

## **ÁRBOLES DE BÚSQUEDA**

### **Taller 3 (Grafos)**

#### **GRUPO 1**

Sophia Aristizábal

(Ingeniería de Sistemas y economía)

Juan Felipe Bríñez

(Ingeniería sistemas)

Iván Darío Orozco Ibáñez

(Ingeniería de Sistemas e Ingeniería Electrónica)

Facultad de Ingeniería, Pontificia Universidad Javeriana

SISTEMAS 4800-1 (2891): Estructuras de Datos (Teo-Prac)

Andrea del Pilar Rueda Olarte

17 de mayo del 2023



Pontificia Universidad  
**JAVERIANA**  
Bogotá

Pontificia Universidad Javeriana  
Estructura de Datos

***Índice***

<b>Desarrollo del taller.....</b>	<b>3</b>
1. Diseño del grafo.....	3
<b>TAD Grafo.....</b>	<b>3</b>
<b>TAD Vertice.....</b>	<b>5</b>
2. Diagrama de relación del grafo.....	7
3. Resultados del algoritmo Prim.....	7
a. <b>Archivo graph01.txt.....</b>	<b>7</b>
b. <b>Archivo graph02.txt.....</b>	<b>9</b>
c. <b>Archivo graph03.txt.....</b>	<b>10</b>
d. <b>Archivo graph04.txt.....</b>	<b>12</b>
e. <b>Archivo graph05.txt.....</b>	<b>15</b>
4. Resultados del algoritmo Dijkstra.....	18
a. <b>Archivo graph01.txt.....</b>	<b>18</b>
b. <b>Archivo graph02.txt.....</b>	<b>20</b>
c. <b>Archivo graph03.txt.....</b>	<b>21</b>
d. <b>Archivo graph04.txt.....</b>	<b>23</b>
e. <b>Archivo graph05.txt.....</b>	<b>26</b>
5. Conclusiones.....	28

## *Desarrollo del taller*

1. Diseño del grafo:

### TAD Grafo

#### Datos mínimos:

- **vertice**, lista de vértices del tipo **Vertice**, representa una lista de vértices o índices del grafo.
- **grafo**, multi-lista de vértices del tipo **Vertice**, representa las aristas del grafo.
- **esDirigido**, booleano, identifica el tipo del grafo. Si es verdadero significa que el grafo es dirigido, en el caso contrario significa que no es dirigido.

#### Operaciones:

- + **Grafo()**, crea un nuevo grafo con atributos vacíos.
- + **Grafo( vertices, graf, tipoGrafo )**, crea un grafo a partir de los valores de una lista de vértices **ind**, una multilista de aristas **graf** y la definición del tipo de grafo **tipoGrafo** por defecto.
- + **~Grafo()**, elimina el grafo actual.
- + **esVacio()**, identifica si el grafo aún no tiene ningún vértice y por lo tanto ninguna arista. Retornando verdadero en el caso que si este vacío y falso en el caso contrario.
- + **InsertarVertice( vertice )**, agrega un nuevo **vertice** a lista de vértices **vertice**, asimismo agrega una lista de vértices disponibles para usar dentro de la multi-lista **grafo**.
- + **InsertarArista( VerticeOrigen, VerticeDestino, peso )**, se inserta una nueva arista, implica partir de la lista de vértices dentro de la multi-lista de vértices **grafo** con el índice **VerticeOrigen**, agregando el vértice **VerticeDestino** dentro de esta.
- + **NumeroVertices()**, Identifica la cantidad de vértices existentes en el grafo actual.
- + **cantidadAristas()**, Identifica la cantidad de aristas existentes en el grafo actual.
- + **BuscarVertice( indice )**, recorre la lista de vértices **vertice** hasta encontrar el vértice mediante el **indice** especificado. Devuelve verdadero en caso de ser encontrado, falso en el caso contrario.



## ÁRBOLES DE BÚSQUEDA

- + **BuscarArista( *VerticeOrigen*, *VerticeDestino* )**, realiza un recorrido sobre la lista de vértices **vertice** hasta encontrar la posición de **VerticeOrigen**, mediante el cual referencia a la multi-lista de vértices **grafo** del cual puede encontrar al **VerticeDestino** especificado. Devuelve verdadero en caso de ser encontrado, falso en el caso contrario.
- + **eliminarVertice( *indice* )**, recorre la lista de vértices **vertice** hasta encontrar el vértice mediante el **indice** especificado. Devuelve verdadero en caso de ser eliminado el vértice especificado, falso en el caso contrario.
- + **eliminarArista( *VerticeOrigen*, *VerticeDestino* )**, realiza un recorrido sobre la lista de vértices **vertice** hasta encontrar la posición de **VerticeOrigen**, mediante el cual referencia a la multi-lista de vértices **grafo** del cual puede encontrar al **VerticeDestino** especificado. Devuelve verdadero en caso de eliminar la arista especificada, falso en el caso contrario.
- + **recorridoPlano()**, recorre la lista de vértices **vertice**, encontrando cada elemento dentro de la misma sin duplicados.
- + **recorridoEnProfundidad( *valor* )**, busca si el vértice **valor** dentro de la lista de vértices **vertice** existe. En caso de ser verdadero se llama al **recorridoEnProfundidad( *Vertices*, *valor*, *Visitados* )** para hacer el recorrido correspondiente.
- + **recorridoEnProfundidad( *Vertices*, *valor*, *Visitados* )**, recursivamente recorre desde el vértice **valor** especificado sus vecinos hasta que no se pueda avanzar más .
- + **recorridoAdyacencias( *valor* )**, recorre desde el vértice **valor** especificado cada vecino hasta que no queden más y luego pasa con los vecinos de sus vecinos, haciendo alusión al recorrido por niveles del grafo.
- + **esHamilton( *inicio* )**, verifica que efectivamente el vértice **inicio** especificado se encuentre dentro de la lista de vértices **vertice**. Devuelve falso en caso de no ser encontrado, en el caso contrario procede a llamar a **esHamilton( *analizar*, *Visitados* )**.
- + **esHamilton( *analizar*, *Visitados* )**, recorre a través de las aristas del grafo cada vértice una sola vez. Devuelve verdadero en caso de lograr el recorrido, falso en el caso contrario.



