Tecnológico de Monterrey

Documentación de código

Programming of data structures

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062

Profesores: Luis Ricardo Peña Llamas
Jorge Gonzalez

15/11/2022

Implementación de grafos

Para la entrega se implementa la estructura de grafos para determinar Outdegree e Indegree dentro de cada nodo del grafo añadido, para esto a partir de la lectura de un archivo se crea una matriz de adyacencia que almacene la relación con cada nodo dentro del grafo, lo que indica una complejidad de O(n^2) ya que es la cantidad de elementos que se evaluarían dentro de la matriz de adyacencia

```
class Graph{    //Grafo de elementos de subIps

    private:
        int numVertices;
        int graphNum;
        bool** adjMatrix;
        list<int>* adjList;
        bool* visited;
        void verifyElement(RNode*, set<int>&, queue<Node*>&);
    public:
        Graph(){
            graphNum = 0;
            numVertices = 0;
            adjList = new list<int>[0];
            visited = new bool[0];
        }
        ~Graph(){
            delete []adjList;
            delete []visited;
            delete []adjMatrix;
        }
        void addEdge(int, int); //O(1)
        void printGraph();  //O(n+m)
        void resetVisited(); //O(n+m)

        void matrixInit();  //O(n)
        void loadGraph(string); //O(n)
        void printMatrix(); //O(n^2)
        void saveMatrix();  //O(n^2)
        int InDegree(int);  //O(n)
        int OutDegree(int); //O(n)
        void saveInOutDegree(); //O(n)
        void saveInDegree();    //O(n)
        void saveOutDegree();   //O(n)

};
```

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062

La función load graph es el encargado de obtener los elementos de un archivo txt y obtener las subIPs de cada registro para crear una relación con el resto de los nodos

Internamente se utiliza una estructura de datos que almacene los datos con complejidad constante y borrar el dato de forma constante, en este caso se utilizó una estructura Queue para este proceso, ya que al mismo tiempo se realiza una clasificación de la cantidad de subIPs (por medio de un set) diferente que se van registrando, de manera que sepamos la longitud y anchura con la que se creara la matriz de adyacencia

```
void Graph::loadGraph(string fileName){
    std::ifstream file;
    file.open(fileName);
    string month, day, hour, ip, reason;
    try{
    queue<Node*> queue;
    set<int> setGraphs;
    set<int>::iterator it;
    int pos1, pos2;
    while (file >> month>> day>> hour>> ip)
        {
            getline(file, reason);
            RNode* newNode = new RNode(month, day, hour, ip, reason);
            verifyElement(newNode, setGraphs, queue);
        }
    int max = 0;
    for (auto it = setGraphs.begin(); it!=setGraphs.end();it++)
        if (max < *it)
        max = *it;

    this->graphNum = max+1;
    this->numVertices = setGraphs.size();
    adjList = new list<int>[graphNum];
    visited = new bool[graphNum];
    matrixInit();
    while (queue.size() > 0)
    {
        Node* current = queue.front();
        queue.pop();
        addEdge(current->n, current->s);
        adjMatrix[current->n][current->s] = true;
    }
    }
    catch(invalid_argument e){
        cerr << e.what() << endl;
    }
}
```

El proceso completo involucra un llenado de una queue, lo que indica O(n) + la verificación del dato en un set, lo cual según la documentación de la estructura para la función usada es O(n) dentro de los datos filtrados. Sin embargo, dentro de esta función se crea la matriz de adyacencia de forma dinámica, lo que el peor de los casos tiene complejidad O(n^2)

Para obtener el Indegree y Outdegree se implementa una función que recorre una posición a lo largo de una columna/renglon, lo que indica una complejidad de O(n) para ambas funciones

```
int Graph::InDegree(int nodeValue){
    int count = 0;
    for (int i = 0; i < graphNum; i++)
        if (adjMatrix[i][nodeValue])
            count++;
    return count;
}

int Graph::OutDegree(int nodeValue){
    int count = 0;
    for (int i = 0; i < graphNum; i++)
        if (adjMatrix[nodeValue][i])
            count++;
    return count;
}
```

