CMPT 225 - Assignment 5: Multi Data Collections

Preamble

- Due Wednesday, April 3 at 3pm.
- Let's start our assignment as early as possible!
- Please, read the entire assignment before starting it!
- This assignment must be done on our own.

Objectives

- The objective of Assignment 5 is for us to gain experience designing a software solution using *multiple data organizations* (i.e., data collections).
- We shall also gain experience expressing the design of our *multiple data organizations* by drawing their underlying data structures.

  To get an idea of what a multiple data organization may be, have a look at the Programming Problem 6 of Chapter 16 in our textbook and pay particular attention to its Figure 16.22.
- Finally, in this assignment, we shall gain experience using Hashing.

Problem Statement

To ease students' task of transferring from one academic institution to another (for example, from Douglas College to SFU), a group of Canadian institutions created a Multi-Institution Student Transfer System.

This Multi-Institution Student Transfer System allows staff in the Registrar's Office of a Canadian university/college to easily obtain information about student transfer applications.

The information kept by the Multi-Institution Student Transfer System is as follows.

For students:

- student last name and first name,
- student mailing address,
- student email address,
- student phone number,
- name of home institution (the institution from which the student is transferring),
- student number from home institution,
- name of transfer institution (the institution to which the student is transferring),

- faculty to which the student is applying (e.g.: Science, Applied Science, etc...),
- programs into which the student is applying (e.g.: Math Major and/or Physics Minor and/or etc...).

For institution:

- institution name,
- institution mailing address,
- email address of the institution's Registrar's Office,
- phone number of the institution's Registrar's Office,
- a listing of all students information (see *students* above) transferring to this institution,
- a listing of all students information (see *students* above) transferring from this institution.

We can add more information to the "student" and/or "institution" entities above.

Requirements

This group of Canadian institutions has asked us to design this Multi-Institution Student Transfer System. Here are their requirements:

This Multi-Institution Student Transfer System must allow the user (staff in the Registrar's Office of a Canadian university or college) to ...

- Select an institution (perhaps from a listing of institutions displayed on the application's window).

- Once an institution has been selected, the Multi-Institution Student Transfer System allows the user to ...

  - ~~Display all the information related to this institution in O(1).~~

  - ~~Display the number of students transferring to this institution in O(1).~~

  - ~~Display the number of students transferring from this institution in O(1).~~

  - ~~Modify all information related to this institution (except the "key(s)" used for "institution") in O(1). This requirement does not include the modification of the content of each student found in either student listings of this institution.~~

  - ~~Display all students transferring to this institution, by ascending alphabetical sort order of last names, in O(n) where n is the number of students transferring to this institution.~~

  - ~~Display all students transferring from this institution, by descending numerical sort order of student numbers, in O(m) where m is the number of students transferring from this institution.~~

  - ~~Insert a student transferring to this institution in O($\log_2$ n), where n is the number of students transferring to this institution.~~

  - ~~Insert a student transferring from this institution in O($\log_2$ m), where m is the number of students transferring from this institution.~~

  - ~~Modify all information related to this student transferring to this institution (except the "key(s)" used for "student") in O(1).~~

- ~~Display all information related to a student transferring to this institution in O(1).~~
- ~~Modify all information related to this student transferring from this institution (except the "key(s)" used for "student") in O(1).~~
- ~~Display all information related to a student transferring from this institution in O(1).~~

Note:

1. "key(s)", in the above list of requirements and in the rest of this assignment, means "indexing key" and/or "search key".
2. During the execution of the Multi-Institution Student Transfer System, removal of institution and student objects are never performed.

What we are asked to do in this Assignment 5

Part 1 - Design

1. We must design the *multiple data organizations* (i.e., data collections) and their underlying data structures that support the Multi-Institution Student Transfer System described above. Our design must satisfy all the given requirements above and their associated time efficiency.

   We must clearly express our design by drawing the multiple data organizations (i.e., data collections) we came up with along with their underlying data structures. The kind of drawing we are asked to do here is very akin to what we were asked to do in Question 2 of our midterm and to the type of drawing Figure 16.22 of Programming Problem 6 of Chapter 16 in our textbook represents.

   We must populate our drawing with sufficient number of "institution" and "student" objects to give a clear idea of the organization of our data. For example, we must sufficiently populate our drawing so as to illustrate whether our data is sorted, and if so, which attribute(s) of the "institution" and the "student" objects is/are used as the "key(s)". Note that we do not have to include the values of all the atributes of the "institution" and the "student" objects, just enough to give a clear idea.

   In our drawing, we must label our data collections and their underlying data structures clearly by writing their names (e.g., of data collections: Sorted List, BST, etc... e.g., of underlying data structures: SHSL list, array, etc...).

   Feel free to add comments and explanations to our drawing, if we feel they are necessary.

   We are free to create our drawing by hand then scan it or create our drawing using a drawing software application.

   Let's name our drawing **DataDesign.pdf**.

