Comp 251: Assignment 2

Answers must be submitted online by October 26th (11:55:00 pm), 2018.

- Your solution must be submitted electronically on MyCourses.
- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will give you an automatic 0.**
- The starter code includes a tester class. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging set of examples. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the myCourses discussion board. Do not include it in your submission.
- Your code should be properly commented and indented.
- **Do not change or alter the name of one of the files you must submit**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded. Make sure to double-check your zip file.
- Do not submit individual files. Include all your files into a .zip file and, when appropriate, answer the complementary quiz online on MyCourses.
- **You will automatically get 0 if the files you submitted on MyCourses do not compile.**
- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments (i.e. as a comment at the beginning of your java source file) the names of the people with whom you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators". If asked, you should be able to orally explain your solution to a member of the course staff.
- It is your responsibility to guarantee that your assignment is submitted on time. We do not cover technical issues or unexpected difficulties you may encounter. Last minute submissions are at your own risk.
- Multiple submissions are allowed before the deadline. We will only grade the last submitted zip file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue.
- Late submissions will receive a penalty of 20% per day. We will not accept any submission more than 72 hours after the deadline. The submission site will be closed, and there will be no exceptions, except medical.
- In exceptional circumstances, we can grant a small extension of the deadline (e.g. 24h) for medical reasons only. However, such request must be submitted before the deadline, and justified by a medical note from a doctor, which must also be submitted to the McGill administration.

- Violation of any of the rules above may result in penalties or even absence of grading. If anything is unclear, it is up to you to clarify it by asking either directly the course staff during office hours, by email at (`cs251@cs.mcgill.ca`) or on the discussion board on myCourses (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on the website. It is your responsibility to monitor the course website and MyCourses for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum on MyCourses. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time. In particular, we will not answer individual emails about the assignment that are sent sent the day of the deadline.
- You should compile all your files as a single package from command line, by using the command `javac -d . *.java`

Exercise 1 (Disjoint sets (20 points)) We want to implement a disjoint set data structure with union and find operations. The template for this program is available on the course website and named `DisjointSets.java`.
In this question, we model a partition of n elements with distinct integers ranging from 0 to $n-1$ (i.e. each element is represented by an integer in $[0, \cdots, n-1]$, and each integer in $[0, \cdots, n-1]$ represent one element). We choose to represent the disjoint sets with trees, and to implement the forest of trees with an array named `par`. More precisely, the value stored in `par[i]` is parent of the element `i`, and `par[i]==i` when `i` is the root of the tree and thus the representative of the disjoint set.
You will and implement union by rank and the *path compression* technique seen in class. The rank is an integer associated with each node. Initially (i.e. when the set contains one single object) its value is 0. Union operations link the root of the tree with smaller rank to the root of the tree with larger rank. In case of the rank of both trees is the same, the rank of the new root increases by 1. You can implement the rank with an specific array (called `rank`) that has been added to the template) or use the array `par` (This is tricky). Note that path compression does not change the rank of a node.
Download the file `DisjointSets.java`, and complete the methods `find(int i)` as well as `union(int i, int j)`. The constructor takes one argument n (a strictly positive integer) that indicates the number of elements in the partition, and initialize it by assigning a separate set to each element. The method `find(int i)` will return the representative of the disjoint set that contains `i` (do not forget to implement path compression here!). The method `union(int i, int j)` will merge the set with smaller rank (for instance `i`) in the disjoint set with larger rank (in that case `j`). In that case, the root of the tree containing `i` will become a child of the root of the tree containing `j`), and return the representative (as an integer) of the new merged set. Do not forget to update the ranks. In case of the ranks are identical, you will merge `i` into `j`.
Once completed, compile and run the file `DisjointSets.java`. It should produce the output available in the file `unionfind.txt` available on the course website.
Note: You will need to complete this question to implement Question 2.

Exercise 2 (Minimum Spanning trees (40 points)) We will implement the Kruskal algorithm to calculate the minimum spanning tree (MST) of a undirected weighted graph. Here, you will use the file `DisjointSets.java` completed in the previous question, and two other files `WGraph.java`, `Kruskal.java` available on the course website. You will need the classes `DisjointSets` and `WGraph` to execute `Kruskal.java`. Your role will be to complete two methods in the template `Kruskal.java`.

