Algoritmos 2023-24

Grado en Ingeniería Informática

Práctica 4

Grafos: problema del camino mínimo

Fecha límite de entrega: sábado, 25 de noviembre

A continuación se presenta un pseudocódigo para calcular el camino mínimo de cada vértice a los restantes en grafos ponderados siguiendo el algoritmo de *Dijkstra*. El argumento es la matriz de adyacencia del grafo y el resultado es una tabla con las distancias mínimas desde cada vértice a los restantes.

```
procedimiento dijkstra( M[1..n,1..n], Distancias[1..n,1..n] );
  para m := 1 hasta n hacer
    noVisitados := { 1, 2, ..., m-1, m+1, ..., n };
    para i := 1 hasta n hacer
      Distancias[m, i] := M[m, i]
    fin para
    repetir n-2 veces:
      v := nodo de noVisitados que minimiza Distancias[m, v];
      noVisitados := noVisitados - { v };
      para cada w en noVisitados hacer
        si Distancias[m, w] > Distancias[m, v] + M[v, w]
        entonces Distancias[m, w] := Distancias[m, v] + M[v, w]
        fin si
      fin para
    fin repetir
  fin para
fin procedimiento
```

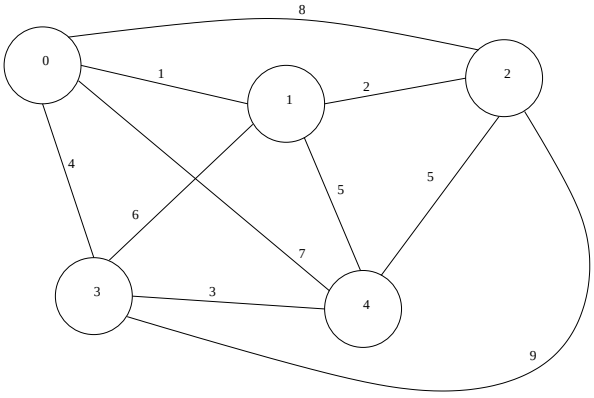
Se pide:

1. Implemente en C el algoritmo presentado (figura 1).
2. Valide el correcto funcionamiento de la implementación. En las figuras 2 y 3 se proponen dos casos de prueba.
3. Usando las funciones de la figura 4 para generar aleatoriamente grafos completos no dirigidos, calcule empíricamente la complejidad computacional del algoritmo para el cálculo de las distancias mínimas.
4. Entregue los ficheros con el código C y el fichero `.txt` con el informe por medio de la tarea *Entrega Práctica 4* en la página de Algoritmos en `https://campusvirtual.udc.gal`. Se recuerda que el límite para completar la tarea es el sábado 25 de noviembre a las 23:59, y una vez subidos los archivos no se podrán cambiar. **Todos los compañeros que forman un equipo tienen que entregar el trabajo**.

```
typedef int ** matriz;

void dijkstra(matriz grafo, matriz distancias, int tam) {
  int n, i, j, min, v=0;
  int *noVisitados = malloc(tam*sizeof(int));
  for (n=0; n<tam; n++) {
    for (i=0; i<tam; i++) {
      noVisitados[i] = 1;
      distancias[n][i] = grafo[n][i];
    }
    noVisitados[n] = 0;
   /*
    ...
    */
  }
  free(noVisitados);
}
```

Figura 1: Parte del procedimiento `dijkstra`



(a) Grafo

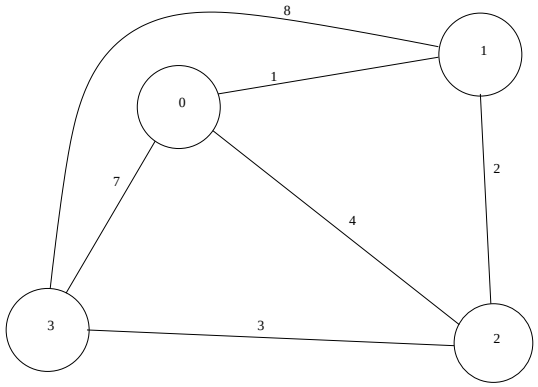
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

	0	1	2	3	4
0	0	1	3	4	6
1	1	0	2	5	5
2	3	2	0	7	5
3	4	5	7	0	3
4	6	5	5	3	0

(c) Distancias mínimas

Figura 2: Primer ejemplo



(a) Grafo

$$\begin{pmatrix} 0 & 1 & 4 & 7 \\ 1 & 0 & 2 & 8 \\ 4 & 2 & 0 & 3 \\ 7 & 8 & 3 & 0 \end{pmatrix}$$

(b) Matriz de adyacencia

	0	1	2	3
0	0	1	3	6
1	1	0	2	5
2	3	2	0	3
3	6	5	3	0

(c) Distancias mínimas

Figura 3: Segundo ejemplo

```
#define TAM_MAX 1000
matriz crearMatriz(int n) {
  int i;
  matriz aux;
  if ((aux = malloc(n*sizeof(int *))) == NULL)
    return NULL;
  for (i=0; i<n; i++)
    if ((aux[i] = malloc(n*sizeof(int))) == NULL)
      return NULL;
  return aux;
}
/* Inicializacion pseudoaleatoria [1..TAM_MAX] de un grafo completo
   no dirigido con n nodos, representado por su matriz de adayencia  */
void iniMatriz(matriz m, int n) {
  int i, j;
  for (i=0; i<n; i++)
    for (j=i+1; j<n; j++)
      m[i][j] = rand() % TAM_MAX + 1;
  for (i=0; i<n; i++)
    for (j=0; j<=i; j++)
      if (i==j)
        m[i][j] = 0;
      else
        m[i][j] = m[j][i];
}
void liberarMatriz(matriz m, int n) {
  int i;
  for (i=0; i<n; i++)
    free(m[i]);
  free(m);
}
```

Figura 4: Las funciones `CrearMatriz`, `iniMatriz`, y `liberarMatriz`