UNIT-2

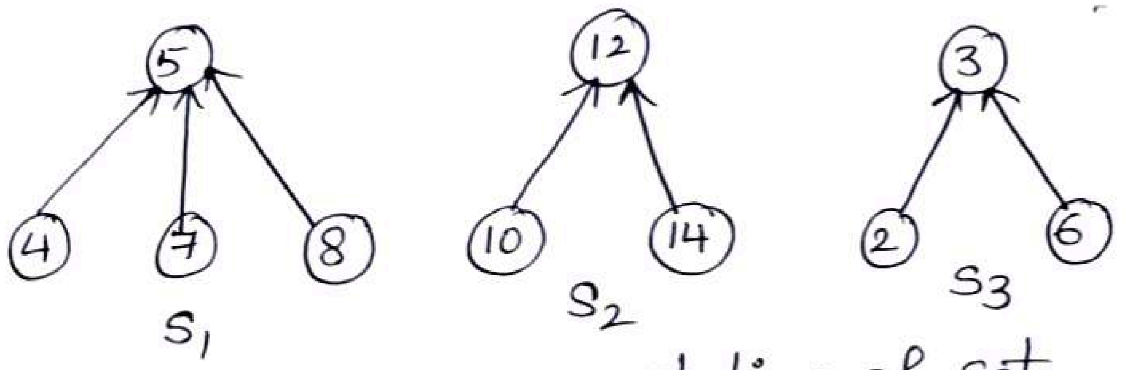
- ⊛ Disjoint sets : Disjoint set operations, union and find algorithms.
- ⊛ Backtracking : General Method, applications, n-queen's problem, sum of subsets problem, graph coloring, Hamiltonian circuits.

Disjoint Sets:-

A disjoint set is a kind of data structure that contains partitioned sets. These partitioned sets are separate non overlapping sets.

For example: If there are $n=10$ elements that can be partitioned into three disjoint sets. S_1, S_2 and S_3 such that no element is common among these sets.

Let, $S_1 = \{5,4,7,9\}, S_2 = \{10,12,14\}, S_3 = \{2,3,6\}$ that each set can be presented as a tree.



Tree representation of sets.

There are two operations that can be performed on the data structure: disjoint set.

These operations are <u>union</u> and <u>find</u>

① Disjoint set union: If there exists two sets S_i and S_j then $S_i \cup S_j = \{$all the elements from set S_i and $S_j\}$. In above example

$S_1 \cup S_2 = \{5,4,7,9,10,12,14\}$.

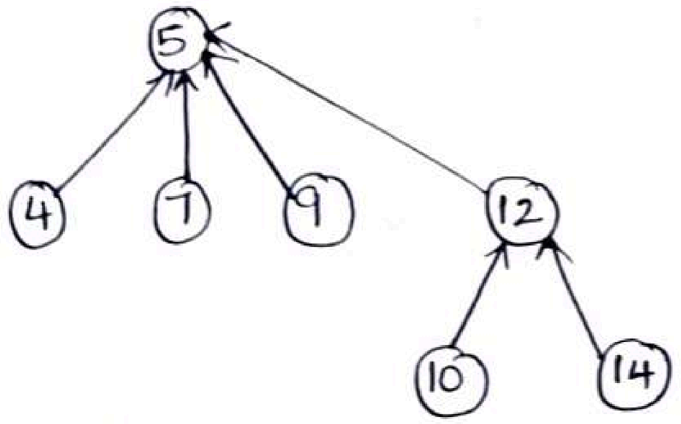
② Find(i): For finding out the element i from given set, is done by this operation.

For example: Element 7 is in S_1, element 14 is in S_2.

Union and Find Operations :-

The Union operation combines the elements from two sets.

