

**INFORME DE INGENIERÍA**  
**PROYECTO FINAL**

**MARIA CAMILA LENIS RESTREPO**  
**JUAN SEBASTIAN PALMA GARCÍA**  
**JAVIER ANDRÉS TORRES REYES**

**ALGORITMOS Y ESTRUCTURAS DE DATOS**

**2018-2**

# INFORME DE INGENIERÍA

## Paso 1: Identificación del problema

### Definición del problema

Implementar una guía de la distribución de la mansión Winchester.

### Justificación

En el día de hoy, la mansión Winchester es un sitio turístico que atrae a las personas gracias al misterio que esta genera, pues nadie sabe ciertamente como está construida y no se tiene planos de ella. Por lo tanto, las personas pueden llegar a perderse sin la certeza de que alguien pueda encontrarlas; es por ello que se decidió implementar una guía para ayudar a las personas a conocer la distribución de dicha mansión. Primero, se ha buscado implementar un nuevo mapa que innove y facilite la movilidad de los visitantes de una sala a otra y que les sirva como guía para llegar a una sala que ellos quieran. Además, han decidido modificar el sistema de inventario de la mansión ya que se han descubierto las misteriosas desapariciones de objetos valiosos de algunas habitaciones y así evitar la pérdida de estos objetos. Inclusive, se ha tomado en cuenta que la mansión al tener una gran antigüedad se requiere la demolición y reconstrucción o adición de nuevas habitaciones para que esta logre mantenerse abierto al público y no imponer algún tipo de riesgo a cualquier visitante. Por último, entre estos nuevos planes se ha hecho una inversión mayor en los sistemas de comunicación dentro de la mansión para permitir una mayor facilidad de transmisión del mensaje de cierre de esta mansión.

### Requerimientos funcionales

1. Dada una habitación de la mansión se debe encontrar el camino más rápido, en minutos, desde esa habitación hasta la salida. Si la habitación no tiene salida, se debe mostrar un mensaje de advertencia.
2. El sistema debe encontrar el camino que pase por menos habitaciones desde un punto a otro de la mansión. El usuario debe ingresar el punto de partida y el de llegada, y recibe una secuencia de habitaciones incluyendo el punto de partida y el de llegada.
3. El sistema debe transmitir el mensaje de cierre a todos los rincones de la casa, de manera que este llegue de la forma más rápida posible teniendo en cuenta lo que se demora cruzar de una habitación a otra. Es importante aclarar que el mensaje, por su misma naturaleza, puede ignorar la dirección de los corredores, de manera que es suficiente que un corredor vaya de B a A para que el mensaje pueda ir de A a B.
4. Añadir una habitación a la mansión. La nueva habitación debe contener el indicador, las habitaciones a las cuales se puede llegar a través de ella, y las habitaciones de las cuales se puede llegar a ella.
5. Dado el indicador de la habitación se debe eliminar la habitación del mapa. Si la habitación contenía tesoros, estos deben quedar en el registrados en el museo.
6. Dado el indicador de dos habitaciones se deben añadir un pasillo entre ella de la siguiente manera: primera habitación indica desde dónde, segunda habitación indica hasta donde y el tiempo de minutos que toma cruzar el pasillo entre ambas habitaciones. Si ya existe un pasillo entre la primera y la segunda, muestra un mensaje de error.

7. Dado el indicador de dos habitaciones se debe eliminar el pasillo que exista entre ellas. Si no existe ningún pasillo se muestra un mensaje de alerta.
8. Dado el indicador de la habitación se deben registrar tesoros encontrados. Se debe añadir el nombre y el valor del tesoro y la habitación a la cual pertenece.
9. Visualizar los tesoros encontrados, ya sea que aún pertenezcan a la habitación o que pertenezcan al museo. Se debe mostrar su nombre, valor, habitación a la que pertenece o, en su defecto, que pertenece al museo.

## **Paso 2: Recopilación de la información**

### **Sobre la mansión Winchester**

Se sabe que la mansión Winchester fue empezada a construir alrededor de los años 1881 en California, luego de la muerte de William Winchester, y no cesó hasta el fallecimiento de Sarah Winchester, quien ponía reglas muy claras a los obreros: nada de planos. Por ende, nadie saber ciertamente como está compuesta, pero de lo que se ha logrado apreciar tiene:

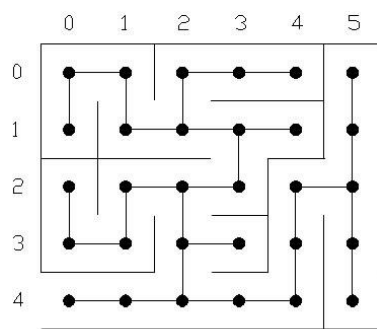
- Cuatro pisos, aunque hubo un tiempo donde alcanzó a tener siete pisos
- Dos hectáreas de longitud.
- 160 cuartos
- 40 recámaras
- Dos salones de baile
- Habitaciones secretas
- Infinitud de ventanas, puertas y muchas de ellas conducen a una pared o al vacío.
- Decoraciones con el número 13 (candelabros con 13 velas, ventanas con 13 vidrios, mosaicos con 13 particiones).
- 476 entradas.

De lo anterior se puede abstraer que para nuestra solución se debe tomar en cuenta que:

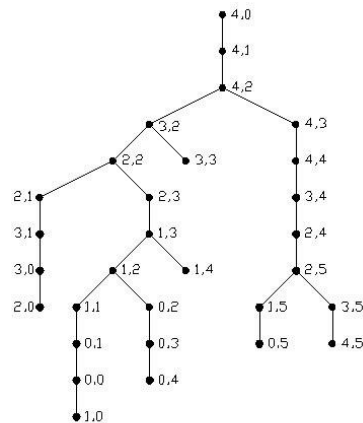
- Existen múltiples entradas, y por ende salidas, de la mansión a la hora de hallar el camino más corto hacia una salida.
- Los tesoros encontrados estarían relacionados con tener alguna característica con el número 13.
- No siempre se va a tener un camino hasta la salida

### **Aproximación de la mansión Winchester**

De manera muy general, la mansión Winchester se aproxima a lo que es un laberinto, y puede ser visto de la siguiente manera:



a)



b)

Donde se puede concluir que los pasos de un lado a otro se conectan y hay puntos de salida y puntos de llegada, lo que puede representarse en un grafo donde a) es una aproximación de una matriz de adyacencia, y en b) como la gráfica del grafo como tal.

## Grafo

Un grafo es una estructura de datos no lineal que se compone de Vértices y Aristas. Los vértices o nodos son los componentes del grafo que contienen un objeto y las aristas son caminos que conectan un vértice a otra. Un grafo puede ser direccionado o no direccionado, en el cual si este es direccionado solo puede ir de un vértice inicial a un final sin poder devolverse a la inicial a menos de que exista una arista que conecte este nodo con el inicial. El grafo no direccionado tiene aristas que permiten el viaje entre vértices mediante el uso de una sola arista ya que no tiene ningún limitante. Un grafo se considera conexo cuando se encuentra completamente conectado y no existe más de un grafo, el no conexo está separado en dos subgrafos.

Como representar un Grafo:

- Lista de Adyacencia: Es una Lista de Listas que contiene todos los vértices y marca dentro de cada una en base a un vértice si tiene aristas que los conectan entre si.
- Matriz de Adyacencia: Es una matriz de vértices que compara cada vértice con otro y si alguno tiene un vértice que lo conecta con otro se marca dentro de la intersección de la matriz la existencia de la arista, con su peso o un uno.

## DFS

DFS o Depth First Search, es un algoritmo que crea un recorrido desde un nodo inicial A, hacia el resto sin repetir los nodos que visita. Este algoritmo recorre la profundidad de un nodo hasta que no existan más nodos por los cuales se puede recorrer, antes de pasar al siguiente. Este método utiliza un arreglo booleano que mantiene un chequeo de los nodos o vértices ya visitados por la búsqueda. Al final se retorna un árbol con los órdenes de los vértices mediante profundidad.

