Dynamic Programming

1. **Longest Common Subsequence Problem**

The Longest Common Subsequence (LCS) problem is finding the longest subsequence present in given two sequences in the same order, i.e., find the longest sequence which can be obtained from the first original sequence by deleting some items and from the second original sequence by deleting other items.

The problem differs from the problem of finding the longest common substring. Unlike substrings, subsequences are not required to occupy consecutive positions within the original string.

For example, consider the two following sequences, `X` and `Y`:

X: ABCBDAB
Y: BDCABA

The length of the LCS is 4
LCS are BDAB, BCAB, and BCBA

2. **Longest Common Substring Problem**

The longest common substring problem is the problem of finding the longest string (or strings) that is a substring (or are substrings) of two strings.

The problem differs from the problem of finding the Longest Common Subsequence (LCS). Unlike subsequences, substrings are required to occupy consecutive positions within the original string.

For example, the longest common substring of strings `ABABC`, `BABCA` is the string `BABC` having length 4. Other common substrings are `ABC`, `A`, `AB`, `B`, `BA`, `BC`, and `C`.

3. **Matrix Chain Multiplication using Dynamic Programming**

   (10x100),(100x5),(5x500) minimum cost is 5000

4. **0–1 Knapsack problem**

   Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
   Output: 3

5. Subset Sum Problem – Dynamic Programming Solution

Given a set of positive integers and an integer `k`, check if there is any non-empty subset that sums to `k`.

For example,

Input:

A = { 7, 3, 2, 5, 8 }
k = 14

Output: Subset with the given sum exists

Subset { 7, 2, 5 } sums to 14

6. Rod Cutting Problem

Given a rod of length `n` and a list of rod prices of length `i`, where `1 <= i <= n`, find the optimal way to cut the rod into smaller rods to maximize profit.

For example, consider the following rod lengths and values:

Input:

length[] = [1, 2, 3, 4, 5, 6, 7, 8]
price[] = [1, 5, 8, 9, 10, 17, 17, 20]

Rod length: 4

Best: Cut the rod into two pieces of length 2 each to gain revenue of 5 + 5 = 10

7. Create optimal cost to construct a binary search tree

If the keys are 10, 20, 30, 40, 50, 60, 70 Answer: 26 is the minimum cost

8. Find the maximum sum of a subsequence with no adjacent elements.

Given an integer array, find the maximum sum of subsequence where the subsequence contains no element at adjacent positions.

Please note that the problem specifically targets subsequences that need not be contiguous, i.e., subsequences are not required to occupy consecutive positions within the original sequences.

For example,

Input: { 1, 2, 9, 4, 5, 0, 4, 11, 6 }
Output: The maximum sum is 26

The maximum sum is formed by subsequence { 1, 9, 5, 11 }

DIVIDE and CONQUER

1. **Inversion count of an array**

Given an array, find the total number of inversions of it. If `(i < j)` and `(A[i] > A[j])`, then pair `(i, j)` is called an inversion of an array `A`. We need to count all such pairs in the array.

For example,

Input: A[] = [1, 9, 6, 4, 5]

Output: The inversion count is 5

There are 5 inversions in the array: (9, 6), (9, 4), (9, 5), (6, 4), (6, 5)

2. **Find the first or last occurrence of a given number in a sorted array.**

Given a sorted integer array, find the index of a given number's first or last occurrence. If the element is not present in the array, report that as well.

For example,

Input:

nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]
target = 5

Output:

The first occurrence of element 5 is located at index 1
The last occurrence of element 5 is located at index 3

Input:

nums = [2, 5, 5, 5, 6, 6, 8, 9, 9, 9]
target = 4

Output:

Element not found in the array

3. **Find the smallest missing element from a sorted array**

Given a sorted array of non-negative distinct integers, find the smallest missing non-negative element in it.

For example,

Input: nums[] = [0, 1, 2, 6, 9, 11, 15]
Output: The smallest missing element is 3

Input: nums[] = [1, 2, 3, 4, 6, 9, 11, 15]
Output: The smallest missing element is 0

Input: nums[] = [0, 1, 2, 3, 4, 5, 6]
Output: The smallest missing element is 7

4. **Find the number of 1's in a sorted binary array**

Given a sorted binary array, efficiently count the total number of 1's in it.

For example,

Input: nums[] = [0, 0, 0, 0, 1, 1, 1]

Output: The total number of 1's present is 3

Input: nums[] = [0, 0, 1, 1, 1, 1, 1]

Output: The total number of 1's present is 5

5. **Find pairs with difference `k` in an array | Constant Space Solution**

Given an unsorted integer array, find all pairs with a given difference `k` in it without using any extra space.

