Comp 251: Assignment 1

Answers must be returned online by February 14th (11:59:59pm), 2023.

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.

- Your solution must be submitted electronically on ed-Lessons.

- 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 14^{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.

- This assignment includes a programming component, which counts for 100% of the grade.

- Multiple submissions are allowed before the deadline. 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.

- 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 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@cs.mcgill.ca`) or on our 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 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 on 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 that tester class, expand it and share it with other students on the 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 the files 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 (65 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 within 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 (40 points)**.** ***Building a Disjoint Set***

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 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.
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 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.

Exercise 3 (95 points)**.** ***Vacuuming ideas (Legos)***

We bought a new Lego kit to our son last Saturday. He opened it on Saturday evening and he started "playing" with it (or that was what we thought). On Sunday morning, I woke up to discover that all the Lego pieces were scattered around the apartment. Then, I took the vacuum with the plan of capturing all the pieces and recovering them at the end from the vacuum bag.

Being accustomed to vacuuming, I already have a well-defined path to traverse my apartment when I am vacuuming (i.e., the pieces of Lego will be collected in a well-defined order). After seeing the Lego pieces lying on the floor, it was clear to me that I should be able to classify the pieces by size. Then, I started vacuuming my apartment and I documented (in an array) the size of every Lego piece that I vacuumed. I noticed that during some time (after starting the vacuuming), I only collected pieces of different sizes. In other words, during some initial time, the bag of my vacuum machine was composed of **unique** sized Lego pieces. I realized that depending on where I decided to start the vacuuming process, this number of **unique** items would change. If I am willing to start vacuuming in any room; however, I must always follows the well-defined path to traverse my apartment (and there will be rooms that will not be cleaned), then I would like to know the following:

What is the **maximum** number of **unique** Lego items that could be in the vacuum bag (i.e., the length of the longest contiguous segment of unique Lego items if I can start at any room)?

Could you please help me create a java function that **efficiently** computes that number? Let's see one example to make sure that the problem is clear.

Example 1: Imagine that the following array represents the sizes of the Lego pieces vacuumed in my apartment `sizes = {1, 2, 3, 2, 1, 2, 4}`. The maximum number of **unique** Lego items that could be found at a specific time in the vacuum bag is 3. The answer represents the following scenario: You start vacuuming in the first room and you collect the first three unique-sized Lego items (i.e., `{1, 2, 3}`). Please notice that once you vacuum the forth item (i.e., `{2}`), you will find a duplicate in the bag. This solution is not unique because you could also start vacuuming in the room where the third piece is located and collect the next thee items (i.e., `{3,2,1}`). Please remember that if j is the first Lego piece that is vacuumed, then the pieces $1 \dots j-1$ will not collected by the vacuum machine.

Understanding that this problem can happen to other parents with kids (who may have huge Lego pieces), I would like to make your solution work with the following conditions:

- Kids can have up to one million pieces of Lego.
- The sizes of the Lego pieces are represented by an integer in the range 0 to 10^9, inclusive.
- Two Lego items are identical if and only if they are identified by the same integer.

I hope the following hints would be useful for you.

- The challenge of this question is not the coding component (which is super-short), but to come up with a clever solution. Please spend your time designing a nice (i.e., correct and efficient) answer, not coding and debugging your ideas. Also, please give yourself the chance to figure out the answer. Asking for (or providing) a solution to your classmates will not help your learning process (or that of your classmate).
- I think a linear solution is possible. Then, to guarantee that your code passes all the test cases (and for personal satisfaction), try to design your code such that it is $O(n)$.
- You are free to use more than one data structure ;).

For this question, you must implement your solution in the function `elements(int[] sizes)` (yes, I know, it is a terrible name for a function, but I spent all my creativity explaining this problem) which is inside the class/file `A1_Q3.java`. Please notice that for this question, the correctness and efficiency of your algorithm will be tested, hence you should code your solution using the appropriate algorithms and data-structures. Please notice that for this question, **it is forbidden to create new classes and/or importing other libraries**.

What To Submit?

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.

Where To Submit?

You need to submit your assignment in ed - Lessons. Please review the tutorial if you still have questions about how to do that (or attend office hours). Please note that you do not need to submit anything to myCourses.

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 submissions. Then, submit your partial work early and you will be able to upload updated versions later (as long as they are submitted before the deadline).

How will this assignment be graded?

Each student will receive an overall score for this assignment. This score is the combination of the passed open and private test cases for the three questions of this assignment. 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.

Student Code of Conduct Assignment Checklist

The instructor provides this checklist with each assignment. The instructor checks the boxes to items that will be permitted to occur in this assignment. If an item is not checked or not present in the list, then that item is not allowed. The instructor may edit this list for their case. A student cannot assume they can do something if it is not listed in this checklist, it is the responsibility of the student to ask the professor (not the TA).

Instructor's checklist of permitted student activities for an assignment:

Understanding the assignment:

- ☒ Read assignment with your classmates
- ☒ Discuss the meaning of the assignment with your classmates
- ☒ Consult the notes, slides, textbook, and the links to websites provided by the professor(s) and TA(s) with your classmates (do not visit other websites)
- ☐ Use flowcharts when discussing the assignment with classmates.
- ☒ Ask the professor(s) and TA(s) for clarification on assignment meaning and coding ideas.
- ☐ Discuss solution use code
- ☐ Discuss solution use pseudo-code
- ☐ Discuss solution use diagrams
- ☐ Can discuss the meaning of the assignment with tutors and other people outside of the course.
- ☐ Look for partial solutions in public repositories

Doing the assignment:

- Writing
  - ☒ Write the solution code on your own
  - ☒ Write your name at the top of every source file with the date
  - ☐ Provide references to copied code as comments in the source code (e.g. teacher's notes, websites, etc.)
  - ☒ Copied code is not permitted at all, even with references
  - ☐ Permitted to store partial solutions in a public repository
- Debugging
  - ☒ Debug the code on your own
  - ☒ Debugging code with the professor
  - ☒ Debugging code with the TA
  - ☒ Debugging code with the help desk
  - ☐ Debugging code with the Internet
  - ☐ You can debug code with a classmate
  - ☐ You can debug code with a tutor or other people outside of the course
- Validation
  - ☒ Share test cases with your classmates
- Internet
  - ☐ Visit stack-overflow (or similar)
  - ☐ Visit Chegg (or similar)

- Collaboration
    - ☐ Show your code with classmates
    - ☐ Sharing partial solutions with other people in the class
    - ☐ Can post code screenshots on the course discussion board
    - ☒ Can show code to help desk

Submitting and cleaning up after the assignment:

- ☐ Backup your code to a public repository/service like github without the express written permission from the professor (this is not plagiarism, but it may not be permitted)
- ☐ Let people peek at your files
- ☐ Share your files with anyone
- ☐ ZIP your files and upload to the submission box
- ☒ Treat your work as private
- ☐ Make public the solutions to an assignment
- ☐ Discuss solutions in a public forum after the assignment is completed