## BFS

BFS o Breath First Search, es un algoritmo que recorre desde un nodo inicial A, a todos los demás, sin repetir visitas. Este algoritmo recorre todos los nodos adyacentes antes de seguir bajando niveles dentro de cada nodo. Este método utiliza un arreglo booleano que mantiene un chequeo de los nodos

o vértices ya visitados por la búsqueda. Al final se retorna un árbol con los órdenes de los vértices mediante su nivel de adyacencia.

### **Recorrido de Camino Mínimo**

Existen tres algoritmos de búsqueda de camino mínimo:

- Dijkstra: Este es un algoritmo que genera un árbol de camino más corto en el cual empieza desde un Vértice Inicial y se asignan todos los valores a cada vértice como infinito o un numero extremadamente grande, y se empieza a recorrer cada vértice sumando sus pesos y asignándoselos al mismo, y evalúa si otro nodo por otro lado llega a ese mismo vértice y evalúa cual es menor. Al final arma un árbol con los caminos más bajos desde el nodo elegido.
- Bellman-Ford: Este es un algoritmo que logra hacer lo mismo que el algoritmo de Disjtrak, solo que este es capaz de manejar los vértices que tienen pesos negativos sin llegar a un bucle infinito. Este método consiste en crear una lista de todos los vértices que contendrá la distancias de estos a la inicial. Todas las distancias se colocan como infinitas excepto la de la inicial y luego se van llenando las distancias con un recorrido, y al mismo tiempo se evalúa si por otro vértice existe una conexión y este camino tiene menor peso al ya establecido
  - Este método maneja los pesos negativos a través de una comparación en la cual la distancia de V es mayor a la distancia de U más la arista entre UV, existe un valor negativo.
- Floyd-Warshall: este método genera un bosque de árboles de caminos más cortos, y lo aplica para cada uno de los vértices. Este algoritmo mediante una matriz de vértices para buscar el camino más corto en cada árbol y así crear el bosque de árboles. Utiliza un método muy parecido al de Disjtrak para evaluar las distancias dentro de los árboles.

### **Arboles de Recubrimiento mínimo**

Los arboles de recubrimiento mínimo son subgrafos provenientes de un grafo conectado y no dirigido que tienen la función de conectar todos los vértices. Estos árboles buscan crear el camino de menor peso entre un vértice y todos los demás mediante la suma mínima de los pesos de las aristas del nodo. Los dos métodos que se utilizan son:

- Prim: Este método toma los vértices adyacentes y compara los pesos. Este método contiene una lista que mantiene la cuenta de vértices que ya han sido descubiertos por el algoritmo y les asigna a todos los vértices un valor infinito y a la inicial de 0, luego revisa los adyacentes y agrega el menor peso, este evalúa después si existe una ruta de menor peso, si la hay reemplaza el actual por el nuevo y repite el proceso con los nuevos nodos descubiertos.
- Kruskal: Este método lista todos los pesos de las aristas de manera ascendente, se agrega la arista más baja y se agrega, luego se toma la siguiente y se verifica que no haya un ciclo, cuando esta condición se cumple lo agrega al árbol. Esto se hace hasta que se recorren todas las aristas y los vértices se encuentran conectados sin la existencia de un ciclo completo.

### **Paso 3: Búsqueda de soluciones creativas**

#### **Para la guía**

Se realizó una lista de atributos para determinar la estructura de datos a usar para implementar la guía en la cual se resumen todas las características que debe cumplir y sus restricciones:

- Un punto donde guardar el objeto habitación.
- Conexión entre habitaciones, las cuales deben permitir ir de una a otra, pero no necesariamente el regreso.
- Habitaciones que no conduzcan a ninguna parte.
- Caminos para llegar a la salida.
- Permitir agregar y borrar conexiones y habitaciones.

Debido a los atributos que la estructura requiere se llegó a la conclusión de que las siguientes alternativas corresponden a una solución para el problema:

1. Grafo simple
2. Multígrafo
3. Pseudografo
4. Grafo dirigido
5. Multígrafo dirigido
6. Árbol n-ario
7. Hash Table

### **Para la recolección de tesoros**

Se realizó una lluvia de ideas para manejar la recolección de tesoros dentro de la guía de la mansión. Cabe resaltar que la funcionalidad de la recolección de tesoros sirve para llevar un registro de los tesoros encontrados empíricamente, no dentro de la aplicación, es decir, la persona ingresa los tesoros que encuentra en la mansión, no los encuentra como tal al momento de la ejecución. Por lo tanto, se tienen las siguientes alternativas para guardar los tesoros:

1. Los tesoros pertenecen a la mansión en una lista enlazada.
2. Los tesoros serían guardados en un HashMap perteneciente a la mansión.
3. Los tesoros serían guardados en una lista enlazada perteneciente a cada habitación.
4. Los tesoros son guardados en un árbol binario de búsqueda.
5. Crear un grafo para mostrar la distribución de los tesoros.
6. Los tesoros deben pertenecer a cada habitación y el museo pertenece a la mansión guardando los datos en una lista enlazada.
7. Utilizar una pila para ir guardando los tesoros encontrados en la habitación.
8. Utilizar una pila para ir guardando los tesoros encontrados en la mansión.

### **Paso 4: Transición de ideas a los diseños preliminares**

#### **Para la guía**

Teniendo en cuenta que las conexiones entre las habitaciones podrían acabar siendo cíclicas, y de esta forma tendría conexiones entre los hijos de una raíz y otra; esto no debe ocurrir en un árbol, de esta manera no cumpliría los requisitos para ser un árbol. Por otro lado, un Hash Table pretende ser un diccionario, el cual puede servir para guardar el índice o clave de la habitación para acceder a ella directamente en memoria, pero no modelaría las conexiones entre habitaciones. Por lo tanto, ambas ideas quedan descartadas.

Las ideas restantes corresponden a tipos de grafos, cuyas características están descritas por la siguiente tabla:

*Tipos de aristas    ¿Admite aristas múltiples?    ¿Admite bucles?*

|                            |              |    |    |
|----------------------------|--------------|----|----|
| <i>Grafo simple</i>        | No dirigidas | No | No |
| <i>Multígrafo</i>          | No dirigidas | Si | No |
| <i>Pseudografo</i>         | No dirigidas | Si | Si |
| <i>Grafo dirigido</i>      | Dirigidas    | No | Si |
| <i>Multígrafo dirigido</i> | Dirigidas    | Si | Si |

### Para la recolección de tesoros

Para efecto de nuestra implementación se llegó a la conclusión de que implementar un grafo para guardar la distribución de los tesoros en la mansión se sale del dominio del problema. Solo necesitamos registrar tesoros teniendo en cuenta su nombre, habitación a la que pertenece y valor, no es necesario tener como tal un esquema de su distribución ni cómo se puede acceder a ellos. Se descartó también la idea del HashMap porque no se necesita utilizar el registro a modo de diccionario, no es pertinente para la solución acceder a ellos directamente, sino listarlos, por ende, tiene un alcance más grande del que tiene nuestra implementación.

Por lo tanto, se decidió explicar más a fondo las ideas restantes:

1. Los tesoros son un atributo de la mansión y se encuentran almacenados en una lista enlazada donde se guardan objetos de tipo Treasure que tienen su habitación a la que pertenece o en su defecto museo y su valor.
2. Los tesoros serían guardados en una lista enlazada perteneciente a cada habitación. Cada habitación tiene su propio atributo List donde registra los tesoros encontrados en dicha habitación.
3. Los tesoros son guardados en un árbol binario de búsqueda. Por lo tanto, realizando un recorrido inorden se puede obtener el listado ordenado de los tesoros por nombre. El árbol le pertenece a la mansión
4. Los tesoros deben pertenecer a cada habitación y el museo pertenece a la mansión guardando los datos en una lista enlazada.
5. Utilizar una pila para ir guardando los tesoros encontrados en la habitación porque así se van almacenando conforme se vayan encontrando tesoros en la habitación. Esto quiere decir que la pila donde se acumulan los tesoros pertenece a la habitación.
6. Utilizar una pila para ir guardando los tesoros encontrados en la mansión porque así se van almacenando conforme se vayan encontrando tesoros en la mansión. Esto quiere decir que la pila donde se acumulan los tesoros pertenece a la mansión.

