ASSIGNMENT -

Ques 1 Discuss between BFS and DFS differences.

Breadth First Search (BFS) :-
→ BFS, is a vertex based technique for finding the shortest path in the graph.

→ It uses a Queue data structure that follows first in first out.

Depth First Search (DFS) :-
→ DFS is an edge-based technique.

→ It uses the Stack data structure & performs two stages, first visited vertices are pushed into stack, & second if there are no vertices then visited vertices are popped.

BFS	DFS
* BFS stands for Breadth First Search.	DFS stands for Depth First Search.
* BFS uses Queue data structure for finding the shortest path	DFS uses Stack data Structure.
* BFS builds the tree level by level	DFS builds the tree sub-tree by sub-tree.

* BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to next level	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes
* It works on the concept of FIFO	It works on the concept of LIFO
* BFS is more suitable for searching vertices closer to the given source	DFS is more suitable when there are solution away from source
* We don't need to backtrack in BFS	We need to follow a backtrack in DFS.
* The amount of memory required for BFS is more	the amount of memory required for DFS is less than BFS
* It is slower.	It is comparatively faster than BFS
* Examples of BFS are- Bipartite graph, shortest path, etc.	Examples of DFS are- acyclic graph, finding strongly connected component, etc.

Date:
P. No:

Q 2 Explain Backtracking & solve 4 Queen's problem and 8 Queen's problem using backtracking method.

BACKTRACKING :-

Backtracking is a problem - solving algorithm technique that involves finding a solution incrementally by trying different options and undoing them if they lead to a dead end.

→ Backtracking can be defined as a general algorithm technique that consider searching every possible combination in order to solve a computational problem.

Applications of Backtracking -

1) N- Queen's problem
2) Graph Coloring
3) Hamiltonian Cycle

4 Queen's problem -

Place each queen one by one in different rows, starting from the topmost row. while placing a queen in a row.
Check for clashes with already placed queens.
For any column, if there is no clash then mark this row & column as part of the sol^n by placing the queen.

Date:
P. No:

The 4 Queens problem consist for placing four queens on a 4x4 chessboard so that no two queens attack each other.

step 1:- Initialize a 4x4 board

4x4

step2:- Put first Queen (Q_1) in (0,0) cell
→ 'x' represent the cell which is not safe.
→ After this move to the next row [0 → 1].

	0	1	2	3
0	Q_1	x	x	x
1	x	x		
2	x		x	
3	x			x

step3 ⇒ Put next Q_2 in the (1,2) cell. After this move to the next. row [1 → 2].

	0	1	2	3
0	Q_1	x	x	x
1	x	x	Q_2	x
2	x	x	x	x
3	x		x	x

step4 → There is still a safe cell in the row 1 i.e. cell (1,3).
→ Put Q_2 at cell (1,3)

	0	1	2	3
0	Q_1	x	x	x
1	x	x	x	Q_2
2	x		x	x
3	x	x		x

steps:⇒ Put Q_3 at cell (2,1).

Step 6 ⇒ There is no any cell to place Queen Q_4 at row.
→ Backtrack & remove Q_3 from row 2
→ Again there is no other safe cell in row 2, so backtrack again & remove Q_2 from row 1
→ Q_1 will be remove from cell (0,0) & move to move to next safe cell i.e (0,1).

Step 7 ⇒ Place Queen Q_1 at cell (0,1) & move to next row.

	0	1	2	3
0	Q_1	x	x	x
1	x	x	x	Q_2
2	x	Q_3	x	x
3	x	x	x	x

Step 8 ⇒ Place Queen Q_2 at cell (1,3) and move to next row.

	0	1	2	3
0	x	Q_1	x	x
1		x	x	
2		x		x
3		x		

Step 9 ⇒ Place Q_3 at cell (2,0), and move to next row.

