16) Subset Sum Problem

06 December 2024 16:04

Given an array of positive integers, **arr[]** and a value, **target**, determine if there is a subset of the given set with sum equal to given target.

From <https://www.geeksforgeeks.org/problems/subset-sum-problem-1611555638/1>

```
bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
    if(target == 0) return true;
    if(index == 0) return (arr[0] == target);
    if(dp[index][target] != -1) return dp[index][target];

    bool notTaken = helper(index-1, arr, target, dp);
    bool taken = false;
    if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index], dp);

    return dp[index][target] = notTaken || taken;
}
bool isSubsetSum(vector<int>& arr, int target) {
    int n = arr.size();
    vector<vector<int>>dp(n, vector<int>(target+1, -1));
    return helper(n-1, arr, target, dp);
}
```

17) Partition Equal Subset Sum (based on q no 16)

06 December 2024 16:16

Given an integer array nums, return true *if you can partition the array into two subsets such that the sum of the elements in both subsets is equal or* false *otherwise*.

From <https://leetcode.com/problems/partition-equal-subset-sum/description/>

```
    bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
        if(target == 0) return true;
        if(index == 0) return (arr[0] == target);
        if(dp[index][target] != -1) return dp[index][target];

        bool notTaken = helper(index-1, arr, target, dp);
        bool taken = false;
        if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index],
dp);

        return dp[index][target] = notTaken || taken;
    }
    bool isSubsetSum(vector<int>& arr, int target) {
        int n = arr.size();
        vector<vector<int>>dp(n, vector<int>(target+1, -1));
        return helper(n-1, arr, target, dp);
    }
    bool canPartition(vector<int>& nums) {
        int totalSum = accumulate(nums.begin(), nums.end(), 0);
        if(totalSum % 2 != 0) return false;
        int target = totalSum/2;
        return isSubsetSum(nums, target);
    }
```

18) Minimum sum partition(based on q 16)

06 December 2024 18:33

Given an array **arr[]** containing **non-negative** integers, the task is to divide it into two sets **set1** and **set2** such that the absolute difference between their sums is minimum and find the **minimum** difference.

From <https://www.geeksforgeeks.org/problems/minimum-sum-partition3317/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
bool helper(int index, vector<int>& arr, int target, vector<vector<int>> &dp){
    if(target == 0) return true;
    if(index == 0) return (arr[0] == target);
    if(dp[index][target] != -1) return dp[index][target];

    bool notTaken = helper(index-1, arr, target, dp);
    bool taken = false;
    if(arr[index] <= target) taken = helper(index-1, arr, target-arr[index], dp);

    return dp[index][target] = notTaken || taken;
}

int minDifference(vector<int>& arr) {
    int n = arr.size();
    int totSum = accumulate(arr.begin(), arr.end(), 0);
    int halfSum = totSum / 2;

    vector<vector<int>> dp(n, vector<int>(halfSum + 1, -1));

    int mini = 1e9;
    for (int s1 = halfSum; s1 >= 0; s1--) {
        int s2 = totSum-s1;
        if (helper(n - 1, arr, s1, dp)) {
            mini = min(mini, abs(s1-s2));
        }
    }

    return mini;
}
```

19) Perfect Sum Problem (same as q16, just the base case changes)

06 December 2024 19:24

Given an array **arr** of non-negative integers and an integer **target**, the task is to count all subsets of the array whose sum is equal to the given target.

From <https://www.geeksforgeeks.org/problems/perfect-sum-problem5633/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=perfect-sum-problem>

```
int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp) {
    // Base case: If the target sum is 0, we found a valid subset
    if (ind == 0){
        if (target == 0 && arr[0] == 0) return 2;
        if (target == 0 || target == arr[0])return 1;
        return 0;
    }

    if (dp[ind][target] != -1) return dp[ind][target];

    int notTaken = findWaysUtil(ind - 1, target, arr, dp);

    int taken = 0;
    if (arr[ind] <= target)
        taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

    return dp[ind][target] = notTaken + taken;
}

int perfectSum(vector<int>& num, int k) {
    int n = num.size();
    vector<vector<int>> dp(n, vector<int>(k + 1, -1));
    return findWaysUtil(n - 1, k, num, dp);
}
```

20) Partitions with Given Difference (based on q no 19)

06 December 2024 19:43

Given an array **arr[]**, partition it into two subsets(possibly empty) such that each element must belong to only one subset. Let the sum of the elements of these two subsets be **sum1** and **sum2**. Given a difference **d**, count the number of partitions in which **sum1** is greater than or equal to **sum2** and the difference between **sum1** and **sum2** is equal to **d**.

From <https://www.geeksforgeeks.org/problems/partitions-with-given-difference/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=partitions-with-given-difference>