## ÁRBOLES DE BÚSQUEDA

- + **esPar( vert )**, verifica si efectivamente el vértice **vert** especificado se caracteriza por tener un grado par.
- + **esEuler( inicio )**, realiza un recorrido de los vértices desde el vértice **inicio** especificado, a través de las aristas del grafo una sola vez. Devuelve verdadero en caso de lograr el recorrido, falso en el caso contrario.
- + **esCircuitoEuler( inicio )**, realiza un recorrido de los vértices desde el vértice **inicio** especificado, a través de las aristas del grafo una sola vez hasta volver desde el mismo punto que partió. Devuelve verdadero en caso de lograr el recorrido, falso en el caso contrario.
- + **Dijkstra( inicio )**, realiza el recorrido siguiendo el algoritmo de Dijkstra para encontrar la ruta más corta a partir de un nodo inicio en el grafo.
- + **eliminarDeCola( cola, inicio )**, elimina un valor de la cola utilizada en la operación de Dijkstra
- + **escogerSiguiente( cola, pesos )**, retorna la posición del vértice con el menor peso de la cola.
- + **encontrarVecinos( inicio )**, retorna los vecinos de un vértice dado.
- + **buscaDistancia( VerticeOrigen, VerticeDestino )**, encuentra la distancia entre dos vértices dados, uno de origen y otro de destino.
- + **llenarCola( cola )**, llena una cola con las posiciones de los elementos en la lista de vértices del grafo.
- + **hacerRecorridos( recorridos, ordenVertices, predecesores )**, hace el recorrido teniendo en cuenta el algoritmo de Dijkstra para cada vértice del grafo.
- + **Prim( inicio )**, realiza el recorrido siguiendo el algoritmo de Prim desde un vértice de inicio.
- + **AnadirVertices( referencia, aux, Visitados, pred )**, agrega el vértice a la lista de vértices.
- + **BuscarPos( valor )**, busca la posición del punto que se está haciendo referencia en la lista de puntos.
- + **Visitado( valor, Visitados )**, retorna verdadero si el punto ya fue visitado.
- + **Buscar( inicio, destino )**, elimina el grafo actual.



## ÁRBOLES DE BÚSQUEDA

- + **ObtenerCosto( VerticeOrigen, VerticeDestino )**, busca una arista dada y retorna el peso de la misma.

## TAD Vertice

### Datos mínimos:

- **Peso**, tipo plantilla, representa el peso de la arista.
- **Índice**, tipo plantilla, representa el índice del vértice.
- **visitado**, booleano, representa si se visitó ese vértice.
- **Grado**, entero , representa cuántas aristas tiene.
- **Predecesor**, entero, representa el predecesor del vértice.

### Operaciones:

- + **Vertice()**, crea un vértice vacío.
- + **Vertice( peso, indice )**, crea un vértice con un peso y un índice dado.
- + **setIndice( indice )**, fija el índice del vértice.
- + **setPeso( peso )**, fija el peso del vértice.
- + **setVisitado( vis )**, fija si se visita el vértice.
- + **getIndice()**, retorna el índice del vértice.
- + **getPeso()**, retorna el peso del vértice.
- + **getPredecesor()**, retorna el predecesor del vértice.
- + **getVisit()**, retorna el estado de visitado del vértice.
- + **getGrado()**, retorna el grado del vértice.
- + **setGrado( grado )**, fija el grado del vértice.
- + **setPredecesor( predecesor )**, fija el predecesor del vértice.



