

Memoria Grafos

Jorge Urbelz Alonso-Cortés

Grado Ingeniería Informática, Universidad de Murcia

Algoritmos y Estructuras de Datos I

Jesús Sánchez Cuadrado


23 de diciembre, 2023









Uso de un comodín (equivale a ejercicio amarillo)

Contenido

| | |
|--|----|
| Tabla resultados Mooshak..... | 3 |
| Envío ejercicios..... | 4 |
| Ejercicios verdes..... | 4 |
| Ejercicio 401 | 4 |
| Ejercicio 402 | 5 |
| Ejercicios azules..... | 7 |
| Ejercicio 403 | 7 |
| Ejercicio 407 | 9 |
| Ejercicios amarillos..... | 11 |
| Conclusiones y valoración del trabajo..... | 12 |
| Bibliografía | 13 |

Tabla resultados Mooshak

| # | País | Equipo | 401 | 402 | 403 | 404 | 405 | 406 | 407 | Total Puntos | |
|---|---|--|-------|-------|-------|-----|-----|-----|-------|--------------|---|
| 1 |  | G3 URBELZ ALONSO CORTES, JORGE | 1 (1) | 1 (2) | 2 (4) | | | | 2 (1) | 4 | 6 |

| # | Tiempo de Concurso | País | Equipo | Problema | Lenguaje | Resultado | Estado |
|----------------------|--------------------|---|--------------------------------|----------|----------|----------------|--------|
| 2214 | 451:23:46 |  | G3 URBELZ ALONSO CORTES, JORGE | 407 | C++ | 2 Accepted | final |
| 830 | 255:40:33 |  | G3 URBELZ ALONSO CORTES, JORGE | 403 | C++ | 2 Accepted | final |
| 824 | 254:53:18 |  | G3 URBELZ ALONSO CORTES, JORGE | 403 | C++ | 1 Wrong Answer | final |
| 819 | 254:40:43 |  | G3 URBELZ ALONSO CORTES, JORGE | 403 | C++ | 1 Wrong Answer | final |
| 817 | 254:38:03 |  | G3 URBELZ ALONSO CORTES, JORGE | 403 | C++ | 1 Wrong Answer | final |
| 499 | 116:29:27 |  | G3 URBELZ ALONSO CORTES, JORGE | 402 | C++ | 1 Accepted | final |
| 490 | 113:28:02 |  | G3 URBELZ ALONSO CORTES, JORGE | 402 | C++ | 0 Wrong Answer | final |
| 52 | 20:17:17 |  | G3 URBELZ ALONSO CORTES, JORGE | 401 | C++ | 1 Accepted | final |

Envío ejercicios

Voy a dividir los ejercicios con respecto a las distintas secciones a entregar, al igual que se muestran en Mooshak. Primero mostraré los ejercicios verdes (401 y 402), luego los azules (403 y 407). Además, he creado la sección con ejercicios amarillos, pero como he usado un comodín esta se encuentra vacía.

Ejercicios verdes

Ejercicio 401

Envíos realizados: 1

Envío válido: 52

```
#include <stdlib.h> // Funcion exit
#include <string.h> // Funcion memset
#include <iostream> // Variables cin y cout
using namespace std;

#define MAX_NODOS 26

////////////////////////////////////
////////////////// VARIABLES GLOBALES //////////////////
////////////////////////////////////

int nnodos;           // Numero de nodos del grafo
int naristas;         // Numero de aristas del grafo
bool G[MAX_NODOS][MAX_NODOS]; // Matriz de adyacencia
bool visitado[MAX_NODOS]; // Marcas de nodos visitados

////////////////////////////////////
////////////////// FUNCIONES DEL PROGRAMA //////////////////
////////////////////////////////////

void leeGrafo (void){
// Procedimiento para leer un grafo de la entrada
cin >> nnodos >> naristas;
if (nnodos<0 || nnodos>MAX_NODOS) {
    cerr << "Numero de nodos (" << nnodos << ") no valido\n";
    exit(0);
}
memset(G, 0, sizeof(G));
char c1, c2;
for (int i= 0; i<naristas; i++) {
    cin >> c1 >> c2;
    G[c1-'A'][c2-'A']= true;
}
}

void bpp(int v){
// Procedimiento recursivo de la busqueda primero en profundidad
// v - primer nodo visitado en la bpp
visitado[v]= true;
cout << char(v+'A');
for (int w= 0; w<nnodos; w++)
    if (!visitado[w] && G[v][w])
        bpp(w);
}
```

```

void busquedaPP (void){
// Procedimiento principal de la busqueda en profundidad
memset(visitado, 0, sizeof(visitado));
for (int v= 0; v<nnodos; v++)
    if (!visitado[v])
        bpp(v);
cout << endl;
}

////////////////////////////////////
//////////////////// PROGRAMA PRINCIPAL ///////////////////
////////////////////////////////////

int main (void){
    int ncasos;
    cin >> ncasos;
    for (int i= 0; i<ncasos; i++) {
        leeGrafo();
        busquedaPP();
    }
}