```
int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp) {
    // Base case: If the target sum is 0, we found a valid subset
    if (ind == 0){
        if (target == 0 && arr[0] == 0) return 2;
        if (target == 0 || target == arr[0])return 1;
        return 0;
    }

    if (dp[ind][target] != -1) return dp[ind][target];

    int notTaken = findWaysUtil(ind - 1, target, arr, dp);

    int taken = 0;
    if (arr[ind] <= target)
        taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

    return dp[ind][target] = notTaken + taken;
}

int perfectSum(vector<int>& num, int k) {
    int n = num.size();
    vector<vector<int>> dp(n, vector<int>(k + 1, -1));
    return findWaysUtil(n - 1, k, num, dp);
}
int countPartitions(vector<int>& arr, int d) {
    int totalSum = accumulate(arr.begin(), arr.end(), 0);
    if(totalSum - d < 0 || (totalSum-d)%2 != 0) return 0;
    return perfectSum(arr, (totalSum-d)/2);
}
```

21) 0/1 Knapsack (based on q no 16)

06 December 2024 20:04

You are given the weights and values of items, and you need to put these items in a knapsack of capacity **capacity** to achieve the maximum total value in the knapsack. Each item is available in only one quantity.
In other words, you are given two integer arrays **val[]** and **wt[]**, which represent the values and weights associated with items, respectively. You are also given an integer **capacity**, which represents the knapsack capacity. Your task is to find the maximum sum of values of a subset of val[] such that the sum of the weights of the corresponding subset is less than or equal to **capacity**. You cannot break an item; you must either pick the entire item or leave it (0-1 property).

From <https://www.geeksforgeeks.org/problems/0-1-knapsack-problem0945/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card>

```
int helper(int index, int W, vector<int> &val, vector<int> &wt, vector<vector<int>> &dp){
    if(index == 0){
        if(wt[0] <= W) return val[0];
        return 0;
    }
    if(dp[index][W] != -1) return dp[index][W];

    int notTaken = helper(index-1, W, val ,wt, dp);
    int taken = INT_MIN;
    if(wt[index] <= W) taken = val[index] + helper(index-1, W-wt[index], val, wt, dp);
    return dp[index][W] = max(notTaken, taken);
}
int knapSack(int capacity, vector<int> &val, vector<int> &wt) {
    int n = val.size();
    vector<vector<int>>dp(n, vector<int>(capacity+1, -1));
    return helper(n-1, capacity, val ,wt, dp);
}
```

22) Coin Change

06 December 2024 20:48

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.
Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return -1.
You may assume that you have an infinite number of each kind of coin.

From <https://leetcode.com/problems/coin-change/description/>

```
int helper(int index, vector<int> &coins, int T, vector<vector<int>> &dp){
    if(index == 0){
        if(T % coins[0] == 0) return T / coins[0];
        else return 1e9;
    }
    if(dp[index][T] != -1) return dp[index][T];
    int notTaken = 0 + helper(index-1, coins, T, dp);
    int taken = 1e9;
    if(coins[index] <= T) taken = 1 + helper(index, coins, T-coins[index], dp);
    return dp[index][T] = min(taken, notTaken);
}
int coinChange(vector<int>& coins, int amount) {
    int n = coins.size();
    vector<vector<int>> dp(n, vector<int>(amount+1, -1));
    int result = helper(n - 1, coins, amount, dp);
    return (result == 1e9) ? -1 : result;
}
```

23) Target Sum (exactly same as q no 20)

06 December 2024 20:59

You are given an integer array nums and an integer target.
You want to build an **expression** out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.

- For example, if nums = [2, 1], you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1".

Return the number of different **expressions** that you can build, which evaluates to target.

From <https://leetcode.com/problems/target-sum/description/>

