CSE344 FINAL PROJECT

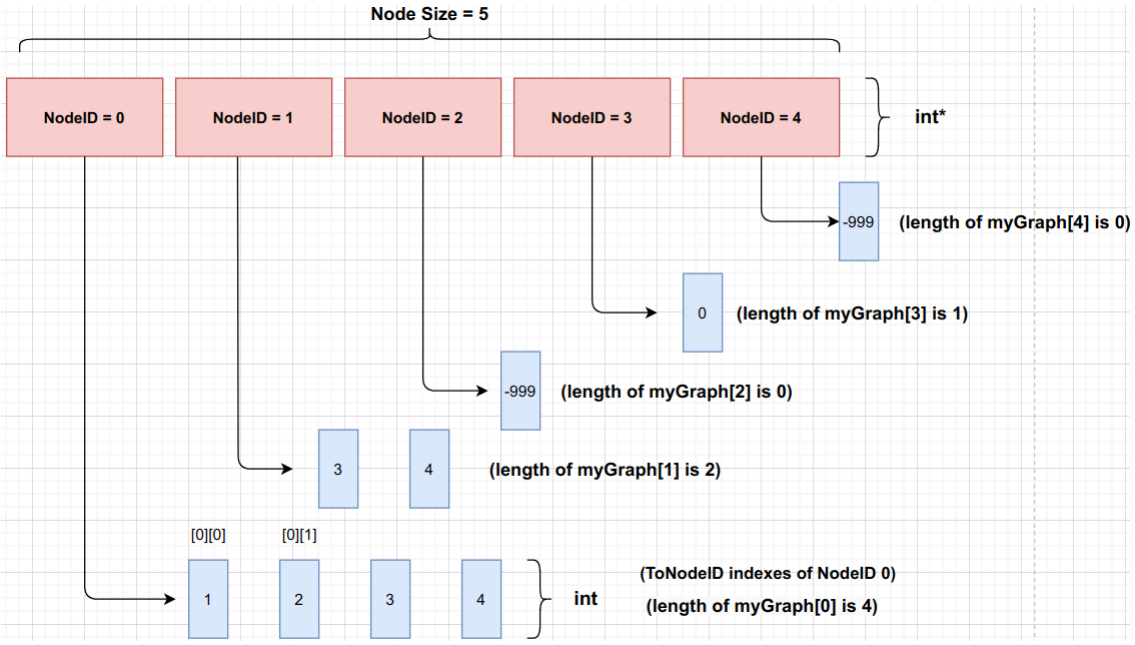
Can Beyaznar - 161044038

June 2020

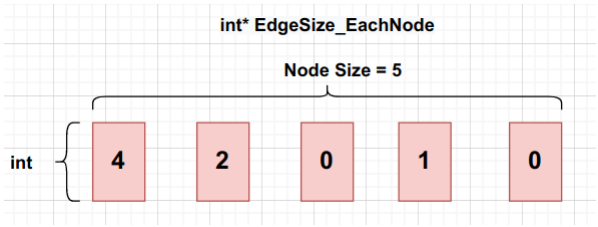
1 Problem Solution

1.1 Server Part

First of all, I checked the values of inputs from the user with the help of getopt in the server part. If it does not comply with the rules specified in the homework, I print the usages to the terminal and end the program. If it's correct, I'm doing a check to prevent the server from running again. For this, I check the file named " `_ISSERVER_WORKING_` " with the access () function. If this file already existed, the server is running. If this file does not exist, I create the file with the fopen () function. Thus, if the server is run again, I prevent multiple servers from running at the same time. And I definitely delete this file at the end of the program. Important note, if, under any error condition, the file " `_ISSERVER_WORKING_` " has not been deleted, please delete this file, otherwise the server will not work.
And now I can allocate global values that I will use on my server. I do this in the `prepare_server` section. Here, for my 2D array named myGraph, I need to determine the size and fill it in. First of all, I read the file entered by the user and find the maximum NodeID in the file. I also terminate the program if there is any mistake in the file entered by the user. These inaccuracies include negative numbers, TAB (ascii value: 9) between two numbers, and space instead. **There must be a TAB between the two numbers or the program will not run.** After learning the maximum NodeID, I create an array named `EdgeSize_EachNode`. This array is size up to NodeID + 1. In this array, I count how many nodes each node points to. After this process, I now learn the size I need to reserve for my 2D graph array. The structure of my graph is as follows.



2D Graph Structure



`EdgeSize_EachNode Array`

After reserving a place for Graph, I re-read the file to fill in the information. Assigns the ToNodeID to the current index by going to the index of FromNodeID read from the file. Thus, our graph becomes completely ready. Then, space is reserved for the parameters required for the threads. In this assignment, I used pipes to facilitate communication between main and thread. There will be a total number of threads in total and these pipes will be updated as they are resized. The `socket_fd` sent by the main using the pipe in the threads is received by the thread in sequence and no requests remain unanswered. After this process, I initialize the mutex that I will use for threads according to `priority_type`. Then my reservation is finished.

