Code & Examples 2

Knapsack Problem

Given n items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible. The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

maxresdefault.jpg

Solution:

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the maximum profit in a knapsack
int knapsack(int W, vector<int> &val, vector<int> &wt) {

    // Initialize dp array with initial value 0
    vector<int> dp(W + 1, 0);

    // Iterate through each item, starting from the 1st item to the nth item
    for (int i = 1; i <= wt.size(); i++) {

        // Start from the back, so we also have data from
        // previous calculations of i-1 items and avoid duplication
        for (int j = W; j >= wt[i - 1]; j--) {
            dp[j] = max(dp[j], dp[j - wt[i - 1]] + val[i - 1]);
        }
    }
    return dp[W];
}

int main() {
    vector<int> val = {1, 2, 3};
```

```
    vector<int> wt = {4, 5, 1};
    int W = 4;

    cout << knapsack(W, val, wt) << endl;
    return 0;
}
```

Explanation:
The code above is already optimized as it saves memory usage and execution time. The analogy for this program is as follows:

- We have a bag (knapsack) with a certain capacity (W).
- We have several items with certain values (val) and weights (wt).
- Our goal is to fill the bag with items so that the total value of items in the bag is maximized without exceeding the bag's capacity.
- We use a dynamic programming approach to solve this problem efficiently.
- First, we iterate through each item, and for each item, we iterate the bag capacity from back to front (e.g., from maximum weight to 0).
- This way, we will find the maximum value that can be put into the bag without exceeding its capacity.
- The code already covers the situation of comparing between using the item with the highest weight or adding an item with a lower weight with the remaining capacity.

Revision #2
Created 2025-12-07 09:21:38 UTC by BH
Updated 2025-12-07 10:05:42 UTC by BH