```

Breve descripción

El ejercicio 401 ya se encontraba resuelto. Simplemente lo copié y lo subí a Mooshak. El código usa el algoritmo de primero en profundidad (BPP). Su eficiencia es del orden del número total de nodos que hay en el grafo $\rightarrow O(n^2)$, ya que recorre la matriz dos veces por nodo.

Ejercicio 402

Envíos realizados: 2

Envío válido: 499

```

#include <stdlib.h> // Funcion exit
#include <string.h> // Funcion memset
#include <iostream> // Variables cin y cout
#include <queue> // Manejo de colas
using namespace std;

#define MAX_NODOS 26

////////////////////////////////////
//////////////////// VARIABLES GLOBALES ///////////////////
////////////////////////////////////

int nnodos; // Numero de nodos del grafo
int naristas; // Numero de aristas del grafo
bool G[MAX_NODOS][MAX_NODOS]; // Matriz de adyacencia
bool visitado[MAX_NODOS]; // Marcas de nodos visitados

////////////////////////////////////
//////////////////// FUNCIONES DEL PROGRAMA ///////////////////
////////////////////////////////////

void leeGrafo (void){
// Procedimiento para leer un grafo de la entrada
cin >> nnodos >> naristas;
if (nnodos<0 || nnodos>MAX_NODOS) {
    cerr << "Numero de nodos (" << nnodos << ") no valido\n";
    exit(0);
}
memset(G, 0, sizeof(G));
char c1, c2;
for (int i= 0; i<naristas; i++) {
    cin >> c1 >> c2;
    G[c1-'A'][c2-'A']= true;
}
}

```

```

void bpa(int v){
// Procedimiento recursivo de la busqueda primero en profundidad
// v - primer nodo visitado en la bpp
    queue<int> Cola;

    cout << char(v+'A');

    visitado[v]= true;
    Cola.push(v);
    while(!Cola.empty()){
        int x=Cola.front();
        Cola.pop();
        for (int y= 0; y<nnodos; y++){
            if (!visitado[y] && G[x][y]){
                visitado[y]=true;
                Cola.push(y);

                cout << char(y+'A');
            }
        }
    }
}

void busquedaPA (void){
// Procedimiento principal de la busqueda en profundidad
memset(visitado, 0, sizeof(visitado));
for (int v= 0; v<nnodos; v++) if (!visitado[v]) bpa(v);
cout << endl;
}

////////////////////////////////////
//////////////////////// PROGRAMA PRINCIPAL //////////////////////////
////////////////////////////////////

int main (void){
    int ncasos;
    cin >> ncasos;
    for (int i= 0; i<ncasos; i++) {
        leeGrafo();
        busquedaPA();
    }
}

```

Breve descripción

Para la realización de este ejercicio, hemos usado el algoritmo de primero en anchura (BPA), tal como aparece en las transparencias del tema 4. Hay una gran similitud con el ejercicio anterior, ya que seguimos usando el array de nodos visitados, así como de la matriz de aristas. El resto del código es el mismo que en el 401.

Debido al uso de bucles infinitos, la eficiencia del ejercicio es del orden del número total de nodos que hay en el grafo $\rightarrow O(n^2)$.

