Mastery III — Data Struct. & Algo. (T. III/18–19)

Name: ____________________________

ID: ______________________________

Directions:

- You have 170 minutes (i.e., 2 hour and 50 minutes) to complete the following examination.
- There are 4 problems. The maximum possible score is 40. We will grade you out of $T \leqslant 30$, where T is yet to be decided. Anything above T is extra credit. You should think of this as *three* real problems plus *one* extra credit.
- No collaboration of any kind whatsoever is permitted during the exam.
- **WHAT IS PERMITTED:**
  - Reading the official Java documentation
  - Accessing Canvas for submission.
- **WHAT IS *NOT* PERMITTED:**
  - Browsing (online) tutorials or reading stack overflow threads.
  - Accessing previously-written code on your own machine.
  - Communicating with other person or using any other aid.
- For each problem, the entirety of you solution must live in one file, named according to the instructions in this handout. When grading a problem, the script will only compile that one file for the problem. **Importantly:** your implementation must not be part of a package.
- We're providing a starter package, which you can download at

  `https://cs.muic.mahidol.ac.th/courses/ds/404.zip`

  The password is "notfound".
  When you unpack the package, you'll see one file for each problem.
- The starter code contains a `graph` package. Do **not** modify the graph package in anyway. If you have to create a new project, copy both the starter files and the graph package (folder) to your new location.
- To submit your work, zip all your Java files as one zip file called `mastery3.zip` and upload it to Canvas.

Problem 1: Rural Roads (10 points)

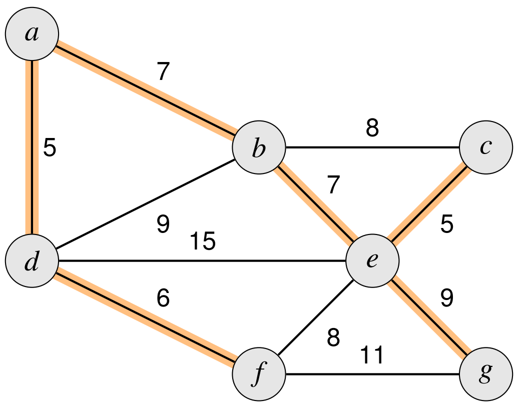
In a rural county far far away from Salaya, there are n isolated villages named $0, 1, 2, \ldots, n-1$. The local government hired a company to survey the cost to build roads to interconnect the villages. The survey result was presented as a weighted graph $G = (V, E, w)$, where $V = \{0, 1, \ldots, n-1\}$ are the vertices, each a village, E are possible intervillage roads (not all pairs of villages are possible), and w is a function $w\colon E \to \mathbb{Z}_+$ such that $w(e)$ is the cost to build a road between the endpoints of e.

Operating on a small budget, the government intended to build the cheapest structure that allows a trip between any pair of villages (that structure, as you know, is a minimum spanning tree). In this task, inside the class `RuralRoads`, you'll write a function

```
public static int totalWeight(List<WeightedEdge> edges)
```

that takes in a list of edges and returns an integer representing the total weight of the cheapest structure the government is building. Each edge is represented as an instance of the `WeightedEdge` class, provided in the starter package. If e is an instance of this class, it expresses the fact that there is an edge between e`.first` and e`.second`, and the cost of this edge is e`.cost`.

Example: On the following graph, the answer is 39.



Performance Expectation: Your code must run in $O(m \log n + n)$ time or faster. As is standard, $n = |V|$ and $m = |E|$. This means, on a graph with about $m = 200{,}000$ edges, your program must finish within 3 seconds. Furthermore, we guarantee that G is a connected graph.

Problem 2: *i*-th Rank on a Sorted Map (10 points)

The *i*-th rank element of an array `xs` is defined to be `sorted(xs)[i]`. Gift wants to give K2 a bit of an extra challenge, however, so instead of giving `xs` as input, Gift has decided to process it as follows:

> Take **long**`[] xs` as input and compute a histogram of the elements of `xs`. This histogram `hist` is kept as a `TreeMap<Long, Integer>` and has the property that `hist.get(k)` is the number of times `k` appears in `xs`.

Your task is to (help K2) implement a static method, inside the class `Rank`,

```
public static Long quickRank(TreeMap<Long, Integer> hist, int i)
```

that takes as input the processed form of `xs` (as described above) and returns the *i*-th rank element of the array. If the index *i* is not a valid index, return **null**.

Example: `quickRank({1: 5, 2: 200, 3: 7}, 100)` should return 2. Moreover, `quickRank({1: 2, 2: 5, 3: 1, 4: 9999}, 6)` should return 4.

