Wenlan Tian

COP 3530

Section 1087, MAEB 211

11/04/2014

Homework 6

"On my honor, I have neither given nor received unauthorized aid in doing this assignment.”

Summary of learning experience:

The easiest part of the task is to understand the theory of the 0-1 knapsack.

The most difficult parts of the task: how to implement the max function which compares the sack(item-1,size) and sack(item-1, size-weight[item])+value(item).

The assignment’s educational objective: we learned the theory in class, the assignment is letting us to practice how to implement it in C++.

How well I think I achieved them: I think I meet all the specifications for the assignment. I used the same data as professor’s example in class to test the program, and the results are same as the professors’.

Wenlan Tian

COP 3530

Section 1087, MAEB 211

11/04/2014

Homework 6

1. Does the program compile without errors?

No error.

2. Does the program compile without warnings?

No warnings.

3. Does the program run without crashing?

No crashing.

4. Describe how you tested the program.

I used the same data as professor’s example from class to test the program, and the results are same as the professors’. More detailed, I use 4 types – A,B,C,D, each has a weight and value associated with it.

5. Describe the ways in which the program does not meet assignment’s specifications.

I think the program meets the assignment’s specifications.

6. Describe all known and suspected bugs.

No bugs detected.

7. Does the program run correctly?

I think it runs correctly, because I used the same data as professor’s example to test the program, and the results are same as the professors’.

Wenlan Tian

COP 3530

Section 1087, MAEB 211

11/04/2014

Homework 6

Homework 6: 0-1 knapsack

Implement (and thoroughly test) a function which takes

* Three equal size arrays

-Weight [int]

-Value [int or double – your choice]

-Name [std::string]

* The number of item type[int]
* The knapsack capacity [int]

Use DP to determine the contents and value of an optimal solution to 0-1 knapsack problem.

#include <iostream>

#include <string>

int max(int a, int b){

return (a >b) ? a:b;

}

int sack(int weight[], int value[], std::string name[], int itemType, int capacity){

int item[itemType+1][capacity+1];

for (int i =0; i <= itemType; ++i){

for (int j =0; j <= capacity; ++j){

if (i == 0 || j == 0) {

item[i][j]= 0;

}else if(weight[i-1]<=j){

item[i][j] = max(value[i-1]+item[i-1][j-weight[i-1]] ,item[i-1][j]);

}else{

item[i][j] = item[i-1][j];

}

}

}

//print the items to be taken

std::cout << "An optimal solution to this problem is to take: " << std::flush;

int j = capacity;

for (int i = itemType; i > 0; --i){

if (item[i][j]!=item[i-1][j]){

std::cout << name[i-1] << " " << std::flush;

j -= weight[i-1];

if (j<0) break;

}

}

std::cout << std::endl;

return item[itemType][capacity];

}

int main(){

int weight[] = {3,6,4,2};

int value[] = {14,30,17,9};

std::string name[] = {"A","B","C","D"};

int capacity = 10;

int itemType = sizeof(value)/sizeof(value[0]);

sack(weight, value, name, itemType, capacity);

return 0;

}

TEST:

UFs-MacBook-Pro:HW6 tianwenlan$ g++ HW06.cpp

UFs-MacBook-Pro:HW6 tianwenlan$ ./a.out

**An optimal solution to this problem is to take: C B**