure (c).

Adjacent vertices to B are {C, D}. The inclusion of C does not lead to a complete solution. All adjacent vertices of C are already members of the path formed so far. So it leads to a dead end.

Backtrack and go to B and explore its next neighbor i.e. D.

The inclusion of D does not lead to a complete solution, and all adjacent vertices of D are already a member of the path formed so far. So it leads to a dead end.

Figure: (b) initial tree

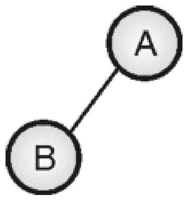


Figure (c): Added node B

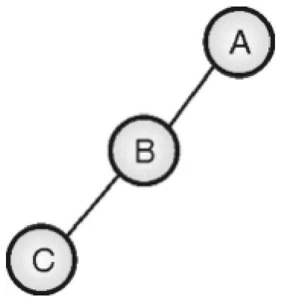


Figure (d): Node C added, dead-end

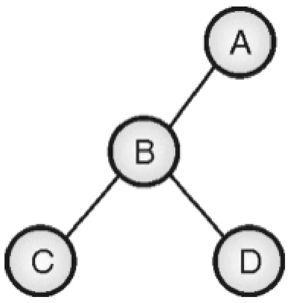
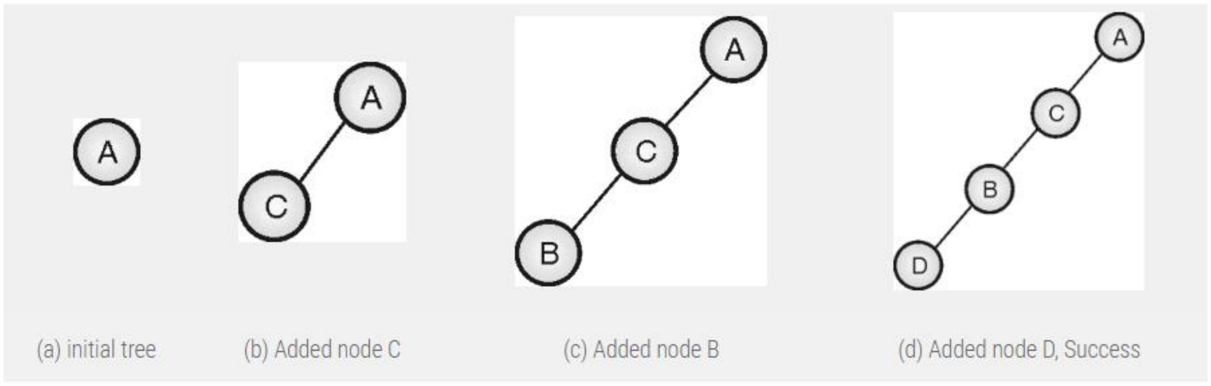


Figure (e): Node D added, dead-end

Backtrack and go to B. Now B does not have any more neighbors, so backtrack and go to A. Explore the next neighbor of A i.e. C. By repeating the same procedure, we get the state space trees as shown below. And path
A – C – B – D – A is detected as the Hamiltonian cycle of the input graph.

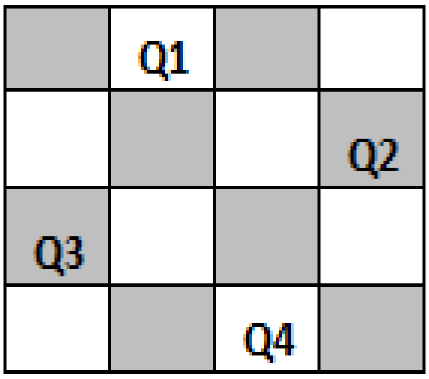


Another Hamiltonian cycle with A as a start vertex is A – D – B – C – A.

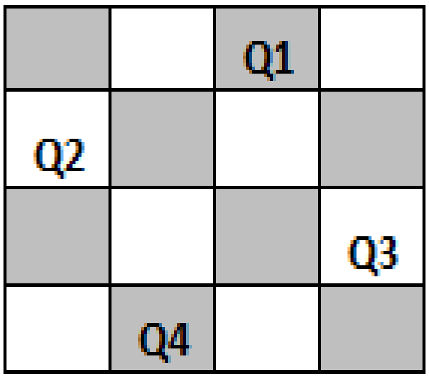
6) Illustrate the solution for 4-queens problem using backtracking method.

