Assignment objectives

The purpose of this assignment is to implement a brute force algorithm to solve the "job assignment" problem. (As a bonus, you get to see a decrease-and-conquer algorithm that generates permutations.)

Introduction

Welcome, young professional, to our not-for-profit consulting firm, "BC Tree Works"! Our mission is to turn the concrete jungle into a lush, green forest, one job at a time.

BC Tree Works has consultants who can perform jobs, and clients who have jobs to be done. Every consultant has different skills, and needs to be matched with the right job, for which the customer will pay a premium price. For every job we do we will earn a certain amount of money, which will translate to our ability to plant a certain number of trees.

We have N consultants, and our clients have N jobs. Every consultant must be assigned to one and only one job. We know, for every possible combination of consultants and jobs, the number of trees we will be able to plant, which we call our "benefit" for assigning that person to that job. We would like to find the matching of consultants to jobs so that the total number of trees we get to plant is as large as possible.

More details

This is, of course, the "job assignment" problem (see Lecture2). We will solve it via a brute force algorithm (see Lecture 2). The algorithm will find the best job assignment by generating *all* possible job assignments and calculating the total benefit of each one. The job assignment with the maximum total benefit is the final answer.

Input for the job assignment problem is a benefit matrix of size NxN. Row X, column Y represents the benefit of assigning person X to job Y.

Output for the job assignment problem consists of (1) a "job assignment" of people to jobs, and (2) the total benefit value of this assignment (a sum of N values).

For example, if the benefit matrix is:

3	2	6	1	2
4	6	8	8	5
4	6	1	2	7
7	9	2	9	2
7	4	9	5	1

Then the maximum job assignment is:

[2, 3, 4, 1, 0]

This shows the jobs that will be assigned to the 5 people. People and jobs are both numbered from 0 to N-1. The "2" in position 0 means that Person 0 gets Job 2 (value 6). Similarly, Person 1 gets Job 3 (value 8), Person 2 gets Job 4 (value 7), Person 3 gets Job 1 (value 9), and Person 4 gets Job 0 (value 7). The total value of this assignment is 6+8+7+9+7 = 37.

The permutations algorithm that you are given represents one permutation as an ArrayList of Integers. The most natural choice for you to represent a "job assignment" is one of these permutations.

Code requirements

Write a Java class called `JobAssignmentFinder` (please note EXACT spelling and capitalization) that provides the public methods listed below. The names and signatures of all public methods MUST BE DECLARED EXACTLY AS SPECIFIED HERE. If your method names are not spelled (and capitalized) EXACTLY as specified, that is a **FAIL**.

<<< NONE OF THESE METHODS EVER PRINTS ANYTHING TO THE CONSOLE. >>>

You are free to declare any private helper methods that you wish to use.

Class name: JobAssignmentFinder

Public method: readDataFile(String), no return value
Takes a filename as an argument and initializes the JobAssignmentFinder with a new "benefit matrix" so that it can be used to find a maximum job assignment. The JobAssignmentFinder object can be (re)initialized to solve another assignment problem by calling this function again with a different data file.
This method MUST NOT modify the String file name in any way. Assume that the full pathname necessary to open the file is already included in the given String.

Public method: getInputSize(), returns int
No arguments. Returns N, the the size of the NxN benefit matrix currently loaded in the JobAssignmentFinder. If readDataFile() has never been called, this function returns -1.

Public method: getBenefitMatrix(), returns int[][]
No arguments. Returns the currently loaded benefit matrix. Note that the return type is a plain, two-dimensional, NxN array of ints. DO NOT RETURN AN ARRAYLIST!

Public method: benefitMatrixToString(), returns String
No arguments. Returns a string representation of the current benefit matrix. No further specification for how it should look. Be creative.

Public method: getMaxAssignment(), returns ArrayList<Integer>
No arguments. The returned list contains a permutation of the numbers from 0..N-1, where N (actually NxN) is the size of the currently loaded benefit matrix. The permutation is the one that represents a job assignment that has the maximum total value.
You may assume that this function will not be called until after readDataFile() has been called.
Note that this function returns an ArrayList. The permutations algorithm that you are given to use represents each permutation as ArrayList of Integers.

