BST, AVL tree, Linked lists

1 Introduction

In this assignment, you are to manage a sequence of records corresponding to different users. Each user can be uniquely identified using a 16-byte character array called uid. The uid is not necessarily terminated using the null character ('\0') and may contain multiple null characters. You need to insert, delete, and search record corresponding to a uid using a BST and an AVL tree. In addition, a user can also have friends. You need to store a list of friends of every user, which essentially means storing the references to its friends' records.

2 Type of record

The type of a user's record is given below.

```
struct record {
  /* character string terminated with '\0'
   * maximum length is 16
   */
  char name[MAX_LEN];
  /* a character array of 16 characters
   * not-necessarily terminated with '\0'
   * a uid may contain multiple '\0''s
   * anywhere in the character array
   */
  char uid[MAX_LEN];

  int age;

  /* location */
  struct location loc;

  /* list of posts */
  struct list_posts *posts;
```

```
  /* list of friends */
  struct list_records *friends;

  /* needed for shortest Path */
  int status;
  struct record *pred;

  /* needed for the tree data-structure */
  int height;
  struct record *left;
  struct record *right;
  struct record *parent;
};
```

The `uid` field is the key used for the BST and the AVL tree. You can use `left` and `right` fields to store the references to left and right subtrees in an AVL or BST node. The `height` field is used for the AVL tree. The `status` field is used when a record is not present during search and delete operations. The `friends` field contains the head of the linked list that stores the references to the records corresponding to a user's friends. The type of `friends` is `struct list_records`, as shown below.

```
struct list_records {
  struct record *record;
  struct list_records *next;
};
```

A node of type `struct list_records` stores a reference to `struct record` and the reference to the next node (using the `next` field). This can be used to implement the list of friends.

You are not allowed to change `struct record` or `struct list_records` in your implementation.

3 BST

The `bst_root` in "pa2.c" points to the root of the BST. Initially, `bst_root` points to an empty tree. You need to implement insert, search, and delete operations that insert search and delete a record of type `struct record` from the BST rooted at `bst_root` using `uid` as the key. In addition, a user may have multiple friends. You need to keep track of the friends of a BST node using a linked list. During deletion, you need to remove the user from the lists of friends of other users.

4 AVL

The `avl_root` in "pa2.c" points to the root of the AVL tree. Initially, `avl_root` points to an empty tree. You need to implement insert, search, and delete operations that insert, search, and delete a record of type `struct record` from the AVL tree rooted at `avl_root` using `uid` as the key. In addition, a user may have multiple friends. You need to keep track of the friends of an AVL node using a linked list. During deletion, you need to remove the user from the lists of friends of other users.

5 Library interface

In this assignment, you need to implement a library that implements all the functionalities we discussed above. The user interface for your library is given in the "pa2.h" file. Below is a short description of these interfaces.

- `get_bst_root`: Return the root of the BST, `bst_root`. This implementation has already been provided. Please don't change it.
- `get_avl_root`: Return the root of the AVL tree, `avl_root`. This implementation has already been provided. Please don't change it.
- `insert_record_bst`: Insert record r in the BST rooted at `bst_root`.
- `insert_record_avl`: Insert record r in the AVL tree rooted at `avl_root`.
- `search_record_bst`: Search the record corresponding to uid in the BST rooted at `bst_root`. If the record is not present, return a dummy record with -1 in the status field; otherwise, return a copy of the record.
- `search_record_avl`: Search the record corresponding to uid in the AVL tree rooted at `avl_root`. If the record is not present, return a dummy record with -1 in the status field; otherwise, return a copy of the record.
- `make_friends_bst`: Make users with uids uid1 and uid2 in the BST rooted at `bst_root` friends of each other if they aren't already friends. The `friends` field in "struct record" stores the head of the linked list of friends of a given user. To make the user with record A a friend of the user with record B, add A to B's list of friends and add B to A's list of friends. Return 1 if uid1 and uid2 are already friends before this call. Return 0 if they become friends during this call.
- `make_friends_avl`: Make users with uids uid1 and uid2 in the AVL tree rooted at `avl_root` friends of each other if they aren't already friends. The `friends` field in "struct record" stores the head of the linked list of friends of a given user. To make the user with record A a friend of the user with record B, add A to B's list of friends and add B to A's list of friends. Return 1 if uid1 and uid2 are already friends before this call. Return 0 if they become friends during this call.

