Universidad del Rosario

MACC Matemáticas Aplicadas y Ciencias de la Computación

Proyecto Algoritmos y Estructuras de Datos

Natalia Chacon - Juan David Martínez - Alejandra Pardo - Sergio Rubio

Mayo 2020

Repositorio

`https://github.com/JUANITOTELO/Proyecto_1_Algoritmos`

Resumen

Usualmente en Teoría de Grafos para solucionar el problema del camino mínimo o máximo de un punto (vértice) a otro se hace uso del algoritmo de Djkstra, el cual consiste en encontrar el camino más corto dado un vértice de origen a uno de los otros vértices de un grafo con pesos en cada arista. Para lograr la implementación de este algoritmo haciendo uso de estructuras de datos, fue necesario aprender conceptos dispuestos en la teoría de grafos, entender de manera apropiada el concepto de algoritmos y estructuras de datos y de esta manera poder identificar la posible solución del problema planteado.

Funcionalidad

Como primer paso para obtener la funcionalidad optima del algoritmo propuesto es la creación de clases; vértice, arista, y grafo, luego de esto se crean las estructuras a utilizar, una estructura que contiene un grafo, una estructura que posee una matriz de tipo entero, otra de tipo carácter y otra estructura que posee el vértice de procedencia y los valores de adyacencia. Posterior a la creación de clases y estructuras se desarrolla cada uno de los métodos para encontrar la solución al problema, estos métodos serán explicados a detalle a continuación.

Descripción

En esta sección se hará una clara descripción de los métodos correspondientes y el cómo se utilizaron y se relacionaron con la implementación del algoritmo de Djkstra. Para la elaboración del grafo se emplean diversidad de ellos:

• Init: es el constructor, el que inicializa el grafo.

• Vacío: determina si el grafo esta vacío o no.

• Size: este método retorna la cantidad de vértices que posee el grafo.

• InsertVertice: este método permite insertar un vértice nuevo y recibe el nombre que se le quiere poner.

• GetVertice: este método que recibe el nombre del vértice que queremos obtener y retorna el vértice.

• EliminarVertice: recibe el vértice a eliminar y lo elimina.

• InsertArista: con este método se puede insertar una arista nueva dando el vértice de origen, el vértice de destino y el peso se la arista.

• GetPeso: este método recibe el vértice de origen, el de destino y retorna el peso de la arista.

• EliminarArista: este método recibe el vértice de origen y el de destino para así eliminar la arista correspondiente.

• ListAdya: este método imprime la lista adyacente utilizando la clase vértice y la clase arista.

• RecordFMmenor: que recibe como parámetro el vértice que apunta al origen, retorna los valores menores y recorre las aristas con menor peso, en este método se hace uso de los "Mapas" estructura de datos que más adelante será explicada detalladamente, también se hace uso de un iterador y de apuntadores.

• RecordFP: este método sirve para recorrer toda la fila y retorna la totalidad de los valores, el método recibe como parámetro el vértice que apunta al origen y cada vértice y sus valores de adyacencia

• CrearMatrizYlistAdyAl: cumple la funcion de crear una matriz lista de adyacencia de forma aleatoria utilizando el método insertVertice, getVertice, insertArista y listAdya.

• CrearMatrizYlistAdyAlSimple: este método crea una matriz y lista de adyacencia aleatoria en la que solo apunta a un nodo y ese nodo no apunta a nada. Para que este método fuera totalmente optimo se hizo uso de los métodos: insertVertice, getVertice, insertArista y listAdya.

• CrearMLUnid: Con este método se crea un grafo unidireccional, usando insertVertice, getVertice, insertArista y listAdya.

• ReCorsFM: Este método recibe el nombre de la fila y la recorre guardando los valores minimos de sus adyacentes utilizando la función recordFMmenor y en ella getVertice para así retornar el mapa con los respectivos valores.

• ReCorsF: Recorre la fila y la guarda en una estructura que contiene el vértice de procedencia y los valores de adyacencia, esto lo hace invocando la funcion recordFP y getVertice.

• IAristas: este método es utilizado con el fin de imprimir la matriz como los caracteres.

• EAristas: este imprime los valores de las aristas en la matriz pero de forma .

• Anular: este metodo es el destructor, el que elimina el grafo.

• Djkstra: el metodo más importante de la implementación, en este se utilizan todos los métodos anteriormente explicados, obteniendo como resultado una lista con las llaves del camino más corto, este metodo recibe el vértice de origen y el vértice de destino al que se quiere llegar de la manera más corta y óptima.