Performance Expectation: The largest test case we'll use contains up to $200,000$ keys in the TreeMap. For every test case, your code should finish within 3 seconds to receive full credit. You should aim for an $O(m)$-time solution, where m is the number of keys in the TreeMap. Partial credit will be given to slower solutions.

Problem 3: The Perfect Hiding Spot (10 points)

The CS student committee for recreation and well-being is renting a huge castle for a week-long party this summer. The entrance opens into Chamber 1. In this chamber, they're planning to set up several large TV monitors so they can play Puyo Puyo Tetris without seeing daylight. Not everyone is on board with this idea, though. Nonny has been persuaded to join the party, but she wants to read by herself, away from the gaming crowd.

Your task is to help Nonny find the perfect hiding spot. You will be presented with a map of the castle. The chambers (vertices) are numbered 1 through n. You'll also be given a list of one-directional passages from one chamber to another (directed edges). Every passage is labeled with its distance. Importantly, each chamber has exactly one passage that opens into it. More precisely, the input is a directed tree where every vertex, except for vertex 1, has precisely one incoming edge. Note: vertex 1 has no incoming edge.

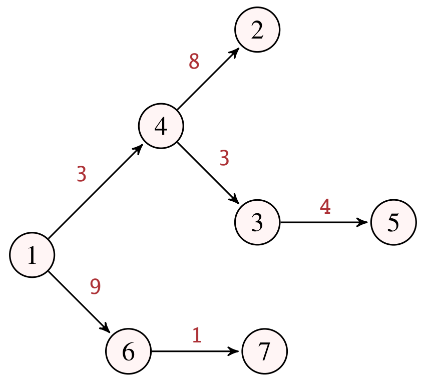
Inside the class `PerfectHiding`, you are to implement a static method

```
public static int bestSpotDistance(int n, List<WeightedEdge> passages)
```

where n is the number of chambers and `passages` is a list of directed edges. If `e` is a `WeightedEdge` in this list, then there is a passage from Chamber `e.first` to Chamber `e.second` of length `e.cost` meters. Notice that because this structure is a tree, the length of the list `passages` is always $n - 1$.

Your static method will compute the distance to the chamber (vertex) farthest away from chamber number 1. If it turns out that you cannot reach any chamber other than 1, the answer will be 0.

Example:



This graph has $n = 7$ nodes and the edges are:

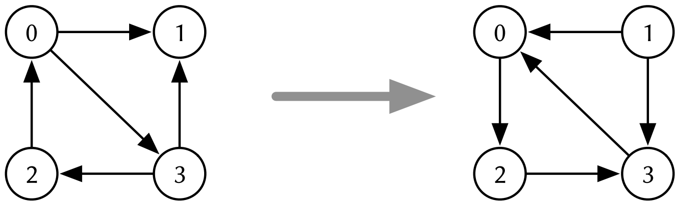
```
(1, 4, 3)   (4, 2, 8)   (4, 3, 3)
(3, 5, 4)   (1, 6, 9)   (6, 7, 1)
```

Here, the notation (u, v, c) denotes a directed edge from u to v with a weight of c meters. Calling `bestSpotDistance` on this input will result in the number 11 because Chamber 2 is the farthest from 1 via $1 \to 4 \to 2$, totaling $3 + 8$ meters.

Performance Expectation: We'll test your program with n up to $100,000$. On this size of input, your program must finish within 3 seconds to receive full credit. The cost of an edge will be a number between 1 and $1,000$ (inclusive).

Problem 4: Graph Reversal (10 points)

Given a **directed** graph $G = (V, A)$, the reverse of G, written G^{R}, is a graph on the same vertices but with the direction on each edge reversed. The figure below shows this transformation on an example graph.



This is a trivial task if the graph is represented as an edge list. For example, if the edges are `[(0, 1), (2, 0), (3, 2), (0, 3), (3, 1)]`, then the edges in the reversed graph will be `[(1, 0), (0, 2), (2, 3), (3, 0), (1, 3)]`.

In this problem, however, the graph is given as an *adjacency table*, i.e., `Map<Integer, Set<Integer>>`, where the outer map maps each vertex (an integer) to the set of its out-neighbors (vertices that it is pointing to). Inside the class `Reversal`, you are to write a static method

```
public static Map<Integer, Set<Integer>> reverseGraph(Map<Integer, Set<Integer>> G)
```

that takes as input a directed graph G as described and returns the reverse of G in the same format. Note: the original graph G should remain intact.

Performance Expectations: The largest test case we'll use contains up to $500,000$ edges. For every test case, your code should finish within 3 seconds to receive full credit. The `Map` here is a `HashMap` and the `Set` here is a `HashSet`. You should aim for an $O(m)$-time solution, where m is the total number of edges. Partial credit will be given to slower solutions.