Algorithms: Dynamic Programming - The Integer Knapsack Problem with C Program Source Code

Quick Links to Various Sections on the Main Page

- The Learning Point
- Computer Science
- Algorithms: An introduction to Dynamic Programming
- CS - Data Structures and Algorithms
- CS - Programming Interviews
- CS - Miscellaneous C Programs
- CS - Intro to DB Queries
- Electrical Science and Engineering
- Electrical Sci - DC Circuits

Given n items of weight wi and value vi, find the items that should be taken such that the weight is less than the maximum weight W and the corresponding total value is maximum. This problem exhibits both overlapping subproblems and optimal substructure and is therefore a good candidate for dynamic programming.

Complexity:

Time complexity is O(n*W), where n is the total number of items and W is the maximum weight.

Complete Tutorial with Examples :

Electrical Sci - Digital Electronics
Mathematics
Mathematics - Linear Algebra
Mathematics - Geometry
Mathematics - Single Variable Calculus
Mathematics - Game Theory
Mathematics - Operations Research
Physics
Physics - Basic Mechanics
Physics - Electrostatics
Sitemap
Recent site activity

These are Quick Links to sections of the Main Page

1

1 / 4

Knapsack(0-1)

Given two n-tuples

$\{v_1, v_2, v_3 \ldots v_n\}$ and $\{w_1, w_2, w_3 \ldots w_n\}$,

and, W>0. We wish to determine a subset T such that

Maximizes - ,

Subject to - $\leq$ W.

Meaning, given n items of weight w_i and value v_i, find the items that should be taken such that the weight is less than the maximum weight W and the corresponding total value is maximum. We can either take the complete item (1) or not (0).

Let A(i,j) represents maximum value that can be attained if the maximum weight is W and items are chosen from 1...i. We have the following recursive definition

$$A(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ A(i-1,j) & \text{if } w_i > j \\ \max\{A(i-1,j), v_i + A(i-1, j-w_i)\} & \text{if } w_i \leq j \end{cases}$$

This problem exhibits both overlapping subproblems and optimal substructure and is therefore a good candidate for dynamic programming.

Algorithm:

Firstly, input the total number of items, the weight and value of each item. Then input the

Integer Knapsack Problem - C Program Source Code

```
#include<stdio.h>
int max(int a,int b)
{
        return a>b?a:b;
}
int Knapsack(int items,int weight[],int value[],int maxWeight)
{
        int dp[items+1][maxWeight+1];
        /* dp[i][w] represents maximum value that can be attained if the maximum weight is w and
           items are chosen from 1...i */
        /* dp[0][w] = 0 for all w because we have chosen 0 items */
        int iter,w;
        for(iter=0;iter<=maxWeight;iter++)
        {
                dp[0][iter]=0;
        }
        /* dp[i][0] = 0 for all w because maximum weight we can take is 0 */
        for(iter=0;iter<=items;iter++)
        {
                dp[iter][0]=0;
        }
        for(iter=1;iter<=items;iter++)
        {
                for(w=0;w<=maxWeight;w++)
                {
                        dp[iter][w] = dp[iter-1][w]; /* If I do not take this item */
                        if(w-weight[iter] >=0)
                        {
                                /* suppose if I take this item */
                                dp[iter][w] = max(dp[iter][w] , dp[iter-1][w-weight[iter]]+value[iter]);
                        }
                }
```

```
        }
        return dp[items][maxWeight];
}
int main()
{
        int items;
        scanf("%d",&items);
        int weight[items+1],value[items+1];
        int iter;
        for(iter=1;iter<=items;iter++)
        {
                scanf("%d%d",&weight[iter],&value[iter]);
        }
        int maxWeight;
        scanf("%d",&maxWeight);
        printf("Max value attained can be %d\n",Knapsack(items,weight,value,maxWeight));
}
```

Rough notes about the Algorithm - as implemented in the code above:

Firstly, input the total number of items, the weight and value of each item. Then input the maximum weight (maxWeight). Lastly calculate the maximum value that can be attained using Knapsack function.

Knapsack function – This function takes total number of items (items), weight of all the items (weight), value of all the items (value) and the maximum weight (maxWeight) as arguments. It returns the maximum value that can be attained.

Declare **dp[items+1][maxWeight+1]**. Where, dp[i][w] represents maximum value that can be attained if the maximum weight
dp[0][w] = 0 for all w because we have chosen 0 items. And, dp[i][0] = 0 for all w because maximum weight we can take
Recurrence:

for i=1 to items
    for w=0 to maxWeight
        dp[i][w] = dp[i-1][w], if we do not tale item i. if w-weight[i] >=0, suppose we take this item then, dp[i][w] = max(dp[i][w] , dp[i-1][w-weight[i]]+value[i]). *Where, max is a function that returns the maximum of the two arguments it takes.*

next

next

Return dp[items][maxWeight]

Related Tutorials (common examples of Dynamic Programming):

Integer Knapsack problem	An elementary problem, often used to introduce the concept of dynamic programming.
Matrix Chain Multiplication	Given a long chain of matrices of various sizes, how do you parenthesize them for the purpose of multiplication - how do you chose which ones to start multiplying first?
Longest Common Subsequence	Given two strings, find the longest common sub sequence between them.

Some Important Data Structures and Algorithms, at a glance:

Arrays : Popular Sorting and Searching			

Algorithms			
Bubble Sort	Insertion Sort	Selection Sort	Shell Sort
Merge Sort	Quick Sort	Heap Sort	Binary Search Algorithm
Basic Data Structures and Operations on them			
Stacks	Queues	Single Linked List	Double Linked List
Circular Linked List	1.		

Tree Data Structures			
Binary Search Trees	Heaps	Height Balanced Trees	
Graphs and Graph Algorithms			
Depth First Search	Breadth First Search	Minimum Spanning Trees: Kruskal Algorithm	Minumum Spanning Trees: Prim's Algorithm
Dijkstra Algorithm for Shortest Paths	Floyd Warshall Algorithm for Shortest Paths	Bellman Ford Algorithm	
Popular Algorithms in Dynamic Programming			
Dynamic Programming	Integer Knapsack problem	Matrix Chain Multiplication	Longest Common Subsequence
Greedy Algorithms			
Elementary cases : Fractional Knapsack Problem, Task Scheduling	Data Compression using Huffman Trees		