## ÁRBOLES DE BÚSQUEDA

2. Diagrama de relación del grafo:

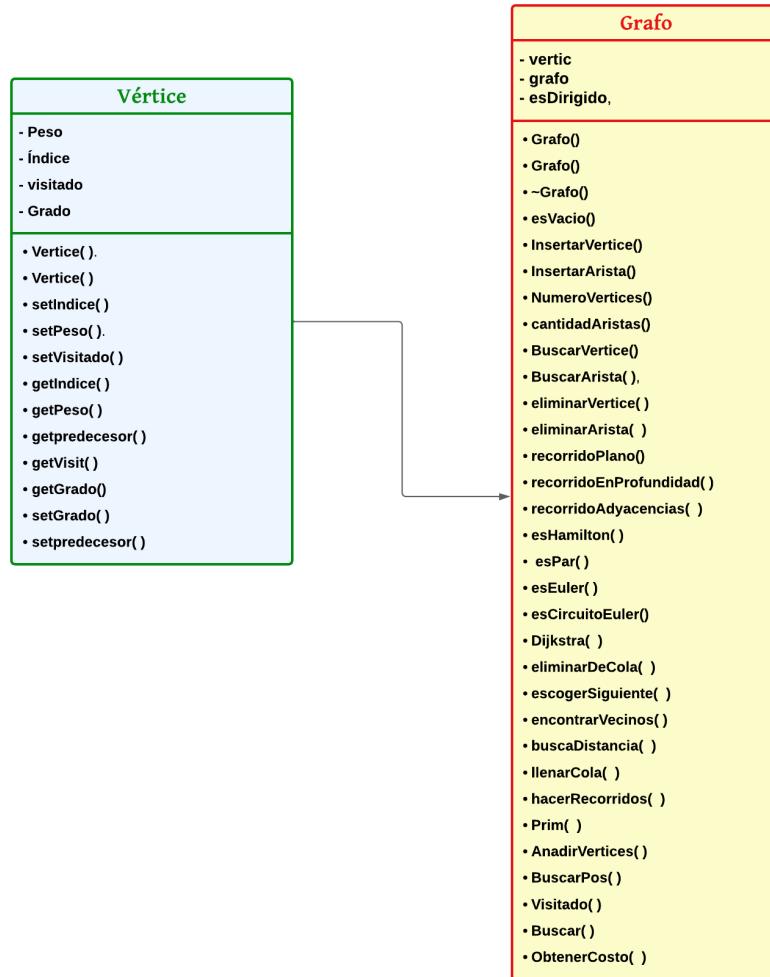


Fig 1. Diagrama de relación del grafo.

3. Resultados del algoritmo Prim:

### a. Archivo graph01.txt

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (25, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Prim puede ofrecer. Se puede visualizar en la Fig 2.



## ÁRBOLES DE BÚSQUEDA

```
[base) nidhood@Ivans-MacBook-Pro tallerGrafo-4 % ./test graph01.txt
Porteria: 25,60
```

Comparacion con Prim

Casa 1:70,60

Distancia lineal a porteria: 45

Camino segun algoritmo de Prim: 0 - 3 - 4 - 1 -

Distancia total recorrida con algoritmo de Prim: 80.3553

Casa 2:5,35

Distancia lineal a porteria: 32.0156

Camino segun algoritmo de Prim: 0 - 2 -

Distancia total recorrida con algoritmo de Prim: 32.0156

Casa 3:50,35

Distancia lineal a porteria: 35.3553

Camino segun algoritmo de Prim: 0 - 3 -

Distancia total recorrida con algoritmo de Prim: 35.3553

Casa 4:70,35

Distancia lineal a porteria: 51.4781

Camino segun algoritmo de Prim: 0 - 3 - 4 -

Distancia total recorrida con algoritmo de Prim: 55.3553

Casa 5:90,35

Distancia lineal a porteria: 69.6419

Camino segun algoritmo de Prim: 0 - 3 - 4 - 5 -

Distancia total recorrida con algoritmo de Prim: 75.3553

Casa 6:25,10

Distancia lineal a porteria: 50

Camino segun algoritmo de Prim: 0 - 2 - 6 -

Distancia total recorrida con algoritmo de Prim: 64.0312

Casa 7:70,10

Distancia lineal a porteria: 67.2681

Camino segun algoritmo de Prim: 0 - 3 - 4 - 7 -

Distancia total recorrida con algoritmo de Prim: 80.3553

Fig 2. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph01.txt .

## ÁRBOLES DE BÚSQUEDA

### b. Archivo graph02.txt

Para el segundo archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Prim puede ofrecer. Se puede visualizar en la *Fig 3*.

```
[base] nidhood@Ivans-MacBook-Pro tallerGrafo-4 % ./test graph02.txt
Porteria: 10,60

Comparacion con Prim
Casa 1:35,60
Distancia lineal a porteria: 25
Camino segun algoritmo de Prim: 0 - 1 -
Distancia total recorrida con algoritmo de Prim: 25

Casa 2:80,60
Distancia lineal a porteria: 70
Camino segun algoritmo de Prim: 0 - 1 - 4 - 2 -
Distancia total recorrida con algoritmo de Prim: 100.967

Casa 3:10,30
Distancia lineal a porteria: 30
Camino segun algoritmo de Prim: 0 - 3 -
Distancia total recorrida con algoritmo de Prim: 30

Casa 4:50,30
Distancia lineal a porteria: 50
Camino segun algoritmo de Prim: 0 - 1 - 4 -
Distancia total recorrida con algoritmo de Prim: 58.541

Casa 5:50,10
Distancia lineal a porteria: 64.0312
Camino segun algoritmo de Prim: 0 - 1 - 4 - 5 -
Distancia total recorrida con algoritmo de Prim: 78.541
```

*Fig 3. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph02.txt .*

## ÁRBOLES DE BÚSQUEDA

### c. Archivo graph03.txt

Para el tercer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (40, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Prim puede ofrecer. Se puede visualizar en la *Fig 4*.



## ÁRBOLES DE BÚSQUEDA

```
[base) nidhood@Ivans-MacBook-Pro tallerGrafo-4 % ./test graph03.txt
Porteria: 40,60

Comparacion con Prim
Casa 1:5,30
Distancia lineal a porteria: 46.0977
Camino segun algoritmo de Prim: 0 - 2 - 1 -
Distancia total recorrida con algoritmo de Prim: 56.6228

Casa 2:30,30
Distancia lineal a porteria: 31.6228
Camino segun algoritmo de Prim: 0 - 2 -
Distancia total recorrida con algoritmo de Prim: 31.6228

Casa 3:40,40
Distancia lineal a porteria: 20
Camino segun algoritmo de Prim: 0 - 2 - 3 -
Distancia total recorrida con algoritmo de Prim: 45.7649

Casa 4:60,30
Distancia lineal a porteria: 36.0555
Camino segun algoritmo de Prim: 0 - 2 - 3 - 4 -
Distancia total recorrida con algoritmo de Prim: 68.1256

Casa 5:80,30
Distancia lineal a porteria: 50
Camino segun algoritmo de Prim: 0 - 2 - 3 - 4 - 5 -
Distancia total recorrida con algoritmo de Prim: 88.1256

Casa 6:40,10
Distancia lineal a porteria: 50
Camino segun algoritmo de Prim: 0 - 2 - 6 -
Distancia total recorrida con algoritmo de Prim: 53.9835
```

*Fig 4. Impresión de resultados utilizando el algoritmo de PRIM para el archivo graph03.txt .*

## ÁRBOLES DE BÚSQUEDA

### d. Archivo graph04.txt

Para el cuarto archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 30) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Prim puede ofrecer. Se puede visualizar en la *Fig 5*, *Fig 6* y *Fig 7*.