Para obtener el ordenamiento de datos por Outdegree e Indegree se hizo la implementación de un Heap, para de esta manera crear un Heap sort que con el que se puede obtener y ordenar los datos de manera rápida

```
class Heap{     //Estructura utilizada para ordenamiento por Indegree o Outdegree
    int len, maxLen;
    DegreeNode** degreeList;
    int condition;
    private:
        void swap(int&, int&);  //O(1)
        void bubbleDown(int&);  //O(Logn)
    public:
        Heap(int num, int condition){
            this->condition = condition;
            len = -1;
            maxLen = num;
            degreeList = new DegreeNode*[num+1];
            };
        void push(DegreeNode*&);  //O(Logn)
        DegreeNode* pop();    //O(Logn)
        void save(string);  //O(nLogn)
        void print();   //O(n)
};
```

```
class DegreeNode{   //Clase que facilita el uso de Heap para ordenamiento por outdegree e indegree
    public:
        int value;
        int outDegree;
        int inDegree;
        DegreeNode(){
            value = -1;
            outDegree = -1;
            inDegree = -1;
        }
        DegreeNode(int value, int outDegree, int inDegree){
            this->value = value;
            this->outDegree = outDegree;
            this-> inDegree = inDegree;
        }
};
```

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062

Como resultado final se obtienen 4 archivos del proceso

Almacenamiento de matriz de adyacencia obtenida en AdjMatrix.csv para análisis de desarrollo

Almacenamiento de las subIps en InOutDegree.txt (Contiene el InDegree y OutDegree ordenado por valor de subIp)

Almacenamiento de las subIps en InDegree.txt (Contiene el InDegree y OutDegree ordenado por InDegree)

Almacenamiento de las subIps en OutDegree.txt (Contiene el InDegree y OutDegree ordenado por OutDegree)

impresión de Top 10 subIp con máximo valor InDegree

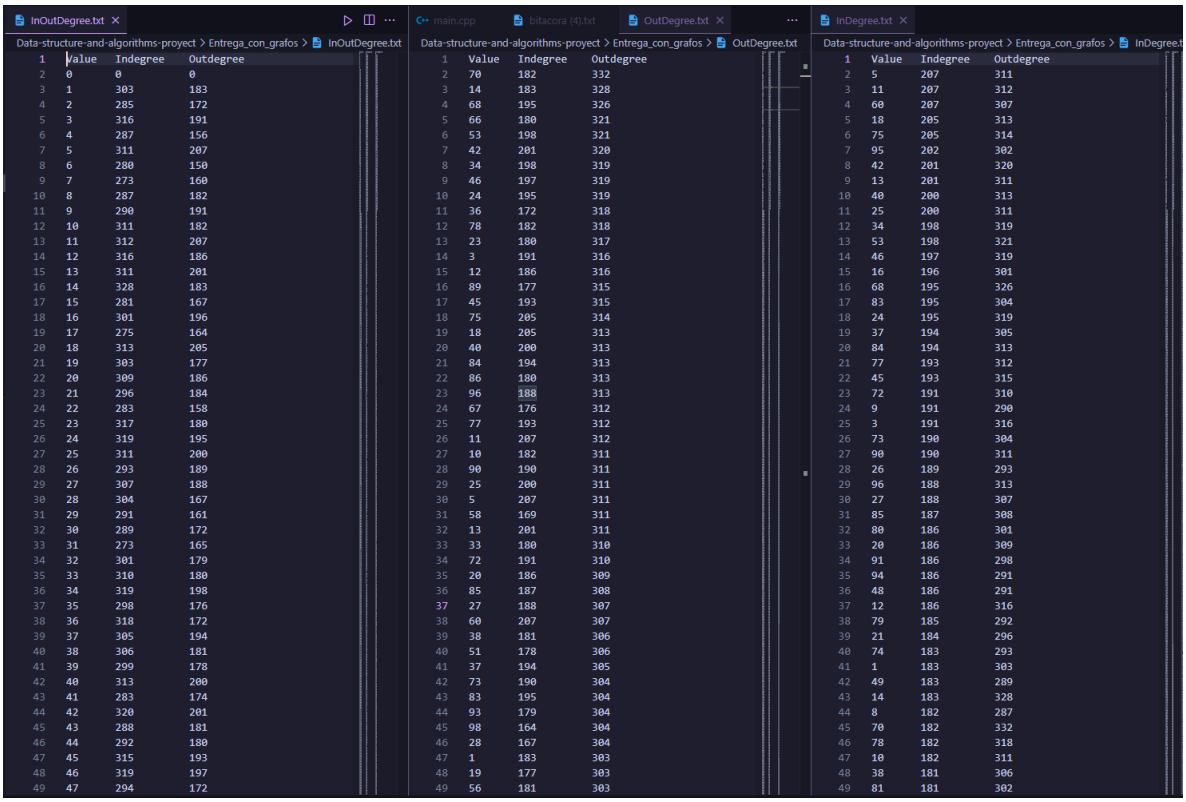
impresión de Top 10 subIp con máximo valor OutDegree

```
int main(){
    Graph g;
    g.loadGraph("bitacora (4).txt");
    g.saveMatrix(); //Almacenado en AdjMatrix.txt
    cout << "\nAnalizando Indegree, Outdegree...\n";
    g.saveInOutDegree();    //Almacenado en InOutDegree.txt
    cout << "\nArchivo Indegree, Outdegree completo\n";
    g.saveOutDegree();      //Almacenado en OutDegree.txt
    cout << "\nArchivo Outdegree completo\n";
    g.saveInDegree();       //Almacenado en InDegree.txt
    cout << "\nArchivo Indegree completo\n";
    readInOutDegreeResults();

    return 0;
}
```