Consider $S_1 = \{5,4,7,9\}$ and $S_2 = \{10,12,14\}$ then $S_1 \cup S_2$ is



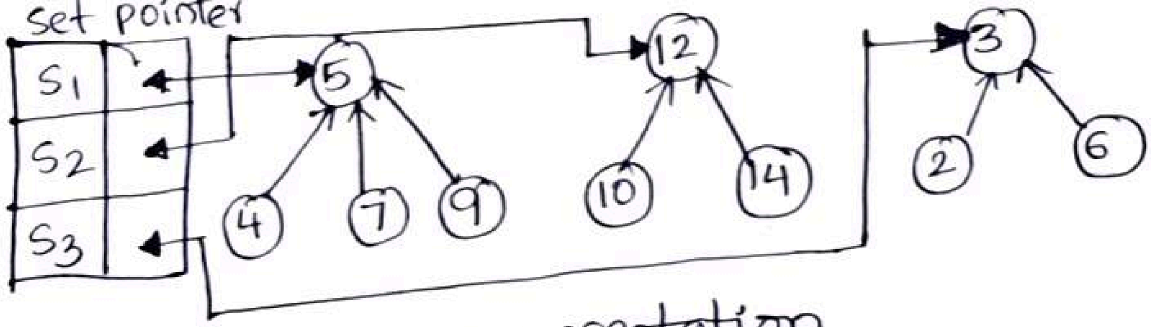
$S_1 \cup S_2$

Union operation

Data Representation of Sets

To perform union or find operations efficiently on sets, it is necessary to represent set elements in a proper manner.

Consider $S_1 = \{5,4,7,9\}$, $S_2 = \{10,12,14\}$ and $S_3 = \{2,3,6\}$



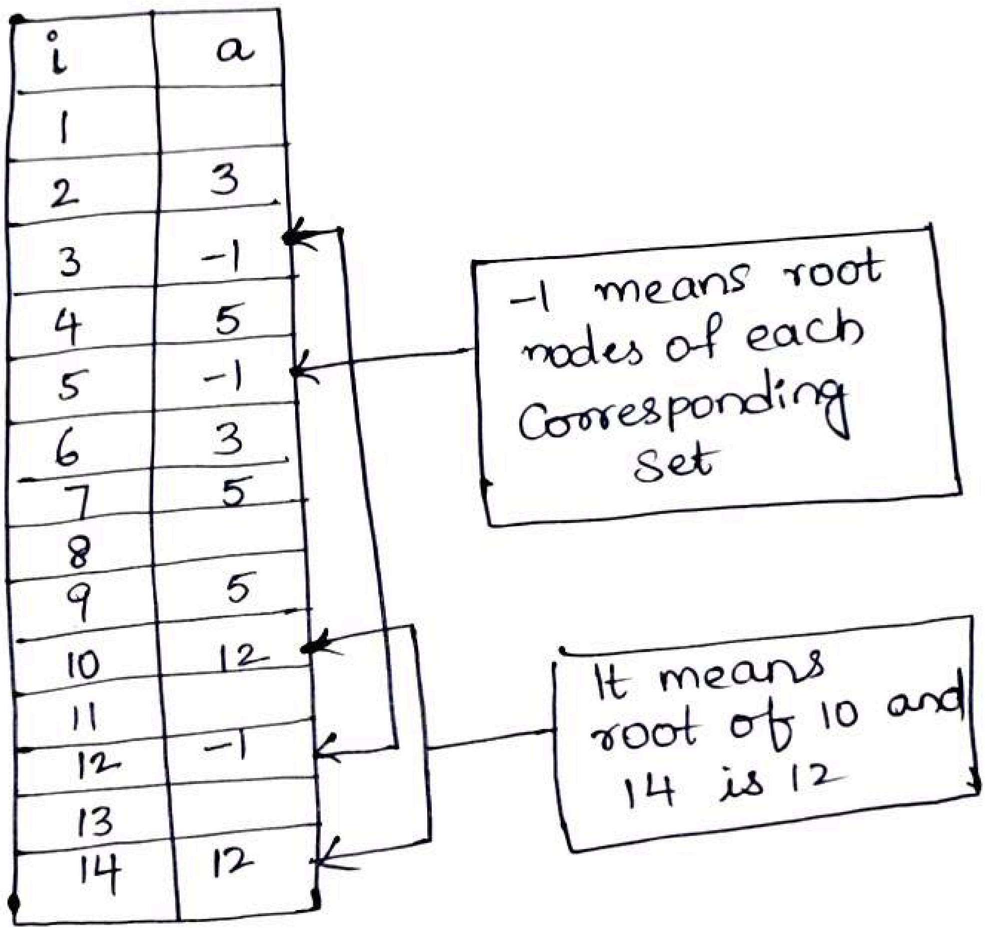
Data representation.

For finding out any desired element,

i. Find the pointer of the root node of Correspondin
Set.

ii. Then find the desired element from that set.