### Paso 5: Evaluación o selección de la mejor solución (Criterios y selección)

#### Para la guía

Criterio A: Conexión entre habitaciones. La conexión entre habitaciones debe permitir llegar de una a otra, pero no necesariamente de permitir el regreso.

- [3] Permite llegar de una habitación a otra, pero no necesariamente el regreso
- [1] Permite llegar de una habitación a otra y viceversa.

Criterio B: Conexión hacia sí mismo. Permite modelar la situación en que una habitación no tiene salida, por lo tanto, su única arista adyacente es hacia el mismo vértice.

- [3] Admite bucles
- [1] No admite bucles

Criterio C: Aristas múltiples. Para llegar de una habitación a otra, directamente, solo existe un pasillo, por ende, no tiene aristas múltiples.

- [3] No admite aristas múltiples
- [1] Admite aristas múltiples

|                     | Criterio A | Criterio B | Criterio C | Total |
|---------------------|------------|------------|------------|-------|
| Grafo simple        | 1          | 1          | 3          | 5     |
| Multígrafo          | 1          | 1          | 3          | 5     |
| Pseudografo         | 1          | 3          | 1          | 5     |
| Grafo dirigido      | 3          | 3          | 3          | 9     |
| Multígrafo dirigido | 3          | 3          | 1          | 7     |

El tipo de grafo escogido para modelar la guía de la mansión es un **grafo dirigido**, porque cumple con la capacidad de conexión restringida entre habitaciones, modelar una habitación sin salida, y no tiene múltiples conexiones entre dos habitaciones.

### Para la recolección de tesoros

Criterio A: Técnica del experto: Los tesoros pertenecen a quien, según la técnica del experto deben pertenecer.

- [3] Los tesoros pertenecen solo a la habitación
- [2] Los tesoros pertenecen solo a la mansión
- [1] Los tesoros pertenecen solo al museo

Criterio B: Inclusión del museo como recolector de tesoros. La solución plateada tiene en cuenta la funcionalidad de que, al eliminar una habitación, si esta contiene tesoros, estos deben quedar perteneciendo al museo.

- [3] Lo tiene en cuenta
- [1] No lo tiene en cuenta

Criterio C: Facilidad de extracción de datos. Permite extraer los datos de la estructura almacenada sin alterarla.

- [3] Al extraer los datos no se altera la estructura usada
- [1] Al extraer los datos se altera la estructura usada

Criterio D: Lugar al que pertenece el museo. El museo pertenece a la clase respectiva según la técnica del experto.

- [3] El museo pertenece a la mansión
- [2] El museo pertenece a una habitación
- [1] No se toma en cuenta el museo como una estructura

|                      | Criterio A | Criterio B | Criterio C | Criterio D | Total |
|----------------------|------------|------------|------------|------------|-------|
| Mansión con lista    | 2          | 3          | 3          | 1          | 9     |
| Habitación con lista | 3          | 1          | 3          | 1          | 8     |

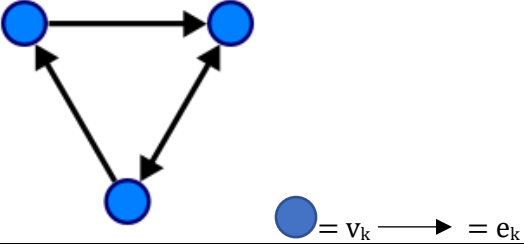


|                     |   |   |   |   |    |
|---------------------|---|---|---|---|----|
| ABB                 | 2 | 3 | 3 | 1 | 9  |
| Museo y habitación  | 3 | 3 | 3 | 3 | 12 |
| Habitación con pila | 3 | 1 | 1 | 1 | 6  |
| Mansión con pila    | 2 | 1 | 1 | 1 | 5  |

Por lo tanto, la manera en que será manejado el registro de tesoros será mediante **listas enlazadas** pertenecientes a cada habitación, donde el valor al guardar es un objeto de tipo `Treasure` que tiene con atributos: nombre, habitación a la que pertenece (solo el nombre) y el valor. Adicionalmente en la mansión se tiene una **lista enlazada** que almacena los tesoros cuando una habitación es eliminada.

## Paso 6: Preparación de informes

### TAD Grafo

| TAD Graph<T>  |  |
|---|--|
| Graph = $\{V = \{v_1, v_2, \dots, v_n\}, E = \{e_1 = (v_{i1}, v_{j1}, w_1), e_2 = (v_{i2}, v_{j2}, w_2), e_m = (v_{im}, v_{jm}, w_m)\}, \text{directed, weighted}\}$  |  |
|   |  |
| Inv: <ol style="list-style-type: none"> <li><math>\forall e_k \in E, v_{ik} \in V \wedge v_{jk} \in V, w_k &gt; 0</math></li> <li><math>\text{directed} = \text{false} \Rightarrow (\forall (a, b) \in E \exists (b, a) \in E, a, b \in V)</math></li> <li><math>\text{weighted} = \text{false} \Rightarrow \forall e_k \in E, w_k = 1</math></li> </ol>  |  |
| Operaciones básicas <ul style="list-style-type: none"> <li>Graph                      Boolean, Boolean <math>\rightarrow</math> Graph</li> <li>addVertex                  Graph x Vertex <math>\rightarrow</math> Graph</li> <li>addEdge                    Graph x Vertex x Vertex <math>\rightarrow</math> Graph</li> <li>addEdge                    Graph x Vertex x Vertex x Double <math>\rightarrow</math> Graph</li> <li>removeVertex              Graph x Vertex <math>\rightarrow</math> Graph</li> <li>removeEdge                Graph x Vertex x Vertex <math>\rightarrow</math> Graph</li> <li>getNeighbors              Graph x Vertex <math>\rightarrow</math> List&lt;Vertex&gt;</li> <li>getNumberOfVertices      Graph <math>\rightarrow</math> Integer</li> <li>getNumberOfEdges        Graph <math>\rightarrow</math> Integer</li> <li>areAdjacent                Graph x Vertex x Vertex <math>\rightarrow</math> Boolean</li> <li>isInGraph                  Graph x T <math>\rightarrow</math> Boolean</li> <li>getEdgeWeight            Graph x Vertex x Vertex <math>\rightarrow</math> Double</li> <li>setEdgeWeight            Graph x Vertex x Vertex x Double <math>\rightarrow</math> Graph</li> <li>getVertices                Graph <math>\rightarrow</math> List&lt;Vertex&gt;</li> <li>searchVertex              Graph x T <math>\rightarrow</math> Vertex</li> <li>isDirected                Graph <math>\rightarrow</math> Boolean</li> <li>isWeighted                Graph <math>\rightarrow</math> Boolean</li> <li>bfs                         Graph x Vertex <math>\rightarrow</math> Graph</li> </ul> |  |

|                 |                                    |
|-----------------|------------------------------------|
| • dfs           | Graph $\rightarrow$ Graph          |
| • dijkstra      | Graph x Vertex $\rightarrow$ Graph |
| • floydwarshall | Graph $\rightarrow$ Double[][]     |
| • prim          | Graph x Vertex $\rightarrow$ Graph |
| • kruskal       | Graph $\rightarrow$ List<Edge>     |
| • getEdges      | Graph $\rightarrow$ List<Edge>     |

