ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 223E
DATA STRUCTURES

HOMEWORK NO	:	3
HOMEWORK DATE	:	07.06.2024

GROUP MEMBERS:

150210100	:	Selin Yılmaz

SPRING 2024

Contents

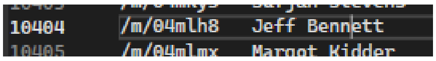
1 INTRODUCTION

In this homework, two *.tsv* files are given (freebase.tsv and mid2name.tsv). *freebase.tsv* contains two *MID*s and a relationship between them (Fig. 1).

```
/m/0kfv9 /tv/tv_program/regular_cast./tv/regular_tv_appearance/actor /m/01
    11sq
```

Figure 1

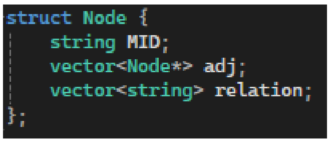
MID is the structure which is used to represent entities. *mid2name.tsv* contains MIDs and their textual equivalents (Fig. 2).



```
10404      /m/04mlh8    Jeff Bennett
10405      /m/04mlmx    Margot Kidder
```

Figure 2: Caption

In the skeleton code, there is a Node structure as in below.



```
struct Node {
    string MID;
    vector<Node*> adj;
    vector<string> relation;
};
```

Figure 3: Caption

adj stores the pointer to the neighbors of the specific node, *relation* stores the type of the relationship.

In the main function of the skeleton, first, the lines from the file *freebase.tsv* are read and parsed appropriately. Second, the MIDs become IDs of nodes, and they are added to *graph_map* (a map of type `map<string, Node*> graph_map = {}`). Then, the graph is built by adding neighbors and relationships between nodes as specified in the file.

Similar steps are implemented when reading *mid2name.tsv*. Lines are read and parsed, and the encountered MIDs and names are added to *mid2name* (a map of type `map<string, string> mid2name = {}`).

2 HELLO NEIGHBOR

The function `void helloNeighbor(string center_MID)` performs the following tasks:

1. **Find the Node**:
   - The function starts by searching for `center_MID` in the `graph_map`.
   - If the `center_MID` is not found, it prints `"MID not found"` and exits the function.
2. **Retrieve the Node**:
   - If the `center_MID` is found, it retrieves the corresponding `Node*` from `graph_map` and assigns it to the variable `center`.
3. **Print Neighbors**:
   - The function prints the number of neighbors the `center` node has.
   - It then iterates through the adjacency list (`adj`) of the `center` node.
4. **Print Neighbor Details**:
   - For each neighbor, it retrieves the neighbor's MID from the adjacency list.
   - It then looks up the neighbor's name using the `mid2name` map.
   - Finally, it prints the neighbor's MID and name.

3 DEGREE CENTRALITY

The function `void degreeCentrality()` calculates and prints the degree centrality of nodes in the graph. The steps are as follows:

1. **Initialize a Vector**:
   - A vector of pairs, `degree_centrality`, is created to store the MID and degree of each node.
2. **Calculate Degrees**:
   - The function iterates through each pair in the `graph_map`.
   - For each node, it retrieves the MID and calculates the degree by determining the size of the node's adjacency list (`adj`).
   - It then adds a pair of the MID and its degree to the `degree_centrality` vector.
3. **Sort by Degree**:
   - The `degree_centrality` vector is sorted in descending order based on the degree of the nodes using a custom comparison lambda function.
4. **Print Top 10 Nodes**:
   - The function prints the top 10 nodes with the highest degree centrality.
   - For each of the top 10 nodes, it retrieves the MID, degree, and name from the `mid2name` map.
   - It then prints the MID, name, and degree.

