

Dynamic Programming | Set 10 (0-1 Knapsack Problem)

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays $val[0..n-1]$ and $wt[0..n-1]$ which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of $val[]$ such that sum of the weights of this subset is smaller than or equal to W . You cannot break an item, either pick the complete item, or don't pick it (0-1 property).

A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subsets whose total weight is smaller than W . From all such subsets, pick the maximum value subset.

1) Optimal Substructure:

To consider all subsets of items, there can be two cases for every item: (1) the item is included in the optimal subset, (2) not included in the optimal set.

Therefore, the maximum value that can be obtained from n items is max of following two values.

- 1) Maximum value obtained by $n-1$ items and W weight (excluding n th item).
- 2) Value of n th item plus maximum value obtained by $n-1$ items and W minus weight of the n th item (including n th item).

If weight of n th item is greater than W , then the n th item cannot be included and case 1 is the only possibility.

2) Overlapping Subproblems

Following is recursive implementation that simply follows the recursive structure mentioned above.

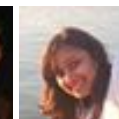
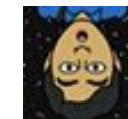
Google™ Custom Search



GeeksforGeeks



53,524 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

[Geometric Algorithms](#)

```

/* A Naive recursive implementation of 0-1 Knapsack problem */
#include<stdio.h>

// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity
int knapSack(int W, int wt[], int val[], int n)
{
    // Base Case
    if (n == 0 || W == 0)
        return 0;

    // If weight of the nth item is more than Knapsack capacity W, then
    // this item cannot be included in the optimal solution
    if (wt[n-1] > W)
        return knapSack(W, wt, val, n-1);

    // Return the maximum of two cases: (1) nth item included (2) not i
    else return max( val[n-1] + knapSack(W-wt[n-1], wt, val, n-1),
                    knapSack(W, wt, val, n-1)
                    );
}

// Driver program to test above function
int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}

```

It should be noted that the above function computes the same subproblems again and again. See the following recursion tree, K(1, 1) is being evaluated twice. Time complexity of this naive recursive solution is exponential (2^n).

In the following recursion tree, K() refers to knapSack(). The two parameters indicated in the following recursion tree are n and W.

The recursion tree is for following sample inputs.

wt[] = {1, 1, 1}, W = 2, val[] = {10, 20, 30}



Popular Posts

[All permutations of a given string](#)

[Memory Layout of C Programs](#)

[Understanding "extern" keyword in C](#)

[Median of two sorted arrays](#)

[Tree traversal without recursion and without stack!](#)

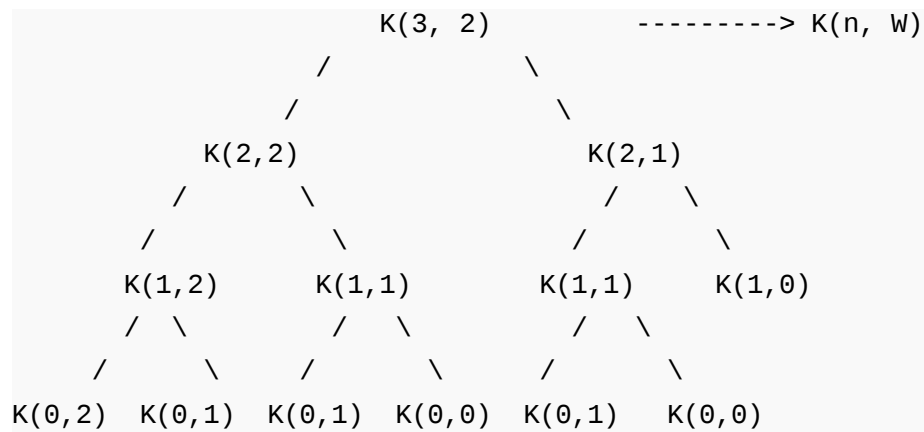
[Structure Member Alignment, Padding and Data Packing](#)

[Intersection point of two Linked Lists](#)

[Lowest Common Ancestor in a BST.](#)

[Check if a binary tree is BST or not](#)

[Sorted Linked List to Balanced BST](#)



Recursion tree for Knapsack capacity 2 units and 3 items of 1 unit weight.

Since subproblems are evaluated again, this problem has Overlapping Subproblems property. So the 0-1 Knapsack problem has both properties (see [this](#) and [this](#)) of a dynamic programming problem. Like other typical **Dynamic Programming(DP) problems**, recomputations of same subproblems can be avoided by constructing a temporary array $K[][]$ in bottom up manner. Following is Dynamic Programming based implementation.

```
// A Dynamic Programming based solution for 0-1 Knapsack problem
#include<stdio.h>

// A utility function that returns maximum of two integers
int max(int a, int b) { return (a > b)? a : b; }

// Returns the maximum value that can be put in a knapsack of capacity
int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;
    int K[n+1][W+1];

    // Build table K[][] in bottom up manner
    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= W; w++)
        {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if (wt[i-1] <= w)
                K[i][w] = max(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);
            else
                K[i][w] = K[i-1][w];
        }
    }
}
```

```

    }

    return K[n][W];
}

int main()
{
    int val[] = {60, 100, 120};
    int wt[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(val)/sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}

```

Time Complexity: $O(nW)$ where n is the number of items and W is the capacity of knapsack.

References:

<http://www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf>

<http://www.cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf>

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



[Subscribe](#)

Recent Comments

Aman Hi, Why arent we checking for conditions...

[Write a C program to Delete a Tree.](#) · 38 minutes ago

kzs please provide solution for the problem...

[Backtracking | Set 2 \(Rat in a Maze\)](#) · 42 minutes ago

Sanjay Agarwal bool

[tree::Root_to_leaf_path_given_sum\(tree...](#)
Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

[Count trailing zeroes in factorial of a number](#) · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

[Find subarray with given sum](#) · 1 hour ago

newCoder3006 Code without using while loop. We can do it...

[Find subarray with given sum](#) · 1 hour ago

AdChoices

[► C++ Code](#)

[► Java Source Code](#)

[► Programming C++](#)

[▶ Knapsack](#)[▶ Int](#)[▶ C++ Example](#)[▶ C++ Program](#)[▶ Test C++](#)[▶ Algorithm](#)

Related Tpoics:

- [Backtracking | Set 8 \(Solving Cryptarithmic Puzzles\)](#)
- [Tail Recursion](#)
- [Find if two rectangles overlap](#)
- [Analysis of Algorithm | Set 4 \(Solving Recurrences\)](#)
- [Print all possible paths from top left to bottom right of a mXn matrix](#)
- [Generate all unique partitions of an integer](#)
- [Russian Peasant Multiplication](#)
- [Closest Pair of Points | O\(nlogn\) Implementation](#)



9



Tweet

1



3

Writing code in comment? Please use [ideone.com](#) and share the link here.

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team