## Operaciones

|   |
|---|
| <b>Graph(Boolean directed, Boolean weighted)</b><br>“Crea un nuevo grafo que puede o no ser dirigido o ponderado”<br>Pre:<br>Post: Graph = {V={}, E={}, directed, weighted}   |
| <b>addVertex(Graph g, Vertex v)</b><br>“Inserta un vértice en el grafo”<br>Pre: $v \notin g.V$<br>Post: $v \in g.V$   |
| <b>addEdge(Graph g, Vertex x, Vertex y)</b><br>“Añade una arista de peso 1 que va de x a y. Si el grafo no es dirigido, también la añade de y a x”<br>Pre: $x, y \in g.V$<br>Post: $e = (x, y, 1) \in g.E$ . Si $g.directed = false$ , $e' = (y, x, 1) \in g.E$   |
| <b>addEdge(Graph g, Vertex x, Vertex y, Double w)</b><br>“Añade una arista de peso w que va de x a y. Si el grafo no es dirigido, también la añade de y a x”<br>Pre: $x, y \in g.V$ , $g.weighted = true$ , $w > 0$<br>Post: $e = (x, y, w) \in g.E$ . Si $g.directed = false$ , $e' = (y, x, w) \in g.E$ |
| <b>removeVertex(Graph g, Vertex v)</b><br>“Elimina a v del grafo”<br>Pre: $v \in g.V$<br>Post: $v \notin g.V$ . Todos los vértices que son incidentes con v $\notin g.E$  |
| <b>removeEdge(Graph g, Vertex x, Vertex y)</b><br>“Elimina la arista que va de x a y en el grafo”<br>Pre: $x, y \in g.V$ , $(x, y, *) \in g.E$<br>Post: $e = (x, y, *) \notin g.E$ . Si $g.directed = false$ , $e' = (y, x, *) \notin g.E$  |
| <b>getNeighbors(Graph g, Vertex x)</b><br>“Devuelve los vértices v tal que hay una arista desde x hasta v”<br>Pre: $x \in g.V$<br>Post: vertices = $\{v_1, v_2, \dots, v_n\} : \forall v_i, (x, v_i, *) \in g.E$ .  |
| <b>getNumberOfVertices(Graph g)</b><br>“Devuelve el número de vértices en el grafo”<br>Pre:<br>Post: $n = size(g.V)$  |
| <b>getNumberOfEdges(Graph g)</b><br>“Devuelve el número de aristas en el grafo”<br>Pre:<br>Post: $n = size(g.E)$  |
| <b>areAdjacent(Graph g, Vertex x, Vertex y)</b><br>“Devuelve si hay una arista de x a y”  |

|   |
|---|
| <p>Pre: <math>x, y \in g.V</math><br/> Post: true si y solo si <math>(x, y, *) \in g.E</math>.</p>  |
| <p><b>isInGraph(T val)</b><br/> “Devuelve si hay un vértice con el valor dado en el grafo”<br/> Pre:<br/> Post: true si y solo si <math>\exists x \in g.V : \text{value}(x) = \text{val}</math>.</p>  |
| <p><b>getEdgeWeight(Graph g, Vertex x, Vertex y)</b><br/> “Devuelve el peso de la arista que va de x a y”<br/> Pre: <math>x, y \in g.V, (x, y, *) \in g.E</math><br/> Post: peso = <math>(x, y).w</math></p>  |
| <p><b>setEdgeWeight(Graph g, Vertex x, Vertex y, Double w)</b><br/> “Cambia el peso de la arista que va de x a y”<br/> Pre: <math>x, y \in g.V, (x, y, *) \in g.E, w &gt; 0</math><br/> Post: <math>(x, y, w) \in g.E</math></p>  |
| <p><b>getVertices(Graph g)</b><br/> “Devuelve la lista de vértices del grafo”<br/> Pre:<br/> Post: <math>\{v_1, v_2, \dots, v_n\} = g.V</math></p>  |
| <p><b>searchVertex(T val)</b><br/> “Devuelve, si existe, el vértice con el valor dado en el grafo”<br/> Pre:<br/> Post: <math>x \in g.V : \text{value}(x) = \text{val}</math>. NIL si no existe.</p>  |
| <p><b>isDirected(Graph g)</b><br/> “Devuelve si el grafo es dirigido”<br/> Pre:<br/> Post: g.directed</p>   |
| <p><b>getWeighted(Graph g)</b><br/> “Devuelve si el grafo es ponderado”<br/> Pre:<br/> Post: g.weighted</p>   |
| <p><b>bfs(Graph g, Vertex s)</b><br/> “Realiza el algoritmo Breadth First Search, ajustando información para los vértices del grafo”<br/> Pre: <math>s \in g.V</math><br/> Post: <math>\forall u \in g.V</math>, añade atributos u.pred y u.d, que corresponden a los añadidos por el algoritmo Breadth First Search</p>                                      |
| <p><b>dfs(Graph g)</b><br/> “Realiza el algoritmo Depth First Search, ajustando información para los vértices del grafo”<br/> Pre:<br/> Post: <math>\forall u \in g.V</math>, añade atributos u.pred, u.d y u.f, que corresponden a los añadidos por el algoritmo Depth First Search</p>  |
| <p><b>dijkstra(Graph g, Vertex s)</b><br/> “Realiza el algoritmo de Dijkstra, tomando como vértice inicial a s”<br/> Pre: <math>s \in g.V</math>, g no tiene pesos negativos<br/> Post: <math>\forall u \in g.V</math>, añade atributos u.pred y u.d, que corresponden respectivamente al predecesor y la distancia añadidos por el algoritmo de Dijkstra</p> |
| <p><b>floydwarshall(Graph g)</b><br/> “Realiza el algoritmo de Floyd-Warshall sobre g”</p>  |

Post: Retorna la matriz dist, donde la posición  $[i,j]$  representa la distancia mínima para ir desde el vértice  $v_i$  hasta  $v_j$

“Realiza el algoritmo de Prim tomando como raíz del árbol a  $r$ , ajustando información para los vértices del grafo”

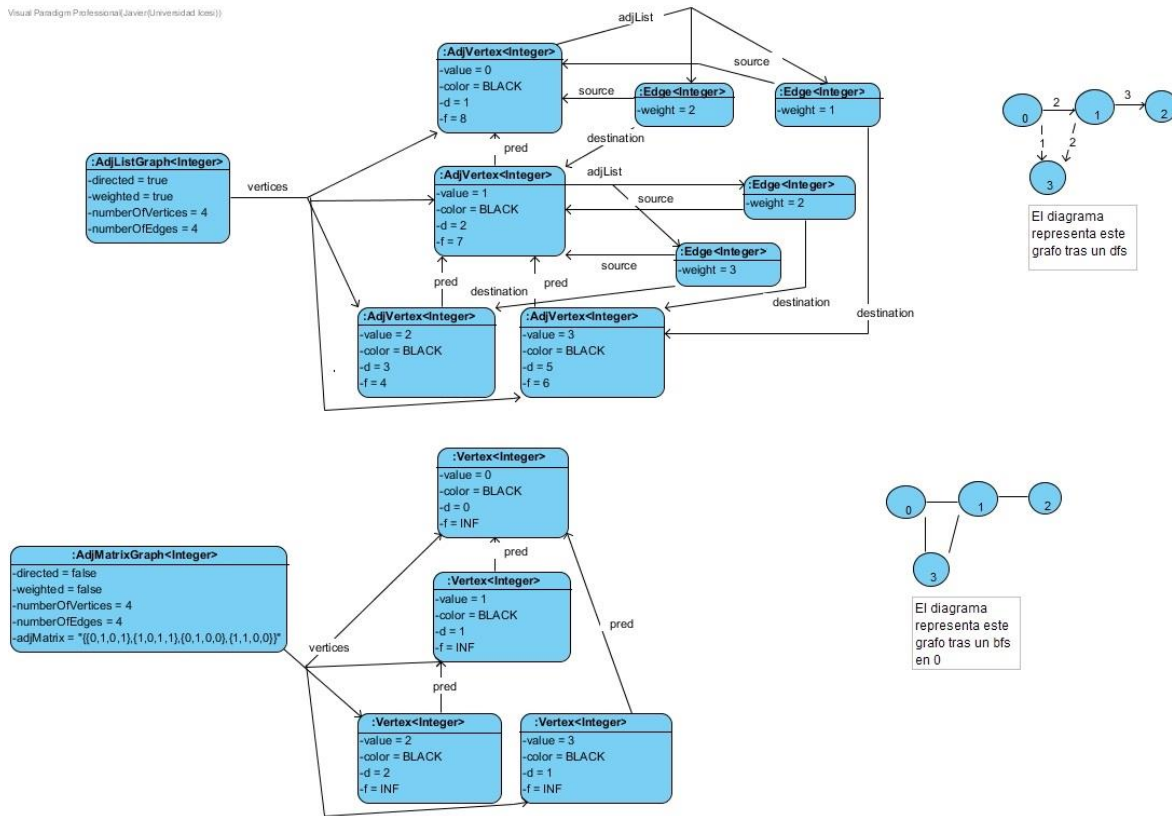
Post:  $\forall u \in g.V$ , añade atributos  $u.pred$  y  $u.d$ , que corresponden respectivamente al predecesor y al key añadidos por el algoritmo de Prim