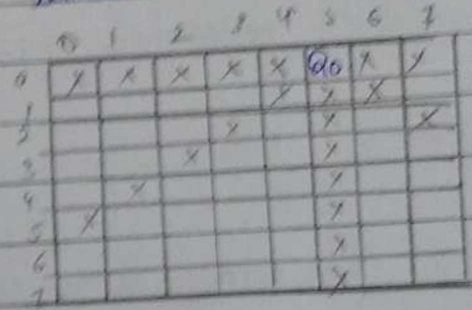
	0	1	2	3
0	x	Q_1	x	x
1	x	x	x	Q_2
2		x	x	x
3		x		x

Step 10 ⇒ Place Q_4 at cell (3,2) & move to next row.
This is one possible configuration of sol^n.

	0	1	2	3
0	x	Q_1	x	x
1	x	x	x	Q_2
2	Q_3	x	x	x
3	x	x	Q_4	x

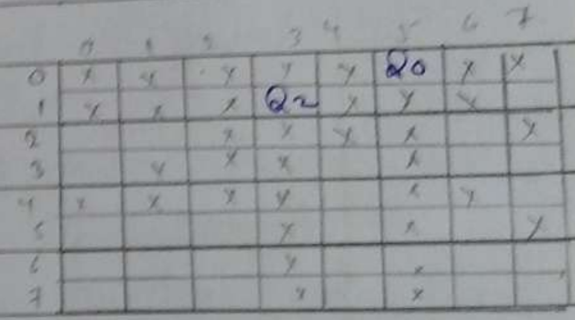
8 Queen's Problem :-

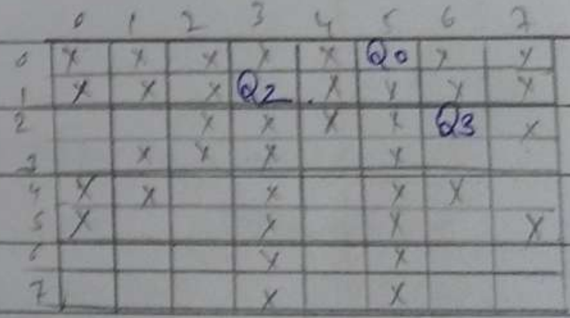
The 8 queen's problem is a classic chess puzzle where you aim to place eight queen on 8 x 8 chessboard in a way that no two queen threaten each other. This means no two queens share the same row, column or diagonal.

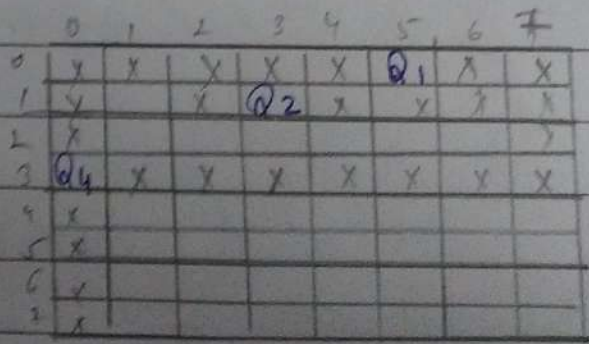


Step1 Initialize a 8 x 8 board.

Step2. Put first Queen (Q_0) in (0,5) cell.







Que 3. Explain Branch and bound technique for solving Travelling Salesman problem.

The branch & bound technique is a method used to solve combinational optimization problems like the Travelling Saleman Problem (TSP).

Here's how it works

1) Branching :- The problem space is divided into smaller subproblem, creating a tree-like structure. Each node represent a partial solⁿ.

2) Bounding :- At each node, a lower bound on the optimal solution is computed. This bound helps in pruning branches of the tree that can't lead to an optimal solⁿ, saving computation time.

