ITU Computer Engineering Department
BLG 223E Data Structures 23-24 Fall
Homework 4 : Skip List

December 18, 2023

1 Introduction

Like in the previous assignments, you will read a csv file to construct a data structure, modify the data according to the given operations. At the end, you will create another csv file and write the final state of your data. In this assignment, the structure that you have to deal with Skip List. You will implement Skip List in two different ways. Also, note that the dataset you will be working is same as your previous homework.

2 Dataset And Operation Files

There are 3 datasets with 10k, 100k and 500k employee records, which are unordered. These records can be easily casted to Employee class, whose definition is given in the following section.

3 Problem Description

Same as in the previous assignments, you need to define a class to represent employees in your program. The class definition given below is a convenient way to represent employees.

```
class Employee{
    private:
        int id;
        int salary;
        int department;
    public:
        Employee(int i, int s, int d);
        // Don't forget getters and getters...
};
```

3.1 4-pointer Skip List Solution

As the first solution, you need to implement a 4-pointer Skip List, meaning that a node of the list has 4 pointers namely *next*, *prev*, *below* and *above*. Using the following declaration of the QuadruplySkipList_Node class, you will be able to encapsulate the Employee class with additional pointers.

```
class QuadruplySkipList_Node{
    private:
        Employee* data;
        QuadruplySkipList_Node* next;
        QuadruplySkipList_Node* prev;
        QuadruplySkipList_Node* below;
        QuadruplySkipList_Node* above;
    public:
        QuadruplySkipList_Node(Employee *data);
        ~QuadruplySkipList_Node();
        // Don't forget getters and getters...
};
```

Using the QuadruplySkipList_Node class definition given above, implement your Skip List as in the following definition.

```
class QuadruplySkipList{
    private:
        QuadruplySkipList_Node* head;
        int height;
    public:
        QuadruplySkipList(int height_in);
        ~QuadruplySkipList();
        void insert(Employee* data);
        void remove(int remove_id);
        Employee* search(int search_id);
        void dumpToFile(ofstream& out_file);
};
```

3.2 2-pointer Skip List Solution

In the second solution, push your implementation a step further by reducing the number of pointers, which will reduce the memory usage, in overall. In this implementation usage of the pointers *next* and *below* is sufficient. Using the following declaration of the DoublySkipList_Node class, you will be able to encapsulate the Employee class with additional pointers.

```
class DoublySkipList_Node{
    private:
        Employee* data;
        DoublySkipList_Node* next;
        DoublySkipList_Node* below;
    public:
        DoublySkipList_Node(Employee *data);
        ~DoublySkipList_Node();
        // Don't forget getters and getters...
};
```

Using the DoublySkipList_Node class definition given above, implement your Skip List as in the following definition.

```
class DoublySkipList{
    private:
        DoublySkipList_Node* head;
        int height;
    public:
        DoublySkipList(int height_in);
        ~DoublySkipList();
        void insert(Employee* data);
        void remove(int remove_id);
        Employee* search(int search_id);
        void dumpToFile(ofstream& out_file);
};
```

4 Implementation

Construct the Skip List **based on the employee IDs**. For example, ID of the next node must be greater than the ID of the current node, and the content of the below node must be same as the current node.

A tricky part for the Skip List is the selection of a proper **height** for the list and a convenient **probabilistic mechanism** to insert data to a level. Firstly, set height as 5 and insert data to level i when the result of a coin-toss is heads (probability=0.5). Then, **arrange the height parameter** to see how it affects the performance of the structure. For an instance, test your implementation with the following heights.

```
height = 2
height = 10
height = 100
height = 1000
```

After all the operations are completed, call the method *dumpToFile* in order to write the content of the structure into a file.

5 Things To Pay Attention

During the implementation **do not change the signatures of the given attributes and methods**. Even though the given definitions are enough to implement a skip list, **you can define your own helper methods** in your implementations. Additionally, **only include related libraries to your code.**

Your implementation will be tested extensively. Make sure you cover all the edge cases as much as possible and **the implementation is well-tuned** in terms of height. It is always a good idea to first implement the structure using dummy data, and to make sure every functionality works as expected. Only then fully implement the structure by using the Employee class.

Be careful about the memory leaks! Make sure you have a delete call corresponding to each allocation (new call). Do not forget to **implement the destructors**. Keep in mind that if your code has memory leaks, it might be degraded heavily!

6 Execution Time Measuring

Print and compare the execution times to the terminal to compare each solution. You can use the following code and time.h library:

```
#include <time.h>    // library
clock_t start = clock();    // start to measure
//.... code for measuring
clock_t end = clock();    // finish measure

//measurement time in milliseconds
(double)(end - start) * 1000 / CLOCKS_PER_SEC;
```

You can examine execution times for insert, remove and search operations with printing to terminal. For example "ADD: array solution 100 milliseconds". After performing these operations on both solutions, find out which solution is faster for each insert, search and remove operations. This will be done just to see comparison of the time it takes to perform different operations on different solutions. Therefore, comment (do not delete) these parts later.

7 Submission Rules

- You cannot use Standard Template Library (STL). Do not use libraries other than those specified.
- Make sure you write your name and number in all of the files of your project, in the following format:
  $/*@Author$

$StudentName :< studentname >$
$StudentID :< studentid >$
$Date :< date > */$

- You will submit **2 C++ files**, one for the **4-pointer Skip List solution**, one for the **2-pointer Skip List solution**.
- Use comments wherever necessary in your code to explain what you did.
- Your program will be checked by using Calico (https://github.com/uyar/calico) automatic checker.
- DO NOT SHARE ANY CODE or text that can be submitted as a part of an assignment.
- Only electronic submissions through Ninova will be accepted no later than deadline.
- You may discuss the problems at an abstract level with your classmates, but you should not share or copy code from your classmates or from the Internet. You should submit your **own, individual homework.**
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- If you have any question about the homework, you can send e-mail to Batuhan Can (canb18@itu.edu.tr).
- Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**