4-Queens problem

- **4-Queens problem is the problem of placing 4-Queens on a 4X4 chessboard such that no two queens attack each other.**
- **following are two solutions for 4 Queen problem:**



Solution 1

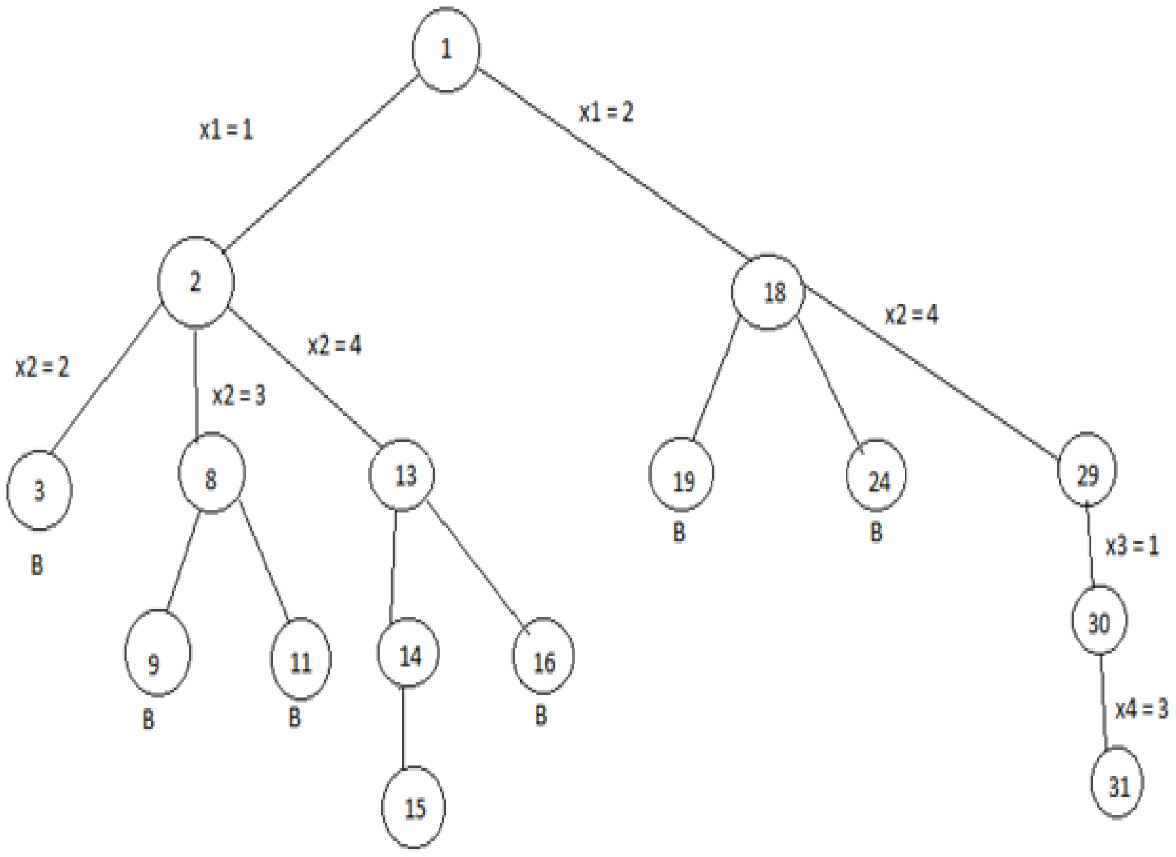


Solution 2

The solutions for 4-Queens problem:

1.(2,4,1,3)

2.(3,1,4,2)



7)Compare and contrast Backtracking and dynamic programming

Backtracking

Backtracking is one of the problem-solving techniques. Using this technique, we can solve our problem. This strategy uses a Brute force approach, and the brute force approach says that for the given problem, we should try out all the possible solutions and pick the desired solution from all the possible solutions. In contrast, dynamic programming is used to solve the optimization problem, but backtracking is not used to solve the optimization problems. When multiple solutions to a given problem exist, then backtracking uses all the solutions to solve the problem.

dynamic programming

Dynamic programming is a technique for solving certain type of complex problems efficiently by breaking them down into simpler subproblems and solving each problem exactly once. Dynamic programming stores the result of a subproblem in a table and reuse them when needed to avoid solving the same problems again and again.

