TÜRKİYE CUMHURİYETİ
YILDIZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

ALGORİTMA ANALİZİ
ÖDEV-4

Öğrenci No: 19011010
Öğrenci Adı Soyadı: Beyda Güler
Öğrenci e-posta: beyda.guler@std.yildiz.edu.tr

Ders / Grup: BLM3021-Algoritma Analizi / Grup:1

Ders Yürütücüsü:
Mine Elif KARSLIGİL
Aralık, 2022

İçindekiler Tablosu

1.YÖNTEM

PROBLEM

Verilen dosyada kişiler ve takip ettikleri kişilerin numaraları bulunmaktadır. Bu girdiyi bir sosyal ağ olarak kabul edip influencer olan kişileri bulmamız istenmektedir.

ÇÖZÜM

```
typedef struct node{
    int vertex;
    int in_degree;
    int deleted;
    char name[20];
    char surname[20];
    struct node* next;
    int connection;
    int isInfluencer;
}NODE;

typedef struct adjList {
    struct node* list;
}ADJLIST;

typedef struct graph{
    int numOfVertices;
    struct adjList* adjLists;
}GRAPH;
```

Dosyada bilgileri verilen her kişiyi bir node olarak ve takip ettiği kişileri de bir *adjacency list* (linked list) olarak tuttum.

Adjacency list'in head düğümü (adjList structındaki list değişkeni) her bir kişi için numara değerini tutmaktadır. Bu düğümden next ile ilerlediğimizde ise kişinin takip ettiği diğer kişilerin numaralarına erişmiş oluruz. Bu şekilde kullanıcılar arasında yönlü bağlantıları oluşturmuş ve *directed* bir graf yapısı elde etmiş oluyoruz.

- **void addEdge(GRAPH* graph, int s, int d)**

Fonksiyona verilen bir source düğümden (s), destination düğüme (d) yönlü bir bağlantı oluşturmak için kullanılan fonksiyon.

- **void calculateInDegree(GRAPH* graph)**

Dosyadan okuduğumuz kişiler için in-degree değerlerini yani takipçi sayılarını hesaplamaya yarayan fonksiyon.

- **void deleteNodes(GRAPH *graph,int M,int i)**

Kişilerin in-degree değerleri kullanıcı tarafından girilen bir M değeriyle karşılaştırılır ve in-degree'si M'den küçük olanlar graph'tan silinir. Bu silinme işlemi deleted adlı flag ile takip edilmiştir. Bu işlemin ardından silinen kullanıcıların takip ettiği kişilerin de in-degree'leri azalacağından bu işlem M'den küçük bir in-degree kalmayana kadar recursive bir şekilde gerçekleştirilmiştir.

- **GRAPH* updateFollowed(GRAPH *graph,int v)**

Takip edilen kişileri tutan adjacency listin deleteNodes fonksiyonu ardından güncellenmesi için kullanılan fonksiyon.

- **void findConnections(GRAPH *graph,int i,int *visited)**

Verilen i değerine karşılık gelen kişinin doğrudan ve dolaylı bütün bağlantılı olduğu kişiler (takipçileri ve takipçilerinin takipçileri) bu fonksiyonda hesaplanır. *DFS yöntemi* takip edilerek bir düğüm için takip ettiği bütün düğümler gezilmiştir.

- **void findInfluencers(GRAPH *graph,int X,int Y)**

deleteNodes fonksiyonu ardından güncellenen in-degree değerleri kullanıcı tarafından girilen bir X değerinden büyük olan, findConnections fonksiyonu ile hesaplanan total bağlantı sayısı yine kullanıcı tarafından girilen bir Y değerinden büyük olan ve delete flag'ı 0'a eşit olan (herhangi bir silme işlemine uğramamış) düğümlerin influencer seçilerek isInfluencer flaglarının 1 yapılmasını sağlayan fonksiyon.

