Programming Assignment #3

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet. If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

The goals of this lab are:

- To recognize algorithms you have learned in class in new contexts.
- Ensure that you understand dynamic programming.

Problem Description

Fruitcake Frank has made quite a living from his life of crime. As part of his retirement he decides that he wants to plan a grand vacation where he can invite his fellow cohort of criminals to party. He will be recruiting your help for solving some of the planning aspects of the vacation and will expect a report about some of your findings.

Part 1: Planning Activities [50 points]

Fruitcake wants to plan activities for his vacation. Fruitcake discussed possible vacation activities with his criminal friends, and for each, they determined an associated "fun level" (i.e., the amount of enjoyment they get from the activity). Quite the fun guy himself, Fruitcake wants to maximize the fun level of the vacation for everyone. Unfortunately, the life of crime has its consequences, and each activity also has an associated risk of getting caught. Fruitcake estimates the total amount of risk he can afford to expel before getting caught (deemed his risk budget) and wants to select activities that maximize the total fun level while not exceeding this risk budget. Fruitcake needs your help in finding the maximum amount of fun that can be had on vacation and the activities that should be selected to do so. Activities must be selected in their entirety or not at all and activities can only be selected once.

Fruitcake provides you with the following:

- An array of strings ACTIVITIES where each element, ACTIVITIES[i], represents the name for activity i (note that each string is unique).
- An array of values* FUNLEVELS where each element, FUNLEVELS[i], represents the funness for the activity i.
- An array of values* RISKS where each element, RISKS[i], represents the risk for the activity i.
- A RISKBUDGET value*: the maximum amount of allowed risk for the vacation.

  *values are nonnegative integers

and expects the following in return:

- A MAXIMUMFUNLEVEL value*: the maximum fun level achievable during the vacation.
- A set of strings SELECTEDACTIVITIES describing a set of activities that achieve the maximum fun level. SELECTEDACTIVITIES should be a subset of ACTIVITIES given in the input.

  *values are nonnegative integers

The following is an example scenario.

Problem:
activities = ["Skydiving","Dance Lessons","Snorkeling"]
funLevels = [60,40,30]
riskLevels = [4,3,3]
riskBudget = 6

Result:
maxFunLevel = 70
selectedActivities = ["Dance Lessons","Snorkeling"]

The following deliverables are required to help Fruitcake Frank plan his vacation activities.

(a) Total Amount of Fun: [Report: 5 points; Implementation: 25 points]

In your report, give the pseudocode for an efficient algorithm determining the maximum fun level that can be achieved based on the given input. State the runtime complexity of this algorithm in Big-O notation.

Implement your algorithm, providing Fruitcake a single value representing the maximum fun level. You will be heavily penalized for a non pseudo-polynomial time algorithm.

(b) Activities on Vacation: [Report: 5 points; Implementation: 15 points]

Modify your algorithm from part (a), so that you are able to determine a selection of activities that provides the maximum fun level. In your report, discuss the changes you made to your algorithm from part (a), and how the runtime complexity may or may not have changed.

Implement the changes, providing Fruitcake with a set of activities that achieve the maximum fun level. Again, you will be penalized if your modification results in a non pseudo-polynomial time algorithm.

Part 2: Scheduling the Vacation [50 points]

Based on all of your help, Fruitcake decided to narrow down his vacation destination choices to Maui and Oahu. On each day, he wants to stay either in Maui or Oahu (where he stays can vary by day). However, being ever so stingy, he wants to make sure to spend as little money as possible over the course of the vacation (after all he is funding this venture for the whole gang). He has determined the cost of a hotel stay on a given day at each of the islands. At first glance, Fruitcake could just

choose the cheaper option for each day. However, there is also a fixed cost for traveling between the two islands. Fruitcake wants you to determine a schedule that minimizes his total vacation cost.

Fruitcake provides you with the following:

- An array of values* MAUICOSTS where each element, MAUICOSTS[i], represents the cost of staying in Maui on day i.
- An array of values* OAHUCOSTS where each element, OAHUCOSTS[i], represents the cost of staying in Oahu on day i.
- A TRANSFERCOST value*: the cost of travelling between the two islands.

  *values are nonnegative integers

and expects the following in return:

- An array of booleans SCHEDULE where each element, SCHEDULE[i], represents whether the day i should be spent in Maui or Oahu. A value of *true* indicates Maui; a value of $false$ indicates Oahu.

The following is an example scenario.

Problem:
mauiCosts = [10,100,10]
oahuCosts = [400,20,400]
transferCost = 50

Result:
schedule = [true,true,true]

The following deliverable is required to help Fruitcake Frank plan his vacation activities.

(a) Minimize Vacation Cost: [Report: 10 points; Implementation: 40 points]

In your report, give the pseudocode for an efficient algorithm determining a schedule that minimizes the total vacation cost based on the given input. State the runtime complexity of this algorithm in Big-O notation.

Implement your algorithm, providing Fruitcake with a schedule indicating the days to spend in Maui and the days to spend in Oahu. You will be heavily penalized for a non-polynomial time algorithm.

Program Details

You will be provided with helper classes that evaluate results and simplify packaging of inputs and outputs as well as code skeletons where you fill in your own solutions. The following Java files are provided:

- Program3.java - This is where you implement your algorithms for part 1 and part 2 in the SELECTACTIVITY and SELECTSCHEDULING methods, respectively. Note that SELECTACTIVITY is used for part 1(a) and part 1(b). Our grader will inspect the contents of the MAXFUNLEVEL and SELECTEDACTIVITIES data structures independently, meaning that as long as your code compiles, you can get credit for (a) if (b) is incorrect, and vice versa. For part 2, you are only concerned with populating a single data structure named SCHEDULE.
- TestRunner.java - The runner class that evaluates your algorithm on provided test cases. We have given a sample test case for each part to help you get started. Note that these test cases are not necessarily indicative of the size or value of inputs used for grading. We highly encourage you to create your own test cases, especially with large input sizes. Our grading process will use a similar runner class with additional test cases. (Hint: you might consider creating your own test generator method to automatically generate many test cases at once.)
- ActivityProblem.java - Value class that holds the inputs to the activity selection algorithm.
- ActivityResult.java - Value class that holds the result of the activity selection algorithm.
- ActivityTestCase.java - Class containing a problem and result (ground truth) that can perform a check against another result (from your algorithm).
- SchedulingProblem.java - Value class that holds the inputs to the scheduling selection algorithm.
- SchedulingResult.java - Value class that holds the result of the scheduling selection algorithm.
- SchedulingTestCase.java - Class containing a problem and result (ground truth) that can perform a check against another result (from your algorithm).

Note that the only file used to evaluate your code is your Program3.java. While you may add helper methods to the Program3 class, DO NOT include additional java class files (any additional files will be ignored).

Submission Details

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains your Program3.java and a typed pdf report `eid_lastname_firstname.pdf`. Do not put these files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your solution must be submitted via Canvas BEFORE 11:59 pm on April 25, 2017. Absolutely no late assignments will be accepted for any reason.

- Make sure your program compiles on the LRC machines before you submit it.
- It is **highly recommended** that you do comment your code and follow good programming style. It will be very unlikely that you receive partial credit if there are not descriptive comments and the TAs cannot understand what your code is/should be doing.