Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación IV

Tema: Algoritmos para la ruta más corta en un Grafo.

Objetivos Específicos

- Definir el concepto de camino o ruta más corta en un grafo.
- Calcular el camino mínimo desde un vértice al resto de los vértices.
- Determinar si existe un camino entre cualquier par de vértices de un grafo.
- Resolver problemas propuestos a través de la implementación de soluciones haciendo uso de grafos.
- Implementar el algoritmo Dijkstra utilizando Visual C#.NET.

Materiales y Equipo

- Guía Número 10.
- Computadora con programa Microsoft Visual C#.NET.

Introducción Teórica

Existen numerosos problemas que se pueden formular en términos de grafos. Ejemplos de ello son la planificación de las tareas que completan un proyecto, encontrar las rutas de menor longitud entre dos puntos geográficos, calcular el camino más rápido en un transporte, determinar el flujo máximo que puede llegar desde una fuente a, por ejemplo, una urbanización; entre otros.

La resolución de estos problemas requiere examinar todos los nodos o todas las aristas del grafo que representa al problema; sin embargo, existen ocasiones en que la estructura del problema es tal que sólo se necesitan visitar algunos de los nodos o bien algunas de las aristas. Los algoritmos imponen implícitamente un orden en estos recorridos: visitar el nodo más próximo o las aristas más cortas, y así sucesivamente; otros algoritmos no requieren ningún orden concreto en el recorrido.

Camino más Corto.

Cuando se trabaja con grafos dirigidos etiquetados o ponderados con factores de peso no negativos, es frecuente buscar el camino más corto entre dos vértices dados; es decir, el camino que nos permita llegar desde un vértice origen a un vértice destino recorriendo la menor distancia o con el menor costo.

Los algoritmos más usados para este fin son: Dijkstra, Floyd y Warshall.

Los tres algoritmos utilizan una matriz de adyacencia ponderada o etiquetada: que es la misma matriz de adyacencia utilizada para representar grafos, pero con la diferencia que en lugar de colocar un número "1" cuando dos vértices son adyacentes, se coloca el peso o ponderación asignado a la arista que los une.

Con frecuencia en la matriz etiquetada suele utilizarse la siguiente notación:

Suponiendo que M [i, j] representa la matriz de adyacencia, tenemos:

M [i, j] = 0, si i = j

M [i, j] = 1000 ó ∞, si no existe un camino de i a j, donde $i \neq j$

M [i, j] = costo de ir del vértice i al vértice j

Otro nombre con el cual suele llamarse a una matriz de adyacencia etiquetada es ***matriz de distancias o matriz de costos.***

El problema de buscar un camino más corto entre dos nodos dados se puede resolver mediante un algoritmo voraz conocido como ***Algoritmo de Dijkstra***.

Algoritmo del camino más corto o ruta más corta (Algoritmo de Dijkstra).

El algoritmo de Dijkstra es un algoritmo eficiente (de complejidad O (n^2), donde "n" es el número de vértices) que sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo. Fue diseñado por el holandés Edsger Wybe Dijkstra en 1959.

Este algoritmo es un típico ejemplo de algoritmo ávido, que resuelve los problemas en sucesivos pasos, seleccionando en cada paso la solución más óptima con el objeto de resolver el problema.

El fundamento sobre el que se basa este algoritmo es el *principio de optimizar*: si el camino más corto entre los vértices "u" y "v" pasa por el vértice "w", entonces la parte del camino que va de "w" a "v" debe ser el camino más corto entre todos los caminos que van de "w" a "v". De esta manera, se van construyendo sucesivamente los caminos de coste mínimo desde un vértice

inicial hasta cada uno de los vértices del grafo, y se utilizan los caminos conseguidos como parte de los nuevos caminos.
Dicho en otras palabras:
“Dado un grafo a cuyos arcos se han asociado una serie de pesos, se define el camino de coste mínimo de un vértice “u” a otro “v”, como el camino donde la suma de los pesos de los arcos que lo forman es la más baja entre las de todos los caminos posibles de “u” a “v”.”

El algoritmo de Dijkstra en cada paso selecciona un vértice “v” cuya distancia es desconocida, entre los que tiene la distancia más corta al vértice origen “s”, entonces el camino más corto de “s” a “v” ya es conocido y marca el vértice “v” como ya conocido. Así, sucesivamente se van marcando nuevos vértices hasta que estén todos marcados; en ese momento es conocida la distancia mínima del origen “s” al resto de los vértices.
Entre las condiciones más importantes que deben considerarse para aplicar el algoritmo están:

- Las aristas deben tener un peso no negativo.
- El grafo debe ser dirigido y por supuesto ponderado.

