COMP 251 Final Programming Assignment

General Information

Deadline:

- To be eligible for the code review (and thus to be eligible to receive above 80% on the project), you must **pass 80% of the tests and schedule your code review** by **December 17th at 11:59 PM**.
- The final deadline is **December 20th at 11:59 PM**.

It is important that you **read the entirety of this document** before beginning. **Sections 2** and **4** contain crucial information about support systems and rules for this project. Some of the guidelines and requirements differ from the mini programming assignments. If you do not read the entire document, you may miss out on important information which can lead to receiving a lower mark.

1 Background

McGill recently received an influx of COMP 251 students complaining about not being able to make it to class on time. The students say that 10 minutes is not enough time to get from lower campus all the way to the Stewart Biology building!

To everyone's surprise, the university decided to tackle this issue and reached out to the *Société de transport de Montréal* (STM) requesting that a campus-wide metro system be constructed to help the students get to class on time. The STM agreed to help under the condition that COMP 251 students would assist with the planning and software for the new metro. They came up with 5 tasks for you, a COMP 251 student, to complete.

Before you start tackling these challenges, it is important to be familiar with the system you will be working on, so the STM has included an on-boarding guide to help you get going.

2 Onboarding

2.1 Academic integrity

- Under no circumstance are you allowed to use artificial intelligence, including LLMs (ChatGPT, Claude, etc.), to assist you with this project. If you have **Copilot** or any other LLM code-autocomplete system installed in your development environment, you must disable it while working on this assignment. **We will be able to tell if you are using LLMs.** It is painfully obvious which students have used them in the mini programming assignments.
- Any resources you use aside from MyCourses and official documentation must be cited in your code. You cannot not directly copy code from outside resources (GeeksForGeeks, W3schools, etc.), but you may use them as **references** *as long as you cite them.* You are expected to **fully understand** how anything you use in your assignment works, and you are expected to be able to explain your code fully during the code review. If we determine that you do not understand the code you wrote, you will be suspected of cheating.
- You are allowed to **brainstorm** solutions with peers **in person**, but the code you submit must be written entirely by yourself. You may not discuss solutions to the assignment online and are not allowed to share code in any capacity (in person or online). Please identify anyone you have spoken to about the assignment in a comment in your code. You will additionally be asked to name them at your code review. This includes course staff.

When the instructors suspect that plagiarism has occurred, they will report the case to the Disciplinary Officer in the student's faculty. For more details on the process, see Section III Articles A.37 (p. 10) and A.48 (p. 13) of the *Code of Student Conduct and Disciplinary Procedures*.

2.2 Code review process

In order to qualify for the code review, you must have a score of at least 80% **and** book your code review by December 17th at 11:59pm. Your highest scoring submission before **that deadline** will be the maximum grade you can receive on this assignment. Scoring below 80% will disqualify you from the code review and the grade on the test cases is the grade you will receive. If you score perfectly in the code review, you will receive the grade you got on the test cases. If, during the code review, it is decided that you did not write the submitted code, you may receive a zero for the code review (capping your grade for this assignment at 80%) or the concern will be escalated to the academic integrity office. This will be handled on a case-by-case basis. Failing to follow the instructions in this document will also result in a fail for the code review.

During the code review, it is expected that you will be able to
- Explain every part of your code and why you decided on your approaches, and
- Provide time complexities and reasoning for each algorithm you use.

All comments will be removed from your code during the code review; you should be able to understand and explain your implementation without them.

If you score 80% or higher and you decide not to attend the code review, you will receive an 80% for this project. However, **all** submissions will still be examined for academic integrity violations.

2.3 Setting up your development environment

The code is available as a tarball on MyCourses which you should download and edit locally. IntelliJ is the recommended IDE which you have free access to through McGill. To get started, open IntelliJ and find `file > open` in the top bar. Navigate to the extracted file from MyCourses and open it as the project root. There will be a tutorial on **Friday, December 13th** going over how to write tests for this assignment as well as some useful features in IntelliJ.