4 SHORTEST DISTANCE

The function `void shortestDistance(string start_MID, string end_MID)` finds and prints the shortest distance and path between two nodes in the graph. The steps are as follows:

1. **Check Existence of Nodes**:
    - The function first checks if `start_MID` and `end_MID` exist in the `graph_map`.
    - If either MID is not found, it prints `"MID not found"` and exits the function.
2. **Initialize BFS Structures**:
    - A queue `q` is initialized to facilitate the breadth-first search (BFS).
    - Two unordered maps, `parent` and `distance`, are initialized to store the parent of each node and the distance from the start node, respectively.
3. **Start BFS**:
    - The `start_MID` is pushed onto the queue, and its parent is set to an empty string.
    - The distance from the start node to itself is set to 0.
4. **BFS Loop**:
    - While the queue is not empty, the function processes the front element of the queue (`current`).
    - It retrieves the node corresponding to `current` from the `graph_map`.
    - For each neighbor of `current`, if the neighbor's MID is not already in the `distance` map:
        - The neighbor's MID is pushed onto the queue.
        - The parent of the neighbor is set to `current`.
        - The distance to the neighbor is set to the distance to `current` plus 1.
        - If the neighbor's MID is `end_MID`, the shortest path has been found:
            - The function prints the shortest distance between `start_MID` and `end_MID`.
            - It then constructs the path from `end_MID` to `start_MID` using the `parent` map.
            - The path is printed in reverse order (from start to end).
            - The function returns, as the shortest path has been found.

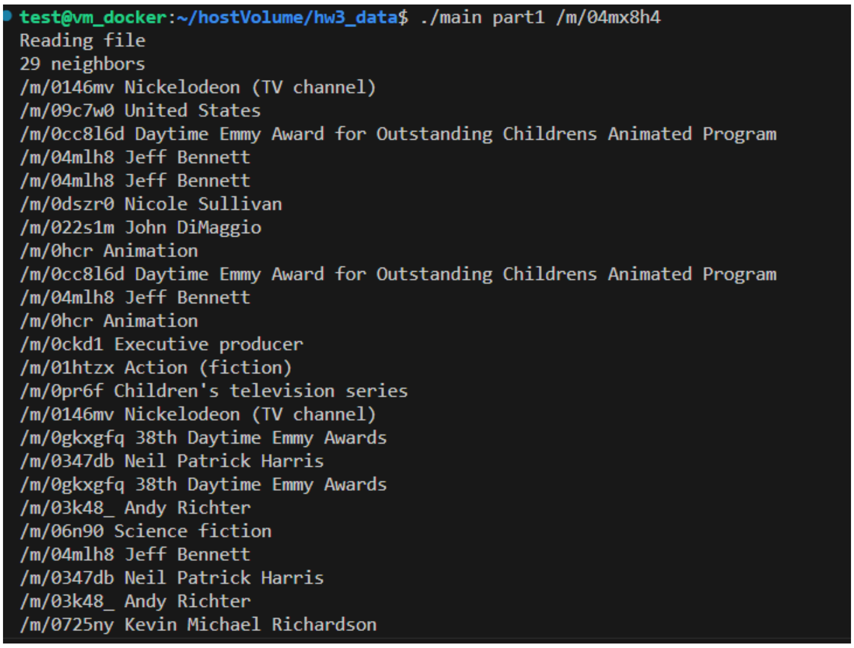
5. **No Path Found**:

   - If the queue is exhausted and no path is found, the function prints `"No path found"`.

5 OUTPUTS

In order to get outputs, argumentation is used. Inside the `int main(int argc, char* argv[])` function, the following steps are implemented:

