Advanced Algorithm's Project

Twitter Followers Project

Team members:

Nourhan Ashraf

Nihal Khalid

Mariam Tarek

Mona Khalid

Shrouk Abdallah

Abdelrahman Hesham

GitHub Repository link:

nourhan-ashraf/Large-Social-Network-Algorthims-Project

Code Explanation and Time Complexity

Main:

```
int main()
{
    ifstream myFile("twitter.csv"); //open the data file

    if (myFile.is_open())
    {
        string line;
        while(getline(myFile,line))
        {
            stringstream ss(line);
            string x;
            string y;
            getline(ss,x,',');
            getline(ss,y,',');
            int follower, account;
            follower = stoi(x);
            account = stoi(y);

           buildGraph(follower, account);
        }

    }
    else{
        cout<<"Cannot open the file!"<<endl;
        return 0;
    }
```

First we open “twitter.csv” file and check if it can be opened, if not then print an error message.

If the file opens correctly then it will enter a while loop to read two values of each line and store the values of each account ID and the number of its followers using “buildGraph” function.

While loop Time complexity: O(Nlog(N))

N is the number of lines in the file.

```
    cout<<"Enter the influncer order: ";
    int num; cin>>num;
    if(num<1 || num>81305){
        cout<<"Wrong input!"<<endl;
        return 0;
    }
    printDegree(num);

    return 0;
}
```

In the main function before we call “printDegree” function

to print the output we check if the input exceeds the number of accounts which is 81305 or it was smaller than 1, then print wrong input, else it will call “printDegree” function.

“buildGraph” function:

```
map<int,int>mapping;
map<int,int>mapping2;
set<int>adjList[810000];

int k=0;
void buildGraph(int follower, int account)
{
        if(!mapping.count(account)){
            k++;
            mapping[account] = k;
            mapping2[k] = account;
        }
        adjList[mapping[account]].insert(follower);
}
```

In this function we asign for each account an index to put it in the list of vertices, the reason we did that is that the IDs value is very big and it will exceed the heap space if we don't assign an index for each ID.

then add list of followers to each vertex's index.

This function will be called N times in the while loop

Count function takes **O(log(N))** in each call.

Insert in set takes **O(log(N)).**

The time complexity for 1 call is **O(log(N))**

N is the number of lines in the CSV file -> 2420766

"printDegree" function:

```
set<pair<int ,int>>ans;
vector<pair<int, pair<int,int>>>v;

void printDegree(int num){
    for (int i = 1; i <=81305; i++) {
        int lst = adjList[i].size();
        ans.insert(make_pair(lst,mapping2[i]));
    }
    int it = 81305;
    for(auto u:ans){
        v.push_back(make_pair(it, make_pair(u.second, u.first)));
        it--;
    }
    for(auto j:v){
        if(num==j.first){
            cout<<"Account ID: "<<j.second.first<<" , "<<"Followers: "<<j.second.second<<endl;
            break;
        }
    }
}
```

This function we used set of pairs that contains the ID and the size of each ID followers list sorted by the size, and vector of pairs of pairs to map between three values (index, ID, size) to assign index for each size after sort it.

Then it prints the ID and size that are mapping to the index that the user entered.

Time complexity is **O(n)**

n: is the number of the IDs.

While loop in the main -> O(Nlog(N))

"printDegree" function in the main -> O(n)

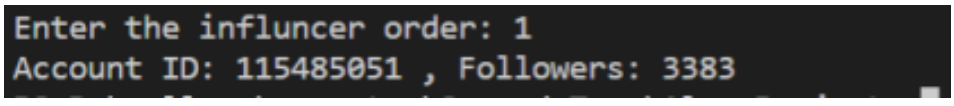
Time complexity (worst case) of the whole code is:

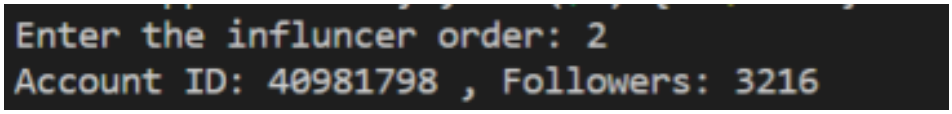
O(Nlog(N))