Differences:

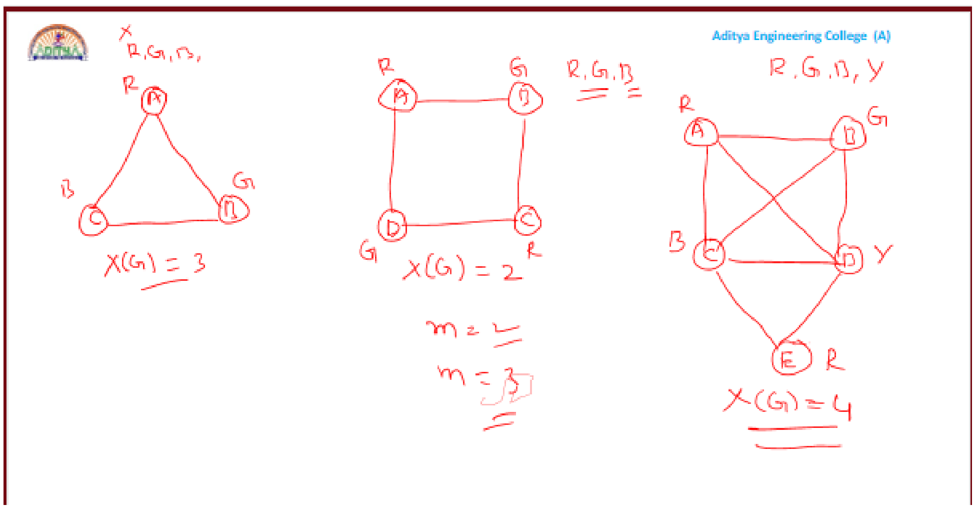
- Dynamic programming is a technique of dividing the complex problem into simpler sub-problems. This technique is applicable to the problems that exhibit the-following-properties:

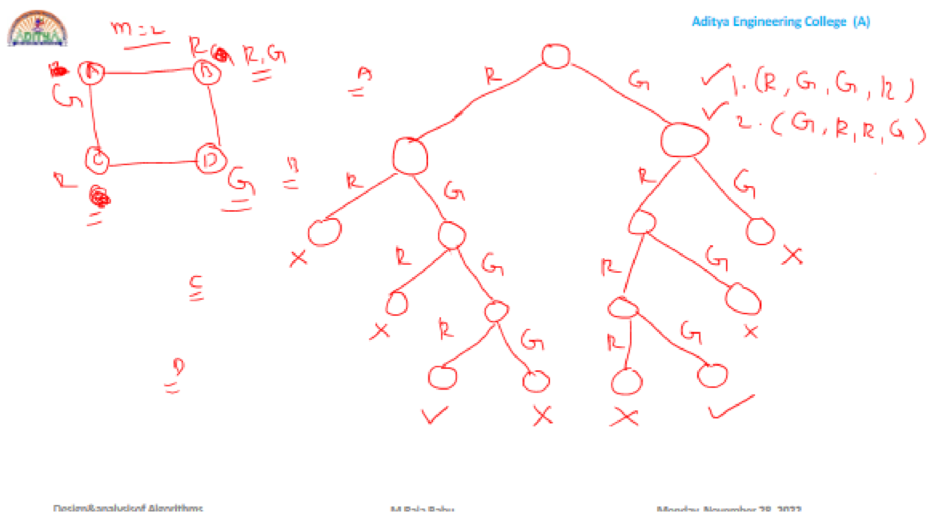
- Overlapping-subproblems
- Optimalsub-structure
- Backtracking is a technique that recursively build a solution incrementally and remove all the solutions that does not satisfy the constraints of the problem at any point of time.
- The major difference between the dynamic programming and backtracking is that the dynamic programming completely relies on the principle of optimality which means that the sub sequence of a sequence should be optimal. In contrast to dynamic programming, backtracking does not guarantee the complete optimal solution.
- Dynamic programming is a technique that solves the optimization problem. Optimization problem uses either minimum or maximum result. In contrast to dynamic programming, backtracking uses the brute force approach without

considering the optimization problem. If we have multiple solutions then it considers all those solutions.

