Comp 251: Assignment 1

Answers must be returned online by February 7th (11:59:59pm), 2025.

General instructions (Read carefully!)

- **Important:** All of the work you submit must be done by only you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. For Comp251, we will use software that compares programs for evidence of similar code. This software is very effective and it is able to identify similarities in the code even if you change the name of your variables and the position of your functions. The time that you will spend modifying your code, would be better invested in creating an original solution.

  Please don't copy. We want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

  Never look at another assignment solution, whether it is on paper or on the computer screen. Never share your assignment solution with another student. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web, or get code from a private tutor, that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in CS involve students who have never met, and who just happened to find the same solution online, or work with the same tutor. If you find a solution, someone else will too. The easiest way to avoid plagiarism is to only discuss a piece of work with the Comp251 TAs, the CS Help Centre TAs, or the COMP 251 instructors.

- 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. At the end of this document, you will find a check-list of the behaviours/actions that are allowed during the development of this assignment.

- This assignment is due on February 7^{th} at 11h59:59 pm. 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 for the coding questions. We will only grade the last submitted file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue. Please notice that for the proof question, only a single submission is allowed.

- Late submissions can be submitted for 24 hours after the deadline, and will receive a flat penalty of 20%. We will not accept any submission more than 24 hours after the deadline. The submission site (Ed-Lessons and MyCourses) 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.

- 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-winter@mcgill.ca`) or on the discussion board (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on one of the communication channels used in the course. It is your responsibility to monitor MyCourses and the discussion board for announcements.

- The course staff will answer questions about the assignment during office hours or in the online forum. 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 the day of the deadline.

Programming component

- 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 or libraries, 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 result in an automatic 0.**
- Public tests cases are available on ed-Lessons. You can run them in your code at any time. If your code fails those tests, it means that there is a mistake somewhere. We highly encourage you to modify our tests and expand them. Do not include it in your submission.
- Your code should be properly commented on and indented.
- **Do not change or alter the name of the files that you must submit or the method headers in these files**. 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.
- **Do not add any package or import statement that is not already provided**
- Please submit only the individual files requested.
- **You will automatically get 0 if the files you submitted on ed-Lessons do not compile, since you can ensure yourself that they do. Note that public test cases do not cover every situation and your code may crash when tested on a method that is not checked by the public tests. This is why you need to add your own test cases and compile and run your code from command line on linux.**

Homework

Exercise 1 (60 points)**. *Building a Hash Table***

We want to compare the performance of hash tables implemented using chaining and open addressing. In this assignment, we will consider hash tables implemented using the multiplication and linear probing methods. Note that the multiplication method described here is slightly different from the one that was seen in class, but the principle remains the same. We will (respectively) call the hash functions h and g and describe them below. Note that we are using the hash function h to define g.

$$\begin{aligned} \text{Collisions solved by chaining (multiplication method):} \quad & h(k) = ((A \cdot k) \mod 2^w) >> (w - r) \\ \text{Open addressing (linear probing):} \quad & g(k, i) = (h(k) + i) \mod 2^r \end{aligned}$$

In the formula above, r and w are two integers such that $w > r$, and A is a random number such that $2^{w-1} < A < 2^w$. In addition, let n be the number of keys inserted, and m the number of slots in the hash tables. Here, we set $m = 2^r$ and $r = \lceil w/2 \rceil$. The *load factor* α is equal to $\frac{n}{m}$.

We want to estimate the number of collisions when inserting keys with respect to keys and the choice of values for A.

We provide you a set of two template files that you will complete. This file contains two classes, one for each hash function. Those contain several helper functions, namely `generateRandom` that enables you to generate a random number within a specified range. Please read the provided code describing the hashtable classes with attention.

Your first task is to complete the two java methods `Open_Addressing.probe` and `Chaining.chain`. These methods must implement the hash functions for (respectively) the linear probing and multiplication methods. They take as input a key k, as well as an integer $0 \leq i < m$ for the linear probing method, and return a hash value in $[0, m[$.

Next, you will implement the method `insertKey` in both classes, which inserts a key k into the hash table and returns the number of collisions encountered before insertion, or the number of collisions encountered before giving up on inserting, if applicable. Note that for this exercise, we define the number of collisions in open addressing as the number of keys encountered, or "jumped over" before inserting or removing a key (*note that this definition only makes sense if the key is in the hash table*). For chaining, we simply consider the number of other keys in the same bin at the time of insertion as the number collisions. You can assume the key is not negative, and that we will not attempt to insert a key that already exists in the hash table.

You will also implement a method `removeKey`, this one only in `Open_Addressing`. This method should take as input a key k, and remove it from the hash table **while visiting the minimum number of slots possible**. Like `insertKey`, it should output the number of collisions if the key is found.

If the key is not in the hash table, the method should simply not change the hash table, and output the **number of slots visited before giving up**.

You will notice from the code and comments that empty slots are given a value of -1. If applicable, you are allowed to use a different notation of your choice for slots containing a deleted element.

Make sure to test your assignment thoroughly by thinking about all the different situations that can occur when dealing with hash tables. Build your own hash table and try inserting and removing keys!

For this question, you will need to submit your `Chaining.java` and `Open_Addressing.java` source files to the `Assignment 1 => Q1 - Hash` lesson in Ed-Lessons. You will not be tested on execution time for this question, but you will be tested on the efficiency of your program in terms of number of steps. **You must implement your own hash table. Using the built-in hash table from Java will result in a 0 on this question.**

Exercise 2 (110 points)**.** ***Disjoint Set***

We want to implement a disjoint set data structure with union and find operations. The template for this program is available on MyCourses 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 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 the case where the rank of both trees is the same, the rank of the new root increases by 1. You can implement the rank with a 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.

