CS575 Programming Assignment 5

Due at 11:59:59PM on May 1, 2023(Monday)

1. [90%] Randomly create a 0/1 Knapsack problem as follows:
    - Create n items where n is an integer randomly selected between 4 and 8. Display the selected n value.
    - Create a list of n items where each item has profit p_i and weight w_i where 1) p_i is a positive integer randomly selected between 10 and 30; and 2) w_i is a positive integer randomly selected between 5 and 20. Set the capacity of the knapsack W = floor(0.6 * $\sum_{i=1}^{n} w_i$).
    - Your program should be invoked as follows and output a text file called knapsack01.txt with the following format.

    $> createkn01 -k knapsack01.txt

    knapsack01.txt

    4 24

Item1	10	7
Item2	13	9
Item3	21	10
Item4	24	15

    The first line contains the number items and the weight capacity of the knapsack. Each of the following lines contains the name of the item (you can just use item1 to item8), the profit of the item, and the weight of the item.

    To solve the created 0/1 knapsack problem, implement the following algorithms discussed in class:

    1) [10%] Implement the brute force method to solve the 0/1 knapsack problem. Show the final solution. Specifically, (a) print the total profit and weight; and (b) print the selected items together with their profits and weights. Your program should be invoked as follows and output a text file called output1.txt. You can modify your program assignment 1 for the implementation.

        $> bruteforce -k knapsack01.txt

        output1.txt

2	37	24
Item2	13	9
Item4	24	15

The first line contains the number items that produced the maximum profit, the produced maximum profit, and the total weight for the items that produced the maximum profit. Each of the following lines contains the name of the item, the profit of the item, and the weight of the item.

2) [40%] Implement the refinement algorithm of dynamic programming approach (slides 38-40 of Ch12 lecture notes): (a) print the entries that are calculated in each row; (b) print the total profit and weight for the final solution; and (c) print the selected items together with their profits and weights for the final solution. Hint: you may need to define an extra variable to keep track of which items to be included in the knapsack comparing to the pseudo code of the general dynamic programming approach for the 0/1 knapsack problem (slide 33 of Ch12 lecture notes). If you have implemented the refinement algorithm of dynamic programming approach correctly, the total profit achieved by this algorithm must be equal to that achieved by the brute force method implemented in 1).
   - Note: You are supposed to implement the algorithm using the refinement algorithm of dynamic programming approach for the 0/1 knapsack problem. You will get no credit if you implement the general dynamic programming by calculating all the entries of matrix B (as shown in slide 33 of Ch12 lecture notes). We did not talk much about space complexity for this problem, and so it is acceptable if you define a full matrix B but set the entries that do not need to be computed to zero.
   - Your program should be invoked as follows and output a text file called entries2.txt and output2.txt. The format of output2.txt should be same as output1.txt. The format of entries2.txt should be as follows.
     $> dynpro -k knapsack01.txt

     entries2.txt (assuming that you have 4 items to choose)
     row1 <col1> <col2> … <coli>
     row2 <col1> <col2> … <colj>
     row3 <col1> <col2>
     row4 <col1>
     You will have one column index for row4 and two column indices for row3.

3) [40%] Implement the backtracking algorithm: (a) print the profit, weight and bound for each node according to the order that is first visited in the implicit pruned state space tree. (b) print the total profit and weight for the final solution; and (c) print the selected items together with their profits and weights for the final solution. Compare the result to the result you have got in 1). If you have implemented the backtracking algorithm correctly, the total profit achieved by this backtracking method must be equal to that achieved by the brute force method implemented in 1).

- Your program should be invoked as follows and output a text file called entries3.txt and output3.txt. The format of output3.txt should be same as output1.txt. The format of entries3.txt should be as follows.

$> backtrack -k knapsack01.txt

entries3.txt (assuming that you have 4 items to choose)

1	0	0	0
2	<profit2>	<weight2>	<bound2>
3	<profit3>	<weight3>	<bound3>

……

Each row starts with the visiting order of node, followed by its profit, weight, and upper bound in the pruned state space tree.

- Note: You are supposed to implement these algorithms correctly for **any 0/1 knapsack problems** as described above. If your program produces correct results for some 0/1 knapsack but doesn't for other knapsacks, you will get no credit.

2. [10%] Coding style: Write meaningful comments, while making your code structured, easy to read, and robust.

All programming must be done using **C or C++ or Java in Linux** where your code will be tested. Create a tar file that includes (1) source code files, (2) a Makefile to produce an executable, and (3) a readme file that clearly describes how to run your code. Submit only the tar file through the Blackboard. The name of the tar file should be yourlastname_yourfirstname _proj5.tar (Do not use special characters like #, @, or &, because they have caused Blackboard problems in the past.) Suppose that your assignment files are under the directory of /your_userid/yourlastname_yourfirstname_proj5/ and you are under that directory right now. To create a tar file under /your_userid directory, do the following in Linux command line:

```
>cd ..
>tar cvf  yourlastname_yourfirstname_proj5.tar  yourlastname_yourfirstname_proj5
```

To view the content of the created tar file, do the following in Linux command line:

```
>tar tvf  yourlastname_yourfirstname_proj5.tar
```

Finally, read the following policies carefully:

- *All work must represent each individual student's own effort. If you show your code or any other part of your work to somebody else or copy or adapt somebody else's work found online or offline, you will get zero and be penalized per the Watson School Academic Honesty Code (http://www.binghamton.edu/watson/about/honesty-policy.pdf).*
- *To detect software plagiarism, everybody's code will be cross-checked using an automated tool.*
- *Your code will be compiled and executed. If your code does not compile or produce any*

runtime errors, such as segmentation faults or bus errors, you will get zero.

- *The instructor and TA will not read or debug your code. The instructor and TA will not take a look at an emailed code. If you need general directions, show your code to a TA during their office hours. The TAs will not do programming or debugging for you though. The TAs will only help you understand algorithms to be implemented and answer basic questions related to implementation.*