Ejercicios azules

Ejercicio 403

Envíos realizados: 4

Envío válido: 830

```

#include <stdlib.h> // Funcion exit
#include <string.h> // Funcion memset
#include <iostream> // Variables cin y cout
#include <queue> // Manejo de colas
#include <sstream>
using namespace std;

#define MAX_NODOS 20000
#define MAX_ADYACENTES 10

////////////////////////////////////
////////////////// VARIABLES GLOBALES //////////////////
////////////////////////////////////

int N; // Numero de sitios del laberinto
int G[MAX_NODOS][MAX_ADYACENTES]; // Matriz de adyacencia
bool visitado[MAX_NODOS]; // Marcas de nodos visitados
queue<int> Cola;
bool k;

////////////////////////////////////
////////////////// FUNCIONES DEL PROGRAMA //////////////////
////////////////////////////////////

void leeGrafo (void){
// Procedimiento para leer un grafo de la entrada
cin >> N;
if (N<0 || N>MAX_NODOS) {
    cerr << "Numero de sitios (" << N << ") no valido\n";
    exit(0);
}
memset(G, 0, sizeof(G));

string linea;
int numadyacente;

for (int i= -1; i<N; i++) {
    getline(cin,linea);
    stringstream lineaAdyacente(linea);

    int j=0;
    while(lineaAdyacente >> numadyacente){
        G[i][j]= numadyacente;
        j++;
    }
}
}

```



```

void salto(int v){
    visitado[v-1]=true;
    Cola.push(v);
    int n;

    if (N!=v){
        for(int p=0;p<MAX_ADYACENTES;p++){
            n=G[v-1][p];
            if(!visitado[n-1] && n!=0){
                salto(n);
                if (!visitado[N-1]&&!k) {
                    k=true;
                    salto(v);
                }
            }else k=false;
            if (visitado[N-1]){
                break;
            }
        }
    }
}

bool nodosVisitados(){
    if (visitado[N-1]){
        return true;
    }else{
        bool nodovisitado =true;
        for(int r=0;r<N;r++) if (!visitado[r]) nodovisitado=false;
        return nodovisitado;
    }
}

void laberinto(){
    memset(visitado, 0, sizeof(visitado));

    salto(1);

    if(nodosVisitados()){
        cout << Cola.size() << endl;
        while (!Cola.empty()){
            cout << Cola.front() << endl;
            Cola.pop();
        }
        //if(Cola.empty()) cout<< "Cola vacia"<<endl;
    }else {
        cout << "INFINITO" << endl;
        while (!Cola.empty()){
            Cola.pop();
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//PROGRAMA PRINCIPAL////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main (void){
    int ncasos;
    cin >> ncasos;
    for (int i= 0; i<ncasos; i++) {
        leeGrafo();
        cout << "Caso " << i+1 << endl;
        laberinto();
    }
}

```

Breve descripción

En este caso no me he basado en ningún algoritmo específicamente. Sigo utilizando el array de nodos visitados, para encontrar algún caso donde se produzcan soluciones infinitas. Además, sigo usando una matriz de adyacencia, con los máximos marcados por el ejercicio. La idea del algoritmo es ir en orden de filas. Siempre empezamos en el nodo 1. De ahí saltamos a la primera fila e introducimos el primer nodo de la fila a la cola. Si ya estaba introducida, volvemos atrás y buscamos en el siguiente, así progresivamente. La eficiencia de este algoritmo es del orden $O(n * m)$, debido a que debemos recorrer la matriz entera hasta encontrar el nodo final ("N" marca el número de nodos, filas de la matriz y el nodo final).