## ÁRBOLES DE BÚSQUEDA

```
|(base) nidhood@Ivans-MacBook-Pro tallerGrafo-4 % ./test graph04.txt
Porteria: 10,30
```

Comparacion con Prim

Casa 1:20,20

Distancia lineal a porteria: 14.1421

Camino segun algoritmo de Prim: 0 - 2 - 1 -

Distancia total recorrida con algoritmo de Prim: 20

Casa 2:20,30

Distancia lineal a porteria: 10

Camino segun algoritmo de Prim: 0 - 2 -

Distancia total recorrida con algoritmo de Prim: 10

Casa 3:30,10

Distancia lineal a porteria: 28.2843

Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 - 3 -

Distancia total recorrida con algoritmo de Prim: 40

Casa 4:30,20

Distancia lineal a porteria: 22.3607

Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 -

Distancia total recorrida con algoritmo de Prim: 30

Casa 5:30,30

Distancia lineal a porteria: 20

Camino segun algoritmo de Prim: 0 - 2 - 5 -

Distancia total recorrida con algoritmo de Prim: 20

Casa 6:40,10

Distancia lineal a porteria: 36.0555

Camino segun algoritmo de Prim: 0 - 2 - 5 - 8 - 6 -

Distancia total recorrida con algoritmo de Prim: 40

Fig 5. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph04.txt .



**ÁRBOLES DE BÚSQUEDA****Casa 7:40,20****Distancia lineal a portería: 31.6228****Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 - 7 -****Distancia total recorrida con algoritmo de Prim: 40****Casa 8:40,30****Distancia lineal a portería: 30****Camino segun algoritmo de Prim: 0 - 2 - 5 - 8 -****Distancia total recorrida con algoritmo de Prim: 30****Casa 9:50,10****Distancia lineal a portería: 44.7214****Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 - 7 - 9 -****Distancia total recorrida con algoritmo de Prim: 50****Casa 10:50,20****Distancia lineal a portería: 41.2311****Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 - 3 - 10 -****Distancia total recorrida con algoritmo de Prim: 50****Casa 11:50,30****Distancia lineal a portería: 40****Camino segun algoritmo de Prim: 0 - 2 - 5 - 8 - 11 -****Distancia total recorrida con algoritmo de Prim: 40****Casa 12:60,10****Distancia lineal a portería: 53.8516****Camino segun algoritmo de Prim: 0 - 2 - 5 - 8 - 6 - 12 -****Distancia total recorrida con algoritmo de Prim: 50****Casa 13:60,20****Distancia lineal a portería: 50.9902****Camino segun algoritmo de Prim: 0 - 2 - 5 - 8 - 11 - 13 -****Distancia total recorrida con algoritmo de Prim: 54.1421***Fig 6. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph04.txt .*

## ÁRBOLES DE BÚSQUEDA

**Casa 14:70,10**

**Distancia lineal a portería: 63.2456**

**Camino segun algoritmo de Prim: 0 - 2 - 1 - 4 - 3 - 10 - 14 -**

**Distancia total recorrida con algoritmo de Prim: 60**

*Fig 7. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph04.txt .*

### e. Archivo graph05.txt

Para el quinto archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 15) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Prim puede ofrecer. Se puede visualizar en la *Fig 8*, *Fig 9* y *Fig 10*.



## ÁRBOLES DE BÚSQUEDA

```
[base] nidhood@Ivans-MacBook-Pro tallerGrafo-4 % ./test graph05.txt
Porteria: 10,15

Comparacion con Prim
Casa 1:15,40
Distancia lineal a porteria: 25.4951
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 1 -
Distancia total recorrida con algoritmo de Prim: 95.1512

Casa 2:15,65
Distancia lineal a porteria: 50.2494
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 - 4 - 3 - 2 -
Distancia total recorrida con algoritmo de Prim: 143.133

Casa 3:30,70
Distancia lineal a porteria: 58.5235
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 - 4 - 3 -
Distancia total recorrida con algoritmo de Prim: 127.321

Casa 4:45,80
Distancia lineal a porteria: 73.8241
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 - 4 -
Distancia total recorrida con algoritmo de Prim: 109.293

Casa 5:80,75
Distancia lineal a porteria: 92.1954
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 - 5 -
Distancia total recorrida con algoritmo de Prim: 111.881

Casa 6:80,40
Distancia lineal a porteria: 74.3303
Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 14 - 6 -
Distancia total recorrida con algoritmo de Prim: 91.2656

Casa 7:75,15
Distancia lineal a porteria: 65
Camino segun algoritmo de Prim: 0 - 10 - 18 - 9 - 8 - 17 - 7 -
Distancia total recorrida con algoritmo de Prim: 99.3398
```

Fig 8. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph05.txt .