Algoritmos y estructuras de datos

En esta sección se exponen los algoritmos y estructuras de datos que se han utilizado en la implementación. En el desarrollo del trabajo se ha utilizado el algoritmo de búsqueda para así encontrar los pesos minimos de las aristas y poder hallar el camino más corto de un vértice dado a otro. Como estructuras de datos, se manejaron las siguientes:

• Lista: es una secuencia de elementos almacenados en espacios de memoria no necesariamente continuos, en la implementación se utiliza para almacenar los vértices que ya han sido "visitados" y los que no, también se almacenan los valores adyacentes de cada vértice, además, para retornar la solución al problema (El camino más corto) también se emplea una lista como se muestra a continuación

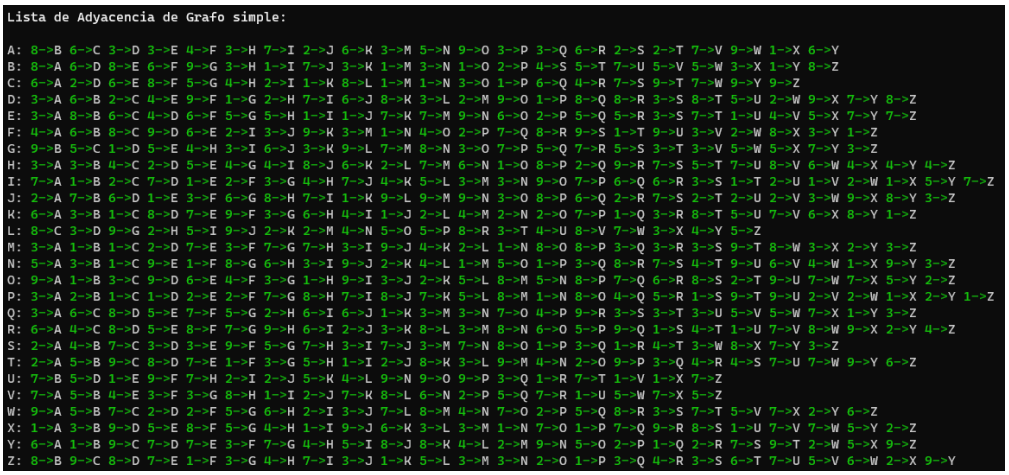

Figure 1: Ejemplo de ista de adyacencia de grafo simple

Figure 2: Ejemplo de solucion del problema

• Mapa: Es una secuencia contable compuesta por (llave, valor) en el dónde los valores de la llave son únicos. En la implementación del trabajo se utilizan para guardar el vértice y su valor.

• Clases: Son tipos definidos por el usuario que se utilizan para encapsular atributos y métodos. En la implementación se definieron las siguientes clases: *Arista*; se crean apuntadores para acceder a los vértices esta clase hereda los métodos de la clase $Grafo$ nombrados más adelante, en la clase $Vértice$ se crean apuntadores con el fin de acceder al vértice y al igual que la anterior clase nombrada hereda los métodos de la clase $Grafo$, y la clase más importante es la clase $Grafo$, con ella se crea el la matriz con los valores de las aristas del grafo y en la que se almacenan la variedad de métodos para obtener la solución al problema planteado.

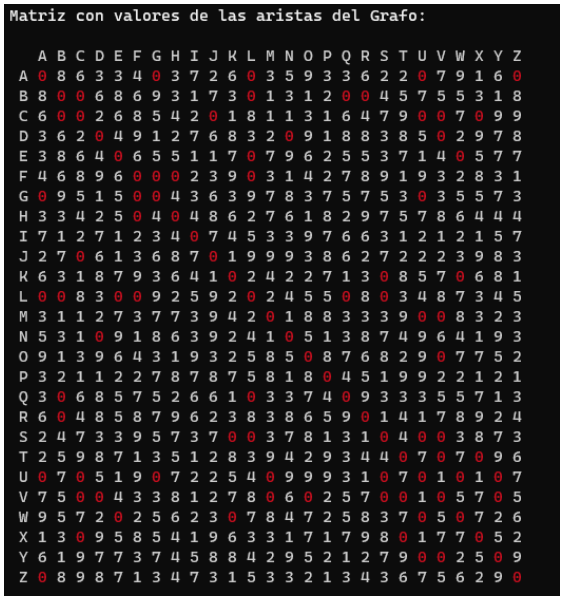

Figure 3: Ejemplo de matriz con el valor de las Aristas

• Estructuras: Estas son usadas para almacenar colecciones de atributos relacionados, en la implementación se desarrollaron las siguientes estructuras: $ArrM$ que contiene dos matrices una de tipo carácter y otra de tipo entero, la estructura $GrMat$ contiene un grafo y una estructura $ArrM$, en la MpV se almacena el vértice de precedencia y lo valores adyacentes, y en la última estructura implementada VmF se almacenan los valores menores en los que ha estado el vértice.