Implementing a college data set - your choices

1. Introduction

Consider a college holding records about its students, where each record contains two integer fields: ID and Age. Given different application scenarios below, for each application, please (1) identify the most appropriate data structures so that the Big-O run time for frequent operations is optimized, (2) provide the implementation code, and (3) analyze the worst-case Big-O run time for each operation. Your possible data structure choices mainly include struct, array, linked list, and hash table. Please note that the college assigns a global unique ID to a student when he/she first enrolls and the ID will never be recycled for other students (Hints: No need to check the uniqueness of a record for insertion).

You can choose <u>any two of the three applications</u> below for implementation.

Application 1:

Assume that the college takes very few transfer students and has most of its students stay for at least one year. (Hints: no frequent insertion and deletion operations). And the maximum student number is given. But it does require a lot of search operations based on student ages. Please note that your search functions may return multiple records when only a student age is given. In addition, the college often requires calculating the largest age gap among its students. The major interfaces provided by your code should include createDataSet, destroyDataSet, searchAge, insertion, deletion, maxAgeGap.

Application 2:

Assume that the college takes very few transfer students and has most of its students stay for at least one year (Hints: no frequent insertion and deletion operations). And the maximum student number is given. But it does require a lot of search operations. Specifically, all of the searches are based on student IDs. The major interfaces provided by your code should include createDataSet, destroyDataSet, searchID, insertion, deletion.

Application 3:

Assume a new college that is expanding frequently and dramatically on its student amount. This leads to two consequences. First, accurately estimating the total number of students is extremely difficult. Second, frequent insertions and deletions are performed. In addition, the college frequently requires calculating the largest age gap among its students. Assume that search operations are not performed frequently. Your implementation has to support searches on either student ID or age. The major interfaces provided by your code should include createDataSet, destroyDataSet, searchAge, searchID, insertion, deletion, maxAgeGap.

2. Implementation

For each application, you need to implement two separate files as college.c (i.e. the “outsider” program containing a main function), and dataset.c (i.e. your dataset ADT with specific data structures).

(1) Implementing a main function (college.c)

For completeness of the project and testing purpose, you need to write your own code with a main function. Please generate a separate code file for your main function (college.c), where the following tasks are performed.

- Create your data set. For application 1 and 2, you can assume the maximum number of students is 3000 and pass this value as a parameter when creating your data set.
- Generate 1000 student records with random ages ranging from 18 to 30. Follow the steps below to generate random unique student IDs.
    a. Implement a random integer generator, which randomly generates an integer number as either 1 or 2.
    b. The first student ID can be determined as the return value of the random integer generator (i.e. either 1 or 2).
    c. For each of the remaining student records, the ID field is determined by the previous student's ID plus the return value of the random integer generator. For example, assuming that the 3rd student ID is 5, and that 2 is returned as a result of the random integer generator, the 4th student ID can be then determined as 5 + 2 = 7.
- Each time when a new student record is generated, insert it into your data set.
- Randomly generate a student record with ID (ranging from 00001 ~ 2000) and age (ranging from 18~30). Search for the student in your data set. Depending on the specific application, you may do search based on either ID or age.
- For application 2 and 3, randomly generate a student ID (ranging from 0001 ~ 2000). Delete the record from your data set if it is found. For application 1, randomly generate a student age. Delete all the records with the given age in your data set.
- For application 1 and 3, call the maxAgeGap interface and print out the return value.
- Destroy your data set.

(2) Printf statement

- Add printf in searchAge, searchID, deletion functions, which prints (1) the Age and ID for the student record that needs to be searched or deleted; and (2) whether the delete has been done successfully or whether the item has been found.
- Add printf within maxAge function, which prints the maximum age gap for the current elements.

(3) Implementing your dataset ADT (dataset.c)

For each of the two applications that you choose, identify the most appropriate data structures so that the Big-O run time for frequent operations is optimized. Implement the interfaces required by the specific application in dataset.c.

(4) Some extra notes for hash table

If you use hash table for your implementation, please use a prime number for the table length. For example, 1009 is the smallest prime number greater than 1000. For simplicity reason, you can just use linear probing to resolve collisions, and an example hash function can be $h(k) = (k + i)\%m$.

3. Submission

Create subdirectories as app1, app2, and app3 for your implementation. Each subdirectory should include college.c, dataset.c and report.txt. Complete the file report.txt containing the results for the worst-case Big-O run time for each of the operations required by your applications, such as creation, destroy, search, insertion, deletion and maxAgeGap. In addition, provide a justification for the selected data structure for each application and possibly

explain why alternate structures would not work. Submit a tar file containing the entire project directory using the online submission system.

4. Grading

Your implementation will be graded in terms of correctness, clarity of implementation, and commenting and style. Your implementation must compile and run on the workstations in the lab. The algorithmic complexity of each function must be documented.