## ÁRBOLES DE BÚSQUEDA

Casa 8:50,5

Distancia lineal a portería: 41.2311

Camino segun algoritmo de Prim: 0 - 10 - 18 - 9 - 8 -

Distancia total recorrida con algoritmo de Prim: 70.1976

Casa 9:35,15

Distancia lineal a portería: 25

Camino segun algoritmo de Prim: 0 - 10 - 18 - 9 -

Distancia total recorrida con algoritmo de Prim: 52.1699

Casa 10:20,25

Distancia lineal a portería: 14.1421

Camino segun algoritmo de Prim: 0 - 10 -

Distancia total recorrida con algoritmo de Prim: 14.1421

Casa 11:30,55

Distancia lineal a portería: 44.7214

Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 - 4 - 3 - 11 -

Distancia total recorrida con algoritmo de Prim: 142.321

Casa 12:50,60

Distancia lineal a portería: 60.208

Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 -

Distancia total recorrida con algoritmo de Prim: 77.1235

Casa 13:60,70

Distancia lineal a portería: 74.3303

Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 12 - 13 -

Distancia total recorrida con algoritmo de Prim: 91.2656

Casa 14:70,55

Distancia lineal a portería: 72.111

Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 - 14 -

Distancia total recorrida con algoritmo de Prim: 73.2379

*Fig 9. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph05.txt .*



**ÁRBOLES DE BÚSQUEDA**

**Casa 15:60,45**

**Distancia lineal a portería: 58.3095**

**Camino segun algoritmo de Prim: 0 - 10 - 18 - 15 -**

**Distancia total recorrida con algoritmo de Prim: 59.0957**

**Casa 16:55,25**

**Distancia lineal a portería: 46.0977**

**Camino segun algoritmo de Prim: 0 - 10 - 18 - 9 - 8 - 17 - 16 -**

**Distancia total recorrida con algoritmo de Prim: 95.5201**

**Casa 17:60,15**

**Distancia lineal a portería: 50**

**Camino segun algoritmo de Prim: 0 - 10 - 18 - 9 - 8 - 17 -**

**Distancia total recorrida con algoritmo de Prim: 84.3398**

**Casa 18:35,35**

**Distancia lineal a portería: 32.0156**

**Camino segun algoritmo de Prim: 0 - 10 - 18 -**

**Distancia total recorrida con algoritmo de Prim: 32.1699**

*Fig 10. Impresión de resultados utilizando el algoritmo de Prim para el archivo graph05.txt .*

4. Resultados del algoritmo Dijkstra:

**a. Archivo graph01.txt**

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (25, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Dijkstra puede ofrecer. Se puede visualizar en la *Fig 11*.



**ÁRBOLES DE BÚSQUEDA**

**Comparacion con Dijkstra:**

**Casa 1:70,60**

**Distancia lineal a porteria: 45**

**Camino segun algoritmo de Dijkstra: 0 - 1 -**

**Distancia total recorrida con algoritmo de Dijkstra: 45**

**Casa 2:5,35**

**Distancia lineal a porteria: 32.0156**

**Camino segun algoritmo de Dijkstra: 0 - 2 -**

**Distancia total recorrida con algoritmo de Dijkstra: 32.0156**

**Casa 3:50,35**

**Distancia lineal a porteria: 35.3553**

**Camino segun algoritmo de Dijkstra: 0 - 3 -**

**Distancia total recorrida con algoritmo de Dijkstra: 35.3553**

**Casa 4:70,35**

**Distancia lineal a porteria: 51.4781**

**Camino segun algoritmo de Dijkstra: 0 - 3 - 4 -**

**Distancia total recorrida con algoritmo de Dijkstra: 55.3553**

**Casa 5:90,35**

**Distancia lineal a porteria: 69.6419**

**Camino segun algoritmo de Dijkstra: 0 - 3 - 4 - 5 -**

**Distancia total recorrida con algoritmo de Dijkstra: 75.3553**

**Casa 6:25,10**

**Distancia lineal a porteria: 50**

**Camino segun algoritmo de Dijkstra: 0 - 2 - 6 -**

**Distancia total recorrida con algoritmo de Dijkstra: 64.0312**

**Casa 7:70,10**

**Distancia lineal a porteria: 67.2681**

**Camino segun algoritmo de Dijkstra: 0 - 3 - 7 -**

**Distancia total recorrida con algoritmo de Dijkstra: 67.371**



## ÁRBOLES DE BÚSQUEDA

*Fig 11. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph01.txt .*

### b. Archivo graph02.txt

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Dijkstra puede ofrecer. Se puede visualizar en la *Fig 12*.

**Comparacion con Dijkstra:**

**Casa 1:35,60**

**Distancia lineal a porteria: 25**

**Camino segun algoritmo de Dijkstra: 0 - 1 -**

**Distancia total recorrida con algoritmo de Dijkstra: 25**

**Casa 2:80,60**

**Distancia lineal a porteria: 70**

**Camino segun algoritmo de Dijkstra: 0 - 2 -**

**Distancia total recorrida con algoritmo de Dijkstra: 70**

**Casa 3:10,30**

**Distancia lineal a porteria: 30**

**Camino segun algoritmo de Dijkstra: 0 - 3 -**

**Distancia total recorrida con algoritmo de Dijkstra: 30**

**Casa 4:50,30**

**Distancia lineal a porteria: 50**

**Camino segun algoritmo de Dijkstra: 0 - 1 - 4 -**

**Distancia total recorrida con algoritmo de Dijkstra: 58.541**

**Casa 5:50,10**

**Distancia lineal a porteria: 64.0312**

**Camino segun algoritmo de Dijkstra: 0 - 5 -**

**Distancia total recorrida con algoritmo de Dijkstra: 64.0312**

*Fig 12. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph02.txt .*



## ÁRBOLES DE BÚSQUEDA

### c. Archivo graph03.txt

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (40, 60) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Dijkstra puede ofrecer. Se puede visualizar en la *Fig 13*.



