Advanced Algorithm's Project

Twitter Followers Project

Team members:

Mohamed Abd-El Ghani Mohamed

Mahmoud Mohamed Attia

Ahmed Bahaa Mohmed

Yasmin Ahmed Mounir

Code Explanation and Time Complexity

Input section: -

```
ios_base::sync_with_stdio(0), cin.tie(0), cout.tie(0);
ifstream infile("twitter.csv");
if (infile.fail())
{
    cerr << "Error Opening file" << endl;
    exit(1);
}
string c;
while (!infile.eof())
{
    string a, b;
    infile >> c;
    a = "", b = "";
    int idx = 0;
    for (int i = 0; i < (int) c.size(); i++)
    {
        if (c[i] == ',')
        {
            idx = i + 1;
            break;
        }
        a.push_back(c[i]);
    }
    for (int i = idx; i < c.size(); i++)
    {
        b.push_back(c[i]);
    }
    int x = str_int(a);
    int y = str_int(b);
    adj_1[x].insert(y);
    adj_2[y].insert(x);
}
infile.close();
```

- Assume that the number of nodes that occurs in the input file = n.

- Assume that the number of edges = m.

- Assume that the length of string s = |s|.

Time complexity: m * 2 |s| * (log(n) + log(n)) = O (m * |s| * log(n)).

Build-tsk-1 function: -

```
void build()
{
    for (auto x : adj_2)
    {
        int ft = x.first, sc = x.second.size();
        influencers.push_back({sc, ft});
    }
    sort(influencers.begin(), influencers.end());
    reverse(influencers.begin(), influencers.end());
}
```

- In the worst-case scenario adj_2 will contain all the nodes.

- Time complexity of sort (built-in function) in C++ is n log(n), where n is the size of data, I want to sort it.

- Time complexity of reverse (built-in function) in C++ is n.

Time complexity is:

O (n + n log(n) + n) =O (2n + n log(n)) = O (n log(n)).

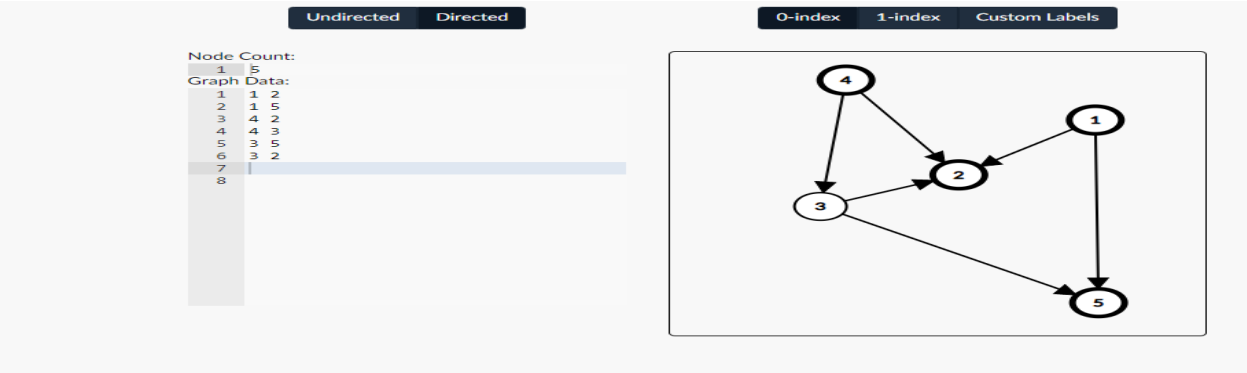
Task-1 function: -

```
void tsk_1()
{
    int number;
    cout << "Enter the rank of influencer you want : " << endl;
    cin >> number;
    number--;
    cout << "The influencer ID = " << influencers[number].second;
    cout << " and the number of followers of the influencer = " << influencers[number].first << endl;
}
```

Time complexity: O (1).

Task-2 function: -

```cpp
void tsk_2()
{
    int number;
    cout << "Enter the ID of influencer you want : " << endl;
    cin >> number;
    int mx = 0, user = -1;
    map<int, int> freq;
    for (auto x:adj_1[number])
    {
        for (auto y:adj_2[x])
        {
            if (y != number && (adj_1[number].lower_bound(y) == adj_1[number].end() || *(adj_1[number].lower_bound(y)) != y))
            {
                freq[y]++;
                if (freq[y] > mx)
                {
                    mx = freq[y];
                    user = y;
                }
            }
        }
    }
    if (user == -1) cout << "No recommendations" << endl;
    else cout << "The ID of influencer : " << user <<" and the number of common influencer : "<<mx<< endl;
}
```



- In the worst-case scenario adj_1 and adj_2 will contain all the nodes.

- we use lower bound built-in function in C++ (log(n)) to check that ID I will increment its frequency is a valid one or not.

- Dealing with map (data structures) is cost log(n).

Time complexity is: O (n^2log(n)).

Task-3 function: -

```
void tak_3()
{
    vector<int>mutual;
    int number_1, number_2;
    cout << "Enter the ID of the first person and the ID of the second one : " << endl;
    cin >> number_1 >> number_2;
    while(number_1==number_2)
    {
        cout<<"Enter two different IDs :"<<endl;
        cin>>number_1>>number_2;
    }
    auto it_1 = adj_1[number_1].begin(), it_2 = adj_1[number_2].begin() ;
    auto lst_1 = adj_1[number_1].end(), lst_2 = adj_1[number_2].end() ;
    while(it_1!=lst_1 && it_2!=lst_2)
    {
        if(*it_1<*it_2)
            it_1++;
        else if(*it_1>*it_2)
            it_2++;
        else
        {
            mutual.push_back(*it_1);
            it_1++;
            it_2++;
        }
    }
    if(mutual.size()!=1)
    {
        if(!mutual.empty())
            cout<<"There are "<<mutual.size()<<" mutual friends between ID = "<<number_1<<" and ID = "<<number_2<<" and the list of mutual friends is : "<<endl;
        else
            cout<<"There are "<<mutual.size()<<" mutual friends between ID = "<<number_1<<" and ID = "<<number_2<<endl;
    }
    else
        cout<<"There are "<<mutual.size()<<" mutual friend between ID = "<<number_1<<" and ID = "<<number_2<<" and the list of mutual friends is : "<<endl;
    if(!mutual.empty())
    {
        for(auto x : mutual)
            cout<<x<<'\n';
        cout<<endl;
    }
}
```

- We will need to use two pointers (one forward to begin and the other forward to the end) to go into adj_1 of each ID and it will cost time of log(n) for initialization of two pointers and 2n to move into the adjacency of each ID.

Time complexity is: O (n).