Dmytro Vernyuk

Practica 5: Kosajaru

Para ejecutar las pruebas, el código dispone de un generador aleatorio de grafos dirigidos. Cada vértice tendrá una arista como mínimo, saliente, entrante o recursiva, y una arista como máximo con el mismo origen y destino. El programa también dispone de un creador de grafos personalizados.

La generación de grafos aleatorios depende de la variable global N_VERTICES (por defecto = 5) que se puede cambiar desde la terminal. Esta indica cuantos números de vértices tendrán los grafos generados.

Para cada grafo generado se nos muestra automáticamente el resultado por pantalla con todos los vértices, el número de aristas y las aristas en sí (si N_VERTICES <= 10) y si queremos imprimirlo manualmente utilizamos la opción 4 (para cualquier caso). La nomenclatura de cada arista es de: “arista x -> y”, donde x es el origen e y es el destino.

Para la resolución propia del problema utilizamos la opción 1. Todas las resoluciones parten siempre del vértice 0 y puede darse el caso que en algunos grafos sea imposible llegar a uno o varios vértices, los cuales el algoritmo recorrerá posteriormente después del primer DFS.

Cuando terminemos de resolver el grafo, se nos imprimirá por pantalla el tiempo utilizado por el algoritmo y el propio resultado con todas las distancias. El resultado nos muestra el grafo acíclico resultante de aplicar el algoritmo. La representación consiste en los conjuntos de vértices SCC (Strongly connected components) y las aristas de cada conjunto. Todas las aristas son filtradas para quedarnos con las adecuadas y, aunque utilizan la notación sin SCC del grafo, se ve a simple vista a que conjunto destino pertenecen.

Algunos ejemplos sencillos de 5 vértices:

```
*Grafo aleatorio resultante:
Vertice 0:
        arista 0 -> 1

Vertice 1:
        arista 1 -> 1
        arista 1 -> 0
        arista 1 -> 4

Vertice 2:
        arista 2 -> 3
        arista 2 -> 1
        arista 2 -> 2

Vertice 3:
        arista 3 -> 2

Vertice 4:
        arista 4 -> 4
```

```
*Grafo aciclico resultante:
Vertice SCC  2 3:
        arista 2 -> 1

Vertice SCC  0 1:
        arista 1 -> 4

Vertice SCC  4:

#Tiempo total 0.0519 ms
```

```
*Grafo aleatorio resultante:
Vertice 0:
        arista 0 -> 3
        arista 0 -> 2

Vertice 1:
        arista 1 -> 4
        arista 1 -> 2
        arista 1 -> 0

Vertice 2:
        arista 2 -> 3
        arista 2 -> 1

Vertice 3:

Vertice 4:
        arista 4 -> 2
```

```
*Grafo aciclico resultante:
Vertice SCC  0 1 2 4:
        arista 0 -> 3
        arista 2 -> 3

Vertice SCC  3:

#Tiempo total 0.0513 ms
```

```
*Grafo aleatorio resultante:
Vertice 0:
        arista 0 -> 2

Vertice 1:
        arista 1 -> 4
        arista 1 -> 1
        arista 1 -> 2

Vertice 2:

Vertice 3:
        arista 3 -> 0
        arista 3 -> 4

Vertice 4:
```