After these processes, I create my threads. I created the `thread_inf` structure to solve problems such as f finding the available thread, learning the id of the thread. And I created a structre named `thread_list` to keep them connected. `thread_list` structure is a linked list that holds `thread_inf` values.

I send my threads to my parameter `thread_pool` of `thread_list` type. After creating threads, I can now switch to the socket creation section. After opening the socket in `AF_INET` type, I am waiting for a request from the client with accept (). After the request, I first check the server load, if it is over 75, I send a character to my resize thread using pipe. When Resize reads this character, it has two options. If the character is 'r', it resizes, if 'e' exits and ends the thread. A similar process is available in the thread calculating BFS. After resizing, the find thread is checked by calling the findAvailableThread () function. Here the int isRunning parameter in my `thread_inf` structure comes into play. If this parameter is 0, it means available, if it is 1, it means busy. The findAvailableThread function returns the index of the thread where it sees the first 1 value. If they are all full, they wait. Then `socket_fd` is sent to the pipe of the thread that should work. And when the threads are running, it takes the file descriptor from its pipe and calculates according to the source and destination sent by the Client. In addition, in order to end my threads calculating BFS, I provide their output by sending -1 to the values sent by pipe. Also, when creating each thread, I send the information of `thread_inf` type as a parameter. Thus, this information can be updated in the thread. And in main I can find out which threads are busy or available. The structure I created for the thread pool is as follows.

```
typedef struct _thread_inf{

    int id;
    //int thread_type;
    int socket_fd;
    int isRunning;
    pthread_t graph_thread;
    //pthread_mutex_t graph_mutex;
    //pthread_cond_t graph_cond_var;

}thread_inf;

struct thread_list{
    thread_inf thread_val;

    struct thread_list* next_thread;
};
```

If we come to the thread function calculating BFS. First of all, I read the values from the main pipe. If all values are -1, the thread terminates itself. If not, I am reading `socket_fd` from main. And I define the char array I will send to the client. I allocated the size of this array as (char *) malloc (sizeof (char) * 40960). However, if the path from BFS is longer than this, the client will definitely not get the correct result of the path. Therefore, please run the program with this problem in mind. Since no value can be sent according to the length of the path to be sent in the sockets, a fixed size has been determined by compelling. With this dimension, a path can contain an average of 5120 numbers in itself. After the reservation, we use our mutex according to the `priority_type` value from the user. The same algorithm is used in 3 possibilities. The only difference is

the way mutex is used and the way it is used. While writing these algorithms, the reader writer problem's algorithm described in the lesson was used. If we come to the path finding algorithm, firstly, the source and destination values are taken by the client with the recv () function. Then, it is checked whether this path is available in the Cache database (Cache structure will be explained later). If not, the path is calculated with BFS and written to the log file and added to the Cache database. If there is no way, an error is given.

Coming to the cache structure, I chose the Binary Search Tree structure to make the searches faster here. I thought that this structure is more suitable because I will use only search and insert in the database. Since it has search and insert log n time complexity, I think I can add and remove database quickly. The structure I created for the cache database is as follows.

```
struct Cache_tree{
    int* path_arr;
    int size;
    int isHead;
    struct Cache_tree* left;
    struct Cache_tree* right;
};
```

If we come to the resize thread, first of all, I read the character coming from main with the help of pipe. If it is 'e', I exit, if 'r', I resize. Before calculating, I calculate how many threads I need to create. If 25% of the current thread count is 0, I create 1 new thread. So I prevent my resize thread from working in vain. Then I create threads as many threads as they need to create.
I keep a parameter for the **SIGINT** signal like I did in previous assignments. When **SIGINT** occurs, I set the value of this parameter to 1. And I can end my program by checking this value in **non-critical** areas of my program.

1.2 Client Part

The client part is simpler than the server part. First of all, I get the inputs that I need to get from the user with getopt and check these values. Source and destination must be positive. After these checks, I contact the server. Then I send the source and destination that should be sent to the server with the help of send. Then I wait for the response from the server with the recv function. As I mentioned in the server part, I got the size of the answer here as much as 40960. Thus, if the answers do not exceed this limit, the path reaches the client's hand correctly. The result is printed on the screen according to the response.

