Documentación y actualización

Pontificia Universidad Javeriana

Estructura de datos

Johan Sebastián Méndez

Profesor: John Corredor

20 de octubre 2024

Resumen:

Este código implementa el algoritmo de Dijkstra, utilizado en teoría de grafos para encontrar la ruta más corta entre un nodo fuente y todos los demás nodos en un grafo ponderado. El grafo está representado por una matriz de adyacencia de 6x6, donde cada celda contiene el peso de la arista entre nodos, si existe una arista.

Funciones:

---

1. int miniDist(int distance[], bool Tset[])

   - Propósito: Encuentra el vértice con la distancia mínima que aún no ha sido incluido en el árbol de rutas más cortas (Tset).

   - Parámetros:

     - distance[]: Array que contiene las distancias actuales más cortas desde el vértice fuente a cada vértice.

     - Tset[]: Array booleano que indica si un vértice está incluido en el árbol de rutas más cortas.

   - Retorno: Índice del vértice con la distancia mínima.

2. void DijkstraAlgo(int graph[6][6], int src)

   - Propósito: Implementa el algoritmo de Dijkstra usando la matriz de adyacencia para encontrar la ruta más corta desde un vértice fuente a todos los demás vértices.

- Parámetros:

  - graph[6][6]: Array 2D que representa la matriz de adyacencia del grafo.

  - src: Vértice fuente desde el cual se calculan las rutas más cortas.

- Proceso:

  - Inicializa el array distance[] con valores infinitos y Tset[] con falso.

  - Establece la distancia del vértice fuente en 0.

  - Selecciona iterativamente el vértice con distancia mínima que aún no se ha procesado y actualiza las distancias de sus vértices adyacentes.

- Salida: Muestra la distancia más corta desde la fuente a cada vértice.

Programa Principal:

- Declara una matriz de adyacencia de 6x6 que representa el grafo ponderado.

- Llama a la función DijkstraAlgo() con el vértice fuente configurado en 0 (primer nodo del grafo).

- Muestra la distancia desde el vértice fuente a cada uno de los otros vértices.

Comentarios del Código:

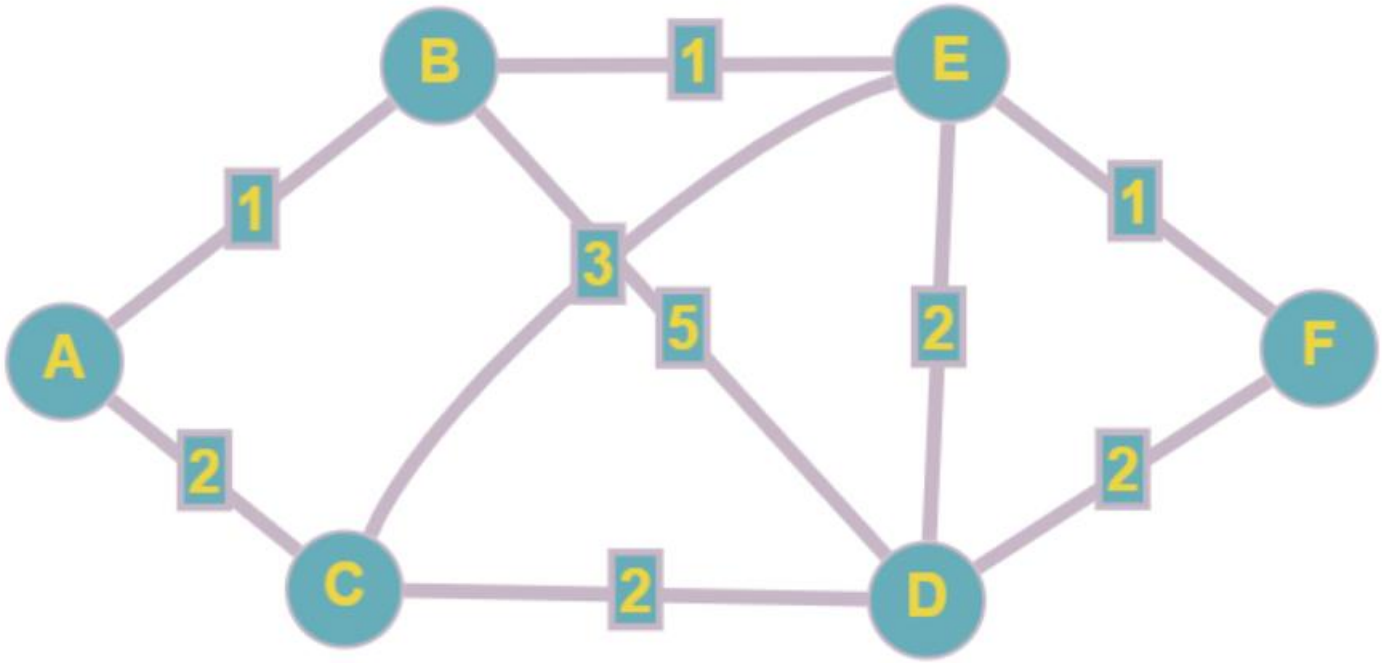
Cada parte del código incluye comentarios que explican el propósito de las variables y los bucles, lo que mejora la legibilidad y el mantenimiento.