Public method: getMaxAssignmentTotalValue(), returns int
No arguments. Returns the total benefit value of the maximum assignment that exists for the currently loaded benefit matrix. You may assume that this function will not be called unless readDataFile() has previously been called.

Public method: getBenefit(int, int), returns int
Assume both arguments are in the range 0..N-1. The first argument is the number of a person (0 to N-1). The second argument is a job (0 to N-1). The return value is the benefit for assigning that person to that job. E.g., if person=x and job=y, this function returns benefitMatrix[x][y].

The permutations algorithm

Note: **You do not need to write Permutations code!** It is already written for you; you just need to plop it in and use it. (See "perms.java".) Think of it as a helper function, and make it a private method in your JobAssignmentFinder class.

However, this permutations algorithm is one that we will study in class (see Lecture 3), so please use this to help you understand what it's all about.

This is a recursive "decrease and conquer" style algorithm to generate all the permutations of the numbers from 0 to N-1. The TLDR is: To solve a problem of size N, you first (recursively!) get the solution to an identical problem of size N-1, and then *extend the smaller result* so that it becomes a solution to the problem of size N.

The trick in writing these algorithms is always "how can you extend the smaller solution to transform it into the bigger?"

So: examine the given Java code. 1) Convince yourself that it follows the decrease-and-conquer pattern. 2) Study the code that it uses to transform the smaller solution into the larger one. Feel free to ask me any questions!

Final note about this algorithm: This is actually a *terrible* algorithm in terms of space efficiency. It stores the entire list of permutations—N! of them—in memory all at once. There are other, more efficient, permutation algorithms (e.g. generating permutations one at a time; given one permutation as input, it will calculate the "next" one according to some predetermined order). But we are using this one just because it exemplifies the decrease-and-conquer technique.

Data files

There are several data files available for your testing purposes. The format of a data file is:

- The first line of the data file contains a single integer (suppose you call it N), indicating how many more lines will follow.
- The next N lines of the data file each contain N integers, separated by spaces.

The example mentioned above with N=5 would have a data file that looks like this:

```
5
3 2 6 1 2
4 6 8 8 5
4 6 1 2 7
7 9 2 9 2
7 4 9 5 1
```

Expected results

This table shows the results you should get for the provided sample data files.

Data file	Max assignment	Total value of max assignment
data0.txt	[2, 3, 4, 1, 0]	37
data1.txt	[3, 2, 6, 4, 1, 5, 0]	58
data2.txt	[5, 0, 9, 2, 4, 3, 7, 6, 8, 1]	8658
data3.txt	[7, 5, 8, 4, 1, 0, 9, 3, 6, 2]	335
data4.txt	[1, 0, 4, 5, 2, 3]	42
data5.txt	[2, 1, 4, 8, 6, 7, 3, 0, 5]	74

Tips

You'll need a main() method to test your code

You can include this in the JobAssignmentFinder class, or you can make a separate driver class whose purpose is to contain "main()". *I strongly recommend making the separate class, because that is the same way I will be testing your program.*

Wherever you decide to put main(), I will not be using your main(). When I test your code, I will have my own main() program in my own separate driver class.

Note also that my code will be running your JobAssignmentFinder over several data files. Be sure you test yours the same way!

If you use a separate driver class, you do not need to submit it. It is only for your own testing purposes.

Submission information

Due date: As shown on Learning Hub.

Submit the following items to the drop box on Learning Hub:

- Your Java source code (JobAssignmentFinder.java)
- Please *do not zip* or otherwise archive your code or your submission. Submit Java files only.
- Please *do not zip* or include your entire IDE project directory.

Marking information

This lab is worth 20 points. 5 points are reserved for conformance with the COMP 3760 Coding Requirements (handout in Learning Hub “Content” section). Write your name and ID in your code comments!