Algoritmos y Estructuras de Datos

Cursada 2014

Grafos – Aplicación Algoritmo de Floyd

Agregue a la clase **Floyd** implementada en el TP 10, el siguiente método:

floydCiclosMinimos (Grafo<T> grafo) : int

El cuál utiliza la información obtenida por el algoritmo de **floyd** para identificar el costo del ciclo con mínimo costo del grafo dirigido ponderado recibido como parámetro.

Grafos – Aplicación Algoritmo de Floyd

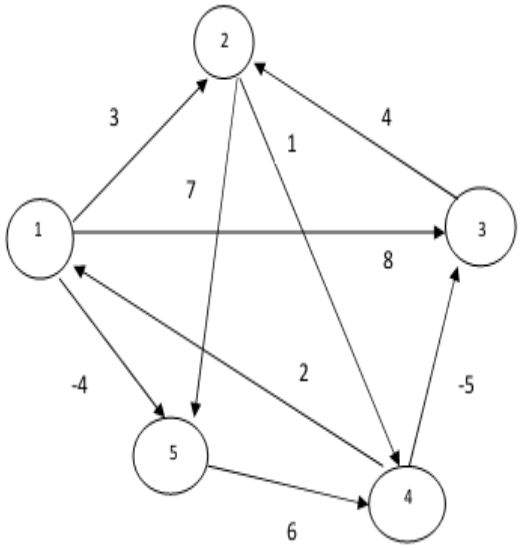
¿Qué estrategia aplicar para calcular el ciclo de mínimo costo?

- Aplicamos el Algoritmo de Floyd: así obtendremos la matriz con todos los costos mínimos entre cualquier par de vértices del grafo
- Luego si recorremos esa matriz de costos sumando los costos de los caminos mínimos para ir de *u* a *v* y de *v* a *u* obtenemos el **costo mínimo del ciclo con origen *u* pasando por el vértice intermedio *v*.** Esto es:

  (u, v) + (v, u) = costo mínimo del ciclo con origen *u*

- Aplicamos el paso anterior a cada uno de los vértices del grafo y nos quedamos con el mínimo!!

Grafos – Aplicación Algoritmo de Floyd



Ejemplo:

En el siguiente dígrafo el costo mínimo del ciclo es 0.
Formado por los vértices 2, 4 y 3

M_0 = Matriz Inicial

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	∞	-5	0	∞
∞	∞	∞	6	0

M_0 = (Pasando por 1)

0	3	8	∞	-4
∞	0	∞	1	7
∞	4	0	∞	∞
2	5	-5	0	-2
∞	∞	∞	6	0

M_2 = (Pasando por 3)

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	-1	-5	0	-2
∞	∞	∞	6	0

M_1 = (Pasando por 2)

0	3	8	4	-4
∞	0	∞	1	7
∞	4	0	5	11
2	5	-5	0	-2
∞	∞	∞	6	0

M_3 = (Pasando por 4)

0	3	-1	4	-4
3	0	-4	1	7
7	4	0	5	11
2	-1	-5	0	-2
8	5	1	6	0

M_4 = (Pasando por 5)

0	1	-1	2	-4
3	0	-4	1	7
7	4	0	5	11
2	-1	-5	0	-2
8	5	1	6	0

Grafos – Aplicación Algoritmo de Floyd


```
package practica10;
//imports
public class Algoritmos<T> {

  private int[][] inicializarTabla(Grafo<T> g) {
    ListaGenerica<Vertice<T>> vertices = g.listaDeVertices();
    int cant_vertices = vertices.tamanio();
    int tabla[][] = new int[cant_vertices][cant_vertices];

        for (int i = 0; i < cant_vertices; i++)
          for (int j = 0; j < cant_vertices; j++)
              if (i == j) tabla[i][j] = 0;
              else tabla[i][j] = Integer.MAX_VALUE;

        for(vertices.comenzar(); !vertices.fin(); vertices.proximo()) {
          Vertice<T> v = vertices.elemento();
          ListaGenerica<Arista<T>> adyacentes = g.listaDeAdyacentes(v);
          for(adyacentes.comenzar(); !adyacentes.fin(); adyacentes.proximo()) {
            Arista<T> ar = adyacentes.elemento();
            tabla[v.getPosicion()][ar.getDestino().getPosicion()] = ar.getPeso();
          }
        }
        return tabla;
  }
```

Grafos – Aplicación Algoritmo de Floyd


```
public int[][] floyd(Grafo<T> g) {
    int tabla[][] = inicializarTabla(g);
    int cant_vertices = g.listaDeVertices().tamanio();
    ListaGenerica<Vertice<T>> vertices = g.listaDeVertices();
    for(int k = 0; k < cant_vertices; k++) {
      for(int i = 0; i < cant_vertices ; i++) {
        for(int j = 0; j < cant_vertices;  j++) {
            if (tabla[i][k] != Integer.MAX_VALUE && tabla[k][j] != Integer.MAX_VALUE &&
                  tabla[i][j] > tabla[i][k] + tabla[k][j])

                tabla[i][j] = tabla[i][k] + tabla[k][j];
        }
      }
    }
    return tabla;

}
```

Grafos – Aplicación Algoritmo de Floyd


```
public int floydCiclosMinimos (Grafo<T> grafo) {

   int costoMinCiclo = Integer.MAX_VALUE;
    int cant_vertices = grafo.listaDeVertices().tamanio();
    int tabla [] []  = floyd (grafo);

    for(int i = 0; i < cant_vertices; i++) {
      for(int j = i+1; j < cant_vertices; j++) {

            int costoAux = tabla[i][j] + tabla[j][i];

            costoMinCiclo = Math.min(costoAux, costoMinCiclo);              }
      }
    }
    return costoMinCiclo ;
}
}
```