Comp 251: Assignment 1

Answers must be returned online by February 14th (11:59:59pm), 2024.

General instructions (Read carefully!)

- **Super Important:** This assignment contains 5 questions. The questions are split in the following components.
  - **Programming Component**: Questions 1 (i.e., Building a Hash Table) and question 2 (i.e., Building a Disjoint Set) belong to the programming component. For this questions, your solution must be submitted electronically on ed-Lessons. Each question in this component is worth 50 points. The questions will be evaluated (by an autograder) mostly based on their correctness.
  - **Real World Component**: Questions 3a (i.e., Minimum distance) and 3b(Maximum different exams) belong to the real world component. For this questions, your solution must be submitted electronically on ed-Lessons. Each question in this component is worth 50 points. The questions will be evaluated (by an autograder) based on their correctness and efficiency.
  - **Proof Component**: Question 4 (i.e., Least common multiple) belongs to the proof component. For this questions, your solution must be submitted electronically on My-Courses. This question is worth 50 points. The question will be marked by one of our T.As.
- **Even more important than Super Important**: This assignment is worth 200 points. The assignment has the following conditions.
  - You can select which four questions (out of the proposed five questions) you will work on.
  - For the grading, we will consider only the four top scores of the solved questions. Let see some scenarios to make it clear:
    - ∗ Student A solved the questions 1, 2, 3a and 4. The student got full marks (50 points) for each of the questions, then the student will get 200 points for the assignment (i.e., full marks of the assignment).
    - ∗ Student B solved the questions 1, 2, 3a, 3b and 4. The student got 25 points in each of the solved questions, then the student will get 100 points for the assignment (i.e., 50% of the assignment full marks). This is due because we are considering the top four scores to compute the assignment mark.
  - If you do not solve the proof question, please do not submit any file to MyCourses. This will make easier the grading process for the fourth question.

- **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 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.

- 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@cs.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 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 our tests and expand them. 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.**

Proof component

- For the proof exercise, the student will have to answer the proposed questions/exercises in the form a report, all contained in one pdf document. Please notice that pdf is the only accepted format. We recommend using the free, online service Overleaf to write the report.

- The student has to provide the answers in the order of the asked/proposed questions/exercises.

- The clarity and presentation of your answers is part of the grading. Answers poorly presented may not be graded. This includes the clarity of the writing or the quality of the image you uploaded. It is your responsibility to ensure that the image has the good resolution and contrast.

Homework

Exercise 1 (50 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 (50 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 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.
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 (100 points)**. *New Exam Protocol at McGill***

The Exam Office at McGill University has commissioned the Comp251 students to validate a new protocol to present the Final exams. The main goal of the new protocol is to decrease the cases of cheating by changing the seating arrangement in the gym. McGill wants all the students to be seated in a long row along a single table, in a way that no student sits next to someone taking the same exam as themselves; this is to ensure that no two students (taking the same exam) can discuss or show their exams between them.

I run some simulations to validate the protocol. I am currently using an array of `integers` to store the possible arrangements. For example, the array `[202,204,202,251,250,321,251]` represents the long row where 7 McGill students are taking exams for the Comp202, Comp204, Comp250, Comp251 and Comp321 courses. Two seats next to each other are said to be separated by a distance of 1. Then, notice that the two students taking the Comp202 exam are separated by 2 and the two students taking the Comp251 exams are separated by 3. I will need your help to solve the following two questions.

a) New Exam Protocol at McGill - Minimum distance [50 points]

For this question, you must compute the number of the smallest distance of separation between two students taking the same course. In our example (`[202,204,202,251,250,321,251]`), that minimum distance is equal to two and it corresponds to the distance between the two students taking the Comp202 exam. If no two students are taking the same exam, the distance is defined to be the number of students taking exams.

For this question, you must implement your solution in the function `protocol_3a(int[] arrangement)` which is inside the class/file `A1_Q3a.java`. Please notice 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. Please notice that for this question, **it is forbidden to create new classes and/or importing other libraries**.

b) New Exam Protocol at McGill - Maximum different exams [50 points]

For this question, we are interested in computing the maximum number of consecutive different exams that you can find in the gym. In our example (`[202,204,202,251,250,321,251]`), the longest contiguous segment of unique exams is in red (`[202,204,202,251,250,321,251]`) and it has a length of 5. Please notice that if we increase the segment by adding the course on the left (i.e., 202) or on the right (i.e., 251), the answer stop to be valid because the duplicates. For this question, you must implement your solution in the function `protocol_3b(int[] arrangement)` which is inside the class/file `A1_Q3b.java`. Please notice 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. Please notice that for this question, **it is forbidden to create new classes and/or importing other libraries**.

General Comments from David

I hope the following comments 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 to yourself the chance to figure it out the answer. Asking for (or providing) a solution to your classmates will not make any good to your learning process (or the one of your classmate).
- I think a linear solution is possible in both cases (i.e., the minimum and the maximum distance). Then, to guarantee that your code passes all the test cases (and for personal satisfaction), try to design your code such that both solutions run in $O(n)$.

- You are free to use more than one data structure ;) if needed.

Exercise 4 (50 points)**.** ***Least common multiple***

This problem aims to study an algorithm that computes, for an integer $n \in \mathbb{N}$, the least common multiple (LCM) of integers $\leq n$.