N is the number of lines in the CSV file -> 2420766

n is the number of the accounts -> 81305

Output:





Bonus code:

Main:

```
int main()
{

    ifstream myFile("twitter.txt");

    if (myFile.is_open())
    {
        string line;
        while(getline(myFile,line))
        {
            stringstream ss(line);
            string x;
            string y;
            getline(ss,x,',');
            getline(ss,y,',');
            int following, account;
            account = stoi(x);
            following = stoi(y);

           buildGraph(account, following);
        }

    }
    else{
        cout<<"Cannot open the file!"<<endl;
        return 0;
    }
```

First we open “twitter.csv” file and check if it can be opened, if not then print an error message.

If the file opens correctly then it will enter a while loop to read the two values of each line and store the values of each account ID and its following list using **“buildGraph”** function.

Time complexity: O(Nlog(N))

Which N is the number of lines in the file.

log(N) -> because of the count function in “buildGraph” function.

```
int ID;
cout<<"Enter ID account: ";
cin>>ID;
bool flag = 0;
for(int i=1;i<=mapping.size();i++){
    if(mapping[ID]) {
        flag = 1;
        break;
    }
}
if(flag){
    searchAccounts(graph,ID);
}
else{
    cout<<"This ID doesn't exist!"<<endl;
}
```

Handle the case if the user entered wrong account ID.

---

buildGraph function:

```
map<int,int>mapping;
set<int>adjList[810000];

int k=0;
void buildGraph(int account, int following)
{

        if(!mapping.count(account)){
            k++;
            mapping[account] = k;
        }
        adjList[mapping[account]].insert(following);

    }
```

In this function we asign for each account an index to put it in the list of vertices, the reason we did that is that the IDs value is very big and it will exceed the heap space if we don't assign an index for each ID.

then add list of following to each vertex's index.

This function will be called N times in the while loop

Count function takes **O(log(N))** in each call.

Insert in set takes **O(log(N)).**

The time complexity for 1 call is **O(log(N))**

N is the number of lines in the CSV file -> 2420766

---

searchAccounts function:

```
void searchAccounts(int account)
{
        unordered_set<int>ansList;
        set<int> lst = adjList[mapping[account]];
        cout <<"Accounts you may know: ";
        auto itr = lst.begin();
        int cnt=3;
        while(itr != lst.end() && cnt!=0){
            set<int> follow = adjList[mapping[*itr]];
            int cnt2=3;
            for(auto d = follow.begin() ; d != follow.end() ;++d){
                if( (!(lst.count(*d))) && cnt2!=0 && (*d)!=account){

                    ansList.insert(*d);
                    //cout << *d<<"  ";
                    cnt2--;
                }
            }
            cnt--;
            ++itr;
        }
        for(auto o:ansList) cout<<o<<" ";

}
```

In this function first we put the following list of the entered account ID in a set “lst” and then

loop on the first 3 IDs in this list,

then loop on the following list of for each ID to find an account ID that the entered account ID doesn’t follow and add it to “ansList” set

repeat this loop 3 times then break if it found three IDs or it can loop on the whole list and does not find any account ID that match our desires in this following list, so it breaks and start another loop with different account ID following list.

It prints the answer after the while loop

The time complexity: O(D*log(D))

Assume D = (*d)

While loop is only repeated for 3 times.

For loop (worst case) is repeated D times and for each D: “count” function in set costs O(log(D)).

D: is the number of the following in the following list (depends on each account ID following list) -> max following list size for all the IDs is 3373.

While loop in the main -> O(Nlog(N))

“searchAccounts” function in the main -> O(D*log(D))

Time complexity (worst case) in the bonus code is:

O(N*log(N))

N is the number of lines in the CSV file -> 2420766.

D: is the number of the following in the following list (depends on each account ID following list) -> max following list size for all the IDs is 3373.

Output:

```
Enter ID account: 40981798
Accounts you may know: 40288959 37788669 19432203 15099384 14607172 14389658 15186688 14994465 14486007
```

```
Enter ID account: 37788669
Accounts you may know: 17868918 17759158 17026571 15846407 14607172 15957907 21362105 818647 813286
```

If we combined the project main question code and the Bonus code

The time complexity (worst case)

Assuming that the user entered valid inputs

O(N*log(N))