Pre:

```
getEdges(Graph g)
```

Pre: g no es dirigido

### Diagrama de objetos



El diagrama de objetos se encuentra ubicado en Bibliografía/Diagramas/Diagrama de objetos.jpg

## Diseño de casos de pruebas unitarias

### Diseños de casos de pruebas unitarias TAD

#### Operaciones estructurales

| Prueba 1: Verifica que el método addVertex añada correctamente un vértice al grafo |                                 |   |                        |   |
|--|---------------------------------|---|------------------------|---|
| Clase  | Método                          | Escenario   | Entrada                | Resultado   |
| Graph  | +addVertex(Graph, Vertex): void | Existe un grafo sin vertices                            | Un vértice con valor 1 | El grafo tiene un vértice con valor 1                               |
| Graph  | +addVertex(Graph, Vertex): void | Existe un grafo con los siguientes vértices:<br>1, 2, 4 | Un vértice con valor 3 | El grafo tiene 4 vértices y contiene el vértice 3                   |
| Graph  | +addVertex(Graph, Vertex): void | Existe un grafo con los siguientes vértices:<br>1, 2, 4 | Un vértice con valor 4 | El grafo tiene 3 vértices (1, 2, 4). No se añadió el nuevo vértice. |

| Prueba 2: Verifica que el método addEdge añada correctamente una arista dirigida y ponderada al grafo |        |           |         |           |
|---|--------|-----------|---------|-----------|
| Clase   | Método | Escenario | Entrada | Resultado |

|       |   |   |                      |   |
|-------|---|---|----------------------|---|
| Graph | +addEdge(Graph, Vertex, Vertex, Double): void | Existe un grafo dirigido con los siguientes vértices: 1,2,5,7   | X= 5<br>Y= 7<br>W= 3 | 7 es vértice adyacente de 5 y su arista pesa 3. No existe una arista de 7 a 5.        |
| Graph | +addEdge(Graph, Vertex, Vertex, Double): void | Existe un grafo no dirigido con los siguientes vértices: 1,2,5,7  | X= 5<br>Y= 7<br>W= 3 | 7 es vértice adyacente de 5 y su arista pesa 3. Existe una arista de peso 3 de 7 a 5. |
| Graph | +addEdge(Graph, Vertex, Vertex, Double): void | Existe un grafo dirigido con los siguientes vértices: 1,2,5,7   | X= 5<br>Y= 5<br>W= 8 | Existe una arista de 5 a 5 (bucle) de peso 8.   |
| Graph | +addEdge(Graph, Vertex, Vertex, Double): void | Existe un grafo no dirigido con los siguientes vértices: 1,2,5,7  | X= 5<br>Y= 5<br>W= 8 | Existe una arista de 5 a 5 (bucle) de peso 8.   |
| Graph | +addEdge(Graph, Vertex, Vertex, Double): void | Existe un grafo dirigido con los siguientes vértices: 1,2,5,7<br>Y las siguientes aristas<br>(1, 2, 3)<br>(1, 5, 6)<br>(5, 2, 3)<br>(7, 5, 5) | X= 5<br>Y= 7<br>W= 3 | Los vértices adyacentes a 5 son 2 y 7, y la arista de 5 a 7 pesa 3.                   |

| Prueba 3: Verifica que el método removeVertex elimina correctamente un vértice del grafo, y por ende todas las conexiones a este. |                                    |  |                           |  |
|---|------------------------------------|--|---------------------------|--|
| Clase   | Método                             | Escenario  | Entrada                   | Resultado  |
| Graph   | +removeVertex(Graph, Vertex): void | Existe un grafo dirigido con los siguientes vértices: 1,2,5,7<br>Y las siguientes aristas<br>(1, 2, 3)<br>(1, 5, 6)<br>(5, 2, 3)<br>(7, 5, 5)<br>(5, 7, 3) | El valor del vértice es 2 | El único vértice adyacente de 1 es 5. El único vértice adyacente de 5 es 7 |
| Graph   | +removeVertex(Graph, Vertex): void | El mismo que el anterior   | El valor del vértice es 1 | No existe el vértice con valor 1.  |
| Graph   | +removeVertex(Graph, Vertex): void | El mismo que el anterior   | El valor de vértice es 5  | El único vértice adyacente de 1 es 2.                                      |

|       |                                    |   |                           |  |
|-------|------------------------------------|---|---------------------------|--|
|       |                                    |   |                           | 7 es un vértice aislado.   |
| Graph | +removeVertex(Graph, Vertex): void | Existe un grafo no dirigido con los siguientes vértices 1,2,3,4<br>Y las siguientes aristas<br>(1,2,1)<br>(2,3,1)<br>(3,4,1)<br>(4,1,1) | El valor del vértice es 2 | El único vértice adyacente a 1 es 4. El único vértice adyacente de 3 es 4. |

| Prueba 4: Verifica que el método removeEdge elimina correctamente una artista del grafo |  |   |              |   |
|---|--|---|--------------|---|
| Clase   | Método                                   | Escenario   | Entrada      | Resultado   |
| Graph   | +removeEdge(Graph, Vertex, Vertex): void | Existe un grafo dirigido con los siguientes vértices: 1,2,5,7<br>Y las siguientes aristas<br>(1, 2, 3)<br>(1, 5, 6)<br>(5, 2, 3)<br>(7, 5, 5)<br>(5, 7, 3)<br>(1, 1, 8) | X= 1<br>Y= 2 | Lo únicos vértices adyacentes de 1 son 5, 1.  |
| Graph   | +removeEdge(Graph, Vertex, Vertex): void | El mismo que el anterior  | X=5<br>Y=7   | Existe una arista de 7 a 5, pero no de 5 a 7.                                       |
| Graph   | +removeEdge(Graph, Vertex, Vertex): void | Existe un grafo no dirigido con los siguientes vértices 1,2,3,4<br>Y las siguientes aristas<br>(1,2,1)<br>(2,3,1)<br>(3,4,1)<br>(4,1,1)                                 | X=1<br>Y=2   | No existe una arista que conecte de 1 a 2, ni de 2 a 1. Existen los vértices 1 y 2. |
| Graph   | +removeEdge(Graph, Vertex, Vertex): void | El mismo que el anterior  | X=3<br>Y=4   | No existe una arista que conecte de 3 a 4 ni de 4 a 3. Existen los vértices 3 y 4.  |

| Prueba 5: Verifica que el método searchVertex devuelve el vértice dado su valor si este se encuentra en el grafo. |        |           |         |           |
|---|--------|-----------|---------|-----------|
| Clase   | Método | Escenario | Entrada | Resultado |

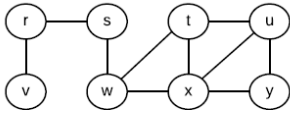

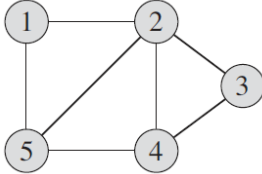
|       |                              |  |        |   |
|-------|------------------------------|--|--------|---|
| Graph | +searchVertex<br>(T): Vertex | Existe un grafo no dirigido con los siguientes vértices<br>1,2,3,4<br>Y las siguientes aristas<br>(1,2,1)<br>(2,3,1)<br>(3,4,1)<br>(4,1,1)                                 | Val= 1 | Retorna un vértice de valor 1 cuyos vértices adyacentes son 2 y 4.    |
| Graph | +searchVertex<br>(T): Vertex | El mismo que el anterior   | Val=5  | Retorna null  |
| Graph | +searchVertex<br>(T): Vertex | Existe un grafo dirigido con los siguientes vértices:<br>1,2,5,7<br>Y las siguientes aristas<br>(1, 2, 3)<br>(1, 5, 6)<br>(5, 2, 3)<br>(7, 5, 5)<br>(5, 7, 3)<br>(1, 1, 8) | Val=2  | Retorna un vértice de valor 2 que no tiene vértices adyacentes.       |
| Graph | +searchVertex<br>(T): Vertex | El mismo que el anterior   | Val=1  | Retorna un vértice con valor 1 cuyos vértices adyacentes son 1, 2, 5. |
| Graph | +searchVertex<br>(T): Vertex | El mismo que el anterior   | Val=8  | Retorna null.   |

| Prueba 6: Verifica que el método areAdjacent retorna true si dos vertices son adyacentes en el grafo. |  |  |            |               |
|---|--|--|------------|---------------|
| Clase   | Método                                       | Escenario  | Entrada    | Resultado     |
| Graph   | +areAdjacent(Graph, Vertex, Vertex): boolean | Existe un grafo no dirigido con los siguientes vértices<br>1,2,3,4<br>Y las siguientes aristas<br>(1,2,1)<br>(2,3,1)<br>(3,4,1)<br>(4,1,1) | X=1<br>Y=2 | Retorna true. |
| Graph   | +areAdjacent(Graph, Vertex, Vertex): boolean | El mismo que el anterior   | X=1<br>Y=3 | Retorna false |
| Graph   | +areAdjacent(Graph, Vertex, Vertex): boolean | Existe un grafo dirigido con los siguientes vértices:  | X=1<br>Y=2 | Retorna true  |