**ÁRBOLES DE BÚSQUEDA**

**Comparacion con Dijkstra:**

**Casa 1:5,30**

**Distancia lineal a porteria: 46.0977**

**Camino segun algoritmo de Dijkstra: 0 - 1 -**

**Distancia total recorrida con algoritmo de Dijkstra: 46.0977**

**Casa 2:30,30**

**Distancia lineal a porteria: 31.6228**

**Camino segun algoritmo de Dijkstra: 0 - 2 -**

**Distancia total recorrida con algoritmo de Dijkstra: 31.6228**

**Casa 3:40,40**

**Distancia lineal a porteria: 20**

**Camino segun algoritmo de Dijkstra: 0 - 2 - 3 -**

**Distancia total recorrida con algoritmo de Dijkstra: 45.7649**

**Casa 4:60,30**

**Distancia lineal a porteria: 36.0555**

**Camino segun algoritmo de Dijkstra: 0 - 4 -**

**Distancia total recorrida con algoritmo de Dijkstra: 36.0555**

**Casa 5:80,30**

**Distancia lineal a porteria: 50**

**Camino segun algoritmo de Dijkstra: 0 - 5 -**

**Distancia total recorrida con algoritmo de Dijkstra: 50**

**Casa 6:40,10**

**Distancia lineal a porteria: 50**

**Camino segun algoritmo de Dijkstra: 0 - 2 - 6 -**

**Distancia total recorrida con algoritmo de Dijkstra: 53.9835**

*Fig 13. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph03.txt*



## ÁRBOLES DE BÚSQUEDA

### d. Archivo graph04.txt

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 30) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Dijkstra puede ofrecer. Se puede visualizar en la *Fig 14* y *Fig 15*.



## ÁRBOLES DE BÚSQUEDA

Comparacion con Dijkstra:

Casa 1:20,20

Distancia lineal a porteria: 14.1421

Camino segun algoritmo de Dijkstra: 0 - 1 -

Distancia total recorrida con algoritmo de Dijkstra: 14.1421

Casa 2:20,30

Distancia lineal a porteria: 10

Camino segun algoritmo de Dijkstra: 0 - 2 -

Distancia total recorrida con algoritmo de Dijkstra: 10

Casa 3:30,10

Distancia lineal a porteria: 28.2843

Camino segun algoritmo de Dijkstra: 0 - 1 - 3 -

Distancia total recorrida con algoritmo de Dijkstra: 28.2843

Casa 4:30,20

Distancia lineal a porteria: 22.3607

Camino segun algoritmo de Dijkstra: 0 - 1 - 4 -

Distancia total recorrida con algoritmo de Dijkstra: 24.1421

Casa 5:30,30

Distancia lineal a porteria: 20

Camino segun algoritmo de Dijkstra: 0 - 2 - 5 -

Distancia total recorrida con algoritmo de Dijkstra: 20

Casa 6:40,10

Distancia lineal a porteria: 36.0555

Camino segun algoritmo de Dijkstra: 0 - 1 - 3 - 6 -

Distancia total recorrida con algoritmo de Dijkstra: 38.2843

Casa 7:40,20

Distancia lineal a porteria: 31.6228

Camino segun algoritmo de Dijkstra: 0 - 1 - 4 - 7 -

Distancia total recorrida con algoritmo de Dijkstra: 34.1421

## ÁRBOLES DE BÚSQUEDA

*Fig 14. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph04.txt .*

Casa 8:40,30

Distancia lineal a portería: 30

Camino segun algoritmo de Dijkstra: 0 - 2 - 5 - 8 -

Distancia total recorrida con algoritmo de Dijkstra: 30

Casa 9:50,10

Distancia lineal a portería: 44.7214

Camino segun algoritmo de Dijkstra: 0 - 1 - 3 - 6 - 9 -

Distancia total recorrida con algoritmo de Dijkstra: 48.2843

Casa 10:50,20

Distancia lineal a portería: 41.2311

Camino segun algoritmo de Dijkstra: 0 - 1 - 4 - 7 - 10 -

Distancia total recorrida con algoritmo de Dijkstra: 44.1421

Casa 11:50,30

Distancia lineal a portería: 40

Camino segun algoritmo de Dijkstra: 0 - 2 - 5 - 8 - 11 -

Distancia total recorrida con algoritmo de Dijkstra: 40

Casa 12:60,10

Distancia lineal a portería: 53.8516

Camino segun algoritmo de Dijkstra: 0 - 1 - 3 - 6 - 9 - 12 -

Distancia total recorrida con algoritmo de Dijkstra: 58.2843

Casa 13:60,20

Distancia lineal a portería: 50.9902

Camino segun algoritmo de Dijkstra: 0 - 2 - 5 - 8 - 11 - 13 -

Distancia total recorrida con algoritmo de Dijkstra: 54.1421

Casa 14:70,10

Distancia lineal a portería: 63.2456

Camino segun algoritmo de Dijkstra: 0 - 2 - 5 - 8 - 11 - 13 - 14 -

Distancia total recorrida con algoritmo de Dijkstra: 68.2843

*Fig 15. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph04.txt.*



## ÁRBOLES DE BÚSQUEDA

### e. Archivo graph05.txt

Para el primer archivo se puede evidenciar el recorrido que se obtiene a partir de la portería desde el punto (10, 15) hasta llegar las demás casas, para de esta encontrar el camino más corto (en términos de pesos o distancias en este caso) que el algoritmo Dijkstra puede ofrecer. Se puede visualizar en la *Fig 16* , *Fig 17* y *Fig 18* .