The sets can be represented using arrays as follows.



i	a
1	
2	3
3	-1
4	5
5	-1
6	3
7	5
8	
9	5
10	12
11	
12	-1
13	
14	12

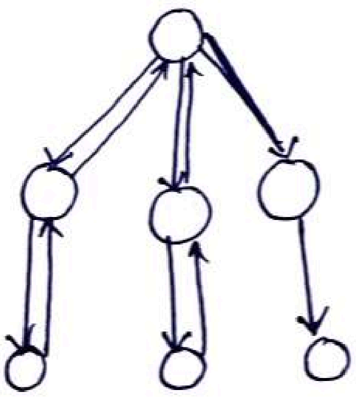
-1 means root nodes of each Corresponding Set
It means root of 10 and 14 is 12

⊛ Backtracking (General Method)

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

⊛ It is Commonly used in situations where we need to explore multiple possibilities to solve a problem, like searching for a path in a maze or solving puzzles like Sudoku.

⊛ when a deadend is reached, the algorithm back-tracks to the previous decision point and explores a different path until a solution is found or all possibilities have been exhausted.



Types of backtracking problems

⊛ problems associated with backtracking can be categorized into 3 categories:

① Decision problems: Here, we search for a feasible solution

② optimization problems: For this type, we search for the best solution.

③ Enumeration problems: We find set of all possible feasible solutions to the problems of this type.

Pseudocode for Backtracking :-

The bestway to implement backtracking is thro
recursion and all backtracking code can be

```
Void FIND_SOLUTIONS(parameters):
 if(valid Solution):
    slore the solution
 Return
    for(all choice):
      if(valid choice):
        APPLY (choice)
 FIND_SOLUTIONS(parameters)
    BACKTRACK (remove choice)
 Return.
```

Applications :-

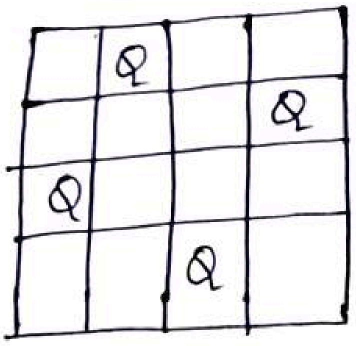
- n-queen's problem
- Sum of subsets problem
- Graph coloring
- Hamiltonian circuits

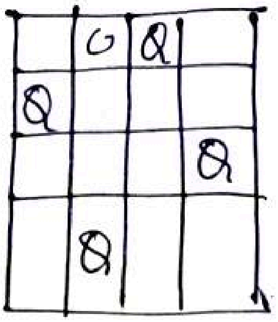
⊛ n-queen's problem:-

→ It is solved by using backtracking algorithm. Given an integer n, the task is to find all the distinct solutions to the n-queen's problem, where n queens are placed on an n*n chessboard such that no two queens can attack each other.

Note:- Each solution is a unique configurations of n queens, represented as a permutation of [1,2,3,...n]. The number at the ith position indicates the row of the queen in the ith column. For example, [3,1,4,2] shows one such Layout.

Input : n=4
o/p : [2,4,1,3], [3,1,4,2]





Time Complexity: O(n!), space complexity : O(n)

step by step implementation:

⊛ Start from the first column and try placing a queen in each row.

⊛ Keep arrays to track which rows are already occupied. Similarly, for tracking major and minor diagonals are already occupied.

⊛ If a queen placement conflicts with existing queens, skip that row and backtrack the queen to try the next possible row

(*) Sum of Subsets problem:-

The sum of subsets problem is a classic problem in backtracking, where the goal is to find all subsets of a given set that sum up to a specific target value.

Set : In mathematical terms, a set is defined as a collection of similar types of objects. The entities or objects of a set must be related to each other through the same rule.

Subset : Suppose there are two sets namely set P and set Q. The set P is said to be a subset of set Q, only if all the elements of set P also belong to the set Q and viceversa need not be true.

Input Output scenario

Suppose the given set and sum value is

Set = {1, 9, 7, 5, 18, 12, 20, 15}

Sum value = 35

Subset {1, 9, 7, 18}, {1, 9, 5, 20}, {5, 18, 12}

Steps to solve subset sum problem using the backtracking approach.