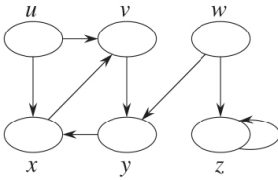
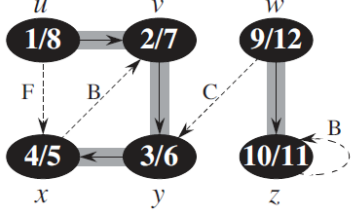


|       |  |  |            |               |
|-------|--|--|------------|---------------|
|       |  | 1,2,5,7<br>Y las siguientes<br>aristas<br>(1, 2, 3)<br>(1, 5, 6)<br>(5, 2, 3)<br>(7, 5, 5)<br>(5, 7, 3)<br>(1, 1, 8) |            |               |
| Graph | +areAdjacent(Graph, Vertex, Vertex): boolean | El mismo que el anterior   | X=2<br>Y=1 | Retorna false |
| Graph | +areAdjacent(Graph, Vertex, Vertex): boolean | El mismo que el anterior   | X=1<br>Y=1 | Retorna true  |

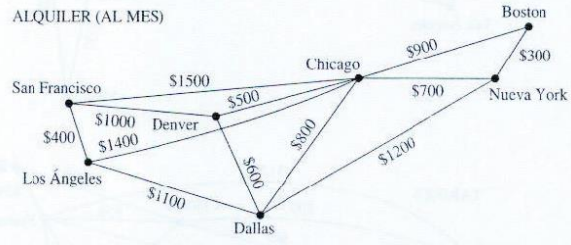
### Algoritmos de recorridos

| Prueba 7: Verifica que el método bfs crea correctamente el árbol bf para encontrar el camino más corto en materia de aristas desde un vértice dado. |                           |  |                           |  |
|---|---------------------------|--|---------------------------|--|
| Clase   | Método                    | Escenario  | Entrada                   | Resultado  |
| Graph   | +bfs(Graph, Vertex): void | Se tiene el siguiente grafo no dirigido:<br> | El valor del vértice es u | El árbol de predecesores queda como se sigue con u como raíz:<br> |
| Graph   | +bfs(Graph, Vertex): void | Se tiene el siguiente grafo dirigido:<br>   | El valor del vértice es 3 | La raíz es 3, su hijo izquierdo es 2 y su hijo derecho es 4. El hijo izquierdo de 2 es 1 y el derecho es 5   |
| Graph   | +bfs(Graph, Vertex): void | Un grafo que tiene 3 vértices: 3, 4, 5   | El valor del índice es 3  | El árbol bf solo está conformado por la raíz 3   |

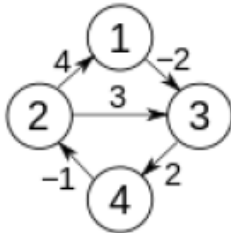
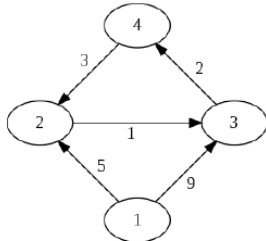
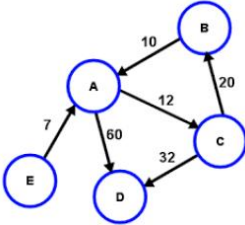
| Prueba 8: Verifica que el método dfs crea un bosque df que provee información acerca de la estructura del grafo |                   |   |         |   |
|---|-------------------|---|---------|---|
| Clase   | Método            | Escenario                               | Entrada | Resultado   |
| Graph   | +dfs(Graph): void | Se tiene el siguiente grafo dirigido no | Ninguna | El siguiente bosque df, donde solo las aristas sombreadas son las pertenecientes a los árboles: |

|       |                      |  |         |  |
|-------|----------------------|--|---------|--|
|       |                      | <p>ponderado</p>  |         |    |
| Graph | +dfs(Graph):<br>void | Se tiene un grafo no dirigido con los siguientes vértices: 1,2,3,4<br>El grafo no posee aristas    | Ninguna | El bosque df está compuesto por 4 árboles donde cada vértice del grafo es la raíz un árbol DF.   |
| Graph | +dfs(Graph):<br>void | Se tiene el siguiente grafo no dirigido:   | Ninguna | El bosque df solo está compuesto por un árbol que en realidad puede verse de la siguiente manera, donde es una secuencia de números y los números en los paréntesis son los timestamps:<br>1(1/10)<br>2(2/9)<br>3(3/8)<br>4(4/7)<br>5(5/6) |

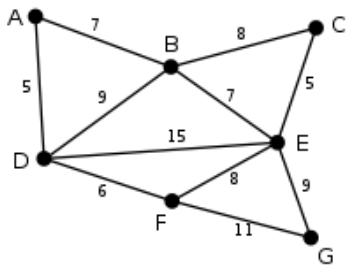
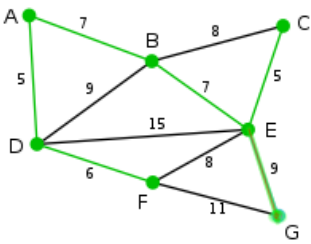
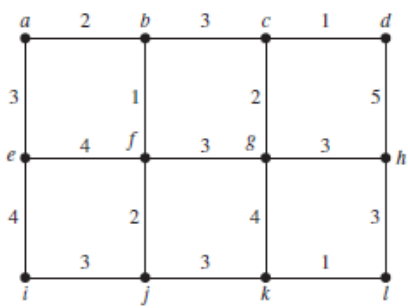
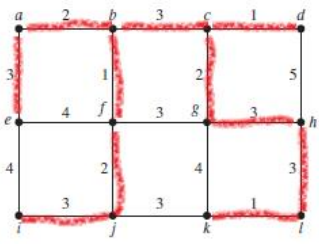
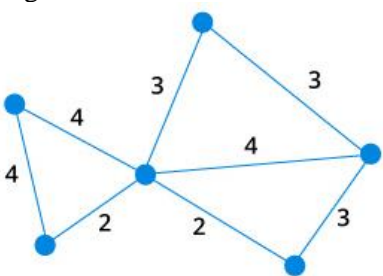
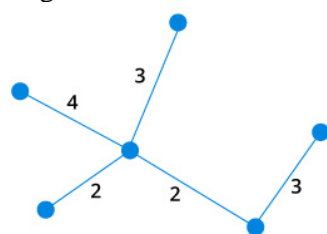
### Algoritmos de camino mínimo

| Prueba 9: Verifica que el método Dijkstra encuentra el camino más corto desde un vértice a otro. |                               |  |                                      |   |
|--|-------------------------------|--|--------------------------------------|---|
| Clase  | Método                        | Escenario  | Entrada                              | Resultado   |
| Graph  | dijkstra(Graph, Vertex): void | <p>ALQUILER (AL MES)</p>  <p>Tomado del libro de Matemática discreta y sus aplicaciones.</p> | El vértice tiene valor Dallas        | El camino mínimo entre Dallas y Boston cuesta 1500        |
| Graph  | dijkstra(Graph, Vertex): void | El mismo que el anterior   | El vértice tiene valor San Francisco | El camino mínimo entre San Francisco y Dallas cuesta 1500 |
| Graph  | dijkstra(Graph, Vertex): void | El mismo que el anterior   | El vértice tiene                     | El camino mínimo  |