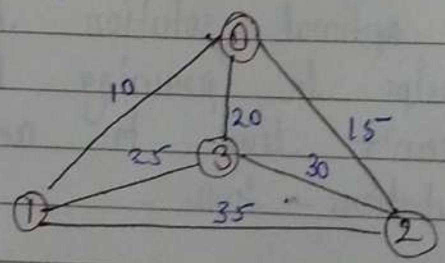
3) Searching :- The algorithm explores tree prioritizing nodes with promising lower bounds. It may use various strategies to decide which nodes to explore first, such as depth-first or best-first search.

4) Backtracking :- If a node cannot lead to a better solution than the current best solution found so far, it is discarded. the algo backtracks to the previous node & explore other branches.

(5) Termination :- The process continues until all branches have been explored or until a predefined termination condition is met, such as reaching a certain depth in the tree or exhaustively searching a certain number of nodes

Travelling Salesman Problem using Branch And Bound

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once & returns to the starting point



Example :-
The graph shown in fig. on right side.
A TSP tour in the graph is 0-1-3-2-0.

The cost of the tour is 10 + 25 + 30 + 15 which is 80.

Q4. What is graph colouring problem? Explain in detail with the help of an example.

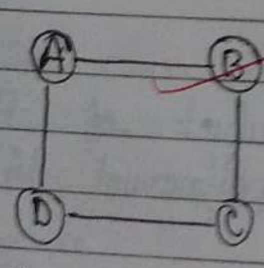
Ans GRAPH COLORING :-

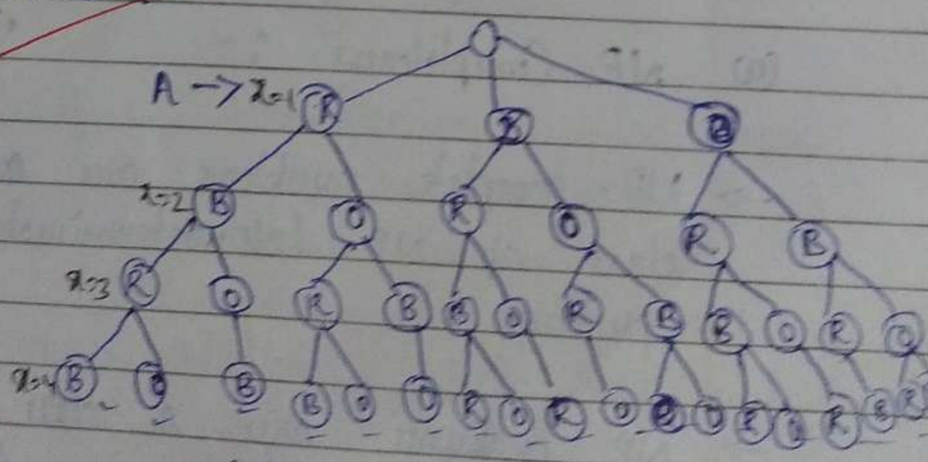
→ Let G be a graph and m be a given positive integer.

→ We want to discover whether the nodes of G can be colored in such a way that no two neighbouring vertex are of same color, yet only m colors are used.
This is termed as m-colorability decision problem.

→ The m-colorability optimization problem asks for the smallest integer m for which the graph G can be colored

Chromatic Number :- The minimum no. of colors needed to color a graph is called its chromatic number





State Space tree

→ Function mColoring is begun by first assigning the graph to its adjacency matrix, setting the array x[ ] to zero, & then invoking the statement mColoring [1];

Analysis :-
An upper bound on the computing time of mColoring can be arrived at by noticing that the no. of internal nodes in the state space tree is $\sum_{i=0}^{n-1} m^i$.

At each internal node, O(mn) time is spent by NextValue to determine the children corresponding to legal colourings.
Hence the total time is bounded by

$$\sum_{i=0}^{n-1} m^{i+1} n = \sum_{i=1}^{n} m^i n = n(m^{n+1}-2)/(m-1) = O(nm^n).$$

Que 5. Write a short note on :-

(a) NP- Completeness :-

→ NP- complete problems are a subset of the larger class of NP [Nondeterministic polynomial time] problem.

