



Proyecto "EL VIAJERO"

El reto es conseguir un ciclo cuyo coste sea menor que 22000.

Alcances del proyecto

Pendiente

En proceso

Terminado

Diapositivas

Resultado 24k

Codificacion
de proyecto en
C++

Floyd

Prim

Informe del
proyecto

Antes de empezar

Para inicializar el proyecto debemos tener herramientas necesarias.

- Compilador `g++` (`G++` es un compilador de línea de órdenes que compila y enlaza programas en C++)
- Editor de código
 - VSCode
 - Vim

1. Compilar el código

```
g++ grafo.cpp
```

2. Ejecutar el programa

```
./a.out
```

3. Verificar si nuestro código es correcto con el archivo `comprobar.c` y generar el script `.out`

```
g++ comprobar.c
```

4. Finalmente, ejecutar el programa

```
./a.out
```

Resultado esperado:

```
Coste: 25186 Nodos: 1373 Nodo1: 712  
El ciclo encontrado es correcto.
```

Propuesta

La resolución se basa en el uso de algoritmos de Floyd-Warshall y Prim.

Implementacion

1. Creacion de la clase Grafo

```
● ● ●
class Grafo
{
    int **ruta; //para la matriz de rutas
    int **matriz; //para la matriz de distancias
    Lista<int> ruta2; //para guardar la ruta de los nodos
    int n_vertices; //numero de vertices
    int longitud; //coste del ciclo

    fstream output; //para el archivo de salida

public:
    Grafo(int);
    void add_arista(int, int, int);
    void floyd_marshall();
    int min_ciclo(int, int, bool);
    int get_ruta(int, int);
    void salida();
};
```

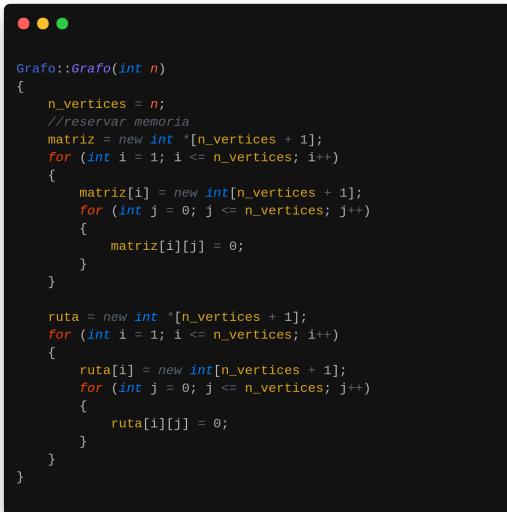
- `Grafo(int vertex)` → Constructor de clase que recibira como parametro el numero total de vertice, el cual creara las matrices de rutas y matriz.
- `void add_arista(int i, j, weight)` → Funcion vacia que creara las relaciones de nuestros grafo, una matriz de adyacencia.

- `void Grafo::floyd_marshall()` → Funcion vacia que hallara el grafo completo con todas las rutas.
- `int min_ciclo(int nodo_inicio, int nodo_siguiente, bool status)` → Funcion entera que se basa en el algoritmo de prim y recibira como
 - 1. Primer parametro nuestro nodo de inicio `712`
 - 2. Segundo parametro cualquier nodo de nuestros grafos de nodos
 - 3. Tercer parametro un booleano, el cual añadira a un lista enlazada donde se almacenara nuestro camino de nodos con la ruta mas optima.
- `int get_ruta(int a, int b)` → Funcion entera que lo que hace es buscar la ruta desde un nodo a otro por medio de sus antecesores, retorna el numero de nodos que visita.
- `void salida()` → Funcion vacia la cual almacenamos nuestros coste del ciclo, numero de vertices y

nuestro ciclo en un archivo de salida `output.txt`.

2. Constructor de clase → `Grafo(int vertex)`

Las variables miembro `matriz` y `ruta` se les reservara memoria dado que son matrices y se les declara como valores 0.

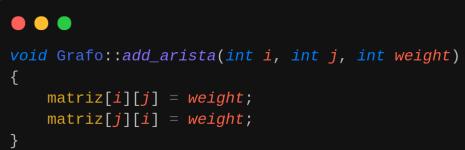


```
● ● ●
Grafo::Grafo(int n)
{
    n_vertices = n;
    //reservar memoria
    matriz = new int *[n_vertices + 1];
    for (int i = 1; i <= n_vertices; i++)
    {
        matriz[i] = new int[n_vertices + 1];
        for (int j = 0; j <= n_vertices; j++)
        {
            matriz[i][j] = 0;
        }
    }

    ruta = new int *[n_vertices + 1];
    for (int i = 1; i <= n_vertices; i++)
    {
        ruta[i] = new int[n_vertices + 1];
        for (int j = 0; j <= n_vertices; j++)
        {
            ruta[i][j] = 0;
        }
    }
}
```

3. Matriz de adyacencia → `void add_arista(int i, j ,weight)`

La variable miembro `matriz` una variable que es una matriz de adyacencia de dos dimensiones.



```
● ● ●
void Grafo::add_arista(int i, int j, int weight)
{
    matriz[i][j] = weight;
    matriz[j][i] = weight;
}
```

