EE360C Algorithms
University of Texas at Austin — Programming Assignment #2
Dr. Santacruz — Due: March 30th, 2022 11:59 PM

Programming Assignment #2

Programming assignments are to be done individually. Do not make your code publicly available (such as a Github repo) as this enables others to cheat and you will be held responsible. 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.

This programming assignment is due **Wednesday, March 30th at 11:59 PM.** If you are unable to complete the assignment by this time, you may submit the assignment late until Friday, April 1 at 11:59 PM for a 20 point penalty.

The goals of this lab are:

- Familiarize yourself with programming in Java.
- Show your understanding of heap implementation.
- Ensure that you understand certain aspects of graphs, greedy algorithms, especially MST.

Problem Description

You are trying to plan a road trip through a deserted wasteland in your car. You are a great mechanic and can upgrade your tank size with necessary parts. In this wasteland, there are several gas stations with unlimited free gasoline and parts to upgrade your tank. Your job is to figure out whether the road trip you are planning is feasible.

There are n gas stations numbered from 0 to $n-1$. You initially have a car that has a tank size that can travel x miles. Each gas station also has a value $k \geq 0$ indicating the value of the parts available at that place that can upgrade your car to travel an additional k miles. You can only use these parts once, but it permanently upgrades your the size of your gas tank.

Each gas station is located at a lattice point (x, y) where x and y are both integer values, measured in miles from each other. Starting at some gas station s, find if you are able to reach another gas station t. (You can use the parts at your start station to upgrade your car)

In this project, you will implement a minimum heap with the `distance` variable in GasStation. You will implement an algorithm to find all reachable stations given a beginning station and tank size. You will also implement a method to find the minimum tank size you will need in order to go from one station to another. We provide you with the following classes.

- `GasStation`
  Keeps track of `double distance` (the closest distance to any other reachable station), and `int id` (the numerical id of the station on the desert map). Remember, even though the coordinates are given in `int`, the distance is measured in `double`.

- `Heap`
  Needs to have all the properties of a heap. This class needs to function independently; the methods we ask you to implement will not all be needed for your algorithm, but we will test them separately.

- `Program2`
  The class that you will be implementing your algorithms in.

- `Driver`
  Reads file input and populates ArrayLists `stations`. You can modify `testRun()` for testing purposes, but we will use our own `Driver` to test your code.

You need to implement the `Program2` class. `Driver.testRun()` can be modified for your own testing purposes, but we will run our own `Driver` and test cases.

Part 1: Write a report [20 points]

Write a short report that includes the following information:

(a) Give the pseudocode for your heap implementation. Give brief explanations of the run time of building the heap, extracting the min, deleting, and changing a key.

(b) Give the pseudocode for your implementation of an algorithm to find the all reachable stations given a start station and an initial tank size. Give the runtime of your algorithm and justify.

(c) Give the pseudocode for your implementation of an algorithm to find the minimum tank size needed to go from start station to destination station. Give the runtime of your algorithm and justify.

Part 2: Implement a Heap [30 points]

Complete `Heap` by implementing the following methods, you can add methods if needed:

- `void buildHeap(ArrayList<GasStaiony> stations)`
  Given an ArrayList of GasStations, build a minimum heap in $O(n)$ time based on each GasStation's `distance`.

- `void insertNode(GasStation in)`
  Insert a GasStation in $O(\log(n))$ time.

- `GasStation findMin()`
  Find minimum in $O(1)$ time.

- `GasStation extractMin()`
  Extract minimum in $O(\log(n))$ time.

- `void delete(int index)`
  Delete the GasStation at a given index in $O(\log(n))$ time.

- `void changeKey(GasStation c, double newCost)`
  Updates a GasStation and rebalances the Heap in $O(\log(n))$ time. (HINT: Try to be more efficient than scanning the entire heap every time you want to update a GasStation.)

Part 3: Finding All Reachable Gas Stations [25 points]

Implement an algorithm to find the all reachable gas stations in the map (including the start gas station itself) given an initial tak size. You will be heavily penalized for a non-polynomial time (in terms of the number of stations) algorithm. The specific inputs provided and outputs expected are provided below.

Input(s):

- A starting gas station `GasStation start`
- Initial size of the gas tank `int init_size`

Output:

- An `ArrayList<GasStation> Reachable` where `Reachable` represents all the gas stations you can get to from GasStation `start`.

Method Signature:

- `ArrayList<GasStation> findAllReachableStations(GasStation start,int init_size);`

Part 4: Finding Minimum Tank Size [25 points]

Implement an algorithm to find the smallest tank size needed to get from a start gas station to an end gas station.