2.4 Familiarize yourself with the codebase

2.4.1 `McMetro`:

This is the main class that houses all the software tools for the metro system. It contains the following fields:
- **`tracks`**: An array of tracks (type: `Track` Section 2.4.2) planned to be built
- **`buildingTable`**: A hashmap mapping `BuildingIDs` to `Buildings` (type: `HashMap<BuildingID, Building>` Section 2.4.3)

There are a 5 of methods you will implement in this class that we will discuss in Section 3.

2.4.2 `Track`:

Tracks represent the metro connections between buildings. They are comprised of the fields:
- **`id`** (`TrackID`): Unique identifier for the track.
- **`startBuilding`** (`BuildingID`): UID for the building at the start of the track
- **`endBuilding`** (`BuildingID`): UID for the building at the end of the track
- **`cost`** (`int`): The cost of constructing the track
- **`capacity`** (`int`): The maximum amount of passengers that can travel on the track at once.

2.4.3 `Building`:

Represents a location that can be connected by a `Track`. Has the fields:

- **`id`** (`BuildingID`): UID for the building
- **`occupants`** (`int`): Number of people in the building

2.4.4 `Disjoint Set`:

This is a naive implementation of a disjoint set data structure that will help you in your solutions. You must implement path compression and union-by-size or union-by-rank; otherwise, your code will be too slow for the STM to use.

2.5 Testing locally

There is a testing template file included with the project files. The intention is that you should test your code locally before submitting. To encourage you to do so, you are **limited to 50 submissions on Ed**. That is, if you submit 50 times, your 50th submission will be counted as your final code.

The example tests provided will help you make sure that your solution is handling the inputs and outputs correctly and give you an idea of how to write your own tests. As mentioned in Section 2.3, there will be a tutorial with extra information on testing.

3 Objectives

All implementations for the below tasks must be optimal in time complexity. For example inputs and outputs, please refer to the testing template, more information in Section 2.5. You may not use any packages which are not already imported. `java.util.*` and `java.lang.Math.*` should contain everything you need. **All helper methods on `McMetro` should be private**.

3.1 `DisjointSet<T>`

You are provided with a generic disjoint set class implemented naively. You must add path compression for the `find` method and union by size or union by rank for the `union` method. While this class will not be tested directly, you will need to use it in one or more of the subsequent tasks. The provided naive implementation will not be efficient enough to pass time complexity checks. **The signature of this class, and the signatures of the methods in this class must remain the same; do not rename the class, you may add private fields and helper methods**. You are not permitted to use a disjoint set library, and must use this class whenever a disjoint set is required. This will not be checked in test cases but could affect your code review grade. Learning about Java's generic types will help you with this task.

3.2 `maxPassengers(BuildingID start, BuildingID end)`:

Given the IDs of two buildings, find the maximum number of people that can be transported from `start` to `end` for the given metro network. This is a method on the `McMetro` class, the `buildingTable` and `tracks` fields define the metro network. The number of people that can travel between two buildings is the minimum between the number of occupants in each of the buildings, and the capacity of the track connecting the two buildings (the numerator in Section 3.3). If there is no track connecting two buildings, no one can travel directly between the buildings. The graph is sparse; this should influence your data structure choice.

3.3 `bestMetroSystem()`

The STM wants to figure out the best metro system they can construct, *now* assuming that tracks move in both directions and can therefore be treated as undirected edges. They do not want to build

more tracks than necessary. You must find a subset of `TrackID`s in the `tracks` field on `McMetro` class such that

- There are no cycles within the returned tracks,
- Every building has at least one track associated with it,
- The subset maximizes the number of people that can be transported taking into account the cost of building each track.