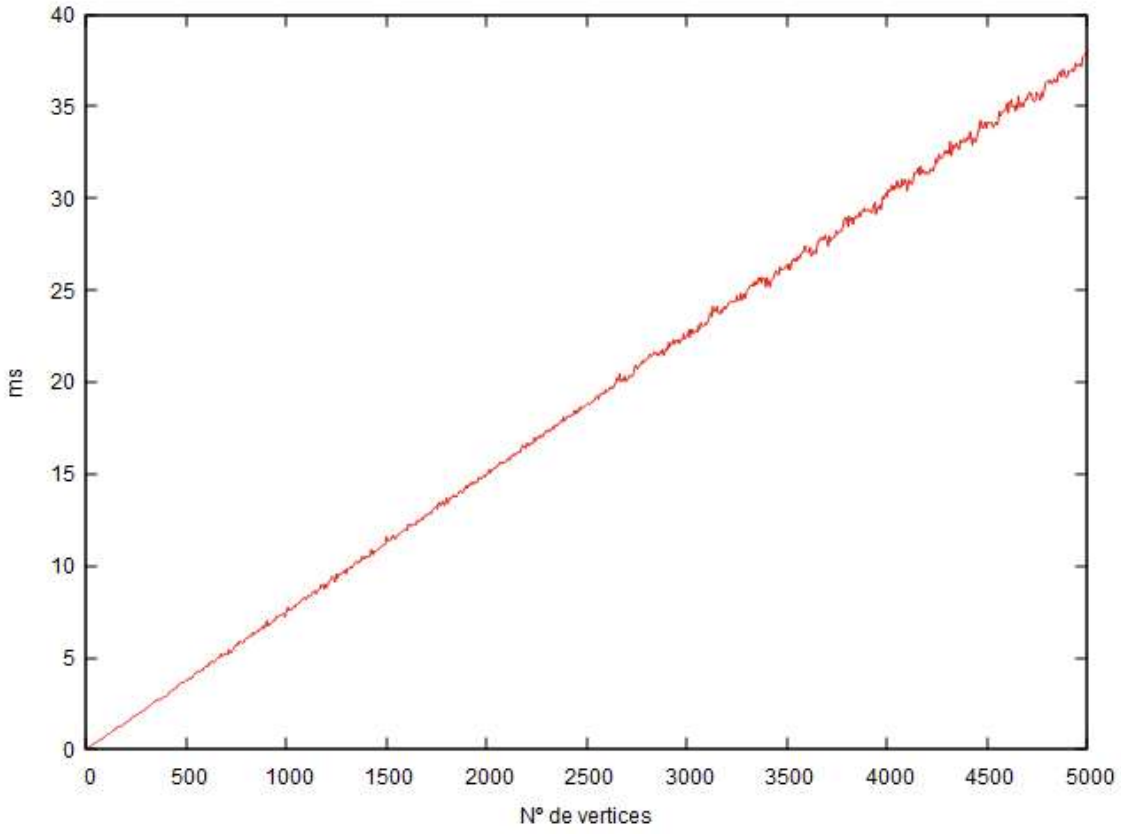
```
*Grafo aciclico resultante:
Vertice SCC  3:
        arista 3 -> 0
        arista 3 -> 4

Vertice SCC  1:
        arista 1 -> 4
        arista 1 -> 2

Vertice SCC  4:

Vertice SCC  0:
        arista 0 -> 2

Vertice SCC  2:

#Tiempo total 0.0416 ms
```

```
*Grafo aleatorio resultante:
Vertice 0:
        arista 0 -> 3
        arista 0 -> 4
        arista 0 -> 0

Vertice 1:
        arista 1 -> 3

Vertice 2:
        arista 2 -> 0

Vertice 3:
        arista 3 -> 1

Vertice 4:
        arista 4 -> 4
```

```
*Grafo aciclico resultante:
Vertice SCC  2:
        arista 2 -> 0

Vertice SCC  0:
        arista 0 -> 3
        arista 0 -> 4

Vertice SCC  4:

Vertice SCC  3 1:

#Tiempo total 0.0457 ms
```

Para la generación de tiempos según el tamaño de la estructura, la aplicación cuenta en el menú con la opción 9, que genera en un archivo “tiempos.txt” los tiempos obtenidos.

Gráfico de tiempos:



Para este grafico se ha generado grafos con vértices desde 5 hasta 5000 sumando +5 en cada iteración. Para cada iteración escogíamos los 3 primeros resultados y hacíamos la media. Si la media resultaba ser menor que 20 ms aumentábamos en 100 los resultados necesarios para la hacer media de esa iteración.

Para finalizar, observe que el generador aleatorio devuelve consistentemente un numero de aristas de orden (1,5*V). Teniendo en cuenta ese dato y que el grafico de la prueba nos muestra una recta de orden de O(n), cuadra perfectamente el coste teórico O(V+E) del algoritmo con los resultados de la implementación.