→ NP problems are a class of computational problem that can be solved in polynomial time by a non-deterministic machine & can be verified in polynomial time by a deterministic Machine.

→ A problem L in NP is NP-complete if all other problems in NP can be reduced to L in polynomial time.

→ A decision problem L is NP-complete if it follow the below two properties :-

- L is in NP (Any solⁿ to NP-complete problems can be checked quickly but no efficient solⁿ is known)
- Every problem in NP is reducible to L in polynomial time (Reduction is defined blew)

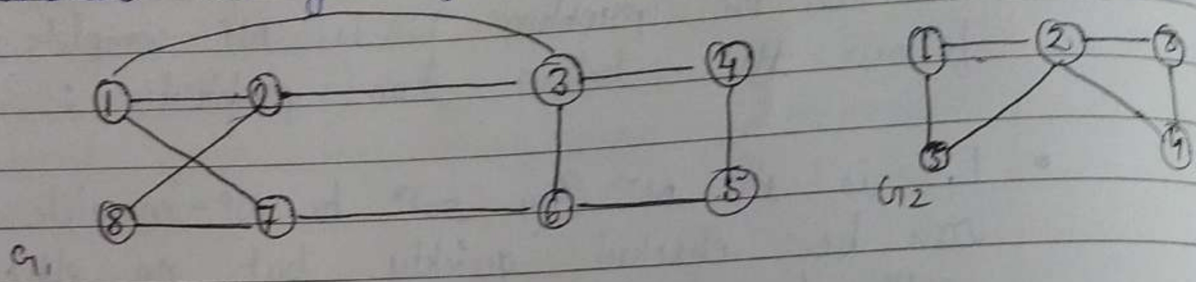
(b) Hamiltonian Cycle :-

Let $G = (V, E)$ be a connected graph with n vertices. A Hamiltonian cycle is a round-trip along a n edges of G that visits every vertex once & return to its starting position.

In other words, if a Hamiltonian cycle begins at some vertex $v_1 \in G$ & the vertices of G are visited in the order $v_1, v_2 \ldots v_{n+1}$, then the edge (v_i, v_{i+1}) are in E, $1 \leq i \leq n$ and the v_i are distinct except for v_1 & v_{n+1}, which are equal.

The graph G_1 of fig contains the Hamiltonian cycle 1, 2, 8, 7, 6, 5, 4, 3, 1.
The graph G_2 contains no Hamiltonian cycle.

There is no known easy way to determine whether a give graph contain a Hamiltonian cycle



→ The backtracking solⁿ vector $(x_1 \ldots x_n)$ is defined x_i i^{th} visited vertex of proposed cycle.

→ By using backtracking we need to determine how to compute the set of possible vertice for x_k if $x_1, x_2, x_3 \ldots x_{k-1}$ have already been chosen

→ If $k=1$ then x_1 can be any of the n vertices

By using "NextValue" algorithm the recursive backtracking scheme to find all Hamiltonian cycles.

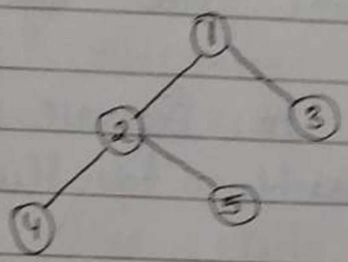
→ The time complexity is given by

$$T(N) = N^* (T(N-1) + O(1))$$

$$T(N) = N^* (N-1)^* (N-2) \ldots = O(N!)$$

(c) Tree Traversal Technique :-

Tree traversal is a form of graph traversal & refers to the process of visiting each node in a tree data structure, exactly once. Such traversals are classified by the order in which the nodes are visited.



Depth First Traversal :-

(a) Inorder (Left, Root, Right) : 4 2 5 1 3.

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First Traversal :- 1 2 3 4 5

DFS :- The aim of DFS algo is to traverse the graph in such a way that it tries to go far from the root node.

