Algoritmos 2023/24

Grado en Inteligencia Artificial

Práctica 4

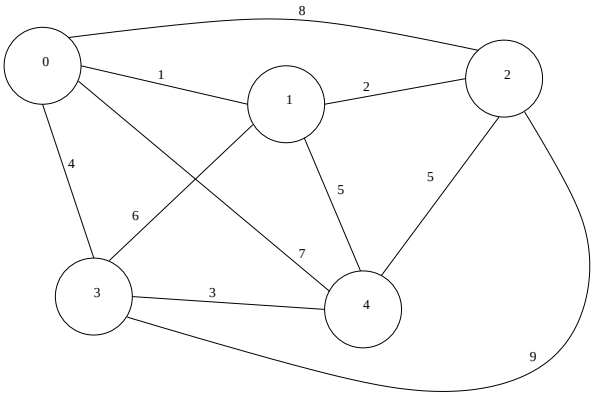
Fecha límite de entrega: viernes, 24 de noviembre, a las 23:59

A continuación se presenta un pseudocódigo para calcular el camino mínimo de cada vértice a los restantes en grafos no dirigidos y ponderados con pesos positivos siguiendo el algoritmo de *Dijkstra*. El argumento de entrada es la matriz de adyacencia del grafo, y la salida es una tabla con las distancias mínimas desde cada vértice a los restantes.

```
función 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 función
```

Se pide:

1. Implemente en PYTHON el algoritmo de Dijkstra utilizando como base el pseudocódigo arriba indicado.
2. Valide que la implementación funcione correctamente. En las figuras 1 y 2 se proponen dos casos de prueba.
3. Calcule empíricamente la complejidad computacional del algoritmo para el cálculo de las distancias mínimas. Genere los grafos completos no dirigidos con pesos aleatorios entre 1 y 1000; puede usar **uno de los dos** códigos propuestos en la figura 3.
4. Entregue los ficheros con el código PYTHON 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 viernes 24 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**.



(a) Grafo

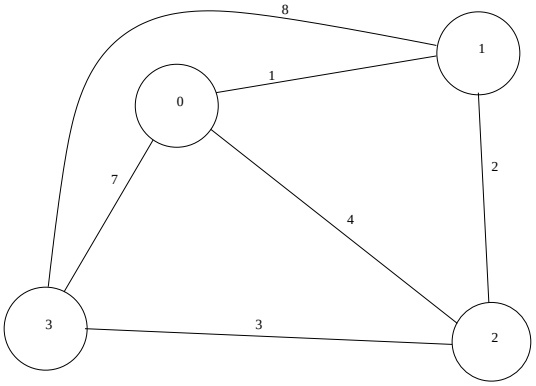
$$\begin{pmatrix} 0 & 1 & 8 & 4 & 7 \\ 1 & 0 & 2 & 6 & 5 \\ 8 & 2 & 0 & 9 & 5 \\ 4 & 6 & 9 & 0 & 3 \\ 7 & 5 & 5 & 3 & 0 \end{pmatrix}$$

(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 1: 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 2: Segundo ejemplo

```
import random
def aleatorio(n):
    l=list(range(n))
    for i in l:
        l[i] = random.randint(1, 1000)
    return l
def matrizAleatoria(n):
    m = []
    for i in range(n):
        m.append(aleatorio(n))
    for i in range(n):
        for j in range(i+1):
            if (i==j):
                m[i][j]=0
            else:
                m[i][j]=m[j][i]
    return m
```

```
import numpy as np
from numpy import random
def matrizAleatoria(n):
    m = random.randint(low=1, high=1000, size=(n,n))
    return(np.tril(m, -1) + np.tril(m, -1).T)
```

Figura 3: Inicializacion aleatoria [1..1000] de una grafo completo no dirigido, representado por una matriz de adyacencia. La segunda versión, a la derecha, usa la biblioteca `NumPy` que da soporte para matrices grandes multidimensionales