Unbounded Knapsack (Repetition of items allowed)

Given a knapsack weight **W** and a set of **n** items with certain value val_i and weight wt_i, we need to calculate minimum amount that could make up this quantity exactly. This is different from classical Knapsack problem, here we are allowed to use unlimited number of instances of an item.

Examples:

```
Input : W = 100
        val[]  = {1, 30}
        wt[] = {1, 50}
Output : 100
There are many ways to fill knapsack.
1) 2 instances of 50 unit weight item.
2) 100 instances of 1 unit weight item.
3) 1 instance of 50 unit weight item and 50
   instances of 1 unit weight items.
We get maximum value with option 2.

Input : W = 8
        val[] = {10, 40, 50, 70}
        wt[]  = {1, 3, 4, 5}
Output : 110
We get maximum value with one unit of
weight 5 and one unit of weight 3.
```

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

Its an unbounded knapsack problem as we can use 1 or more instances of any resource. A simple 1D array, say dp[W+1] can be used such that dp[i] stores the maximum value which can achieved using all items and i capacity of knapsack. Note that we use 1D array here which is different from classical

knapsack where we used 2D array. Here number of items never changes. We always have all items available.

We can recursively compute dp[] using below formula

```
dp[i] = 0
dp[i] = max(dp[i], dp[n-wt[j]] + val[j]
                   where j varies from 0
                   to n-1 such that:
                   wt[j] <= i

result = d[W]
```

Below is the implementation of above idea.

C++

```
// C++ program to find maximum achievable value
// with a knapsack of weight W and multiple
// instances allowed.
#include<bits/stdc++.h>
using namespace std;

// Returns the maximum value with knapsack of
// W capacity
int unboundedKnapsack(int W, int n, int val[], int wt[])
{
    // dp[i] is going to store maximum value
    // with knapsack capacity i.
    int dp[W+1];
    memset(dp, 0, sizeof dp);

    int ans = 0;

    // Fill dp[] using above recursive formula
    for (int i=0; i<=W; i++)
      for (int j=0; j<n; j++)
         if (wt[j] <= i)
            dp[i] = max(dp[i], dp[i-wt[j]]+val[j]);

    return dp[W];
}

// Driver program
int main()
{
    int W = 100;
    int val[] = {10, 30, 20};
    int wt[] = {5, 10, 15};
    int n = sizeof(val)/sizeof(val[0]);

    cout << unboundedKnapsack(W, n, val, wt);

    return 0;
}
```

Run on IDE

Java

```java
// Java program to find maximum achievable
// value with a knapsack of weight W and
// multiple instances allowed.
public class UboundedKnapsack {

    private static int max(int i, int j) {
            return (i > j) ? i : j;
    }

    // Returns the maximum value with knapsack
    // of W capacity
    private static int unboundedKnapsack(int W, int n,
                                int[] val, int[] wt) {

        // dp[i] is going to store maximum value
        // with knapsack capacity i.
        int dp[] = new int[W + 1];

        // Fill dp[] using above recursive formula
        for(int i = 0; i <= W; i++){
            for(int j = 0; j < n; j++){
                if(wt[j] <= i){
                    dp[i] = max(dp[i], dp[i - wt[j]] +
                                val[j]);
                }
            }
        }
        return dp[W];
    }

    // Driver program
    public static void main(String[] args) {
        int W = 100;
        int val[] = {10, 30, 20};
        int wt[] = {5, 10, 15};
        int n = val.length;
        System.out.println(unboundedKnapsack(W, n, val, wt));
    }
}
// This code is contributed by Aditya Kumar
```

Run on IDE

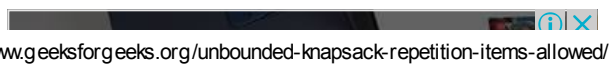
Output:

```
300
```

Asked in: Amazon,Google

This article is compiled using inputs from **Shubham Gupta**, **Shubham Joshi** and **Ashish kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



GATE CS Corner Company Wise Coding Practice

Dynamic Programming knapsack

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Dynamic Programming | Set 10 (0-1 Knapsack Problem)
Ways to write n as sum of two or more positive integers
Dynamic Programming | Set 7 (Coin Change)
Finding the maximum square sub-matrix with all equal elements

A Space Optimized DP solution for 0-1 Knapsack Problem
K maximum sums of non-overlapping contiguous sub-arrays
Largest rectangular sub-matrix having sum divisible by k
Minimum number of deletions to make a string palindrome | Set 2
Count ways to reach the nth stair using step 1, 2 or 3
Newman-Conway Sequence

(Login to Rate)

3.2 Average Difficulty : **3.2/5.0**
Based on **33** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!