- `get_friends_list_bst`: The `friends` field in "struct record" stores the head of the linked list of friends of a given user. Return the head of the linked list of friends (i.e., the `friends` field) of the user with uid `uid` in the BST rooted at `bst_root`. If the corresponding record doesn't exist, return NULL.
- `get_friends_list_avl`: The `friends` field in "struct record" stores the head of the linked list of friends of a given user. Return the head of the linked list of friends (i.e., the `friends` field) of the user with uid `uid` in the AVL tree rooted at `avl_root`. If the corresponding record doesn't exist, return NULL.
- `delete_record_bst`: Delete record (say n) corresponding to uid from the BST rooted at `bst_root`. Also, remove n from the lists of friends of other records and release the memory for the linked list nodes. Release memory for all the nodes in the list of friends of n. Return a copy of the value of the deleted node. If the node is not present, return a dummy record with -1 in the status field.
- `delete_record_avl`: Delete record (say n) corresponding to uid from the AVL tree rooted at `avl_root`. Also, remove n from the lists of friends of other records and release the memory for the linked list nodes. Release memory for all the nodes in the list of friends of n. Return a copy of the value of the deleted node. If the node is not present, return a dummy record with -1 in the status field.
- `get_num_bst_records`: Return the total number of records in the BST rooted at `bst_root`.
- `get_num_avl_records`: Return the total number of records in the AVL tree rooted at `avl_root`.
- `destroy_bst`: Release memory for all BST nodes and their lists of friends. Make `bst_root` points to an empty tree.
- `destroy_avl`: Release memory for all AVL nodes and their lists of friends. Make `avl_root` points to an empty tree.

6 Compilation and running the test cases

Clone the assignment repository using:

```
git clone https://github.com/Systems-IIITD/DSALAB.git
```

Implement everything in the "PA2/pa2.c" file. Don't change any other files. Use `printf` to debug your code. Run "make" in the "PA2" folder to compile your library and test cases. There are four test cases. To run the first test cases: use "./test1 10". It will test your program for ten records. Once your implementation works for small sizes, test and debug it for large sizes. To run

the second test for size 10, use "./test2 10". To run the third test case for size 10, use "./test3 10". To run the fourth test case for size 10, use "./test4 10". We will test your implementation for large input sizes. So make sure to test them for large inputs as well. You are not allowed to use `malloc` and `free` directly in your library. Use `allocate_memory` and `free_memory` routines provided to you instead of `malloc` and `free`.

6.1 How to submit

Remove all `printf` statements from your library before submitting. Create a report in pdf format that contains the output of "make submit1", "make submit2", "make submit3", and "make submit4". Submit the "pa2.c" file along with your report. A sample format of the report is shown below. Use the same format in your submission.

Sample report file.

```
The output of make submit1:
echo "Compiling test-case 1"
Compiling test-case 1
gcc -g -Werror -O3 -L. -Wl,-rpath=. -o test1 test1.c -ldsa -lpa2 -lm
./test1 100000
Creating 100000 uids took 79 ms.
adding 100000 records took 31 ms.
making 599982 friends took 229 ms.
search 100000 records took 25 ms.
Test-case-1 passed
./test1 1000000
Creating 1000000 uids took 1548 ms.
adding 1000000 records took 808 ms.
making 5999982 friends took 6091 ms.
search 1000000 records took 860 ms.
Test-case-1 passed

The output of make submit2:
echo "Compiling test-case 2"
Compiling test-case 2
gcc -g -Werror -O3 -L. -Wl,-rpath=. -o test2 test2.c -ldsa -lpa2 -lm
./test2 100000
Creating 100000 uids took 81 ms.
adding 100000 records took 32 ms.
making 599982 friends took 237 ms.
deleting 50000 records took 465 ms.
Test-case-2 passed
./test2 1000000
Creating 1000000 uids took 1562 ms.
adding 1000000 records took 815 ms.
```

```
making 5999982 friends took 6092 ms.
deleting 500000 records took 10456 ms.
Test-case-2 passed

The output of make submit3:
echo "Compiling test-case 3"
Compiling test-case 3
gcc -g -Werror -O3 -L. -Wl,-rpath=. -o test3 test3.c -ldsa -lpa2 -lm
./test3 100000
Creating 100000 uids took 78 ms.
adding 100000 records took 32 ms.
making 599982 friends took 199 ms.
search 100000 records took 23 ms.
Test-case-3 passed
./test3 1000000
Creating 1000000 uids took 1600 ms.
adding 1000000 records took 641 ms.
making 5999982 friends took 4231 ms.
search 1000000 records took 590 ms.
Test-case-3 passed

The output of make submit4:
echo "Compiling test-case 4"
Compiling test-case 4
gcc -g -Werror -O3 -L. -Wl,-rpath=. -o test4 test4.c -ldsa -lpa2 -lm
./test4 100000
Creating 100000 uids took 78 ms.
adding 100000 records took 32 ms.
making 599982 friends took 197 ms.
deleting 50000 records took 455 ms.
Test-case-4 passed
./test4 1000000
Creating 1000000 uids took 1549 ms.
adding 1000000 records took 630 ms.
making 5999982 friends took 4158 ms.
deleting 500000 records took 10068 ms.
Test-case-4 passed
```