COMP3506/7505 Project – Part B

Due: 18th October 2021 @ 4:00 PM AEST Version: 1.0

Report Template

	Full Name	Student ID
Details	**Alexander F Viller**	**45375325**

1. Overview

This document is the ***mandatory*** template which must be used to submit the report section of your project.

This template is automatically synced with Gradescope to identify the location of each section of the report; therefore, it is imperative that the overall format/layout of this document not be modified. Modification of the template **will** result in a penalty being applied.

You are permitted to make changes inside the purple boxes for each question provided however the overall size of the box cannot change. Your report should easily fit within the boxes provided however please be aware the minimum font size allowed is Arial 10pt. If you are exceeding the box size, then this may be a good indication your response is not succinct.

2. Submission

Once you have completed your report this document must be exported as a pdf and uploaded to the Gradescope Report Portal for Part B of this assignment. This document **should not** be uploaded to the autograder.

3. Marking

The report will be hand marked by the teaching team. Information regarding the rubrics used while marking will be made available after grades are released. While this report will indicate the relative weighting of each section of the report, should there be any discrepancy with the official assignment specification, the assignment specification shall take precedent.

4. Plagiarism

The University has strict policies regarding plagiarism. Penalties for engaging in unacceptable behaviour can range from cash fines or loss of grades in a course, through to expulsion from UQ. You are required to read and understand the policies on academic integrity and plagiarism in the course profile (Section 6.1).

If you have any questions regarding acceptable level of collaboration with your peers, please see either the lecturer or your tutor for guidance. Remember that ignorance is not a defence!

5. Task

You are required to complete all sections of this report in line with the programming tasks completed in the project.

Security Database (20 Marks)

1. State the best and worst case time complexity of adding and removing a passenger. Briefly explain your answer.

For adding a passenger:

- Best case, O(1), this is due to being able to find the correct index for the passenger straight away and not needing to do any linear probing and not having to resize the array, but the array should only need resizing in the absolute worst case
- Worst case, O(n), this is because in the worst case you need to resize the array and recalculate all hash codes and replace them in the array, or if the array doesn't need resizing there is a chance through linear probing that you need to look through every index to find a free space.

For removing a passenger:

- Best case, O(1), this is because the passenger is found straight away and then their location is just set to null and no linear probing is required
- Worst case, O(n), this is because of the same reason as for adding with the chance that linear probing needs to iterate through the whole array

Use the following box to insert snippets of the code you refer to in the above question.

```java
public boolean addPassenger(String name, String passportId) {
    Passenger passenger = new Passenger(name, passportId);
    int tmp = compressionFunction(calculateHashCode(passportId));
    int i = tmp;

    do {
        if (keys[i] == null) {
            keys[i] = passportId;
            values[i] = passenger;
            currentSize++;
            if (currentSize == size()) {
                int k = size();
                maxSize = 1021;
                String[] tmp1 = new String[maxSize];
                Passenger[] tmp2 = new Passenger[maxSize];
                for (int j = 0; j < k; j++) {
                    tmp1[j] = keys[j];
                    tmp2[j] = values[j];
                }
                keys = tmp1;
                values = tmp2;
                int index = 0;      You, a week ago • part 1 done

                for (index = (index + 1) % size(); keys[index] != null; index = (index + 1) % size()) {
                    Passenger tmp3 = values[index];
                    keys[index] = null;
                    values[index] = null;
                    currentSize--;
                    addPassenger(tmp3.getName(), tmp3.getId());
                }
            }
            return true;
        }

        if (keys[i].equals(passportId)) {
            if (values[i].getName() == name) {
                values[i].incrementCounter();
                if (values[i].isSuspicious()) {
                    return false;
                } else {
                    return true;
                }
            } else {
                System.err.println("Suspicious behaviour");
            }
        }
        i = (i + 1) % size();
    } while (i != tmp);
    return false;
}
```

```java
public boolean remove(String passportId) {
    if (!contains(passportId)) {
        return false;
    }

    int index = compressionFunction(calculateHashCode(passportId));
    while (!(passportId == keys[index])) {
        index = (index + 1) % size();
    }
    keys[index] = null;
    values[index] = null;

    for (index = (index + 1) % size(); keys[index] != null; index = (index + 1) % size()) {
        Passenger tmp = values[index];
        keys[index] = null;
        values[index] = null;
        currentSize--;
        addPassenger(tmp.getName(), tmp.getId());
    }
    currentSize--;
    return true;
}
```

2. Describe the advantage(s) of the cumulative component sum hashcode function over a regular summation of the ASCII values.

