CS375 Design and Analysis of Algorithms

Spring 2015

Programming Assignment 4

Assigned: Monday April 6, 2015
Due: Midnight Friday April 24, 2015 See Blackboard for detailed submission instructions.

Remember to include the following statement in your readme.txt file. "I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment for my first offense and that I will receive a grade of "F" for the course for any additional offense."

Prelude

For Programs 1 and 3, you implemented several different solutions to the 0/1 Knapsack Problem, applied to the baseball card story. In particular, you have implemented:

- A ***brute force*** algorithm that enumerated all possible subsets of baseball cards and selected the most profitable subset of cards. Your implementation of this algorithm is not able to handle large inputs because of its exponential asymptotic runtime complexity.
- A ***greedy*** algorithm that selected items in order of profit per unit cost. This algorithm proved very scalable, but was not able to find the optimal solution for all inputs. A minor adjustment (for "Greedy 2") fixed this problem for one special kind of input, but may still miss optimal solutions.
- A ***backtracking*** algorithm takes the brute force approach but makes it significantly more scalable for many reasonable inputs, by recognizing when proceeding down some paths cannot lead to improved solutions.

Assignment

For Program 4, you will implement one more algorithm for the Integer 0/1 Knapsack Problem, an algorithm that uses ***dynamic programming***. Implement the algorithm described in the class lecture notes, modified to store only two rows of the dynamic programming table in memory, rather than the entire table. In the dynamic programming algorithm, each row of the table is computed using values that reside only in the row above it. Once row *k* is used to compute the values in row *k*+1, the values on row *k* can be discarded. With two rows worth of storage space, you should be able to compute the maximum profit.

Input

Your program will support the following usage:

```
$> ./program4 –m <market-price-file> -p <price-list-file> -o <output-file-name> [3]
```

Note that users will invoke this program exactly the same way that they invoked your program 3, except that the name of the executable is program4, and the integer parameter is 3, instead of {0|1|2}.

Input file formats for the specified *market price file* and *price list file* are exactly the same as for Programs 1 and 3. Please feel free to re-use *your own* file parsing code, and please use the sample input files that we provided for Program 3 to test your dynamic programming algorithm.

If it is easier for you, you may decide to add the dynamic programming implementation to your Program 3 code, to build a new program that supports all four different algorithms (the three from Program 3, plus the new dynamic programming algorithm for Program 4). Or you may simply reuse your Program 3 code to read input files and write output, and remove the implementations of the three Program 3 algorithms. We will test Program 4 only for the dynamic programming algorithm, not for the algorithms from Program 3.

So that we can identify and check your Dynamic Programming code, please place the following comment immediately above your implementation of the new algorithm:

```
// *** DYNAMIC PROGRAMMING CODE STARTS HERE ***
```

Output

Your output file format should also match the output format for Program 3; the initial line of output should be labeled "`Dynamic Programming: `". For this algorithm, you should *not* try to output the number of cards nor the set of cards that generate the maximum profit; generate one line of output per problem, formatted as follows:

```
Dynamic Programming: 25 422 3.87
```

The 25 corresponds to the number of cards in the price list file (*n*), the 422 is the maximum profit achieved, and the 3.87 is the runtime of the algorithm measured in seconds.

Testing

Please test your program using the test cases from Program 3, including and especially the "Large" test case. For *n*=1,000,000 and C=9157 (the input parameters of the "Large" test case), the full dynamic programming table would require (*n*+1)*(C+1) = approximately 37 GB for 4-byte integers.

Submission Instructions

You may write the code using C, C++, or Java. Your program must compile and run on harvey.cc.binghamton.edu. No exceptions. It should be purely a command line program; GUIs will not be accepted.

Please submit a *.tar.gz* file to Blackboard. The file should be named (lower case) as follows:

```
<lastname>_<firstinitial>_p4.tar.gz
```

When the file is unzipped it should produce a single directory named `<lastname>_<firstinitial>_p4`.

When the file is unzipped it should contain a directory with the same name as the zip file. The directory should contain:

1. File(s) with the source code for the program, and possibly a makefile.
2. A read-me file named *readme.txt* which should contain:
   Line 1: C, or C++, or Java
   Line 2+: Either a comma-delimited list of files, which will be compiled and linked with the appropriate compiler (gcc, g++, or javac) and then executed (make sure that for Java the first file has the main method) or the single word "make" which will execute the makefile in the unzipped directory. The makefile will need to produce an executable called "program4" or a "Program4" class in the java case, which will be interpreted/executed. The remainder of the file should describe any assumptions that limit the applicability of the program, any known bugs, and the honesty statement.

Grading

This assignment is worth a total of 100 points.

Plagiarism Policy

All your code will be checked for similarity to other submissions using Moss. Programmers tend to reproduce the same code that they have seen before. So you are advised not to look at each other's code. Please review the course's plagiarism policy.