For example,

Input:

arr = [1, 5, 2, 2, 2, 5, 5, 4]
k = 3

Output:

(2, 5) and (1, 4)

6. **Find `k` closest elements to a given value in an array**

Given a sorted integer array, find the `k` closest elements to `target` in the array where `k` and `target` are given positive integers.

The `target` may or may not be present in the input array. If `target` is less than or equal to the first element in the input array, return first `k` elements. Similarly, if `target` is more than or equal to the last element in the input array, return the last `k` elements. The returned elements should be in the same order as present in the input array.

For example,

Input: [10, 12, 15, 17, 18, 20, 25], k = 4, target = 16
Output: [12, 15, 17, 18]

Input: [2, 3, 4, 5, 6, 7], k = 3, target = 1
Output: [2, 3, 4]

Input: [2, 3, 4, 5, 6, 7], k = 2, target = 8
Output: [6, 7]

7. **Longest Common Prefix (LCP) Problem**

Write an efficient algorithm to find the longest common prefix (LCP) between a given set of strings.

For example,

Input: technique, technician, technology, technical
Output: The longest common prefix is techn

Input: techie delight, tech, techie, technology, technical
Output: The longest common prefix is tech

8. **Find the frequency of each element in a sorted array containing duplicates.**

Given a sorted array containing duplicates, efficiently find each element's frequency without traversing the whole array.

For example,

Input: [2, 2, 2, 4, 4, 4, 5, 5, 6, 8, 8, 9]
Output: {2: 3, 4: 3, 5: 2, 6: 1, 8: 2, 9: 1}

Explanation:

2 and 4 occurs thrice
5 and 8 occurs twice
6 and 9 occurs once

9. **Maximum Subarray Sum using Divide and Conquer**

Given an integer array, find the maximum sum among all subarrays possible.

The problem differs from the problem of finding the maximum subsequence sum. Unlike subsequences, subarrays are required to occupy consecutive positions within the original array.

For example,

Input: nums[] = [2, -4, **1, 9, -6, 7**, -3]

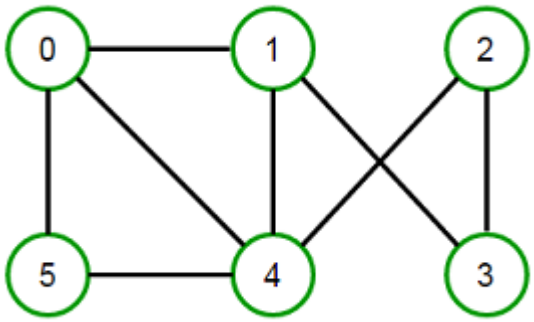
Output: The maximum sum of the subarray is 11 (Marked in **Green**)

GREEDY Algorithms

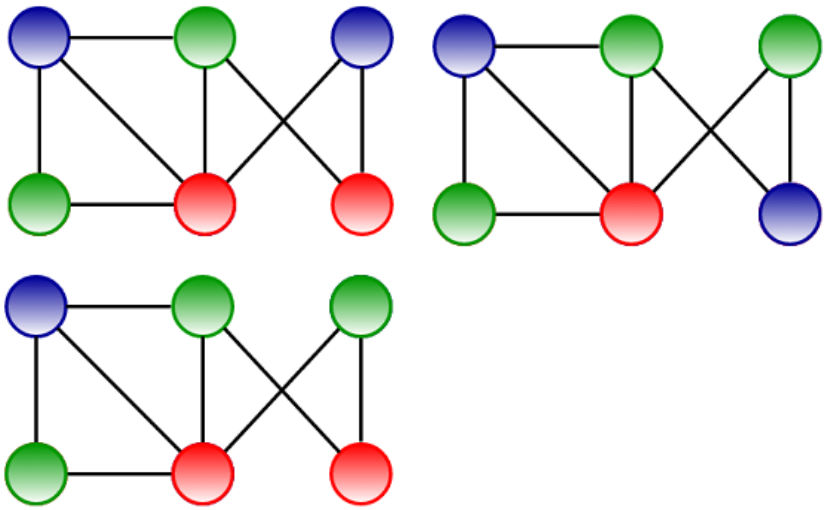
1. **Graph Coloring Problem**

Graph coloring (also called vertex coloring) is a way of coloring a graph's vertices such that no two adjacent vertices share the same color. Implement a greedy algorithm for graph coloring and minimize the total number of colors used.

For example, consider the following graph:



We can color it in many ways by using the minimum of 3 colors.



2. Huffman Coding Compression Algorithm

Huffman coding (also known as Huffman Encoding) is an algorithm for doing data compression, and it forms the basic idea behind file compression. Implement the variable-length encoding, uniquely decodable codes, prefix rules, and Huffman Tree construction.

3. Kruskal's and Prim's Algorithm for finding Minimum Spanning Tree

Given a connected and weighted undirected graph, construct a minimum spanning tree out of it using Kruskal's and Prim's Algorithm.