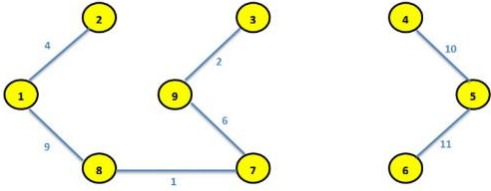
|       |                                   |   |                                      |  |
|-------|-----------------------------------|---|--------------------------------------|--|
|       |                                   |   | valor<br>Chicago                     | entre<br>Chicago<br>y Los<br>Ángeles<br>es la<br>arista que<br>los<br>conecta. |
| Graph | dijkstra(Graph<br>, Vertex): void | Existe el siguiente grafo no dirigido con los<br>siguientes vértices y las siguientes aristas:<br>1,2,3,4<br><br>(1,2,4)<br>(1,3,2) | El<br>vértice<br>tiene el<br>valor 1 | No existe<br>un<br>camino<br>mínimo<br>entre 1 y<br>4.                         |

| Prueba 10: Verifica que el método Floyd-Warshall encuentra el camino mínimo entre todos los vértices. |                                 |  |         |  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|---|---------------------------------|--|---------|--|---------|---|-----|---|---|---|---|---|----|----|----|-----|---|----|-----|----|-----|-----|----|----|----|-----|----|-----|---|-----|-----|-----|---|-----|---|---|----|----|----|----|---|---|--|
| Clase   | Método                          | Escenario  | Entrada | Resultado  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| Graph   | Floydwarshall():<br>double [][] | Se tiene el siguiente grafo:<br>           | Ninguna | El siguiente grafo, representado por una matriz de la siguiente manera:<br><table data-bbox="1029 930 1245 1173"><tr><td colspan="2"><math>k = 4</math></td><td colspan="5"><math>j</math></td></tr><tr><td></td><td></td><td>1</td><td>2</td><td>3</td><td>4</td><td></td></tr><tr><td rowspan="4"><math>i</math></td><td>1</td><td>0</td><td>-1</td><td>-2</td><td>0</td><td></td></tr><tr><td>2</td><td>4</td><td>0</td><td>2</td><td>4</td><td></td></tr><tr><td>3</td><td>5</td><td>1</td><td>0</td><td>2</td><td></td></tr><tr><td>4</td><td>3</td><td>-1</td><td>1</td><td>0</td><td></td></tr></table> | $k = 4$ |   | $j$ |   |   |   |   |   |    | 1  | 2  | 3   | 4 |    | $i$ | 1  | 0   | -1  | -2 | 0  |    | 2   | 4  | 0   | 2 | 4   |     | 3   | 5 | 1   | 0 | 2 |    | 4  | 3  | -1 | 1 | 0 |  |
| $k = 4$   |                                 | $j$  |         |  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|   |                                 | 1  | 2       | 3  | 4       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| $i$   | 1                               | 0  | -1      | -2   | 0       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|   | 2                               | 4  | 0       | 2  | 4       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|   | 3                               | 5  | 1       | 0  | 2       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|   | 4                               | 3  | -1      | 1  | 0       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| Graph   | Floydwarshall():<br>double [][] | Se tiene el siguiente grafo:<br>          | Ninguna | La matriz resultante sería:<br><table data-bbox="1019 1218 1338 1402"><tr><td></td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>0</td><td>5</td><td>6</td><td>8</td></tr><tr><td>2</td><td>inf</td><td>0</td><td>1</td><td>3</td></tr><tr><td>3</td><td>inf</td><td>5</td><td>0</td><td>2</td></tr><tr><td>4</td><td>inf</td><td>3</td><td>4</td><td>0</td></tr></table>   |         | 1 | 2   | 3 | 4 | 1 | 0 | 5 | 6  | 8  | 2  | inf | 0 | 1  | 3   | 3  | inf | 5   | 0  | 2  | 4  | inf | 3  | 4   | 0 |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
|   | 1                               | 2  | 3       | 4  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| 1   | 0                               | 5  | 6       | 8  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| 2   | inf                             | 0  | 1       | 3  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| 3   | inf                             | 5  | 0       | 2  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| 4   | inf                             | 3  | 4       | 0  |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| Graph   | Floydwarshall():<br>double [][] | Se tiene el siguiente grafo dirigido:<br> | Ninguna | La matriz resultante sería:<br><table data-bbox="1019 1551 1382 1770"><tr><td></td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td></tr><tr><td>A</td><td>0</td><td>32</td><td>12</td><td>44</td><td>inf</td></tr><tr><td>B</td><td>10</td><td>0</td><td>22</td><td>54</td><td>inf</td></tr><tr><td>C</td><td>30</td><td>20</td><td>0</td><td>32</td><td>inf</td></tr><tr><td>D</td><td>inf</td><td>inf</td><td>inf</td><td>0</td><td>inf</td></tr><tr><td>E</td><td>7</td><td>39</td><td>19</td><td>51</td><td>0</td></tr></table>  |         | A | B   | C | D | E | A | 0 | 32 | 12 | 44 | inf | B | 10 | 0   | 22 | 54  | inf | C  | 30 | 20 | 0   | 32 | inf | D | inf | inf | inf | 0 | inf | E | 7 | 39 | 19 | 51 | 0  |   |   |  |
|   | A                               | B  | C       | D  | E       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| A   | 0                               | 32   | 12      | 44   | inf     |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| B   | 10                              | 0  | 22      | 54   | inf     |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| C   | 30                              | 20   | 0       | 32   | inf     |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| D   | inf                             | inf  | inf     | 0  | inf     |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| E   | 7                               | 39   | 19      | 51   | 0       |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |
| Graph   | Floydwarshall():<br>double [][] | Un grafo con tres vértices: 1, 2, 3  | Ninguna | La diagonal de la matriz es de 0 y el resto de la matriz es inf.   |         |   |     |   |   |   |   |   |    |    |    |     |   |    |     |    |     |     |    |    |    |     |    |     |   |     |     |     |   |     |   |   |    |    |    |    |   |   |  |

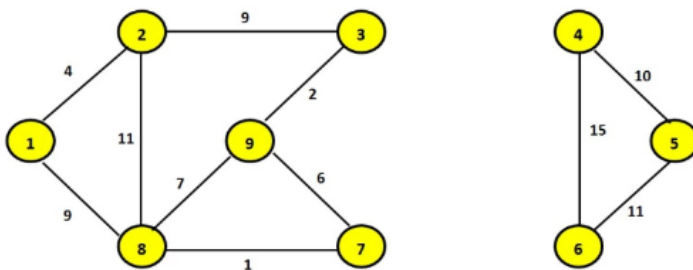
### Algoritmos de árbol de recubrimiento mínimo

| Prueba 11: Verifica que el método Prim crea el árbol generador mínimo de un grafo dado. |                   |  |         |   |
|---|-------------------|--|---------|---|
| Clase   | Método            | Escenario  | Entrada | Resultado   |
| Graph   | +prim()<br>: void | Se tiene el siguiente grafo no dirigido<br>   | Ninguna | El árbol resultante es el siguiente resaltado con verde:<br> |
| Graph   | +prim()<br>: void | Se tiene el siguiente grafo no dirigido<br>  | Ninguna | El árbol generador mínimo es el siguiente<br>               |
| Graph   | +prim()<br>: void | Se tiene el siguiente grafo no dirigido<br> | Ninguna | El árbol generador mínimo es el siguiente<br>              |

| Prueba 11: Verifica que el método Kruskal crea el árbol generador mínimo, o el bosque generador mínimo, de un grafo dado. |                          |   |         |   |
|---|--------------------------|---|---------|---|
| Clase   | Método                   | Escenario                                     | Entrada | Resultado                                     |
| Graph   | +kruskal(Graph):<br>void | El del primer caso de prueba de la prueba 10. | Ninguna | El del primer caso de prueba de la prueba 10. |

|       |                          |   |         |  |
|-------|--------------------------|---|---------|--|
| Graph | +kruskal(Graph):<br>void | El del segundo caso de prueba de la prueba 10.  | Ninguna | El del segundo caso de prueba de la prueba 10.   |
| Graph | +kruskal(Graph):<br>void | El del tercer caso de prueba de la prueba 10.   | Ninguna | El del tercer caso de prueba de la prueba 10.  |
| Graph | +kruskal(Graph):<br>void | Se tiene el siguiente grafo no conexo, descrito en el escenario 1 del final de la página. | Ninguna | Se generan los siguientes árboles  |