Input(s):

- A starting gas station `GasStation start`
- A destination gas station `GasStation dest`

Output:

- An `int tankSize` where the returned value `tankSize` represents the minimum gas tank size needed at the beginning of the trip in order to reach the destination gas station `start`. You should also use the upgrade available at the start station. *Note that the distance calculated between gas stations should be* `double`*, so round up* `int tankSize` *at the end.*

Method Signature:

- `int findMinimumTankSize(GasStation start, GasStation dest);`

Of the files we have provided, `GasStation`, `Heap`, and `Program2` are what will be used in grading. Feel free to add any methods or fields but be careful not to remove any to ensure that your classes remain compatible with our grading. Also, feel free to add any additional Java files (of your own authorship) as you see fit.

Input File Format

The first line of file is the number of total gas stations `int numV`. In the next `numV` lines represent the gas stations: The first number on each line is the x coordinate of the GasStation, the second number is the y coordinate of that GasStation. The following number is the value of upgrade you can get at that GasStation.
In the next line, the number `testSize` represents the number of test cases we will do for `testReachables` and `testMinTSize`.

The following `testSize` lines are for `testReachables`, where the first number is the id of start GasStation, and the second number is the initial tank size you obtain.

The following `testSize` lines are for `testMinTSize`, where the first number is the id of the start GasStation, and the second number is the id of the destination GasStation.

For example, if the input file is as follows:

```
4
6 -15 3
-15 -5 3
0 -12 4
1 -7 5
3
1 2
2 3
2 0
3 0
0 0
2 1
```

Then there are 4 gas stations.
GasStation 0 has coordinates of (6,15) and parts value of 3.
GasStation 1 has coordinates of (-15,-5) and parts value of 3.
GasStation 2 has coordinates of (0,-12) and parts value of 4.
GasStation 3 has coordinates of (1,-7) and parts value of 5.

There are 3 test cases each for `testReachable` and `testMinTSize`.
Test all reachable station from station 1 with initial tank size of 2.
Test all reachable station from station 2 with initial tank size of 3.
Test all reachable station from station 2 with initial tank size of 0.
Test the minimum tank size needed to go from station 3 to station 0.
Test the minimum tank size needed to go from station 0 to station 0.(which should be 0)
Test the minimum tank size needed to go from station 2 to station 1.

We will parse the input files for you, so you don't need to worry too much about the input file format except when trying to make your own testcases.

Instructions

- Download and import the code into your favorite development environment. We will be grading in Java 1.8 on GradeScope. Therefore, we recommend you use Java 1.8 and NOT other versions of Java, as we can not guarantee that other versions of Java will be compatible with our grading scripts. If you have doubts, email a TA or post your question on Piazza.
- Make sure you don't add the files into a package (keep them in the default package). Some IDEs will make a new package for you automatically. If your IDE does this, make sure that you remove the package statements from your source files.
- If you do not know how to download Java or are having trouble choosing and running an IDE, email a TA, post your question on Piazza or visit the TAs during Office Hours.
- There are several `.java` files, you can make modifications to them or add additional source files in your solution if you so desire. You were allowed to add your own fields to the given .java files (e.g. GasStation.java), but do a quick run through to make sure you didn't end up changing up any variable names from the starter code by accident.
- `Driver.java` is the main driver program. Modify `testRun()` to suit your liking, but we will use our own `Driver` to test your code. A main portion of the lab is debugging—be sure to leave time for that.
- As we ask you to implement Heap, note that for this assignment that you should **NOT** be using Hashmaps/Hashsets or java.util.PriorityQueue.
- All your java files should be in the root of the folder you submit. No nested directories or packages. You should be submitting `Program2.java` and `Heap.java` at very least and `GasStation.java` if it was modified. You need to submit any other .java files you made, but do **not** submit Driver.java.
- We will be checking programming style. A penalty of up to 10 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).
- When you submit, you will receive results for a few visible test cases. If you have not passed the visible test cases, it is strongly suggested that you keep working on your program. If you do pass all the visible test cases, please keep in mind that the test cases we will use to grade could be harder, therefore that does not guarantee a perfect score.
- Before you submit, be sure to turn your report into a PDF.

What To Submit

You should submit to GradeScope `Program2.java` and `Heap.java`. If you modified it, include `GasStaion.java` (and, any extra `.java` files you added or modified). Please do not submit `Driver.java`. Failure to follow these instructions will result in a penalty of up to 10 points.

Your PDF report should be legibly scanned and submitted to GradeScope. Both your code and PDF report must be submitted by 11:59 PM on Wednesday, March 30th, 2022. If you are unable to complete the assignment by this time, you may submit the assignment late until Friday, April 1st, 2020 at 11:59 PM for a 20 point penalty.