Soluciones - práctico 8

Algoritmos voraces (Greedy)

Ejercicio 2

a) Los caminos más cortos entre los vértices 5 y 1 son:

i. 5, 6, 1
ii. 5, 4, 6, 1
iii. 5, 4, 3, 2, 1

b) En este caso la aplicación de Dijkstra retorna el camino iii. , ya que el algortimo, selecciona en cada paso la arista adyacente de menor costo. Por lo tanto estando en el nodo 4, la selección es la arista que va al nodo 3 (costo 1) en vez que la que va a 6 (costo 4). Notese que la solución de Dijkstra puede variar según la implementación dependiendo de como se seleccionen aristas de igual (y menor) costo.

c) La solución se basa en la modificación de la tabla de Dijkstra, cada vez que se encuentra una solución de camino alternativo más corto se incorpora a la tabla agregando el predecesor en cada caso.

Para esta implementación, se suponen definiciones de los TADs Lista y Grafo con las operaciones usuales (Grafo con aristas con costo).

Grafo de enteros con costo en las aristas:

```
typedef struct CNode * Grafo;

Grafo construirGrafo (int cantNodos);

void agregarAristaGrafo (Grafo & grafo, int nodoOrigen, int nodoDestino, int costo);

bool existeAristaGrafo (Grafo &grafo, int nodoOrigen, int nodoDestino);

int obtenerCostoArista(Grafo &grafo, int nodoOrigen, int nodoDestino);

int cantNodosGrafo(Grafo &grafo);

Lista adyacentesNodoGrafo (Grafo &grafo, int nodo);
```

Tabla auxiliar:

```
typedef struct TDijkstra * TablaDijkstra;

int recuperarMinimoNoMarcado(TablaDijkstra &tabla);

void marcarNodo(TablaDijkstra &tabla, int nodo);

void agregarPredecesor(TablaDijkstra &tabla, int actual, int anterior);

void actualizarCostoNodo (TablaDijkstra &tabla, int actual, int costo);

void borrarPredecesores(TablaDijkstra &tabla, int actual);

int obtenerCostoNodo(TablaDijkstra &tabla, int actual);

Lista obtenerListaNodos(TablaDijkstra tabla, int actual);

void inicializarTabla(TablaDijkstra &tabla, Grafo g, int comienzo);

void imprimirTabla(TablaDijkstra &tabla);
```

Algoritmo:

```
   void Dijkstra(Grafo g, TablaDijkstra &tabla)
   {
       int cantNodos = cantNodosGrafo(g);
       for (int i=0; i < cantNodos; i++)
       {
              int actual = recuperarMinimoNoMarcado(tabla);

              marcarNodo(tabla, actual);
              Lista listaAdy = adyacentesNodoGrafo(g, actual);
              while (listaAdy != NULL)
              {
                     int adyacente = obtenerPrimeraLista(listaAdy);
                     int costo = obtenerCostoArista(g, actual, adyacente);

                     if (costo + obtenerCostoNodo(tabla, actual) <
                         obtenerCostoNodo(tabla, adyacente))
                     {
                            /* Notar que al encontrar un mejor camino, se deben eliminar
                                   todos los posibles caminos anteriores */
                            borrarPredecesores(tabla, adyacente); /*1*/
                            agregarPredecesor(tabla, adyacente, actual);
                            actualizarCostoNodo(tabla, adyacente, costo +
                                    obtenerCostoNodo (tabla, actual));
                     }
                     else if (costo + obtenerCostoNodo(tabla, actual) ==
                              obtenerCostoNodo(tabla, adyacente))
                     {
                            agregarPredecesor(tabla, adyacente, actual); /*2*/
                     }
                     listaAdy = obtenerRestoLista(listaAdy);
              }
       }
   }

   int main(int argc, char *argv[]){

       int cantNodos, vertice_inicio;
       Grafo g = construirGrafo(cantNodos);
   //  agregarAristaGrafo
       TablaDijkstra tabla;
       inicializarTabla(tabla, g, vertice_inicio);
```

```
        Dijkstra(g, tabla);
        imprimirTabla(tabla);
    }
```

Las diferencias que se pueden ver en esta solución con respecto a las clásicas implementaciones del algoritmo de Dijkstra están dadas por:

- /*1*/ borrar todos los predecesores al nodo **actual** cuando se encuentra un camino de costo menor.
- /*2*/ agregar una solución alternativa cuando se encuentran dos caminos de igual costo.