   In general, we know that objects (instantiated from classes) must not be duplicated in memory. Why? Therefore, our drawing must not include duplicated objects of "institution" and "student" classes.

2. As part of our design, we must also design the classes that represent the "student" and "institution" objects. To communicate our design, we must include, in our design document, the private section of the classes that represent the "student" and "institution" objects. In other

words, we must include the section of the header files that declare the private attribute members for each of our "student" and "institution" classes.

Assumptions:

- We can assume *ideal* hashing, i.e., 1-to-1 mapping. In other words, we can assume that our hashing functions will never create collisions.
- We can assume the total number of institutions this Multi-Institution Student Transfer system will ever deal with is 1000 and the total number of students this system will ever manage is 4,000,000.
- We cannot assume that student numbers are assigned to students such that when the "student" objects are sorted by last names, they are also automatically sorted by student numbers.

  For example, we cannot assume that the first student number assigned would be assigned to the first student, if the students were ordered by last names, then the second student number assigned to the second student, then the third student number assigned to the third student, etc... This would be rather unusual.

Note that in this Part 1 of this assignment, we are not asked to implement the Multi-Institution Student Transfer System.

Part 2 - Hash Function Analysis

In this Part 2, we shall no longer assume that our hashing data collections can be implemented such that we achieve *ideal* hashing, i.e., 1-to-1 mapping. In other words, we shall no longer assume that our hash functions never create collisions.

Therefore, we shall analyse and report on the *goodness* of one of our hash functions. More specifically, we shall report on how easy this particular hash function is to compute and whether it evenly distribute indexing keys across the range of hash table array indices.

So, the first thing we need to do is to select one of our hashing data collections from our design (if we have more than one). Then, let's analyse its performance as follows:

1. First, let's make sure that each of our hashing data collections in our design diagram is uniquely named. (Let's update the drawing of our design accordingly.)
2. In a text file or WORD document (from which we shall create the document **HashFunctionAnalysis.pdf**), list a) the name of the hashing data collection we are analysing, b) the indexing key we have chosen to use for this particular hashing data collection and c) a brief explanation as to why we have chosen this attribute (or these attributes) as an indexing key.
3. Then, implement (write code) its hash function and put this hash function in a file containing a main function. We shall use this main function to test our hash function. Note that we are not asked to create a HashTable ADT class.

   Let's call this file **HashFunction.cpp**.

   Let's create a data file containing > 100 instances of indexing keys and call this file **IndexingKeys.txt**. The main function then opens and reads this data file and calls our hash function for each of these indexing keys. Let's record the frequency at which each resulting hash table array index is produced using a mechanism similar to our frequency counter (or table) from our Assignment 4. If our hash function evenly distributes indexing keys across the range of hash table array indices, the frequencies stored in our frequency counter will be 1 for the hash table array indices produced (and 0 for the hash table array indices not produced). If some of frequencies stored in our frequency counter are > 1 then our hash function does not quite evenly distribute indexing keys across the range of hash table array indices.

Let's report our results in our analysis document. What does the range of hash table array indices produced by our hash function look like? Find an efficient to report our results. Perhaps by drawing an axis representing the entire range of array indices and indicating where the produced indices "landed" on the axis. Can we predict where all the other indices (not produced in our experiment) would "land"? Does our hash function evenly distribute indexing keys across the range of hash table array indices? Do we obtain a 1-to-1 mapping? Or is our hash function producing collisions? Why? What do synonyms look like? Give some examples of synonyms.

Let's now modify our hash function with the idea of spreading the hash table array indices produced by our hash function a little more, if possible. Make sure we keep the first version of our hash function in the file HashFunction.cpp by commenting it out. This way, we will be able to compare and contrast both versions of our hash function.

Again, let's report our new results (results obtained from the version 2 of our hash function).

Include a copy of the file HashFunction.cpp containing both versions of our hash function in this analysis document. Make sure we have clearly label the two versions of our hash function. Finally, indicate, in our analysis document, how easy each version is to compute, by expressing its time efficient using the Big O notation.

Marking Scheme

When marking Assignment 5, we shall consider some or all of the following criteria:

- Design: Does our design satisfy the requirements stated in this assignment? Make sure the drawing of our design clearly communicate our design?
- Analysis: Is the analysis clear, sound and accurate?

Remember ...

We can use any C++ IDE we wish to do our assignments in this course as long as the code we submit compiles and executes on the CSIL computers. Why? Because our assignments will be marked using this platform (Linux and C++ - version running on CSIL computers). So, the suggestion is to compile and execute our code using our makefile or the g++ command at the Linux command line in CSIL before we submit our assignment to CourSys.

Submission

- Assignment 5 is due Wednesday, April 3 at 3pm.

- In the interest of fairness to our classmates, and expedient marking, late assignments will only be marked for feedback, and therefore will receive 0 marks.
- This assignment must be done on our own.
- Submit the 4 files: DataDesign.pdf, HashFunctionAnalysis.pdf, HashFunction.cpp (containing the version 1 and version 2 of our hash function) and IndexingKeys.txt

---