2.UYGULAMA

MENU

```
            ****** WELCOME ******

            BEYDA GULER  19011010

            ***********************
            1 - Normal Mode
            2 - Deatiled Mode
            3 - EXIT
            ***********************
            Choose the mode -> 1

      Please enter the value of M --> 1

      Please enter the value of X --> 2

      Please enter the value of Y --> 5
```

a) Normal Mod

```
                        ****NORMAL MODE****

                      HERE IS THE INFLUENCERS!

6  Lieven Vandenberghe  he/she has 11 connections and 2 in-degree value.
8  Jorge Nocedal        he/she has 11 connections and 4 in-degree value.
10  Stephen Wright      he/she has 11 connections and 2 in-degree value.
11  Philippe Salembier  he/she has 11 connections and 2 in-degree value.
12  Robert Stevenson    he/she has 11 connections and 2 in-degree value.

BYE:)
```

b) Detaylı Mod

```
                ****DETAILED MODE*****

            HERE IS THE DETAILS YOU OVERTHINKER!

           THIS IS HOW IT SEEM AT THE BEGINNING...

         1  Michael  Jordan      -> 2 -> 3 -> 5 -> null

         2  Stephen  Boyd        -> 3 -> 1 -> null

         3  Kalyanmoy  Deb       -> 1 -> 2 -> null

         4  David  Johnson       -> 6 -> null

         5  Scott  Kirkpatrick   -> 6 -> 7 -> null

         6  Lieven  Vandenberghe -> 8 -> null

         7  Fabian  Pedregosa    -> 8 -> null

         8  Jorge  Nocedal       -> 11 -> 10 -> 9 -> null

         9  Clifford  Stein      -> 4 -> null

         10  Stephen  Wright     -> 8 -> 12 -> null

         11  Philippe  Salembier -> 12 -> null

         12  Robert  Stevenson   -> 8 -> 10 -> 11 -> null

                        IN-DEGREE VALUES

            Vertex Number : 1     In-degree : 2
            Vertex Number : 2     In-degree : 2
            Vertex Number : 3     In-degree : 2
            Vertex Number : 4     In-degree : 1
            Vertex Number : 5     In-degree : 1
            Vertex Number : 6     In-degree : 2
            Vertex Number : 7     In-degree : 1
            Vertex Number : 8     In-degree : 4
            Vertex Number : 9     In-degree : 1
            Vertex Number : 10    In-degree : 2
            Vertex Number : 11    In-degree : 2
            Vertex Number : 12    In-degree : 2
```

```
        NODES THAT ARE NOT DELETED AFTER DELETION(M=1)

        Vertex:1        In-degree 2     Michael Jordan
        Vertex:2        In-degree 2     Stephen Boyd
        Vertex:3        In-degree 2     Kalyanmoy Deb
        Vertex:4        In-degree 1     David Johnson
        Vertex:5        In-degree 1     Scott Kirkpatrick
        Vertex:6        In-degree 2     Lieven Vandenberghe
        Vertex:7        In-degree 1     Fabian Pedregosa
        Vertex:8        In-degree 4     Jorge Nocedal
        Vertex:9        In-degree 1     Clifford Stein
        Vertex:10       In-degree 2     Stephen Wright
        Vertex:11       In-degree 2     Philippe Salembier
        Vertex:12       In-degree 2     Robert Stevenson
```

```
                        INFLUENCER INFORMATIONS

Lieven Vandenberghe -->  has 11 total (directed/undirected) connections and 2 in-degree value.
Jorge Nocedal -->        has 11 total (directed/undirected) connections and 4 in-degree value.
Stephen Wright -->       has 11 total (directed/undirected) connections and 2 in-degree value.
Philippe Salembier -->   has 11 total (directed/undirected) connections and 2 in-degree value.
Robert Stevenson -->     has 11 total (directed/undirected) connections and 2 in-degree value.
```

