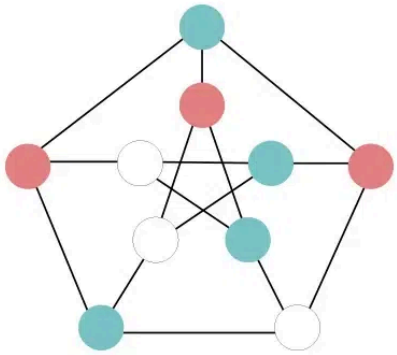
## UNIT V – BRANCH AND BOUND AND BACKTRACKING

Backtracking: N-Queens problem - Hamiltonian cycles – Graph coloring – Sum of subset. Branch and bound: The method – FIFO branch and bound- LC branch and bound – 0/1 Knapsack problem - Traveling salesman problem.

---

## GRAPH COLORING

Given an undirected graph represented by an adjacency matrix. The graph has n nodes, labeled from 1 to n. The task is to assign colors to each node in such a way that no two adjacent nodes have the same color. The challenge is to solve this problem using the minimum number of colors.



**Step-by-step algorithm:**

- Create a recursive function that takes the graph, current index, number of vertices, and color array.
- If the current index is equal to the number of vertices. Print the color configuration in the color array.
- Assign a color to a vertex from the range **(1 to m)**.
    - For every assigned color, check if the configuration is safe, (i.e. check if the adjacent vertices do not have the same color) and recursively call the function with the next index and number of vertices else return **false**
    - If any recursive function returns **true** then break the loop and return true
    - If no recursive function returns **true** then return **false**

---

## SUM OF SUBSET

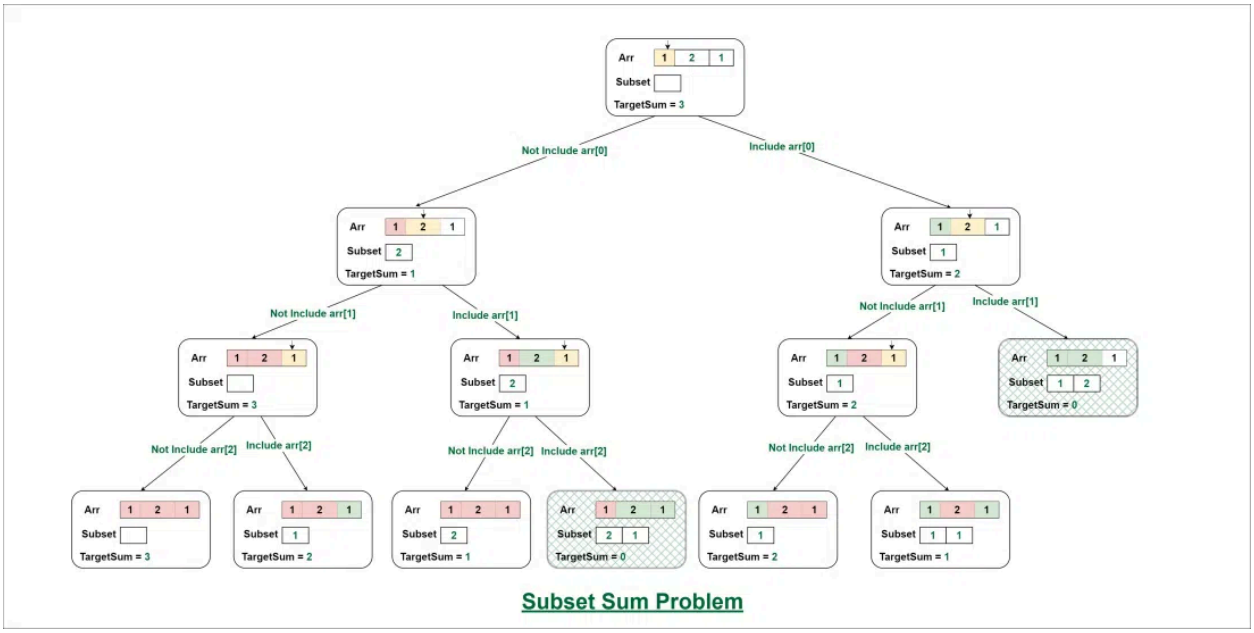
Given a set[] of non-negative integers and a value sum, the task is to print the subset of the given set whose sum is equal to the given sum.

Subset sum can also be thought of as a special case of the 0–1 Knapsack problem. For each item, there are two possibilities:

Include the current element in the subset and recur for the remaining elements with the remaining Sum.

Exclude the current element from the subset and recur for the remaining elements.

Finally, if Sum becomes 0 then print the elements of the current subset. The recursion's base case would be when no items are left, or the sum becomes negative, then simply return.




Subset Sum Problem

**Backtracking Approach to solve Subset Sum Problem**

In the naive method to solve a subset sum problem, the algorithm generates all the possible permutations and then checks for a valid solution one by one. Whenever a solution satisfies the constraints, mark it as a part of the solution.

In solving the subset sum problem, the backtracking approach is used for selecting a valid subset. When an item is not valid, we will backtrack to get the previous subset and add another element to get the solution.

In the worst-case scenario, the backtracking approach may generate all combinations, however, in general, it performs better than the naive approach.

Follow the below 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.

**Complexity analysis:**

- **Time Complexity:** O(2n) The above solution may try all subsets of the given set in the worst case. Therefore the time complexity of the above solution is exponential.
- **Auxiliary Space:** O(n) where n is recursion stack space.

---