Una posible aplicación de este algoritmo se presenta cuando se desea encontrar la ruta más corta entre dos ciudades; cada vértice representa una ciudad y las aristas representan la duración de los vuelos.

A continuación se presenta el algoritmo.
Para la explicación general se tomará como referencia los siguientes pasos:

1. Seleccionar vértice de partida, es decir un origen.
2. Marcar el punto de partida como el punto de inicio.
3. Determinar los caminos especiales desde el nodo de partida, es decir, el de inicio.
4. **Camino especial** es aquel que solo puede trazarse a través de los nodos o vértices ya marcados.
5. Para cada nodo no marcado, se debe determinar si es mejor usar el camino especial antes calculado o si es mejor usar el nuevo camino especial que resulte al marcar este nuevo nodo.
6. Para seleccionar un nuevo nodo no marcado como referencia, deberá tomarse aquel cuyo camino especial para llegar a él es el mínimo, por ejemplo si anteriormente marqué el nodo o vértice 2, el cual tiene dos nodos adyacentes 3 y 4 cuyo peso en la arista

corresponde a 10 y 5 respectivamente, se tomará como nuevo nodo de partida el 4, ya que el peso de la arista o camino es menor.

7. Cada camino mínimo corresponde a la suma de los pesos de las aristas que forman el camino para ir del nodo principal al resto de nodos, pasando únicamente por caminos especiales, es decir nodos marcados.

Procedimiento

Ejemplo 1. Implementaremos el algoritmo de Dijkstra en C#.

Le aplicaremos el algoritmo de Dijkstra al siguiente dígrafo:

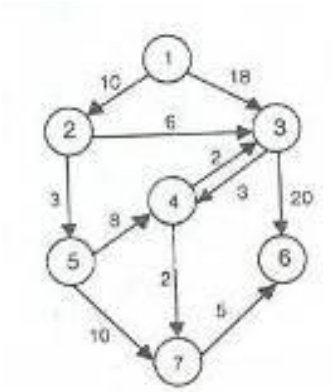

Figura 1.

El primer paso a realizar es construir la matriz de adyacencia, por conveniencia para las posiciones en la matriz en las cuales no existe una arista entre dos vértices pondremos un valor negativo “-1” y cuando exista una arista se colocará el peso de la misma.

La matriz de adyacencia asociada al dígrafo mostrado en la figura es:

		0 **1**	1 **2**	2 **3**	3 **4**	4 **5**	5 **6**	6 **7**
0	**1**	-1	10	18	-1	-1	-1	-1
1	**2**	-1	-1	6	-1	3	-1	-1
2	**3**	-1	-1	-1	3	-1	20	-1
3	**4**	-1	-1	2	-1	-1	-1	2
4	**5**	-1	-1	-1	8	-1	-1	10
5	**6**	-1	-1	-1	-1	-1	-1	-1
6	**7**	-1	-1	-1-	-1	-1	5	-1

A continuación, implementaremos el algoritmo Dijkstra en un proyecto de Visual C#:

1. Crear un proyecto de consola, se sugiere nombrarlo “Algoritmo Dijkstra”.
2. Cambiamos el nombre de “Program.cs” a “Dijkstra.cs”.

3. Agregar el siguiente código a la clase “Dijkstra”:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Algoritmo_Dijkstra
{
   class Dijkstra
   {              // Declaración de variables a utilizar
      private int rango = 0;
      private int [ , ] L;          // matriz de adyacencia
      private int [ ] C;            // arreglo de nodos
      public int [ ] D;             // arreglo de distancias
      private int trango = 0;

         // Algoritmo Dijkstra
      public Dijkstra (int paramRango, int [ , ] paramArreglo)
      {
         L = new int [paramRango, paramRango];
         C = new int [paramRango];
         D = new int [paramRango];
         rango = paramRango;

         for (int i = 0; i < rango; i++)
         {
            for (int j = 0; j < rango; j++)
            {
               L [i, j] = paramArreglo [i, j];
            }
         }

         for (int i = 0; i < rango; i++)
         {
            C [i] = i;
         }

         C [0] = -1;

         for (int i = 1; i < rango; i++)
            D [i] = L [0, i];
      }

          // Rutina de solución Dijkstra
      public void SolDijkstra ( )
      {
         int minValor = Int32.MaxValue;
         int minNodo = 0;
```