3.SONUÇ

ZAMAN KARMAŞIKLIKLARI

- calculateInDegree() fonksiyonunun karmaşıklığı

```
void calculateInDegree(GRAPH* graph){
    int size = graph->numOfVertices,v;
    int vertex;
    NODE* list;
    for(v=1;v<=size;v++){
        list = graph->adjLists[v].list;
        if(list->deleted != 1){
            list = list->next;
            while(list){
                vertex = list->vertex;
                graph->adjLists[vertex].list->in_degree++;
                list=list->next;
            }
        }
    }
}
```

N : düğüm sayısı ve **E** : edge (bağlantı) sayısı olmak üzere;

Zaman Karmaşıklığı : O(N*E) ile ifade edilir.

- deleteNodes() fonksiyonunun karmaşıklığı

```
void deleteNodes(GRAPH *graph,int M,int i){

    NODE *list=graph->adjLists[i].list;
    if(list->deleted == 1)
        return;
    else{
        if(list->in_degree < M){
            list->deleted = 1;
            graph = updateFollowed(graph,list->vertex);
            list=list->next;
            while(list){
                graph->adjLists[list->vertex].list->in_degree--;
                deleteNodes(graph , M , list->vertex);
                list = list->next;
            }
        }
    }
}
```

N : düğüm sayısı ve **E** : edge (bağlantı) sayısı olmak üzere;

Zaman Karmaşıklığı : O(N+E)'dir fakat bu fonksiyon ile tek bir node için işlem yapılmaktadır. Bütün node'lar için işlem yapılması main fonksiyonundaki for döngüsü ile sağlandığından zaman karmaşıklığını O(N*(N+E)) ile ifade etmek daha doğru olacaktır.

- updateFollowed() fonksiyonunun karmaşıklığı

```
GRAPH* updateFollowed(GRAPH *graph,int v){
    int i;
    NODE* tmp;
    for(i=1;i<=graph->numOfVertices;i++){
        tmp = graph->adjLists[i].list;
        if(tmp->deleted != 1){
            tmp = tmp->next;
            while(tmp){
                if(tmp->vertex == v){
                    tmp->deleted = 1;
                }
                tmp = tmp->next;
            }
        }
    }
    return graph;
}
```

N : düğüm sayısı ve **E** : edge (bağlantı) sayısı olmak üzere,

Zaman Karmaşıklığı : O(N*E)'dir.

- findConnections() fonksiyonunun karmaşıklığı

```
void findConnections(GRAPH *graph,int i,int *visited){

    visited[i] = 1;
    NODE* tmp = graph->adjLists[i].list;
    tmp=tmp->next;
    while(tmp){
        if(visited[tmp->vertex] == 0 && tmp->deleted == 0){
            graph->adjLists[tmp->vertex].list->connection++;
            visited[tmp->vertex] = 1;
            findConnections(graph,tmp->vertex,visited);
        }
        tmp = tmp->next;
    }
}
```

N : düğüm sayısı ve **E** : edge (bağlantı) sayısı olmak üzere,

Zaman Karmaşıklığı : O(N+E)'dir fakat main'de for döngüsü ile her düğüm için çağrıldığı için (N defa) O(N*(N+E)) şeklinde ifade edebiliriz.

- findInfluencers() fonksiyonunun karmaşıklığı

```
void findInfluencers(GRAPH *graph,int X,int Y){
    int i;
    for(i=1;i<=graph->numOfVertices;i++){
        if((graph->adjLists[i].list->in_degree >= X) && (graph->adjLists[i].list->connection >= Y) && (graph->adjLists[i].list->deleted == 0))
            graph->adjLists[i].list->isInfluencer = 1;
        else
            graph->adjLists[i].list->isInfluencer = 0;
    }
}
```

N : düğüm sayısı olmak üzere,

Zaman Karmaşıklığı : O(N)'dir.

VİDEO LİNKİ

https://youtu.be/NmOIqfJFgPo