Single-source shortest-paths

May 26, 2023

1 Introduction

In this assignment, you are to implement Dijkstra's algorithm for single-source shortest-paths to find shortest paths to different users from a given user. A user's data is stored in `struct record`. The location of the user is also stored in `struct record`. A user can have multiple friends. All the friends of a given user U are reachable from U. All the users that are reachable from the friends of U are also reachable from U. In this assignment, for a given user, U, you need to calculate the shortest distances and shortest paths to all other users that are reachable from U.

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;

  /* used for verification */
  int verify;
```

```
  /* 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;
  double distance;

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

The `loc` field of type `struct location` contains the location of the user. `struct location` contains two fields, `lat` and `lon`, that correspond to the `latitude` and `longitude`, respectively. You need to use the `distance` function to compute the distance between two users. The other fields that are relevant in this assignment are `friends`, `status`, `pred`, and `distance`. 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 existing structures (e.g., `struct record`, `struct list_records`, etc.) in your implementation. The `distance` and `pred` fields contain the shortest distance and the predecessor on a shortest path computed by your algorithm.

3 Single-source Shortest-paths

The `struct record` of a user U stores the references to other `struct record` nodes that correspond to the friends on U in a linked-list. The `friends` field in

the `struct record` contains the head of this linked-list. If V is in the friends' list of U, then U is also in the friends' list of V. If two users are friends, they are connected via an undirected edge whose length is the distance between them. Two users are reachable from each other if a path exists between them. In this part, you need to compute a shortest path and the shortest distance to each user v that is reachable from a source user u. The shortest distance is the shortest distance between u and v. The shortest path is a shortest path between u and v. You need to implement Dijkstra's algorithm for single-source shortest-paths. You can use a linear scan instead of a min-heap to find a vertex with the minimum distance in your implementation. For min-heap or linear scan implementation, you can use `min_heap_arr`. `min_heap_arr` is large enough to store all vertices in the graph.

4 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 "pa4.h" file. Below is a short description of these interfaces.

- `make_friends(struct record *r1, struct record *r2)`: Make users `r1` and `r2` friends of each other if they aren't already friends. The `friends` field in "`struct record`" stores the head of the linked-list that stores references to friends. To make `r1` a friend of the `r2`, insert `r1` in the linked-list `r2->friends` and insert `r2` in the linked-list `r1->friends`. Return 1 if `r1` and `r2` are already friends before this call. Return 0 if they become friends during this call.
- `get_friends_list(struct record *r)`: Return the list of friends of `r`, i.e., `r->friends`.
- `compute_sssp(struct record *r)`: Compute the shortest distance and a shortest path to all the users reachable from `r`. The `status` fields in all the records are set to zero before this function call. `distance` and `pred` may contain garbage values. At the end of this procedure, the `distance` and `pred` fields corresponding to the users reachable from `r` must contain the shortest distance from `r` and the predecessor on the corresponding shortest path. You can use the `min_heap_arr` array to implement the min-heap. The elements in this array are of type `struct heap_elem`. `struct heap_elem` can store a reference to a vertex. `min_heap_arr` is large enough to store all vertices reachable from the source node.
- `delete_friends_list(struct record *r)`: Remove all friends from `r->friends` and release the corresponding memory.

5 Compilation and running the test cases

Clone the assignment repository using:

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

Implement everything in the "PA4/pa4.c" file. Don't change any other files. Use `printf` to debug your code. Run "make" in the "PA4" folder to compile your library and the test case. There is only one test case. To run the 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. 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`. In this assignment, you are allowed to allocate memory only for the linked-list nodes in the list of friends.

5.1 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". Submit the "pa4.c" file along with your report. Don't submit anything apart from these two files; otherwise, you will get a zero on this assignment. A sample format of the report is shown below. Use the same format in your submission. Late submissions are not allowed. Ask about any assignment-related doubts after the lecture or during my office hours.

Sample report file.

```
echo "Compiling test-case 1"
Compiling test-case 1
gcc -g -Werror -O3 -L. -Wl,-rpath=. -o test1 test1.c -ldsa -lpa4 -lm
./test1 1000
Creating 1000 uids took 0 ms.
adding 1000 records took 0 ms.
making 5982 friends took 0 ms.
computing all shortest paths took 1608 ms.
Test-case-1 passed
./test1 5000
Creating 5000 uids took 2 ms.
adding 5000 records took 0 ms.
making 29982 friends took 1 ms.
computing all shortest paths took 86368 ms.
Test-case-1 passed
```