You can assume the network is connected. The order in which the `TrackID`s are returned does not matter. The "goodness" of a track can be calculated with the formula

$$\left\lfloor \frac{\min(\text{start building, end building, track capacity})}{\text{track cost}} \right\rfloor$$

3.4 `static addPassenger(String name)` + `static searchForPassengers(String firstLetters)`

In this this task, you will implement a system to add passengers and search for passengers in `McMetro`'s system. The `addPassenger` method will add a new passenger to the search system and the `searchForPassengers` method will return all the passengers in the system whose names begin with `firstLetters`. You can assume that we are only storing first names in the system (no spaces) and that if multiple users with identical names exist in the system, we only return one of them. When searching for passengers, the letter case of the names and of `firstLetters` should not matter (`Alex == aLeX`), but when we return search results, the first letter of each name should be capitalized. Names should be returned in alphabetical order.

3.4.1 A note on the `trie` data structure

In order to achieve optimal time complexity for this task, you will need to implement a **prefix tree**, also known as a **trie**.

The idea is to start with a tree data structure where a path from the root node to an "end node" in the tree represents a word. When we add a word to the trie, the first letter of the word will be added as a child of the root node (call it `c1`), the second letter will be added as a child of `c1` (call it `c2`). When we get to the last letter of the word, if the word is `k` letters long, we mark `ck` as an end node.

Think about how using this data structure will allow you to find all of the names corresponding to some prefix very quickly. Feel free to do your own research on this data structure to get a better understanding before starting on your implementation. A good resource to look at is the **stringology** lecture notes from COMP 252. Do not forget to add any resources you use as references in your code.

3.5 `static int hireTicketCheckers(int[][] schedule)`

The metro system will be very expensive to build. In order to protect their ROI, McGill insists that there be some level of security to check that passengers riding the metro have bought tickets, so they have decided to hire **ticket checkers**. They received many qualified applicants for this role, and the applicants have each indicated the times they are free. It was decided that there does not always need to be someone checking tickets, and there never needs to be more than one person checking tickets.

To reduce worker fatigue, the maximum number of ticket checkers should be hired such that their schedules should never overlap. As input you receive a list of all the schedules provided by the workers. Each entry in the schedule is a list with two elements of the form `[start, end]` where `start <= end` representing one applicant's availability. Due to the high number of applicants, each

applicant only provides one interval. The schedules need not be disjoint at the endpoints to be considered non-overlapping; that is, `[1,2]` and `[2,3]` are considered non-overlapping. We still hire someone if `start == end`, and `start` and `end` may be negative so long as their absolute values are at most `100000`. These constraints don't really work with the story, but please obey them anyways :)

The method returns the maximum number of workers we can hire with non overlapping schedules such that an applicant is hired over the entire interval that they provide.

4 Support

4.1 Ed

As always, we will be accessible on Ed for questions regarding the project. If your question contains information about your implementation or ideas, please make the post **private**, but if it is asking for clarification about the rules or guidelines, it can be made public.

4.2 Tutorials

Josh will be hosting a tutorial on how to test your code locally on **December 13th**. It is possible that more tutorial dates will be scheduled; all of these will be announced on Ed.

A video tutorial may be posted on the subject of **tries**. Again, this will be announced on Ed.

Although these tutorials may be helpful, you should be able to begin the assignment without attending them. You should not wait until December 13th to begin the assignment.

4.3 Office hours

Office hours will continue to be hosted throughout the remaining duration of the semester. Some of our TAs may be returning home early, in which case their office hours will be moved online. This will be reflected in the course calendar which can be found on MyCourses.

Will and Josh will begin hosting office hours together from 10-12 AM on Fridays, starting on December 13th. It is possible that Will and Josh will hold additional office hours in the earlier half of the week if there is enough demand for it.

5 Acknowledgements

This assignment was created by Joshua Dall'Acqua in collaboration with Will Zahary Henderson and Giulia Alberini.

Again, please don't cheat (we will know), and have fun!

6 Edits

- (Dec 5, 6:16PM) Clarified some wording on the notion of "connectedness" in `bestMetroSystem`. Clarified wording on signatures in `DisjointSet`.
- (Dec 6, 9:30AM) Added floor division to Section 3.3 to remove ambiguity