4. Algoritmo de wharsall → `void Grafo::floyd_marshall()`

Primera Parte

En los primeros dos `for` tenemos tres condicionales

1. Primer condicional → `if (i != j && matriz[i][j] == 0)`

Si en nuestra matriz de adyacencia esta en la diagonal principal o si el valor de la

```

● ● ●
void Grafo::floyd_marshall()
{
    for (int i = 1; i <= n_vertices; i++)
    {
        for (int j = 1; j <= n_vertices; j++)
        {
            if (i == j && matriz[i][j] == 0)
                matriz[i][j] = INF;

            if (matriz[i][j] == INF)
                ruta[i][j] = -1;
            else
                ruta[i][j] = j;
        }
    }

    for (int k = 1; k <= n_vertices; k++)
    {
        for (int i = 1; i <= n_vertices; i++)
        {
            for (int j = 1; j <= n_vertices; j++)
            {
                if (matriz[i][k] == INF || matriz[k][j] == INF)
                    continue;

                if (matriz[i][j] > matriz[i][k] + matriz[k][j])
                {
                    matriz[i][j] = matriz[i][k] + matriz[k][j];
                    ruta[i][j] = ruta[i][k];
                }
            }
        }
    }
}

```

Segunda Parte

Siguiendo con la explicacion tenemos dos condicionales.

1. Primer condicional →

```
matriz[i][k] == INF || matriz[k]
[j] == INF
```

Si en nuestra matriz de adyecencia `matriz[j][k]` o `matriz[k][j]` tiene un valor `INF`

Entonces, continue la ejecucion del codigo

2. Segundo condicional →

```
(matriz[i][j] > matriz[i][k] +
matriz[k][j])
```

Si en nuestra matriz de adyecencia el valor de la matriz `matriz[i][j]` es mayor entonces.

Entonces, el valor de la `ruta[i][j]` actualizara el valor con la suma de `ruta[i][k]` y `ruta[k][j]`. Por

posicion de nuestra matriz `matriz[i][j]` es `0`.

Entonces, el valor de la matriz `matriz[i][j]` toma un valor infinito `INF`.

2. Segundo condicional →

```
if (matriz[i][j] == INF)
```

Si en nuestra matriz de adyecencia el valor de la matriz `matriz[i][j]` es un valor infinito `INF`.

Entonces, el valor de la `ruta[i][j]` tomara un valor infinito `-1` que indicara que la ruta no existe para `ruta[i][j]`

3. Tercer condicional

Caso contrario al segundo caso `ruta[i][j]` tomara el valor de `j` "el nodo apuntado por `i`".

ultimo `ruta[i][j]` tomara el valor de `ruta[i][k]`

5. Algoritmo de Prim → `int min_ciclo(int nodo_inicio, int nodo_siguiente, bool status)`

Funcion entera que se basa en el algoritmo de prim

```

int Grafo::min_ciclo(int begin_node, int second_node, bool status)
{
    longitud = matriz[begin_node][second_node];
    if (status)
    {
        ruta2.insertar(begin_node);
        ruta2.insertar(second_node);
    }

    bool visited[n_vertices + 1];
    for (int i = 1; i <= n_vertices; i++)
    {
        visited[i] = false;
    }

    int q;
    q = second_node;
    visited[q] = true;

    int vis, min = INF;
    int cambio = 1;
    //Para encontrar un arbol de expansion minima
    while (q != 0 && cambio < n_vertices)
    {
        vis = q;
        q = 0;

        for (int i = 1; i <= n_vertices; i++)
        {
            if (matriz[vis][i] != 0 && (!visited[i]))
            {
                if (matriz[vis][i] < min)
                {
                    q = i;
                    min = matriz[vis][i];
                }
            }
        }
        if (status)
            ruta2.insertar(q);
        visited[q] = true;
        cambio++;
        longitud = longitud + matriz[vis][q];
        min = INF;
    }
    if (status)
        ruta2.insertar(begin_node);
    longitud = longitud + matriz[q][begin_node];
    return longitud;
}

```

1. El atributo de clase

`longitud` es lo que retornada en nuestra funcion

2. Si status es true nuestro agrera a los dos nodos al vector `ruta2` "vector de nuestro ciclo de menor coste"

3. Dentro de nuestro `while` si nuestro cambio es menor a `n_vertices` "total de vertices" y q sea diferente de 0.

1. Dentro de nuestro for comparamos

1. Primer condicional

→ `if (matriz[vis][i] != 0 && (!visited[i]))`

Lo cual indicaria que si ese nodo noha sido visitado y su valor es diferente de node pasa al siguiente condicional

- `if(matriz[vis][i] < min)`

Solo si `matriz[vis]`
[i] es menor
entonces se
actualizn los
valores con los
valores minimos.

2. Antes de salir del while
 1. Nuestro arreglo booleano `visited` se actualiza
 2. `cambio` incrementa
 3. longitud se actualiza con el valor de la matriz de adyacencia minima.
 4. Por ultimo min toma el valor de infinito para volver encontrar el menor nodo en nuestro for.
 4. Por ultimo, nuestra longitud tomara el anterior nodo con nuestro nodo de inicio 712 para volver al inicio de nuestro ciclo. Y asi obtenemos el coste de minimo de nuestro ciclo.

Observaciones

1. Como parte del proyecto se nos dio un archivo `entrada.txt` segun las indicaciones:

"A continuación vienen N líneas, una por cada nodo. Cada línea i-ésima contiene dos enteros separados por un espacio en blanco: Xi, Yi, que indican las coordenadas relativas de la ciudad i-ésima en el mapa. Esta información se ofrece simplemente a efectos de visualización del grafo."

Se trabaja con el archivo `input.txt` que es una replica de `entrada.txt` sin N lineas.

2. El costo computacional de la función miembro `void floyd_marshall` de la clase `Graph` es de n^5 , el cual para un entorno profesional no es recomendable.

Resultados

El costo que se obtuvo en este reto es de 25186.

Referencias

1. [Algoritmo de Prim](#)
2. [Algoritmo de Floyd-Warshall](#)