```
        for (int i = 0; i < rango; i++)
        {
            if (C [i] == -1)
                continue;

            if (D [i] > 0 && D [i] < minValor)
            {
                minValor = D [i];
                minNodo = i;
            }
        }

        C[minNodo] = -1;

        for (int i = 0; i < rango; i++)
        {
            if (L [minNodo, i] < 0)              // si no existe arco
                continue;

            if (D [i] < 0)                       // si no hay un peso asignado
            {
                D [i] = minValor + L [minNodo, i];
                continue;
            }

            if ((D [minNodo] + L [minNodo, i]) < D [i])
                D [i] = minValor+ L [minNodo, i];
        }
    }

        // Función de implementación del algoritmo
    public void CorrerDijkstra ( )
    {
        for (trango = 1; trango < rango; trango++)
        {
            SolDijkstra ( );
            Console.WriteLine ("Iteracion No." + trango);
            Console.WriteLine ("Matriz de distancias: ");

            for (int i = 0; i < rango; i++)
                Console.Write (i + "  ");

            Console.WriteLine (" ");

            for (int i = 0; i < rango; i++)
                Console.Write (D [i] + " ");

            Console.WriteLine (" ");
            Console.WriteLine (" ");
        }
    }
```

```
        static void Main (string [ ] args)
        {
            // Definición de la matriz de adyacencia del digrafo mostrado en la figura 1
            int [ , ] L = {
                {-1, 10, 18, -1, -1, -1, -1},
                {-1, -1,  6, -1,  3, -1, -1},
                {-1, -1, -1,  3, -1, 20, -1},
                {-1, -1,  2, -1, -1, -1,  2},
                {-1, -1, -1,  6, -1, -1, 10},
                {-1, -1, -1, -1, -1, -1, -1},
                {-1, -1, 10, -1, -1,  5, -1}
            };

            Dijkstra prueba = new Dijkstra ((int) Math.Sqrt (L.Length), L);
            prueba.CorrerDijkstra ( );

            Console.WriteLine
                ("La solucion de la ruta mas corta tomando como nodo inicial el NODO 1 es: ");

            int nodo = 1;
            foreach (int i in prueba.D)
            {
                Console.Write ("Distancia minima a nodo " + nodo + " es ");
                Console.WriteLine ( i );
                nodo++;
            }

            Console.WriteLine ( );
            Console.WriteLine ("Presione la tecla Enter para salir.");
            Console.Read ( );
        }
    }
}
```

Análisis de resultados

Ejercicio 1.

Tomando como referencia el código de ejemplo proporcionado, desarrollar la implementación del algoritmo Dijkstra como un Simulador, en una interfaz gráfica de formulario (Windows Forms), de tal manera que el usuario pueda agregar (dibujar) los nodos y arcos que forman el dígrafo al cual se le aplicará el algoritmo.

Se debe preguntar al usuario el nodo inicial para determinar la ruta más corta hacia el resto de los nodos del dígrafo.

Investigación Complementaria

Para la siguiente semana:

Continuar con la implementación de algoritmos de la ruta más corta en el Simulador de grafos en una interfaz gráfica de formulario (Windows Forms).

Al proyecto elaborado durante el desarrollo de esta guía, agregar la opción de poder encontrar los caminos más cortos a partir de un nodo inicial hacia el resto de los nodos de un grafo, utilizando el algoritmo de Floyd-Warshall.

Se sugiere tener un menú donde el usuario seleccione con cual método desea encontrar la ruta más corta:

a. Algoritmo Dijkstra.
b. Algoritmo Floyd-Warshall.

Guía 10: Algoritmos para la ruta más corta en un Grafo.

Hoja de cotejo: 10

Alumno:

Máquina No:

Docente:

GL:

Fecha:

EVALUACIÓN					
	%	1-4	5-7	8-10	Nota
CONOCIMIENTO	Del 20 al 30%	Conocimiento deficiente de los fundamentos teóricos	Conocimiento y explicación incompleta de los fundamentos teóricos	Conocimiento completo y explicación clara de los fundamentos teóricos	
APLICACIÓN DEL CONOCIMIENTO	Del 40% al 60%				
ACTITUD	Del 15% al 30%	No tiene actitud proactiva.	Actitud propositiva y con propuestas no aplicables al contenido de la guía.	Tiene actitud proactiva y sus propuestas son concretas.	
TOTAL	100%				