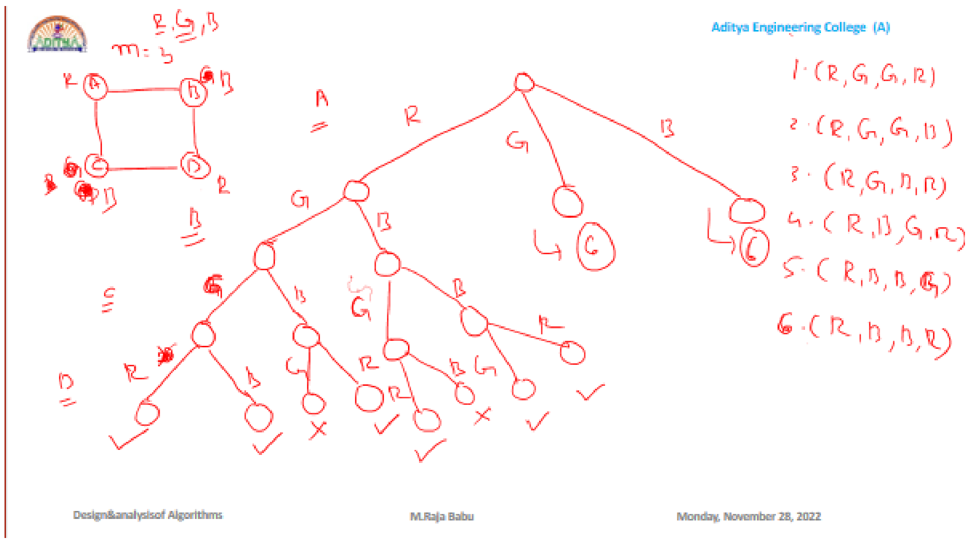
8)Apply graph m-coloring problem for the given graph

- **The problem is to find if it is possible to assign nodes with m different colors, such that no two adjacent vertices of the Graph G are of the same colors. If the solution exists, then display which color is assigned on which vertex.**





Aditya Engineering College (A)
m = 2
R,G
1. (R,G,G,R)
2. (G,R,R,G)
Design&analysis of Algorithms
M Raja Babu
Monday, November 28, 2022



9)Differentiate between Deterministic and Non-deterministic Algorithms

Key	Deterministic Algorithm	Non-deterministic Algorithm
Definition	The algorithms in which the result of every algorithm is uniquely defined are known as the Deterministic Algorithm. In other words, we can say that the deterministic algorithm is the algorithm that performs fixed number of steps and always get finished with an accept or reject state with the same result.	On other hand, the algorithms in which the result of every algorithm is not uniquely defined and result could be random are known as the Non-Deterministic Algorithm.
Execution	In Deterministic Algorithms execution, the target machine executes the same instruction and results same outcome which is not dependent on the way or process in which instruction get executed.	On other hand in case of Non-Deterministic Algorithms, the machine executing each operation is allowed to choose any one of these outcomes subjects to a determination condition to be defined later.
Type	On the basis of execution and outcome in case of Deterministic algorithm, they are also classified as reliable algorithms as for a particular input instructions the machine will give always the same output.	On other hand Non deterministic algorithm are classified as non-reliable algorithms for a particular input the machine will give different output on different executions.

Execution Time	As outcome is known and is consistent on different executions so Deterministic algorithm takes polynomial time for their execution.	On other hand as outcome is not known and is non-consistent on different executions so Non-Deterministic algorithm could not get executed in polynomial time.
Execution path	In deterministic algorithm the path of execution for algorithm is same in every execution.	On other hand in case of Non-Deterministic algorithm the path of execution is not same for algorithm in every execution and could take any random path for its execution.

10)Explain NP - Hard and NP Complete class problems

NP-Hard and NP-Complete Problem:

Let P denote the set of all decision problems solvable by deterministic algorithm in polynomial time. NP denotes set of decision problems solvable by nondeterministic algorithms in polynomial time. Since, deterministic algorithms are a special case of nondeterministic algorithms, P ⊆ NP. The nondeterministic polynomial time problems can be classified into two classes. They are

1. NP Hard and
2. NP Complete

NP-Hard: A problem L is NP-Hard iff satisfiability reduces to L i.e., any nondeterministic polynomial time problem is satisfiable and reducable then the problem is said to be NP-Hard.
Example: Halting Problem, Flow shop scheduling problem

NP-Complete: A problem L is NP-Complete iff L is NP-Hard and L belongs to NP (nondeterministic polynomial).

A problem that is NP-Complete has the property that it can be solved in polynomial time iff all other NP-Complete problems can also be solved in polynomial time. (NP=P)

If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time. All NP-Complete problems are NP-hard, but some NP-hard problems are not known to be NP-Complete.
Normally the decision problems are NP-complete but the optimization problems are NP-Hard.

However if problem L1 is a decision problem and L2 is an optimization

problem, then it is possible that L1α L2.

Example: Knapsack decision problem can be reduced to knapsack optimization problem.

There are some NP-hard problems that are not NP-Complete.

11)Discuss cooks theorem

Cook's theorem
Cook's Theorem implies that any NP problem is at most polynomially harder
than SAT(satisfiability problem). This means that if we find a way of solving SAT in polynomial
time, we will then be in a position to solve any NP problem in polynomial
time.

Stephen Cook presented four theorems in his paper "The Complexity of Theorem Proving Procedures". These theorems are stated below. We do understand that many unknown terms are being used in this chapter, but we don't have any scope to discuss everything in detail.