Ejercicio 407

Envíos realizados: 1

Envío válido: 2214

```

#include <stdlib.h> // Funcion exit
#include <string.h> // Funcion memset
#include <iostream> // Variables cin y cout
#include <vector> // Para trabajar con vectores.
#include <unordered_map> // Para usar un mapa no ordenado que asocia nombres de ciudades con números de nodo.
#include <algorithm> // Para la función min

using namespace std;

#define MAX_NODOS 200 // Numero máximo de ciudades.

////////////////////////////////////
// VARIABLES GLOBALES
////////////////////////////////////

int nciudades; // Numero de ciudades
const int INF = 1e9; // Constante infinita

////////////////////////////////////
// FUNCIONES DEL PROGRAMA
////////////////////////////////////

void floyd(vector<vector<int>>& dist, int n){ // La funcion del algoritmo Floyd
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
            }
        }
    }
}

void leeGrafo (void){
    cin >> nciudades;
    cin.ignore();

    if (nciudades<0 || nciudades>MAX_NODOS) { //Comprobamos que el numero de ciudades es mayor que 0 y menor que 200
        cerr << "Numero de ciudades (" << nciudades << ") no valido\n";
        exit(0);
    }

    unordered_map<string, int> ciudadToNumero; // Mapa que asocia el nombre de una ciudad a un numero
    vector<string> numeroToCiudad; // Vector que asocia numero con nombre de ciudades
    string linea;
    for (int i = 0; i < nciudades; i++) {
        getline(cin >> ws, linea);
        ciudadToNumero[linea] = i;
        numeroToCiudad.push_back(linea);
    }

    vector<vector<int>> dist(nciudades, vector<int>(nciudades, INF)); // Matriz de distancias
    for (int i=0;i<nciudades;i++) dist[i][i]=0; // A 0 la diagonal de la matriz

    int A, B, L;
    while (true) {
        cin >> A >> B >> L;
        if (A == 0 && B == 0 && L == 0) break;
        dist[A][B] = dist[B][A] = L; // Introducimos las distancias en la matriz
    }

    floyd(dist, nciudades); // Llamamos a la función floyd

    int minExcentricidad = INF; // Excentricidad a infinito
    string centro; // Centro ciudad

    // Calcula la excentricidad de cada ciudad y encuentra el centro
    for (int i = 0; i < nciudades; i++) {
        int maxDist = 0;
        for (int j = 0; j < nciudades; j++) maxDist = max(maxDist, dist[i][j]);

        if (maxDist < minExcentricidad || (maxDist == minExcentricidad && numeroToCiudad[i] < centro)) {
            minExcentricidad = maxDist;
            centro = numeroToCiudad[i];
        }
    }
    cout << centro << endl << minExcentricidad << endl; // Se imprime el resultado
}

```

```

////////////////////////////////////
////////////////////      PROGRAMA PRINCIPAL      //////////////////////
////////////////////////////////////

int main (void){
    int ncasos;
    cin >> ncasos;
    for (int i= 0; i<ncasos; i++) {
        leeGrafo();
    }
}

```

Breve descripción

Para terminar, en el ejercicio 407 ha sido basado en el algoritmo de Floyd. El objetivo es encontrar la ciudad más céntrica de las dadas. Mediante Floyd, calculo la distancia entre cada ciudad. Para el algoritmo, he copiado tal cual el que aparece en las transparencias del tema 4. La función “leeGrafo” lo he basado en el que he utilizado en el resto de los ejercicios. La matriz de distancias utilizada es un vector. Mediante el uso de un mapa que asocia ciudades con números, he creado el último bucle del programa, donde va buscando la excentricidad por fila y asignando el nuevo centro. Como tiene que recorrer cada fila y columna de la matriz, su eficiencia es $O(n^2)$.

Ejercicios amarillos

Como ya he dejado claro, decidí hacer uso de uno de mis comodines, el cual equivale a la realización de un ejercicio amarillo. Debido a esto, esta sección se haya vacía y por tanto se llega a la conclusión de práctica.

Conclusiones y valoración del trabajo

Los grafos son una manera muy interesante de aprender nuevos algoritmos para programar. Dijkstra, Floyd, en general todos son usados como solución para problemas. Por ejemplo, en la asignatura Arquitectura de Redes he aprendido que se usan estos antiguos pero completos algoritmos a la hora de enrutar redes y completar las tablas de rutas buscando caminos más cortos o donde el peso sea menor. Pese a que las horas invertidas en esta práctica sean menores que otras, tengo que decir que el camino a seguir está muy bien marcado. Tiene los tintes perfectos para empezar a entender el funcionamiento de los grafos. Seguro que podré usar lo aprendido en el futuro.

Bibliografía

C++ Pilas (stacks) y colas (queues) | Aprende programación competitiva. (s. f.).

<https://aprende.olimpiada-informatica.org/node/373>

Hernandez, P. P. G. (s. f.). *Camino más corto para cada pareja: Floyd-Warshall.*

<https://pier.guillen.com.mx/algorithms/10-graficas/10.5-floyd.htm>