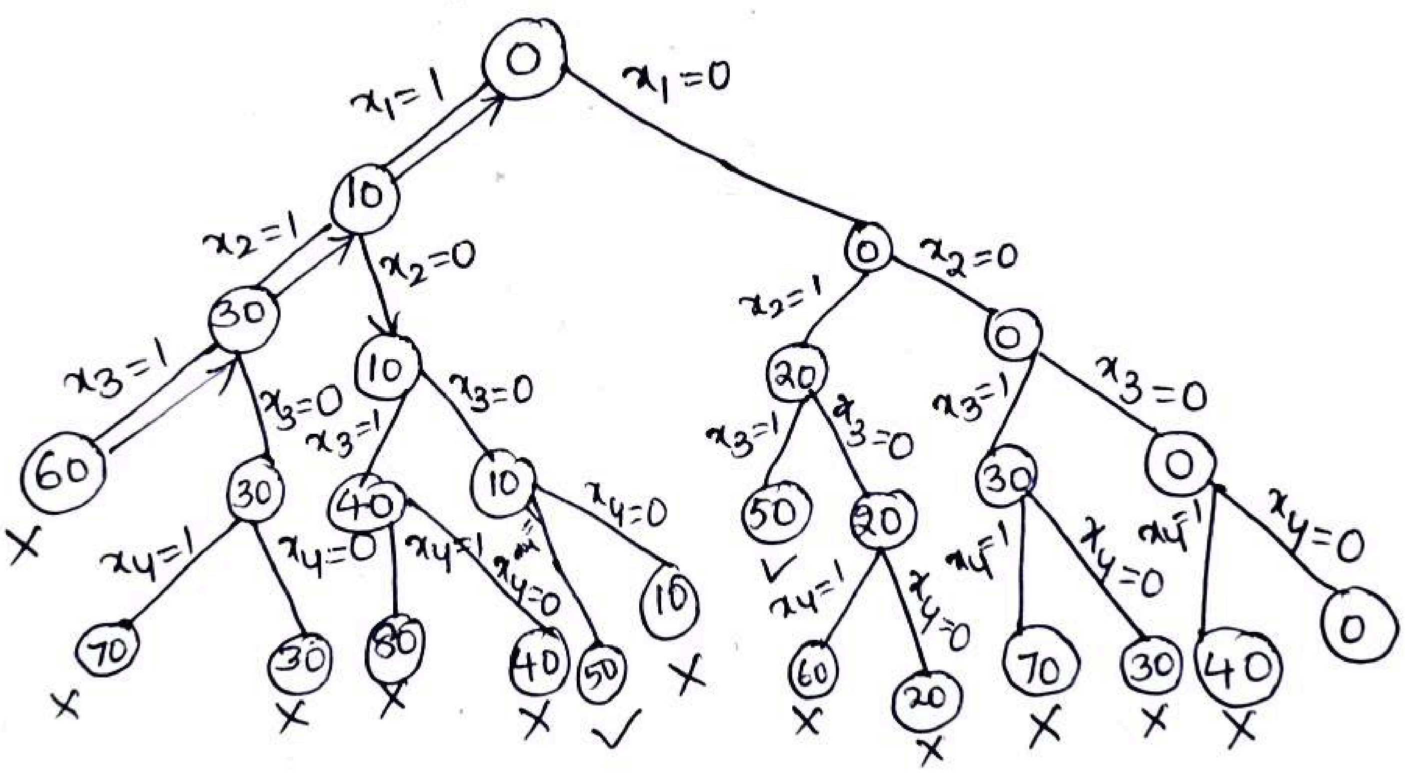
- (*) First, take an empty subset
- (*) Include the next element, which is at index 0 to the empty set.
- (*) If the subset is equal to the sum value, mark it as a part of the solution.
- (*) If the subset is not a solution and it is less than the sum value, add next element to the subset until a valid solution is found

(*) Now, move to the next element in the set and Check for another Solution until all combinations have been tried.

State space tree:-
A State space tree is a tree representation of all possible states (partial Solutions) explored by the backtracking algorithm.

→ Each node represents a decision point.

Example:- Let us consider a set $W = \{10, 20, 30, 40\}$ with x_1, x_2, x_3, x_4 and the target sum $M = 50$

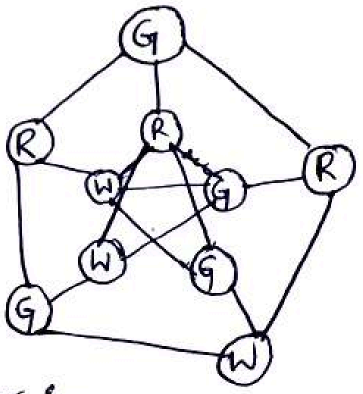


So, In the above example the possible sets for target Sum = 50 is $\{10, 40\}, \{20, 30\}$.

