CS375 Design and Analysis of Algorithms

Fall 2020

***Programming Assignment* 1**

Assigned: August 28, 2020
Due: Midnight Tuesday, September 15, 2020

Prelude

One day, while procrastinating on a term paper for a gen. ed. class, wishing your CS professors would assign more challenging programming assignments, you discover an online ad placed by Gertrude McGullible:

> **For Sale**: Baseball Cards! 84-year-old Grandmother looking to rid my garage of my children's old baseball card collection to pay and make room for a pool table, poker table, and kegerator, for a planned Gran Cave.

The ad then proceeds to list a few sample cards and prices that you immediately recognize as impossibly low! You have two thoughts:

(1) What a great cause!
(2) What an awesome money-making opportunity!

Baseball card market values are readily available online. With some planning, you hope to maximize your profit. You call Gertrude and set up a time to look through her collection. She tells you that she will give you a complete list of every card she has, along with her prices, when you arrive at her house. (Gertrude has a lot of time on her hands.) Ideally, you would buy and sell every card she has (all of her prices are below market value), but you don't have enough money for that. The more money you can bring to her house, the more profit you can make in the transaction, so you plan to scrape up every penny you can.

Given some fixed maximum amount of money to spend, the market values of baseball cards, and Gertrude's baseball card prices, how much money could you make by purchasing baseball cards from Gertrude, and which cards should you buy?

Assignment

The goal of this assignment is to experience the growth of time used by functions with exponential run time. You will implement a ***brute force algorithm*** for the problem described above. Later this semester we will study a number of different approaches to solve this problem, including greedy, backtracking, and dynamic programming algorithms.

Formal problem description: Given n items (baseball cards) numbered 1 through n, item j has a potential benefit (profit) p_j and a weight (cost) w_j. Assume that the profits and costs are positive integers. The maximum weight (in this case, the maximum amount of money that you can spend) is W. Select a subset of items such that (a) the items' total weight does not exceed W, and (b) the sum of the profits is maximum among all such subsets.

Mathematically, the problem can be formulated as:

$$\text{Maximize} \quad \sum_{j=1}^{n} p_j x_j \text{ subject to } \sum_{j=1}^{n} w_j x_j \le W \text{ and } x_j \in \{0,1\}$$

$x_j = 1$ if item j is selected (i.e. purchased from Gertrude and included in the subset); $x_j = 0$ if it is not.

Discussion

Brute force algorithms generate every possible solution (for an input of size *n*), to find the best one. For this problem, your program will need to generate every possible subset of *n* items. Note that the number of subsets of the *n* items can be very large. For example, for $n = 30$, the number of subsets is $2^{30} = (1024)^3$ which is approximately 10^9. If we assume that 30 bits are needed to store each subset, approximately $30*10^9$ bits, or approximately $4*10^9$ bytes (4 GB) are needed to store all the subsets. Since you may run out of memory, it is not a good idea to first generate all the subsets and then find the best one.

One simple brute force way to compute the maximum profit is with the following algorithm:

```
COMPUTEMAXPROFIT (I, W)
        maxProfit = 0;
        S = the empty set;
        M = the empty set;
        if the sum of the weights in I <= W
             return the profit of the items in I, and I;
        while more subsets to generate
             if the sum of the weights in S <= W
                  if the profit of the items in S > maxProfit
                        update maxProfit;
                        copy S to M;
             generate the next subset S;
        return maxProfit and M;
```

- I is the set of all items available to include (Gertrude's baseball cards).
- W is the total weight constraint for the problem (the money you have available to spend).
- S is the current set of items under consideration.
- M is the subset of items that maximizes the profit, among all of the subsets that have been considered.
- maxProfit is the total value of that profit.

What is the Worst Case computation time of this algorithm? What is the Best Case time complexity of the algorithm? *DO NOT try to improve the algorithm!* The goal of this assignment is to realize that when an algorithm has exponential run time, it becomes too slow for even small values of *n*.

Program Usage

Your program should be invoked as follows

```
$> program1 –m <market-price-file> -p <price-list-file>
```

Market Price File: `<market-price-file>` is the name of a file that contains baseball card market prices (that is, the price at which you could sell each card). The first line of the file contains only an integer that indicates how many cards and prices reside in the rest of the file, one card per line.

A sample market price file appears below.

```
4
HonusWagner1933 2000000
JimRice1975 1000
DerekJeter1996 300
RickyHenderson1983 2000
```

The baseball card identifier is a string that contains at most 64 characters with no embedded whitespace. The market value of the card, an integer representing dollars, follows a single space.

Price Lists File: `<price-list-file>` is the name of a file that contains a sequence of "price lists" for Gertrude's baseball cards. You will use the price lists file to test your program for several potential price lists, so that you are sure you are ready for whatever price list Gertrude gives you, and for however much money you will be able to spend.

So the price lists file includes one or more problems. The lines that correspond to problem *j* will be formatted as follows:

Line 1 for problem *j* contains two numbers: n_j W_j

- n_j is the number of cards in the price list for problem *j*, one per line
- W_j is the maximum amount of money you can spend on this set of cards

Each of the next n_j lines (on lines 2 to line $(n_j + 1)$) also contains two things: <*card*> *p*

- <*card*> is a string name of a baseball card in this price list
- *p* is Gertrude's price for this card, in integer dollars.

To compute potential profits from selling a card, you need to look up its market value from information in the market price file.

If your program finds a card in the price list file that does not appear in the market price file, your program should print an informative error message, and exit.

Sample price lists file:

```
2 300
JimRice1975 100
RickyHenderson1983 150
3 200
HonusWagner1933 202
DerekJeter1996 120
JimRice1975 80
```

This file contains two problems. The first one has $n = 2$ baseball cards listed, and $W = 300$ (the maximum amount you can spend). The second has three cards, and you have $200 to spend. You should create your own more interesting and extensive input files for testing.

Output File

Each line in the output file corresponds to a single problem and will contain four numbers, in the following order:

- The size of the input (the number of cards in the price list for this problem)
- The computed maximum profit, in integer dollars
- The number of cards that you could purchase to achieve the maximum profit
- The time (in seconds) it took to solve the problem.

Each of the cards in the set that produced the maximum should then follow, one per line. If there are two or more sets of cards that produce the same maximum profit, your program only needs to output one of them.

Submission

You may write the code using C, C++, or Java. Your program must compile and run on remote.cs.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>_p1.tar.gz`

When the file is unzipped it should produce a single directory named `<lastname>_<firstinitial>_p1`.

The directory should contain the following files:

1. The output file output.txt
2. File(s) with the source code for the program and possibly a makefile.
3. 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 "program1" or a "Program1" class in the java case, which will be interpreted/executed.

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.