For a given integer $n \in \mathbb{N}$, let $P_n = p_1^{x_1} p_2^{x_2} \cdots p_k^{x_k}$, where $p_1, p_2, \cdots, p_k$ is a strictly increasing sequence of prime numbers between 2 and n and for each $i \in \{1, \cdots k\}$, x_i is the integer such that $p_i^{x_i} \leq n < p_i^{x_i+1}$. For example, $P_9 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$.

More precisely, we're going to compute all P_j, $j \in \{1, \cdots, n\}$ and store pairs of integers (p^α, p) in a heap, a binary tree where the element stored in the parent node is **strictly smaller** than those stored in children nodes. For two given pairs of integers (a, b) and (a', b'), $(a, b) < (a', b')$ if and only if $a < a'$. Let h denotes the tree height, we admit that $h = \Theta(\log n)$. All levels of the binary tree are filled with data except for the level h, where elements are stored from the left to the right. After computing P_j, all pairs (p^α, p) are stored in the heap such that p is a prime number smaller or equal to j and α is the **smallest** integer such that $\underline{\mathbf{j} < \mathbf{p}^\alpha}$. For instance, after computing P_9, we store $(16, 2)$, $(27, 3)$, $(25, 5)$, and $(49, 7)$ in the heap.

The algorithm is iterative. We store in the variable **LCM** the least common multiple computed so far. At first, **LCM**$= 2$ is the LCM of integers smaller than 2 and the heap is constructed with only one node with value $(4, 2)$. After finish the $(j-1)$-th step, we compute the j-th step as follows:

1. If j is a prime number, multiply **LCM** by j and insert a new node (j^2, j) in the heap.
2. Otherwise, if the root (p^α, p) satisfies $j = p^\alpha$, then we multiply **LCM** by p, change the root's value by $(p^{\alpha+1}, p)$, and reconstruct the heap.

We're going to prove, step by step, that the time complexity of this algorithm is $O(n\sqrt{n})$.

4.1 - 15 points
Please report/draw the heap trees that are computed when you run the algorithm for P_9. In other words, please report/draw each heap tree that results after finishing the $j-th$ step (as defined above) when computing $P_2, P_3 \ldots P_8, P_9$.. Please also report the value of the **LCM** variable in each of those steps.

4.2 - 2.5 points
In operation 1, a new node is inserted and possibly reconstructed. What is the complexity of this operation in the best and worst case scenario? Please justify your answer.

4.3 - 2.5 points
In operation 2, the heap is reconstructed. What is the time complexity of this operation in the best and worst case scenario? Please justify your answer.

4.4 - 10 points
The number of prime numbers smaller than n, concerned in the operation 2, is less than $\sqrt{n}$. We will prove that the number of times N we need to execute operation 2 to compute P_n is $N < \log n\sqrt{n}$. We came up with the following true claims; however, those claims are missing an

explanation of why they are true. Then, your job for this question is to add clear justifications that support each of the provided claims. In other words, we are giving to you the main steps of the proof, then your job is to create a clear and justified proof based on those claims.

1. The total number needed N is $\sum_i \alpha_i - 1$
2. We have $\alpha_i \sim \frac{\log n}{\log p_i}$
3. Furthermore, the number of prime numbers concerned is smaller than $\sqrt{n}$ (*i.e.*, the prime number p_i with $\alpha_i \geq 1$).
4. From claims 1, 2 and 3, we can conclude that $N < \log n\sqrt{n}$

4.5 - 5 points

The next step is to show that $\log n\sqrt{n}$ is asymptotically negligible compared to n. In order to do that we need to check that N is $o(n)$. For this question, you will plot the two functions (n and $\log n\sqrt{n}$) for different values of n to visually inspect that n is an upper bound of $\log n\sqrt{n}$. Please report the plot and justify the procedure (not the code) that you followed to produce the plot. Note: We recommend using XChart, a free, easy to use java plotting library; however, the student can also use any graphing software they choose (even if it is not in java).

4.6 - 5 points

Assume the complexity of assessing whether an integer is a prime number is $\sqrt{n}$ and suppose multiplication has a time complexity of 1. Please report the complexity (in big O notation) for each of the following steps of the algorithm and a brief justification explaining why the reported complexity is a tight upper bound. Hint: Please remember that the algorithm's complexity is $O(n\sqrt{n})$, then at least one of the following steps must have that complexity.

- The total number of times needed for the operation 1 is equal to the number of prime number times the heap reconstruction time, thus complexity for the first operation is:
- From previous questions, we can derive that the complexity for the second operation is:
- In the algorithm, we verify if each number is prime, then the complexity for this step(s) is:
- We need to multiply the variable **LCM** $\sum_i \alpha_i$ times, so the complexity is:

To conclude, the complexity of the algorithm is $O(n\sqrt{n})$.

4.7 - 10 points

Prove that, for a given heap of height h with n nodes, we have $h = \Theta(\log n)$. No partial credit will be awarded.

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.

For the proof exercise: Some of our T.As will go over your report and they will grade it.

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). Please notice that you are not allowed to copy code from the internet.
    - ☐ 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. Please notice that this is allowed to debug syntax errors, no logic errors.
    - ☐ 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

- [x] Visit stack-overflow (or similar). Please notice that this is allowed only to debug syntax errors, no logic errors.
- [ ] 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
    - [x] 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
- [x] Treat your work as private
- [ ] Make public the solutions to an assignment
- [ ] Discuss solutions in a public forum after the assignment is completed