Cumulative component sum helps avoid the chance where two different Ids create the same code, e.g. “ABC123” == “CBA321” where just summing ASCII values, whereas “ABC123” != “CBA321” when computing the cumulative component sum.

3. The hash table you have implemented for the security database starts at a fixed size, M, then will only increase once to the maximum size of 1021. For a regular hash table implementation using the doubling strategy though, why do we need to expand the size of the hash table when it reaches a load factor of 75% and not when it nears 100%?

To avoid the worst case with linear probing where the next available index is the one before what the hash function points to

4. If separate chaining was used instead of linear probing for collision handling, which data structure could be used instead of linked lists to make the implementation more efficient? Briefly explain why your chosen data structure is or is not more efficient.

An array would be more efficient for separate chaining as you could use the key for the index within the array and then have it be an array of linked lists. This would mean you calculate the hash function (O(1)) then go to the index (O(1)) then put it in its place within the linked list (O(1)). This would be more efficient for adding things into the array but would be potentially less efficient for finding elements. I don’t think this would be more or less efficient, and would instead just be a different way of implementing our hash table.

Shuttle Recommendation System (20 Marks)

1. Describe the data structure you have used to represent a graph and explain its advantages over alternative data structures.

I used a hash map which used the terminals as keys which pointed to a linked list of shuttles that connect to that terminal. The main advantage of this is using a hash map which is very efficient for addition and removal and searching for the elements and also doesn't require as much space as some other data structures.

2. Describe the algorithms you used to calculate the shortest path and the fastest path, and state their complexities.

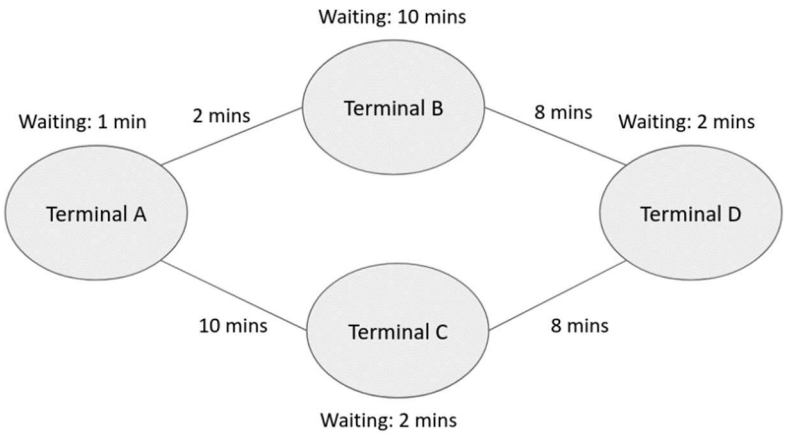
Each algorithm was just Dijkstra's shortest path algorithm but when finding the shortest path it ignored the time and just made each weight be 1 to find the smallest number of shuttles needed. But when finding the quickest path it used the time needed on each edge as the weight for Dijkstra's. The complexity of each is (with T being num terminals, and S being num shuttles) O((T+S) log T)

3. An adaptive algorithm takes advantage of some properties of the input. For example, some algorithms do not need to traverse the remaining input once the desired output has been found. Explain which of your shortest/fastest path algorithms (if any) are adaptive and why.

They aren't adaptive but would have been if implemented a bit better. As it stands, they just calculate the distance to all other terminals and returns the one that it is looking for. A better implementation would be to stop once the right terminal has been found and return then, but I couldn't work out how to implement this properly.

4.

A) Depending on the graph, there may exist two equally fastest paths. For example, in the graph below, there are two fastest paths from terminal A to terminal D: ABD and ACD. Which of these two paths will be chosen by your implementation of the fastest path algorithm, and why?



It would depend on the order that the shuttles were added, as that decides what order they end up in the queue. So, whichever was added to the system first would be the path returned as it only checks if the path is faster, not faster, or equal to it.

B) Explain how you would modify your algorithm such that it would choose the other path instead.

Change the check to be looking for the path that is <= to the current faster, rather than strictly <.

END OF REPORT

ALIGNMENT TEST BOX
DO NOT EDIT