Following are the four theorems by Stephen Cook –

Theorem-1

If a set **S** of strings is accepted by some non-deterministic Turing machine within polynomial time, then **S** is P-reducible to {DNF tautologies}.

Theorem-2

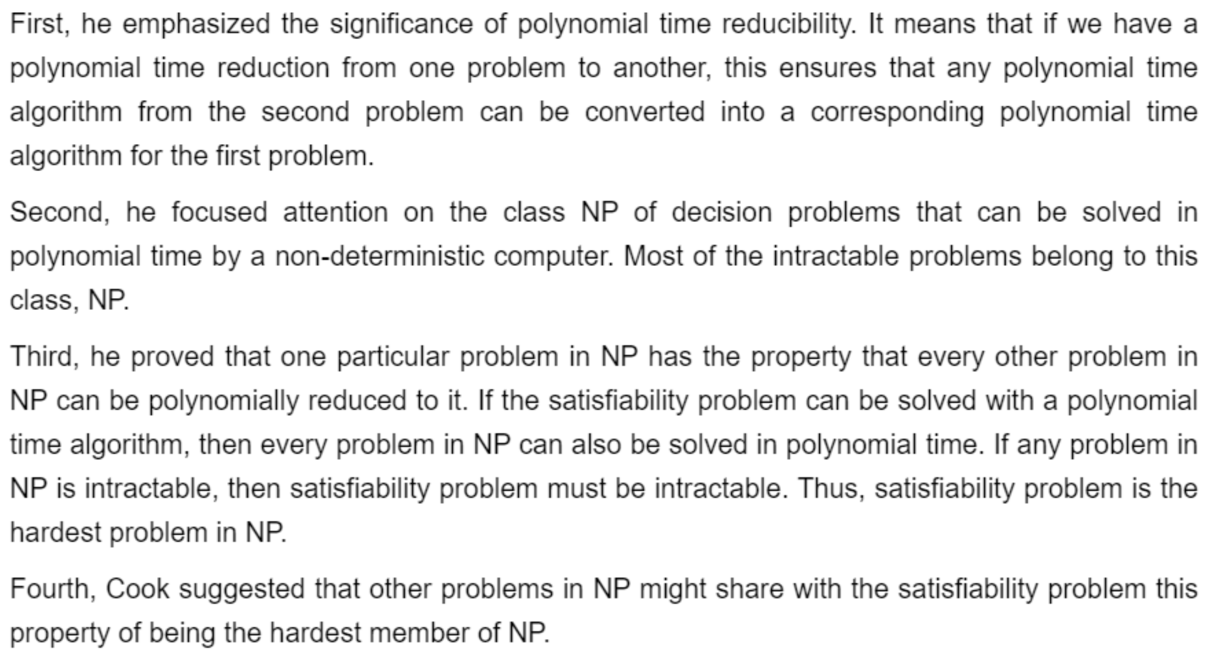
The following sets are P-reducible to each other in pairs (and hence each has the same polynomial degree of difficulty): {tautologies}, {DNF tautologies}, D3, {sub-graph pairs}.

Theorem-3

- For any ***T_Q(k)*** of type **Q**, TQ(k)k√(logk)2TQ(k)k(logk)2 is unbounded
- There is a ***T_Q(k)*** of type **Q** such that TQ(k)⩽2k(logk)2TQ(k)⩽2k(logk)2

Theorem-4

If the set S of strings is accepted by a non-deterministic machine within time ***T(n) = 2^n***, and if ***T_Q(k)*** is an honest (i.e. real-time countable) function of type **Q**, then there is a constant **K**, so **S** can be recognized by a deterministic machine within time ***$T_Q(K8^n)$***.



First, he emphasized the significance of polynomial time reducibility. It means that if we have a polynomial time reduction from one problem to another, this ensures that any polynomial time algorithm from the second problem can be converted into a corresponding polynomial time algorithm for the first problem.

Second, he focused attention on the class NP of decision problems that can be solved in polynomial time by a non-deterministic computer. Most of the intractable problems belong to this class, NP.

Third, he proved that one particular problem in NP has the property that every other problem in NP can be polynomially reduced to it. If the satisfiability problem can be solved with a polynomial time algorithm, then every problem in NP can also be solved in polynomial time. If any problem in NP is intractable, then satisfiability problem must be intractable. Thus, satisfiability problem is the hardest problem in NP.

Fourth, Cook suggested that other problems in NP might share with the satisfiability problem this property of being the hardest member of NP.