ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 223E

DATA STRUCTRES
HOMEWORK REPORT

HOMEWORK NO : 3

DATE : 06.06.2024

150210087 : HİLAL KARTAL

SPRING 2024

Contents

1 INTRODUCTION **1**

2 METHODS **1**

2.1 CHANGES TO SKELETON 1

2.2 MAP OF MID_NAMES 1

2.3 HELLO NEIGHBOUR 1

2.4 DEGREE CENTRALITY 2

2.5 SHORTEST PATH 3

3 RESULTS **4**

3.1 Part 1 5

3.2 Part 2 6

3.3 Part 13 6

1 INTRODUCTION

In this homework we were asked to do given operation on a given graph.

1. Print the neighbours of a given vertice.
2. Find the 10 most central vertices.
3. Find the shortest distance between two vertices.

2 METHODS

2.1 CHANGES TO SKELETON

While reading the files there were some errors, to fix them I have made the given changes:

- While reading from "freebase.tsv" for $ent2$, I have changed `$line.find("\t")$` to `$remain.find("\r")$`. So, code takes the remain, takes of the nr and assigns it to the $ent2$.
- While reading from "mid2name.tsv" for $name$, I have changed `$remain.find("\t")$` to `$remain.find("\r")$`. So, code takes the remain, takes of the `\r$`. and assigns it to the $name$.

2.2 MAP OF MID_NAMES

MID names and their human-readable names are stored in a map called MID_names. Keys are the MID's.

```
1     map<string, string> MID_names = {};
```

While reading we have to use the first occurence of the MID's. This is checked with a if statement

```
1     if (MID_names.find(MID) == MID_names.end()) {
2         MID_names[MID] = name;
3     }
```

2.3 HELLO NEIGHBOUR

In this part we have to print out the neighbours of the given MID. $HelloNeighbour$ function takes a string.

- An iterator is assigned to MID's position in graph_map.

```
auto it = graph_map.find(center_MID);
```

- A node pointer is assigned to it's second which is a node pointer.
- For this node it's adjancency's size is printed out for neighbour count.

```
cout << given_MID->adj.size() <<" neighbours" <<endl;
```

- To print out the neighbours starting from the adjancency's begin, we iterate till end and print out the MID and it's corresponding name.

```
for(auto it = given_MID->adj.begin(); it != given_MID->adj.end(); ++it){
    Node* neigbour = *it;
    cout << neigbour->MID <<" " << MID_names[neigbour->MID] << endl;
}
```

2.4 DEGREE CENTRALITY

In this part we have to print out the 10 most central degrees.

- I have used a priority queue for this part.
- It stores pairs which include the degree of the node and the node itself.

```
priority_queue<std::pair<int,Node*>> max_cent;
for(auto vertice: graph_map){
    max_cent.push({nodeDegree(vertice.second),vertice.second});
}
```

- Node degree is calculated with the function *nodeDegree*. It takes a node pointer and for each node in it's adjancency, it increases the degree.

```
int nodeDegree(Node* vertice){
    int degree = 0;
    for(auto neighbour : vertice->adj){
        degree++;
    }
    return degree;
}
```

2.5 SHORTEST PATH

In this part we are asked to print out the shortest path between two nodes.

1. We need to use BFS. To do so I have declared the function *ShortestPath* which takes in two strings and prints the path.

   - First it finds the corresponding nodes in the graph according to given MID's

```
auto it1 = graph_map.find(MID1);
auto it2 = graph_map.find(MID2);
if (it1 == graph_map.end() || it2 == graph_map.end()) {
    cout << "MID is not in the graph." << endl;
    return;
}
```

   - A queue is initialized for the BFS and two maps are initialized for the visited and parennt nodes. Parent nodes will be used to print out the path.

```
queue<Node*> q;
map<Node*, bool> visited;
map<Node*, Node*> parent;
```

   - Then the algoritm starts. It takes node out marks it as visited and for each node it visits all the neighbours.

```
q.push(start);
visited[start] = true;
while (!q.empty()) {
Node* current = q.front();
q.pop();
if (current == end) {
    break;
}
for (int i = 0; i < current->adj.size(); i++) {
    Node* neighbour = current->adj[i];
    if (!visited[neighbour]) {
        visited[neighbour] = true;
        parent[neighbour] = current;
        q.push(neighbour);
    }
}
}
```

- Then the path is reconstructed using parent map and it is printed out

```
if (visited[end]) {
    vector<Node*> path;
    Node* current = end;
    while (current != start) {
        path.push_back(current);
        current = parent[current];
    }
    path.push_back(start);
    cout << "Shortest path between " << MID1 <<" ("<<MID_names[MID1] <<")
    and " << MID2 <<" ("<< MID_names[MID2] <<") is " << path.size()-1
    << " edges." << endl;
    cout << "Path: ";
    for (int i = path.size()-1; i >= 0; i--) {
        cout << path[i]->MID << " (" << MID_names[path[i]->MID] << ")";
        if (i != 0) {
            cout << " -> ";
        }
    }
    cout << endl;
} else {
    cout << "There is no path between " << MID1 << " and " << MID2
    << "." << endl;
}
```

3 RESULTS

Results of the parts could be seen below

3.1 Part 1

```
test@vm_docker:~/hostVolume/data_hom3/src$ ./main part1 /m/04mx8h4
Part 1:
29 neighbours
/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
/m/01htzx Action (fiction)
/m/0cc8l6d Daytime Emmy Award for Outstanding Childrens Animated Program
/m/0725ny Kevin Michael Richardson
/m/0ckd1 Executive producer
/m/05p553 Anarchic comedy film
```

Figure 1: Results for part1

3.2 Part 2

```
test@vm_docker:~/hostVolume/data_hom3/src$ ./main part2
Part 2:
Ten most central nodes:
Vertice: /m/09c7w0 degree: 9606
Vertice: /m/09nqf degree: 6366
Vertice: /m/04ztj degree: 5526
Vertice: /m/02hrh1q degree: 4512
Vertice: /m/0jbk9 degree: 3927
Vertice: /m/02sdk9v degree: 3796
Vertice: /m/02nzb8 degree: 3743
Vertice: /m/02_j1w degree: 3566
Vertice: /m/0dgrmp degree: 3102
Vertice: /m/05zppz degree: 2999
```

Figure 2: Results for part2

3.3 Part 13

```
test@vm_docker:~/hostVolume/data_hom3/src$ ./main part3 /m/0xn6 /m/0y09
Part 3:
Shortest path between /m/0xn6 (Arabic alphabet) and /m/0y09 (Analgesic) is 5 edges.
Path: /m/0xn6 (Arabic alphabet) -> /m/02hxcvy (Urdu) -> /m/08bqy9 (Feroz Khan) -> /m/0qcr0 (Cancer) -> /m/09d11 (Meningitis) -> /m/0y09 (Analgesic)
```

Figure 3: Results for part3