→ Stack is used in the implementation of the depth first search.

Date:
P. No:

step 1:- Push the root node in the stack.
step 2:- Loop until stack is empty.
step 3:- Peek the node of the stack.
step 4:- If the node has unvisited child node, get the unvisited child node, mark it as traversal and push it on stack.
step 5:- If the node does not have any unvisited child nodes, the node from the stack.

BFS :- This aims to traverse the graph as close as possible to the root node.

→ Queue is used in the implementation of BFS

step 1:- Push the root node in the Queue.
2:- Loop until the queue is empty.
3:- Remove the node from the Queue.
4:- If the removed node has unvisited child node, mark them as visited & inserted the unvisited children in the queue.

(d) B-Tree :-

- B-tree is a self-balancing search tree.
- The main idea of using B-Trees is to reduce the no. of disk access.
- Most of the tree operation (search, insert, delete, max, min... etc) require O(h) disk accesses where h is height of the tree.
- Height of B-Trees is kept low by putting max. possible keys in a B-Tree node.

Properties of B-Tree

(1) All leaves are at same level.
(2) A B-Tree is defined by the term minimum degree 't'.
(3) Every node except root must contain at least t-1 keys. Root may contain min 1 key.
(4) All nodes may contain at most 2t-1 keys.
(5) No. of children of a node is equal to the no. of keys in it plus 1.
(6) All keys of a node are sorted in increasing order. The child b/w two keys k_1 & k_2 contains all keys in range from k_1 & k_2.
(7) B-tree grows & shrinks from root which is unlike Binary Search tree.
(8) Time complexity to search, insert and delete is O(log n).

(e) **Height Balanced Tree:**

AVL tree is a self-balancing Binary Search Tree (BST) where the difference b/w height of left & right subtrees cannot be more than one for all nodes.

→ The height of an AVL tree is always O(log n).

→ Most of the BST operation take O(h) time where h is height of the BST.

→ An AVL tree is a balanced binary search tree.

→ In an AVL tree, balance factor of every node is either -1, 0 or +1

Balance factor = height of Left Subtree - Height of Right ST

AVL Tree Rotation :- Rotation is the process of moving the nodes to either left or right to make tree balanced.

There are four rotation & they are classified into 2 types

① Single Rotation
- Left rotation
- Right rotation

② Double Rotation
- Left - Right rotation
- Right - Left rotation

Qu 6 What are 2-3 Trees used for? Why are 2-3 trees better than BST? limitation.

→ A 2-3 tree is a specific form of a B-tree

→ A 2-3 tree is a search tree.

Properties :-
- Each node has either one value or two value
- a node with one value is either a ~~left~~ leaf node or has exactly two children.

Date:
P. No:

- a node with two values is either a leaf node or has exactly three children.

Insertion algorithm :-

Into a two - three tree is quite different from the Insertion algorithm into a binary search tree.

- If the tree is empty, create a node & put value into the node.
- Otherwise find the leaf node where the value belong.
- If the leaf node has only one value, put the new value into the node.
- If the leaf node has more than two values, split the node & promote the median of the three value to parent.
- If the parent then has three values, continue to split & promote, forming a new root node if necessary.

Delete Operation :-

Deleting key k is similar to inserting, there is a special case when T is just a single node containing k, otherwise, the parent of the node to be deleted is found, then the tree is fixed up if necessary so that it is still a 2-3 trees.

Complexity Analysis :-

- Keys are stored only at leaves, ordered left - to right.
- non-leaf nodes have 2 or 3 children.
- non - leaf nodes also have leftMax & MiddleMax values.
- All leaves are at the same depth.
- the height of the tree is O (log N).
- at least half the nodes are leaves, so the height of the tree is also O(log M) for M = # values stored in tree.
- the lookup, Insert & delete methods can all be implemented to run in time O(logN)

10/5/24