**Comparacion con Dijkstra:**

**Casa 1:15,40**

**Distancia lineal a porteria: 25.4951**

**Camino segun algoritmo de Dijkstra: 0 - 1 -**

**Distancia total recorrida con algoritmo de Dijkstra: 25.4951**

**Casa 2:15,65**

**Distancia lineal a porteria: 50.2494**

**Camino segun algoritmo de Dijkstra: 0 - 1 - 2 -**

**Distancia total recorrida con algoritmo de Dijkstra: 50.4951**

**Casa 3:30,70**

**Distancia lineal a porteria: 58.5235**

**Camino segun algoritmo de Dijkstra: 0 - 1 - 11 - 3 -**

**Distancia total recorrida con algoritmo de Dijkstra: 61.7083**

**Casa 4:45,80**

**Distancia lineal a porteria: 73.8241**

**Camino segun algoritmo de Dijkstra: 0 - 1 - 11 - 3 - 4 -**

**Distancia total recorrida con algoritmo de Dijkstra: 79.7361**

**Casa 5:80,75**

**Distancia lineal a porteria: 92.1954**

**Camino segun algoritmo de Dijkstra: 0 - 10 - 18 - 15 - 14 - 5 -**

**Distancia total recorrida con algoritmo de Dijkstra: 95.5985**

**Casa 6:80,40**

**Distancia lineal a porteria: 74.3303**

**Camino segun algoritmo de Dijkstra: 0 - 9 - 16 - 6 -**

**Distancia total recorrida con algoritmo de Dijkstra: 76.5154**

*Fig 16. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph05.txt .*



**ÁRBOLES DE BÚSQUEDA****Casa 7:75,15****Distancia lineal a portería: 65****Camino segun algoritmo de Dijkstra: 0 - 9 - 8 - 17 - 7 -****Distancia total recorrida con algoritmo de Dijkstra: 72.1699****Casa 8:50,5****Distancia lineal a portería: 41.2311****Camino segun algoritmo de Dijkstra: 0 - 9 - 8 -****Distancia total recorrida con algoritmo de Dijkstra: 43.0278****Casa 9:35,15****Distancia lineal a portería: 25****Camino segun algoritmo de Dijkstra: 0 - 9 -****Distancia total recorrida con algoritmo de Dijkstra: 25****Casa 10:20,25****Distancia lineal a portería: 14.1421****Camino segun algoritmo de Dijkstra: 0 - 10 -****Distancia total recorrida con algoritmo de Dijkstra: 14.1421****Casa 11:30,55****Distancia lineal a portería: 44.7214****Camino segun algoritmo de Dijkstra: 0 - 1 - 11 -****Distancia total recorrida con algoritmo de Dijkstra: 46.7083****Casa 12:50,60****Distancia lineal a portería: 60.208****Camino segun algoritmo de Dijkstra: 0 - 1 - 11 - 12 -****Distancia total recorrida con algoritmo de Dijkstra: 67.3238***Fig 17. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph05.txt .*

**ÁRBOLES DE BÚSQUEDA****Casa 13:60,70**

Distancia lineal a portería: 74.3303

Camino segun algoritmo de Dijkstra: 0 - 1 - 11 - 12 - 13 -

Distancia total recorrida con algoritmo de Dijkstra: 81.466

**Casa 14:70,55**

Distancia lineal a portería: 72.111

Camino segun algoritmo de Dijkstra: 0 - 10 - 18 - 15 - 14 -

Distancia total recorrida con algoritmo de Dijkstra: 73.2379

**Casa 15:60,45**

Distancia lineal a portería: 58.3095

Camino segun algoritmo de Dijkstra: 0 - 10 - 18 - 15 -

Distancia total recorrida con algoritmo de Dijkstra: 59.0957

**Casa 16:55,25**

Distancia lineal a portería: 46.0977

Camino segun algoritmo de Dijkstra: 0 - 9 - 16 -

Distancia total recorrida con algoritmo de Dijkstra: 47.3607

**Casa 17:60,15**

Distancia lineal a portería: 50

Camino segun algoritmo de Dijkstra: 0 - 9 - 8 - 17 -

Distancia total recorrida con algoritmo de Dijkstra: 57.1699

**Casa 18:35,35**

Distancia lineal a portería: 32.0156

Camino segun algoritmo de Dijkstra: 0 - 10 - 18 -

Distancia total recorrida con algoritmo de Dijkstra: 32.1699

*Fig 18. Impresión de resultados utilizando el algoritmo de Dijkstra el archivo graph05.txt .*

## 5. Conclusiones:

El algoritmo de Prim se enfoca en encontrar un árbol de expansión mínima en un grafo ponderado, mientras que el algoritmo de Dijkstra se enfoca en encontrar la ruta más corta desde un vértice fuente a todos los demás vértices. Ambos algoritmos son herramientas

## ÁRBOLES DE BÚSQUEDA

valiosas en la teoría de grafos y en la resolución de problemas de optimización de rutas como se puede observar de mejor manera en las figuras *Fig 19* y *Fig 20*.