The file `WGraph.java` implements two classes `WGraph` and `Edge`. An object of `Edge` stores all informations about edges (i.e. the two vertices and the weight of the edge), which are used to build graphs. The class `WGraph` has two constructors `WGraph()` and `WGraph(String file)`. The first one creates an empty graph and the second uses a file to initialize a graph. Graphs are encoded using the following format. The first line of this file is a single integer n that indicates the number of nodes in the graph. Each vertex is labelled with an number in $[0, \cdots, n-1]$, and each integer in $[0, \cdots, n-1]$ represents one and only one vertex. The following lines respect the syntax "n_1 n_2 w", where n_1 and n_2 are integers representing the nodes connected by an edge, and w the weight of this edge. n_1, n_2, and w must be separated by space(s). An example of such file can be found on the course website with the file `g1.txt`. These files will be used as an input in the program `Kruskal.java` to initialize the graphs. Thus, an example of a command line is `java Kruskal g1.txt`.
The class `WGraph` also provide a method `addEdge(Edge e)` that adds an edge to a graph (i.e. an object of this class). Another method `listOfEdgesSorted()` allows you to access the list of edges of a graph in increasing order of their weight.
You task will be to complete the two static methods `isSafe(DisjointSets p, Edge e)` and `kruskal(WGraph g)` in `Kruskal.java`. The method `isSafe` considers a partition of the nodes p and an edge e, and should return `True` if it is safe to add the edge e to the MST, and `False` otherwise. The method `kruskal` will take a graph object of the class `WGraph` as an input, and return another `WGraph` object which will be the MST of the input graph.
Once completed, compile all the java files and run the command line `java A2.Kruskal g1.txt`. An example of the expected output is available in the file `mst1.txt`. You are invited to run other examples of your own to verify that your program is correct.

Exercise 3 (Greedy algorithms (30 points)) In this exercise, you will plan out your homework with a greedy algorithm. You are given as input a list of homeworks defined by two arrays, an array of weights (the relative importance of the homework towards your final grade), and an array of deadlines. Those arrays are not sorted. Each index on those arrays, which are of the same size, represents a single homework to submit. Weights and deadlines are both integers between 1 and 100. Each homework takes exactly one hour to complete. Your task is to output a `homeworkPlan`, which will take the form of an array of length equal to the weights and deadlines. Each index in this array represents the same homework as each index in the input arrays. For each homework, indicate the time at which you plan on completing the homework. This time should be defined as an integer between 1 and the latest deadline of your homeworks, divided into slots of one hour. If you do not plan on completing a specific homework at all, indicate 0 in the corresponding slot of the `homeworkPlan` . The homework is considered due at the end of the time slot. In other words, if the homework is due at t=14, then you can complete it before or during the slot t=14. For example, `Homework 2` is defined as the homework with deadline `deadlines[2]` and weight `weights[2]` If your solution plans on doing `Homework 2` first, then `homeworkPlan[2]=1` should appear in your output. You can only complete a single homework in a 1 hour slot. Note that sometimes you will be given too much homework to complete in time, and that is okay.

Your homework plan should maximize the sum of the weights of completed assignments.

To organize your schedule, we give you a class `HW_sched.java`, which defines an `Assignment` object, with a number (it's index in the input array), a weight and a deadline.In addition, we provide you the file `GreedyTester.java` that demonstrates how to use this class to initialize an ArrayList of homework objects of the appropriate size.

In order to organize your schedule efficiently, you will have to sort the homeworks. For your convenience, we provide a `compare` method, which takes as input two assignments, compares them, and

outputs the order they should appear it. You have to determine what comparison criterion you want to use to compare assignments in this problem. Given two assignments A1 and A2, the method should output:

- 0, if the two items are equivalent
- 1, if a1 should appear before a2 in the sorted list
- -1, if a2 should appear before a1 in the sorted list

Your `compare` method should be the only tool you use to re-organize lists and arrays in this problem.

You will submit all the Java files you modified in this assignment, in a single zip file.

Exercise 4 (10 points) *You will answer to this section through MyCourses. Note that you MUST use your own results to answer those questions. Answers to this quiz that would not match the output of your program will be considered as plagiarism (refer to course outline).*