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062

```
Analizando Indegree, Outdegree...

Archivo Indegree, Outdegree completo

Archivo Outdegree completo

Archivo Indegree completo
................................
Top 10 sub Ip con mayor InDegree:

Value           Indegree        Outdegree
5               207             311
11              207             312
60              207             307
18              205             313
75              205             314
95              202             302
42              201             320
13              201             311
40              200             313
25              200             311

Top 10 sub Ip con mayor OutDegree:

Value           Indegree        Outdegree
70              182             332
14              183             328
68              195             326
66              180             321
53              198             321
42              201             320
34              198             319
46              197             319
24              195             319
36              172             318
```

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062



InOutDegree.txt

Data-structure-and-algorithms-proyect > Entrega_con_grafos > InOutDegree.txt

Value	Indegree	Outdegree
0	0	0
1	303	183
2	285	172
3	316	191
4	287	156
5	311	207
6	280	150
7	273	160
8	287	182
9	290	191
10	311	182
11	312	207
12	316	186
13	311	201
14	328	183
15	281	167
16	301	196
17	275	164
18	313	205
19	303	177
20	309	186
21	296	184
22	283	158
23	317	180
24	319	195
25	311	200
26	293	189
27	307	188
28	304	167
29	291	161
30	289	172
31	273	165
32	301	179
33	310	180
34	319	198
35	298	176
36	318	172
37	305	194
38	306	181
39	299	178
40	313	200
41	283	174
42	320	201
43	288	181
44	292	180
45	315	193
46	319	197
47	294	172

main.cpp | bitacora (4).txt | OutDegree.txt

Data-structure-and-algorithms-proyect > Entrega_con_grafos > OutDegree.txt

Value	Indegree	Outdegree
70	182	332
14	183	328
68	195	326
66	180	321
53	198	321
42	201	320
34	198	319
46	197	319
24	195	319
36	172	318
78	182	318
23	180	317
3	191	316
12	186	316
89	177	315
45	193	315
75	205	314
18	205	313
40	200	313
84	194	313
86	180	313
96	188	313
67	176	312
77	193	312
11	207	312
10	182	311
90	190	311
25	200	311
5	207	311
58	169	311
13	201	311
33	180	310
72	191	310
20	186	309
85	187	308
27	188	307
60	207	307
38	181	306
51	178	306
37	194	305
73	190	304
83	195	304
93	179	304
98	164	304
28	167	304
1	183	303
19	177	303
56	181	303

InDegree.txt

Data-structure-and-algorithms-proyect > Entrega_con_grafos > InDegree.t

Value	Indegree	Outdegree
5	207	311
11	207	312
60	207	307
18	205	313
75	205	314
95	202	302
42	201	320
13	201	311
40	200	313
25	200	311
34	198	319
53	198	321
46	197	319
16	196	301
68	195	326
83	195	304
24	195	319
37	194	305
84	194	313
77	193	312
45	193	315
72	191	310
9	191	290
3	191	316
73	190	304
90	190	311
26	189	293
96	188	313
27	188	307
85	187	308
80	186	301
20	186	309
91	186	298
94	186	291
48	186	291
12	186	316
79	185	292
21	184	296
74	183	293
1	183	303
49	183	289
14	183	328
8	182	287
70	182	332
78	182	318
10	182	311
38	181	306
81	181	302

Inés Alejandro Garcia Mosqueda A00834571
Jesus Fong Ruiz A01254062