(50 points) Part A: Building a Disjoint Set

Download the file `DisjointSetsB.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 initializes 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 the case where 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 MyCourses.

(60 points) Part B: Modifying a Disjoint Set

For this part of the assignment, we will modify the Disjoint Set data structure coded in the previous question. In particular, we want now also to support the following two additional operations/methods.

- The method `move(int i, int j)` will move the `i` to the set containing `j`. Please notice that if `i` and `j` are already in the same set, the command must be ignored.
- The method `sum_elements(int i)` must return the sum of elements in the set containing `i`.

Let's see an example reporting a sequence of commands to make sure that the information is clear.

- `new DisjointSets(6):  {0}, {1}, {2}, {3}, {4}, {5}`
- `union(1,2):  {0}, {1,2}, {3}, {4}, {5}`
- `move(3,4):  {0}, {1,2}, {3,4}, {5}`
- `union(3,5):  {0}, {1,2}, {3,4,5}`
- `sum_elements(4): 12`
- `move(4,1):  {0}, {1,2,4}, {3,5}`
- `sum_elements(4): 7`
- `sum_elements(3): 8`

Please notice the following important information.

- Complete your code using the `DisjointSetsB.java` template provided in MyCourses.
  - The main difference of this new file (with respect to `DisjointSetsA.java`) is that it contains the header for the `move(int i, int j)` and `sum_elements(int i)` functions.
  - You will need to copy and paste the code that you already generated for the `find(int i)` and `union(int i, int j)` functions in the previous question.
  - For this question, you are allowed to add helper functions and to modify (if needed) the class attributes in the provided code; however, you should not change the methods or class headers. It is your responsibility to guarantee that your modifications do not make the autograder to crash.
- Given that this question represents a modification of a `Union Find` data structure, you are **not** required to guarantee that the `move(int i, int j)` and `sum_elements(int i)` operations run in $O(\alpha(n))$. Your code will be evaluated in terms of correctness and not efficiency; however, your implementation must still run on the time resources provided by the autograder.

Exercise 3 (80 points)**. *Ed-discussion board***

The company who created our discussion board (Ed) contacted us to develop a new feature in their software. The feature must identify important topics discussed on Ed by filtering via specific keywords. Given a list of Ed messages, these keywords correspond to the words that were used by all users in their messages. The feature is expected to return a sorted list of keywords, from most to least used words (i.e., the word with the highest frequency must be the first one). In case of a frequency tie, the word needs to be sorted in alphabetical order.

Let us see now some features of the discussion board posts. The list of posts will be provided to you as an array of strings (`String[]`), where every slot in the array will contain a message. All messages will have the following characteristics.

- Each message is represented in Java as a `String`
- Each message begins with a user name of no more than 20 characters.
- After the name, each message continues with the content of that user's post all in lower cases.
- Each word in the content of the user's post is separated by a single space character.
- The total number of characters across all messages, including spaces, will not exceed $2 * 10^6$

Let's see now two examples to make sure that everything is clear. Given the following list of posts:

```
David no no no no nobody never
Alexia why ever not
Parham no not never nobody
Brian no never know nobody
Jeremy why no nobody
Jeremy nobody never know why nobody
David never no nobody
Alexia never never nobody no
```

Your algorithm must return the array `[no, nobody, never]` (exactly in that order). Those three words were used by every single user of our discussion board and they are reported in order of frequency (i.e., `"no"` is the most frequently used word). In case of a tie, the order was decided lexicographically.

Now, if the following list of posts is given to you:

```
David comp
Ziqi music
```

Your algorithm must return an empty array `[]` given that any of the words in the post were used by every single user.

For this question, you must implement your solution in the function `Discussion_Board(String[] posts)` which is inside the class/file `A1_Q3.java`. Please note that for this question the correctness and efficiency of your algorithm will be tested, then it is of your interest to code your solution using the right algorithms and data-structures.

What To Submit?

For the coding exercises: Attached to this assignment are Java template files. You have to submit only this java files. Please DO NOT zip (or rar) your files, and do not submit any other files.

For the proof exercise: You have to submit a PDF document (in the form of a report) answering the proposed questions/exercises. Please DO NOT zip (or rar) your files, and do not submit any other files.

Where To Submit?

For the coding exercises: You need to submit your assignment in ed - Lessons. Please note that you do not need to submit anything to myCourses.

For the proof exercise: You need to submit your assignment in myCourses. Please note that you do not need to submit anything to ed - Lessons.

When To Submit?

Please do not wait until the last minute to submit your assignment. You never know what could go wrong during the last moment. Please also remember that you are allowed to have multiple submission. Then, submit your partial work early and you will be able to upload updated versions later (as far as they are submitted before the deadline).

How will this assignment be graded?

Please remember that we are only considering the highest 4 scores as the overall grade of your assignment.

For the coding exercises: Each student will receive an overall score. This score is the combination of the passed open and private test cases. The open cases correspond to the examples given in this document plus other examples. These cases will be run with-in your submissions and you will receive automated test results (i.e., the autograder output) for them. You MUST guarantee that your code passes these cases. In general, the private test cases are inputs that you have not seen and they will test the correctness of your algorithm on those inputs once the deadline of the assignment is over; however, for this assignment you will have information about the status (i.e., if it passed or not) of your test. Please notice that not all the test cases have the same weight.