Ejemplo de Uso:

Al ejecutar este código, se muestran las distancias más cortas desde el vértice 0 (fuente) a cada uno de los otros vértices en el grafo.

Primer grafo implementado:

```
int graph[6][6] = {
    {0, 1, 2, 0, 0, 0},
    {1, 0, 0, 5, 1, 0},
    {2, 0, 0, 2, 3, 0},
    {0, 5, 2, 0, 2, 2},
    {0, 1, 3, 2, 0, 1},
    {0, 0, 0, 2, 1, 0}
};
```



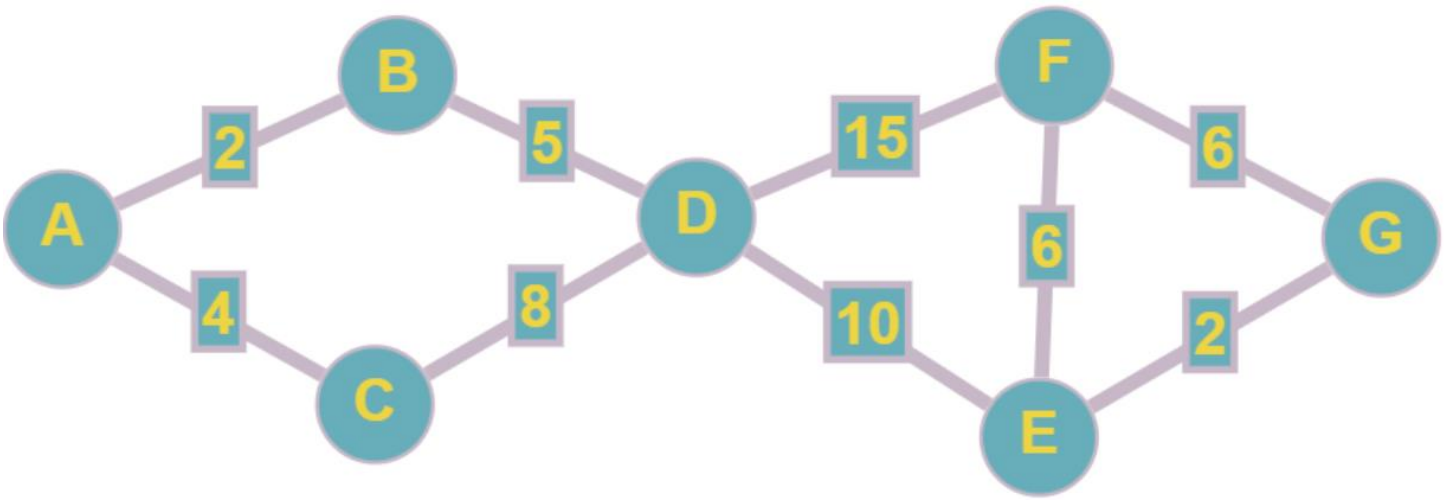
Salida:

```
Vertice                  Distancia desde la fuente al Vertice
A                        0
B                        1
C                        2
D                        4
E                        2
F                        3

--------------------------------
Process exited after 0.01161 seconds with return value 0
Presione una tecla para continuar . . .
```

Segundo grafo implementado:

```
int graph[7][7] = {
    {0, 2, 4, 0, 0, 0, 0},
    {2, 0, 0, 5, 0, 0, 0},
    {4, 0, 0, 8, 3, 0, 0},
    {0, 5, 8, 0, 10, 15, 0},
    {0, 0, 3, 10, 0, 6, 2},
    {0, 0, 0, 15, 6, 0, 6},
    {0, 0, 0, 0, 2, 6, 0}
};
```



Salida:

```
Vertice                  Distancia desde la fuente al Vertice
A                        0
B                        2
C                        4
D                        7
E                        17
F                        22
G                        19
```

Notas:

- El tamaño del grafo se establece en 6 y 7 vértices, pero se puede ajustar para adaptarse a diferentes estructuras de grafos.

- Para facilitar la lectura, los vértices están etiquetados como letras (A, B, C, ...) correspondientes a su posición en la matriz de adyacencia.