(*) Time Complexity:
Worst case: $O(2^n)$
Space Complexity: $O(n)$

⊛ Graph Coloring :-

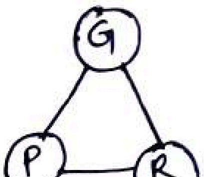
Graph coloring refers to the problem of Coloringvertices of a graph in such a way that no adjacent vertices have the same color. This is also called the vertex coloring problem. If Coloring is done using at most m colors, it is called m-Coloring.



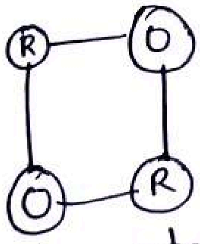
R: Red
G: Green
W: white
P: pink
O: orange
V: violet

Chromatic Number :-

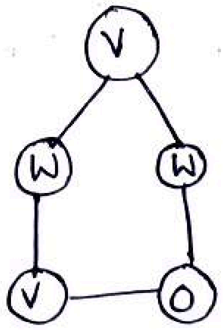
The minimum number of colors needed to color a graph is called its chromatic number. The following can be colored a minimum of two colors.



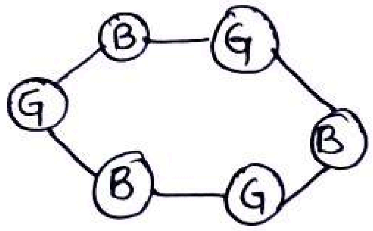
chromatic Number: 3



Chromatic Number = 2



CN = 3



Example of chromatic Numbers in a Cyclic graph.

Algorithm of Graph coloring using backtracking:-

Assign colors one by one to different vertices, Starting from vertex 0. Before assigning a color, check if the adjacent vertices have the Same color or not. If there is any color assignment that does not violate the conditions, mark the color assignment as part of the Solution. If no assignment of color is possible then backtrack and return false.

Applications of Graph Coloring:-

- Design a time table
- Sudoku
- Register allocation in the compiler
- Map Coloring
- Mobile radio frequency assignment

⊛ Hamiltonian Cycle or Circuit :-

Hamiltonian cycle or circuit in a graph G is a cycle that visits every vertex of G exactly once and returns to the starting vertex.

→ If a graph contains a Hamiltonian cycle, it is called Hamiltonian graph othercoise it is non-Hamiltonian.

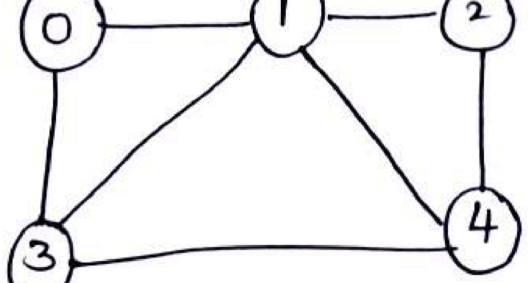
→ The Hamiltonian cycle problem has practical applications in various fields, such as logistics, network design, and computer science.

Hamiltonian path :-

Hamiltonian path in a Gragh G is a path that visits every vertex of G exactly once and its doesn't have to return to the starting vertex. It's an open path.

→ Hamiltonian paths have applications in various fields, such as finding optimal routes in transportation networks, circuit design, and graph theory research.

Ex :- Input graph[][] = [[0, ~~1, 0, 1~~, 0], [1, 0, 1, 1, 1], [0, 1, 0, 0, 1], [1, 1, 0, 0, 1], [0, 1, 1, 1, 0]]



output : 0 1 2 4 3 0

- (*) Start with the node 0.
- (*) Apply Depth First Search for finding the Hamiltonian path.
- (*) When base case reach (i.e., total no of node traversed == n (total vertex)):

Check whether current node is a neighbour of starting node.
As node 2 and node 0 are not neighbours of each other so return from it.
Like wise we need to check all the nodes and whether it satisfyies the conditions.

Time Complexity: O(n!), backtracking tries all Permutations of n vertices.
Each recursive call explores (n-1).
Space Complexity: O(n), path array takes O(n) space and recursion stack also takes O(n).