**Laboratorio S15. AYED-02**

Jefer Alexis González Romero

**DOCUMENTO TÉCNICO**

**Implementación del algoritmo de Prim para la representación de matriz y lista de adyacencia**

**Requisitos**

**Especificación**

1

7

5

3

**a**

**e**

**b**

**d**

**c**

6

2

4

grafo =

1

7

5

3

**a**

**e**

**b**

**d**

**c**

6

2

4

grafo.prim(“a”)

(b, e, 3)

(e, d, 2)

(b, c, 5)

(None, a, 0)

(a, b, 1)

**Entrada:**

Se tiene un grafo y se recibe un vértice.

**Salida:**

Encuentra los arcos que forman el árbol de expansión mínimo de un grafo

**Diseño**

**Estrategia**

Con el atributo “data\_props” se le asignará dos propiedades a cada vértice: “key” que da el peso del arco del árbol de expansión que se encontró de primeras para ese nodo y “phi” que da el nodo con el cuál se hizo esa unión.

Al vertice arbitrario que se escoja se le pone 0 como key, ya que se comienza desde el, los demás tendrán infinito como key y todos None en phi. Se creará una lista llamada Q donde estarán desde un principio todos los vertices y con esta mientras no este vacía se van a ir eligiendo los vértices que tenga key menor, para esto se crea una función llamada “extract\_min” donde se revisa el valor de todos los keys de los vértices en Q y se escoge el menor, este se saca de Q y se retorna para posteriormente encontrar sus vecinos y asignarles su phi y key correspondientes, esto se hace hasta ya se encuentren en el árbol de expansión todos los vértices.

Para la matriz y lista de adyacencia es lo mismo, ya que, lo único que cambia es la búsqueda de los vecinos y arcos que se vio en algoritmos anteriores.

Adjunto el programa con nombre “Algoritmo de Prim”.

**Casos prueba**

Para los casos prueba se va a usar el siguiente árbol:

grafo =

1

7

5

3

**a**

**e**

**b**

**d**

**c**

6

2

4

|  |  |  |
| --- | --- | --- |
| Entrada | Justificación | Salida |
| grafo.bfs(“b”) | Vértice con tres vecinos. | (e, d, 2)  (b, e, 3)  (b, c, 5)  (None, b, 0)  (b, a, 1) |
| grafo.bfs(“d”) | Vértice dos vecinos | (e, b, 3)  (None, d, 0)  (b, a, 1)  (b, c, 5)  (d, e, 2) |

**Análisis**

**Temporal**

En el mejor de los casos T(n) = Ω(1)

En el peor de los casos T(n) = O(V2)

Si el grafo se presenta en una lista de adyacencia:

En el peor de los casos T(n) = O(E logV)

**Código**

**Documentación**

Dentro del código

**Prototipo**

Dentro del programa con nombre “Algoritmo de Prim”.

La gráfica que ilustra el grafo usado en el prototipo es el siguiente:

1

7

8

11

10

**d**

**f**

**a**

**c**

**h**

**i**

**g**

**b**

**e**

4

8

7

9

14

4

2

6

2

**Casos de prueba**

1. Comienza desde la componente a

1

7

8

11

10

**d**

**f**

**a**

**c**

**h**

**i**

**g**

**b**

**e**

4

8

7

9

14

4

2

6

2

(c, i, 2)

(f, g, 2)

(c, f, 4)

(b, c, 8)

(d, e, 9)

(a, b, 4)

(c, d, 7)

(None, a, 0)

(g, h, 1)

1. Comienza desde la componente c

Como se puede ver a continuación no es exactamente igual al anterior, en este caso se incluye la conexión a-h y no la b-c lo cual tiene sentido, porque con cualquiera de la dos da que la longitud de cable que se necesita es 37 y además se incluyen en los dos todos los componentes.

1

7

8

11

10

**d**

**f**

**a**

**c**

**h**

**i**

**g**

**b**

**e**

4

8

7

9

14

4

2

6

2

(c, i, 2)

(f, g, 2)

(c, f, 4)

(None, c, 0)

(d, e, 9)

(a, b, 4)

(c, d, 7)

(h, a, 8)

(g, h, 1)

1. Comienza desde la componente e

1

7

8

11

10

**d**

**f**

**a**

**c**

**h**

**i**

**g**

**b**

**e**

4

8

7

9

14

4

2

6

2

(c, i, 2)

(f, g, 2)

(c, f, 4)

(d, c, 7)

(None, e, 0)

(a, b, 4)

(e, d, 9)

(h, a, 8)

(g, h, 1)