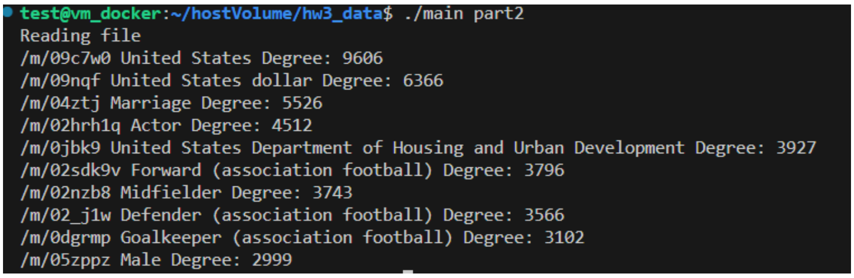
1. **Check for Arguments**:
    - The function checks if at least one argument is provided. If not, it prints the usage message: `"Usage: ./main part1|part2|part3"` and exits.
2. **Choose Part**:
    - The second argument, `argv[1]`, determines which part of the program to execute.
3. **Part 1: Hello Neighbor**:
    - If `part` is `"part1"`, the function checks if exactly three arguments are provided. If not, it prints the usage message: `"Usage: ./main part1 [MID]"` and exits.
    - The function then calls `helloNeighbor(argv[2])` to print the neighbors of the specified MID.



```
test@vm_docker:~/hostVolume/hw3_data$ ./main part1 /m/04mx8h4
Reading file
29 neighbors
/m/0146mv Nickelodeon (TV channel)
/m/09c7w0 United States
/m/0cc8l6d Daytime Emmy Award for Outstanding Childrens Animated Program
/m/04mlh8 Jeff Bennett
/m/04mlh8 Jeff Bennett
/m/0dszr0 Nicole Sullivan
/m/022s1m John DiMaggio
/m/0hcr Animation
/m/0cc8l6d Daytime Emmy Award for Outstanding Childrens Animated Program
/m/04mlh8 Jeff Bennett
/m/0hcr Animation
/m/0ckd1 Executive producer
/m/01htzx Action (fiction)
/m/0pr6f Children's television series
/m/0146mv Nickelodeon (TV channel)
/m/0gkxgfq 38th Daytime Emmy Awards
/m/0347db Neil Patrick Harris
/m/0gkxgfq 38th Daytime Emmy Awards
/m/03k48_ Andy Richter
/m/06n90 Science fiction
/m/04mlh8 Jeff Bennett
/m/0347db Neil Patrick Harris
/m/03k48_ Andy Richter
/m/0725ny Kevin Michael Richardson
```

Figure 4: Part 1 - Output

4. **Part 2: Degree Centrality**:
    - If `part` is `"part2"`, the function checks if exactly two arguments are provided. If not, it prints the usage message: `"Usage: ./main part2"` and exits.



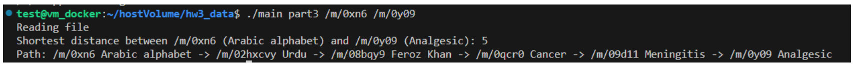
```
test@vm_docker:~/hostVolume/hw3_data$ ./main part2
Reading file
/m/09c7w0 United States Degree: 9606
/m/09nqf United States dollar Degree: 6366
/m/04ztj Marriage Degree: 5526
/m/02hrh1q Actor Degree: 4512
/m/0jbk9 United States Department of Housing and Urban Development Degree: 3927
/m/02sdk9v Forward (association football) Degree: 3796
/m/02nzb8 Midfielder Degree: 3743
/m/02_j1w Defender (association football) Degree: 3566
/m/0dgrmp Goalkeeper (association football) Degree: 3102
/m/05zppz Male Degree: 2999
```

Figure 5: Part 2 - Output

- The function then calls `degreeCentrality()` to calculate and print the top 10 nodes with the highest degree centrality.

5. **Part 3: Shortest Distance**:

    - If `part` is `"part3"`, the function checks if exactly four arguments are provided. If not, it prints the usage message: `"Usage: ./main part3 [MID] [MID]"` and exits.
    - The function then calls `shortestDistance(argv[2], argv[3])` to find and print the shortest path and distance between the two specified MIDs.



```
test@vm_docker:~/hostVolume/hw3_data$ ./main part3 /m/0xn6 /m/0y09
Reading file
Shortest distance between /m/0xn6 (Arabic alphabet) and /m/0y09 (Analgesic): 5
Path: /m/0xn6 Arabic alphabet -> /m/02hxcvy Urdu -> /m/08bqy9 Feroz Khan -> /m/0qcr0 Cancer -> /m/09d11 Meningitis -> /m/0y09 Analgesic
```

Figure 6: Part 3 - Output

6. **Invalid Part**:

    - If the second argument does not match `"part1"`, `"part2"`, or `"part3"`, the function prints the usage message: `"Usage: ./main part1|part2|part3 [MID] [MID] (adding MIDs are proportional to part choice)"` and exits.