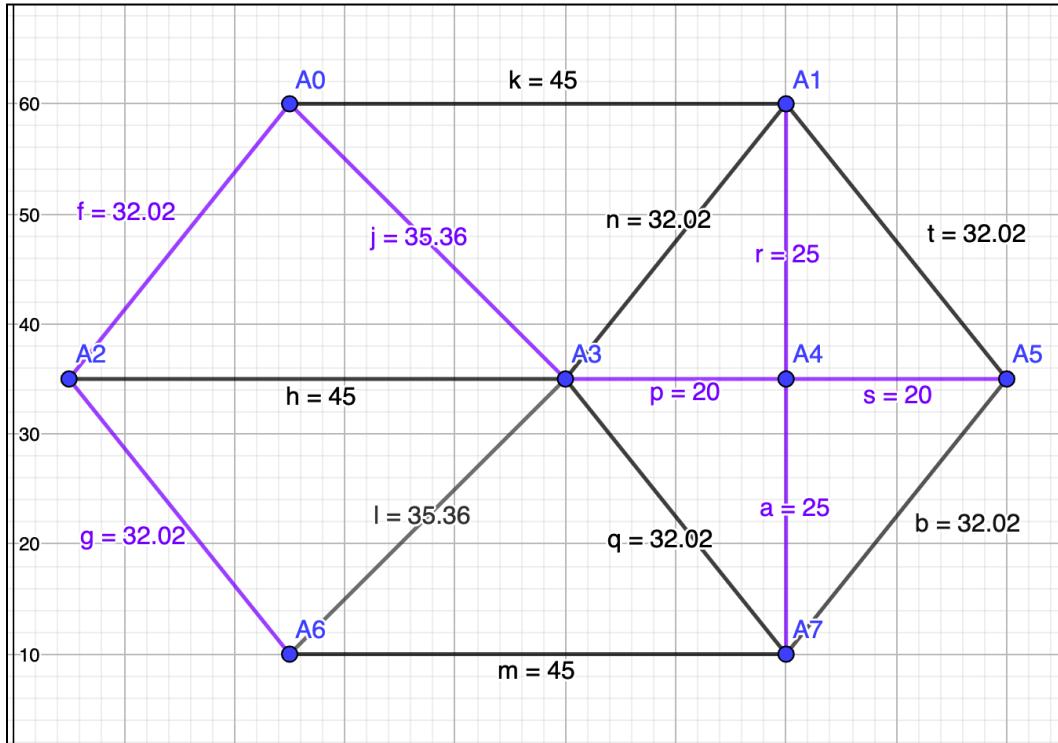


Fig 19. Gráfica de visualización de recorridos utilizando el algoritmo de Prim para el archivo graph01.txt .



## ÁRBOLES DE BÚSQUEDA

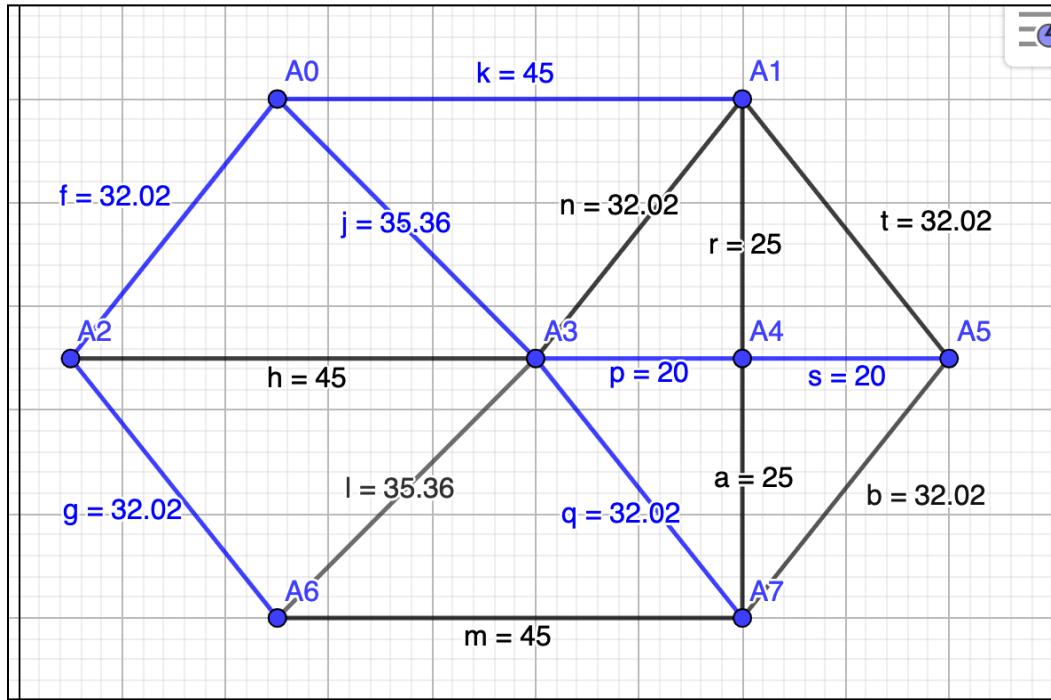


Fig 12. Gráfica de visualización de recorridos utilizando el algoritmo de Dijkstra para el archivo graph01.txt .

Mediante un arduo análisis con el grupo, suponemos que no es fácil decidir que un algoritmo sea mejor que el otro de manera general, ya que el algoritmo de Prim y el algoritmo de Dijkstra tienen aplicaciones y objetivos diferentes, donde en el caso del algoritmo de Prim es más eficiente en términos de complejidad temporal cuando el grafo es denso (tiene muchas aristas) y cuando se requiere encontrar un subconjunto mínimo de aristas para conectar todos los vértices. En cambio el algoritmo de Dijkstra es más eficiente en términos de complejidad temporal cuando el grafo es disperso (tiene pocas aristas) y cuando se necesita encontrar la ruta más corta entre un solo par de vértices.