

CS2040C Semester 1 2018/2019
Data Structures and Algorithms

Tutorial 06 - Midterm Test Debrief, Table ADT 1, Hash Table

For Week 8 (Week Starting on 8 October 2018)

Document is last modified on: October 2, 2018

1 Introduction and Objective

In the first 20m of the tutorial, the tutor will discuss some selected questions in last week's CS2040C Midterm Test.

Then, we will continue our discussion about Non-Linear Data Structures. Next up is Hash Table, **one possible efficient** implementation of Table ADT (unordered). We will discuss about the Hash Function and the next Tutorial will cover about the Collision Resolution Techniques.

2 Midterm Test Discussion

This topic is open to the class. Please suggest and recommend some questions that you would like to ask the Tutor regarding the Midterm test.

3 Tutorial 06 Questions

Hash Table Basics

Q1). A good hash function is essential for good Hash Table performance. A good hash function is easy/efficient to compute and will evenly distribute the possible keys. Comment on the flaw (if any) of the following hash functions. Assume that the load factor $\alpha = \text{number of keys } N / \text{Hash Table size } M = 0.3$ (i.e. low enough) for all the following cases:

1. $M = 100$. The keys are positive even integers. The hash function is $h(\text{key}) = \text{key} \% 100$.
2. $M = 1009$. The keys are valid email addresses. The hash function is $h(\text{key}) = (\text{sum of ASCII values of the last 10 characters}) \% 1009$. See <http://www.asciitable.com> for ASCII values.

3. $M = 101$. The keys are integers in the range of $[0, 1000]$. The hash function is $h(\text{key}) = \text{floor}(\text{key} * \text{random}) \% 101$, where $0.0 \leq \text{random} \leq 1.0$.

Possible answers:

1. No key will be hashed directly to odd-numbered slots in the table, resulting in wasted space, and a higher number of collisions in the remaining slots than necessary. Aside from the hash function, the hash table size may not be good as it is not a prime number. If there are many keys that have identical last two digits, then they will all be hashed to the same slot, resulting in many collisions.
2. Keys are not evenly distributed because many email addresses have the same domain names, e.g. 'u.nus.edu', 'gmail.com'. Many email addresses will be hashed to the same slot, resulting in many collisions.
3. This hash function is not deterministic. The hash function does not work because, while using a given key to retrieve an element after inserting it into the hash table, we cannot always reproduce the same address.

Key message to be shared with students: If we are just hashing a simple built-in data type: an integer, a double (but probably problematic – see Q2.4), or a string, just use C++11/java built-in hash functions for those two native data types. That's it, for C++, we can use `unordered_map<int, int> mapper1`, `unordered_map<double, int> mapper2`, or `unordered_map<string, int> mapper3` directly without defining our own hash function. For Java, we can use `HashMap<Integer, Integer> mapper1`, `HashMap<Double, Integer> mapper2`, or `HashMap<String, Integer> mapper3` directly without defining our own hash function.

Q2). Hashing or No Hashing: Hash Table is a Table ADT that allows for `search(v)`, `insert(new-v)`, and `delete(old-v)` operations in $O(1)$ average-case time, **if properly designed**. However, it is not without its limitations. For each of the cases described below, state if Hash Table can be used. If not possible to use Hash Table, explain why is Hash Table not suitable for that particular case. If it is possible to use Hash Table, describe its design, including:

1. The `<Key, Value>` pair
2. Hashing function/algorithm

The cases are:

1. A mini-population census is to be conducted on every person in your (not so large) neighbourhood. We are only interested in storing every person's name and age. The operations to perform are: retrieve age by name and retrieve name(s) by age.

2. A much larger population census is also conducted across the country, similarly containing only every person's name and age. The operation to perform is: retrieve names of people eligible for voting. Only people above legal age 17 years old (or older) are eligible for voting. However, we still need to store the rest of the data.
3. A different population census similarly contains only the name and age of every person. The operation to perform is: Retrieve people with a given last name.
4. A grades management program stores a student's index number and his/her final marks in one GCE 'O' Level subject. There are 100,000 students, each scoring final marks in $[0.0, 100.0]$. The operation to perform is: Retrieve a list of students who passed in ranking order (highest final marks to passing marks). A student passes if the final marks are more than 65.5. Whether a student passes or not, we still need to store all students' performance.

Possible Answers:

1. Yes, we can use 2 Hash Tables for efficient lookup both ways (nobody is restricting us, so we can use more than one data structure if that simplifies our life :O).
 <Key, Value> Pair: <name, age> and <age, names>, to find age and names respectively
 Hashing Algorithm: $h(\text{name}) = \text{standard string hashing}$, and, $h(\text{age}) = \text{age}$ (direct addressing)
2. Yes, we can actually use Direct Addressing Table as in part (1)'s <age, name> Hash Table. The operation can be performed by iterating through age (from age 0 to maybe age 100?, which is not that large), traversing the chain/list for each age, starting from the minimum eligible age.
3. Yes, if we can extract the last name from a name. <Key, Value> Pair: <last name, pair<full name, age>s> Hashing Algorithm: $h(\text{name}) = \text{standard string hashing}$
4. Two issues: First, floating-point precision. If the precision is fixed, say 1 or 2 decimal places, the number can be converted to an integer, and stored as in part (2). Second issue is the requirement to retrieve a list of students who passed in **ranking order**. Hash Table is NOT designed for this ordered operations, we need to use the data structure discussed in the next Tut07.

Table ADT

Q3). What is/are the main difference(s) between List ADT basic operations (see <https://visualgo.net/en/list?slide=2-1>) versus Table ADT basic operations (see <https://visualgo.net/en/hashtable?slide=2-1>)?