```
    int findWaysUtil(int ind, int target, vector<int>& arr, vector<vector<int>>& dp)
{
        if (ind == 0){
            if (target == 0 && arr[0] == 0) return 2;
            if (target == 0 || target == arr[0])return 1;
            return 0;
        }

        if (dp[ind][target] != -1) return dp[ind][target];

        int notTaken = findWaysUtil(ind - 1, target, arr, dp);

        int taken = 0;
        if (arr[ind] <= target)
            taken = findWaysUtil(ind - 1, target - arr[ind], arr, dp);

        return dp[ind][target] = notTaken + taken;
    }

    int perfectSum(vector<int>& num, int k) {
        int n = num.size();
        vector<vector<int>> dp(n, vector<int>(k + 1, -1));
        return findWaysUtil(n - 1, k, num, dp);
    }
    int countPartitions(vector<int>& arr, int d) {
        int totalSum = accumulate(arr.begin(), arr.end(), 0);
        if(totalSum - d < 0 || (totalSum-d)%2 != 0) return 0;
        return perfectSum(arr, (totalSum-d)/2);
    }
    int findTargetSumWays(vector<int>& nums, int target) {
        return countPartitions(nums, target);
    }
```

24) Coin Change II

06 December 2024 21:31

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.
Return *the number of combinations that make up that amount*. If that amount of money cannot be made up by any combination of the coins, return 0.
You may assume that you have an infinite number of each kind of coin.
The answer is **guaranteed** to fit into a signed **32-bit** integer.

From <https://leetcode.com/problems/coin-change-ii/description/>

```
int helper(int index, vector<int> &coins, int T, vector<vector<int>> &dp){
    if(index == 0) return (T % coins[0] == 0);
    if(dp[index][T] != -1) return dp[index][T];
    int notTaken = 0 + helper(index-1, coins, T, dp);
    int taken = 0;
    if(coins[index] <= T) taken = helper(index, coins, T-coins[index], dp);
    return dp[index][T] = taken + notTaken;
}
int change(int amount, vector<int>& coins){
    int n = coins.size();
    vector<vector<int>> dp(n, vector<int>(amount+1, -1));
    return helper(n - 1, coins, amount, dp);
}
```

25) unbounded Knapsack (based on prev knapsack problem)

07 December 2024 17:39

Given a set of items, each with a weight and a value, represented by the array **wt** and **val** respectively. Also, a knapsack with a weight limit **capacity**.
The task is to fill the knapsack in such a way that we can get the maximum profit. Return the maximum profit.

From <https://www.geeksforgeeks.org/problems/knapsack-with-duplicate-items4201/1?utm_source=youtube&utm_medium=collab_striver_ytdescription&utm_campaign=knapsack-with-duplicate-items>

```
int helper(vector<int>& val, vector<int> &wt, int W, int index, vector<vector<int>> &dp){
    if(index == 0) return ((int)(W/wt[0]))*val[0];

    if(dp[index][W] != -1) return dp[index][W];

    int notTaken = helper(val, wt, W, index-1, dp);
    int taken = 0;
    if(wt[index] <= W){
        taken = val[index] + helper(val, wt, W-wt[index], index, dp);
    }
    return dp[index][W] = max(taken, notTaken);
}
int knapSack(vector<int>& val, vector<int>& wt, int capacity) {
    // code here
    int n = val.size();
    vector<vector<int>> dp(n, vector<int>(capacity+1, -1));
    return helper(val, wt, capacity, n-1, dp);
}
```

26) Rod cutting (q no 16)

07 December 2024 18:04

Given a rod of length **n** inches and an array of prices, **price**. price[i] denotes the value of a piece of length i. Determine the **maximum** value obtainable by cutting up the rod and selling the pieces.

From <https://www.geeksforgeeks.org/problems/rod-cutting0840/1>

```
int helper(int n, int index, vector<int> &price, vector<vector<int>> &dp){
    if(index == 0) return n*price[0];

    if(dp[index][n] != -1) return dp[index][n];

    int notTaken = 0 + helper(n, index-1, price, dp);
    int taken = INT_MIN;
    int rodLength = index+1;
    if(rodLength <= n) taken = price[index] + helper(n-rodLength, index, price, dp);

    return dp[index][n] = max(notTaken, taken);
}
int cutRod(vector<int> &price) {
    int n = price.size();
    vector<vector<int>> dp(n, vector<int>(n+1, -1));
    return helper(n, n-1, price, dp);
}
```