1.3 Example Output

```
:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./client -a 127.0.0.1 -p 8080 -s 3 -d 6100
05:43:26 27/06/2020 Client (26531) connecting to 127.0.0.1
05:43:26 27/06/2020 Client (26531) connected and requesting a path from node 3 to 6100
05:43:26 27/06/2020 Server's response to (26531): 3->1287->3140->4159->5361->5794->5987->6100, arrived in 0.0 seconds.
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./client -a 127.0.0.1 -p 8080 -s 18 -d 300
05:43:47 27/06/2020 Client (26534) connecting to 127.0.0.1
05:43:47 27/06/2020 Client (26534) connected and requesting a path from node 18 to 300
05:43:47 27/06/2020 Server's response (26534): NO PATH, arrived in 0.0 seconds, shutting down
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./client -a 127.0.0.1 -p 8080 -s 0 -d 23
05:44:06 27/06/2020 Client (26580) connecting to 127.0.0.1
05:44:06 27/06/2020 Client (26580) connected and requesting a path from node 0 to 23
05:44:06 27/06/2020 Server's response to (26580): 0->3->1287->1945->1228->23, arrived in 0.0 seconds.
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./client -a 127.0.0.1 -p 8080 -s 0 -d 7
05:44:14 27/06/2020 Client (26584) connecting to 127.0.0.1
05:44:14 27/06/2020 Client (26584) connected and requesting a path from node 0 to 7
05:44:14 27/06/2020 Server's response to (26584): 0->7, arrived in 0.0 seconds.
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./client -a 127.0.0.1 -p 8080 -s 3 -d 6100
05:44:25 27/06/2020 Client (26590) connecting to 127.0.0.1
05:44:25 27/06/2020 Client (26590) connected and requesting a path from node 3 to 6100
05:44:25 27/06/2020 Server's response to (26590): 3->1287->3140->4159->5361->5794->5987->6100, arrived in 0.0 seconds.
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$
```

```
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ ./server -i p2p-Gnutella0
8.txt -p 8080 -o mylog.txt -s 4 -x 24 -r 0
canbey@canbey-PC:~/Desktop/Home works/Sistem Pr
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$ kill -2 26520
canbey@canbey-PC:~/Desktop/Home works/Sistem Prog./final/final_project/code$
```

```
05:43:08 27/06/2020 Executing with parameters:
05:43:08 27/06/2020 -i p2p-Gnutella08.txt
05:43:08 27/06/2020 -p 8080
05:43:08 27/06/2020 -o mylog.txt
05:43:08 27/06/2020 -s 4
05:43:08 27/06/2020 -x 24
05:43:08 27/06/2020 Loading graph...
05:43:08 27/06/2020 Graph loaded in 0.0 seconds with 6301 nodes and 20777 edges.
05:43:08 27/06/2020 Thread #0: waiting for connection
05:43:08 27/06/2020 Thread #1: waiting for connection
05:43:08 27/06/2020 Thread #2: waiting for connection
05:43:08 27/06/2020 A pool of 4 threads has been created
05:43:08 27/06/2020 Thread #3: waiting for connection
05:43:26 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:43:26 27/06/2020 Thread #0: searching database for a path from node 3 to node 6100
05:43:26 27/06/2020 Thread #0: no path in database, calculating 3->6100
05:43:26 27/06/2020 Thread #0: path calculated:3->1287->3140->4159->5361->5794->5987->6100
05:43:26 27/06/2020 Thread #0: responding to client and adding path to database
05:43:47 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:43:47 27/06/2020 Thread #0: searching database for a path from node 18 to node 300
05:43:47 27/06/2020 Thread #0: no path in database, calculating 18->300
05:43:47 27/06/2020 Thread #0: path not possible from node 18 to 300
05:44:06 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:44:06 27/06/2020 Thread #0: searching database for a path from node 0 to node 23
05:44:06 27/06/2020 Thread #0: no path in database, calculating 0->23
05:44:06 27/06/2020 Thread #0: path calculated:0->3->1287->1945->1228->23
05:44:06 27/06/2020 Thread #0: responding to client and adding path to database
05:44:14 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:44:14 27/06/2020 Thread #0: searching database for a path from node 0 to node 7
05:44:14 27/06/2020 Thread #0: no path in database, calculating 0->7
05:44:14 27/06/2020 Thread #0: path calculated:0->7
05:44:14 27/06/2020 Thread #0: responding to client and adding path to database
05:44:25 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:44:25 27/06/2020 Thread #0: searching database for a path from node 3 to node 6100
05:44:14 27/06/2020 Thread #0: path calculated:0->7
05:44:14 27/06/2020 Thread #0: responding to client and adding path to database
05:44:25 27/06/2020 A connection has been delegated to thread id #0, system load 25.0%
05:44:25 27/06/2020 Thread #0: searching database for a path from node 3 to node 6100
05:44:25 27/06/2020 Thread #0: path found in database: 3->1287->3140->4159->5361->5794->5987->6100
05:45:54 27/06/2020 Termination signal received, waiting for ongoing threads to complete.
05:45:54 27/06/2020 All threads have terminated, server shutting down.
```

1.4 Important Notes

NodeIDs should not be negative.
The server program should always be run in the same location. (To check `_ISSERVER_WORKING_` file)
There must be TAB between 2 NodeIDs.