Escenario 1:



***Diseños de casos pruebas unitarias solución del problema:***

| Prueba 1: Verifica que el método addRoom añade una habitación si esta no existe en la mansión. |                                  |   |                                    |  |
|--|----------------------------------|---|------------------------------------|--|
| Clase  | Método                           | Escenario   | Entrada                            | Resultado  |
| Mansion  | +addRoom(String , boolean): void | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>- Room, false<br>-Kitchen, false | Name:<br>Bathroom<br>isExit: false | El hashMap de las habitaciones es de tamaño 4<br>El baño pertenece al HashMap mapRooms |
| Mansion  | +addRoom(String , boolean): void | El mismo que el anterior pero implementado bajo matriz de adyacencia.   | Name:<br>Bathroom<br>isExit: false | El hashMap de las habitaciones es de tamaño 4<br>El baño pertenece al HashMap mapRooms |

|         |                                 |                        |                                |   |
|---------|---------------------------------|------------------------|--------------------------------|---|
| Mansion | +addRoom(String, boolean): void | El mismo que el caso 1 | Name: Kitchen<br>isExit: false | Lanza la excepción RoomAlreadyExistsException |
| Mansion | +addRoom(String, boolean): void | El mismo que el caso 2 | Name: Kitchen<br>isExit: false | Lanza la excepción RoomAlreadyExistsException |

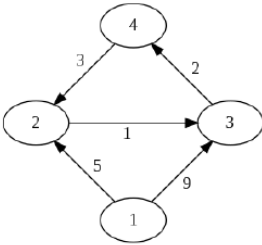
| Prueba 2: Verifica que el método deleteRoom elimina una habitación de la mansión correctamente. |                           |  |                 |  |
|---|---------------------------|--|-----------------|--|
| Clase   | Método                    | Escenario  | Entrada         | Resultado  |
| Mansion   | +deleteRoom(String): void | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>-Room, false<br>-Kitchen, false | Room: Room      | El hashMap de las habitaciones es de tamaño 2. La habitación no pertenece al HashMap mapRooms  |
| Mansion   | +deleteRoom(String): void | El mismo que el anterior pero implementado bajo matriz de adyacencia   | Room: Room      | El hashMap de las habitaciones es de tamaño 2. La habitación no pertenece al HashMap mapRooms. |
| Mansion   | +deleteRoom(String): void | El mismo que el caso 1   | Room: Main exit | Lanza la excepción NotFoundException   |
| Mansion   | +deleteRoom(String): void | El mismo que el caso 2   | Room: Main exit | Lanza la excepción NotFoundException   |
| Mansion   | +deleteRoom(String): void | El mismo que el caso 1   | Room: Bath      | Lanza la excepción NotFoundException   |
| Mansion   | +deleteRoom(String): void | El mismo que el caso 2   | Room: Bath      | Lanza la excepción NotFoundException   |

| Prueba 3: Verifica que el método addCorridor añade un pasillo ponderado entre dos habitaciones. |   |   |   |   |
|---|---|---|---|---|
| Clase   | Método  | Escenario   | Entrada                                   | Resultado   |
| Mansion   | +createCorridor(String, String, double): void | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>-Room, false<br>-Kitchen, false<br>Y las siguientes conexiones<br>(Main exit, Room, 1)<br>(Room, Kitchen, 3) | From: Main exit<br>To: Kitchen<br>Time: 3 | Main exit y Kitchen son adyacentes.<br>Kitchen y Main exit no son adyacentes. |
| Mansion   | +createCorridor(String, String, double): void | El mismo que el anterior pero implementado bajo matriz de adyacencia.   | From: Main exit<br>To: Kitchen            | Main exit y Kitchen son adyacentes.   |

|         |   |                        |  |   |
|---------|---|------------------------|--|---|
|         |   |                        | Time: 3                                  | Kitchen y Main exit no son adyacentes.            |
| Mansion | +createCorridor(String, String, double): void | El mismo que el caso 1 | From: Room<br>To: Kitchen<br>Time: 2     | Lanza la excepción CorridorAlreadyExistsException |
| Mansion | +createCorridor(String, String, double): void | El mismo que el caso 2 | From: Room<br>To: Kitchen<br>Time: 2     | Lanza la excepción CorridorAlreadyExistsException |
| Mansion | +createCorridor(String, String, double): void | El mismo que el caso 1 | From: Bathroom<br>To: Kitchen<br>Time: 2 | Lanza la excepción NotFoundException              |
| Mansion | +createCorridor(String, String, double): void | El mismo que el caso 2 | From: Kitchen<br>To: Bathroom<br>Time: 2 | Lanza la excepción NotFoundException              |

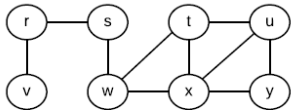
| Prueba 4: Verifica que el método deleteCorridor elimina un pasillo entre dos habitaciones si este existe. |                                       |  |                               |                                      |
|---|---------------------------------------|--|-------------------------------|--------------------------------------|
| Clase   | Método                                | Escenario  | Entrada                       | Resultado                            |
| Mansion   | +deleteCorridor(String, String): void | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>-Room, false<br>-Kitchen, false<br>Y las siguientes conexiones<br>(Main exit, Room, 1)<br>(Room, Cocina, 3) | From: Main exit<br>To: Room   | Main exit y Room no son adyacentes.  |
| Mansion   | +deleteCorridor(String, String): void | El mismo que el anterior pero implementado bajo matriz de adyacencia.  | From: Main exit<br>To: Room   | Main exit y Room no son adyacentes   |
| Mansion   | +deleteCorridor(String, String): void | El mismo que el caso 1   | From: Kitchen<br>To: Room     | Lanza la excepción NotFoundException |
| Mansion   | +deleteCorridor(String, String): void | El mismo que el caso 2   | From: Kitchen<br>To: Room     | Lanza la excepción NotFoundException |
| Mansion   | +deleteCorridor(String, String): void | El mismo que el caso 1   | From: Bathroom<br>To: Kitchen | Lanza la excepción NotFoundException |

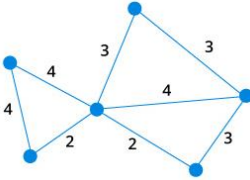
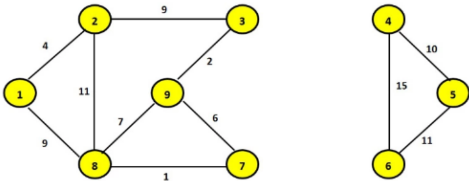
|         |                                       |                        |                               |                                      |
|---------|---------------------------------------|------------------------|-------------------------------|--------------------------------------|
| Mansion | +deleteCorridor(String, String): void | El mismo que el caso 2 | From: Bathroom<br>To: Kitchen | Lanza la excepción NotFoundException |
| Mansion | +deleteCorridor(String, String): void | El mismo que el caso 1 | From: Kitchen<br>To: Bathroom | Lanza la excepción NotFoundException |
| Mansion | +deleteCorridor(String, String): void | El mismo que el caso 2 | From: Kitchen<br>To: Bathroom | Lanza la excepción NotFoundException |

| Prueba 5: Verifica que el método shortestWayOut encuentra el camino más corto entre una habitación y la salida si esta tiene salida. |                                      |   |         |   |
|--|--------------------------------------|---|---------|---|
| Clase  | Método                               | Escenario   | Entrada | Resultado   |
| Mansion  | +shortestWayOut (String): List<Room> | <p>Se tiene una mansión implementada bajo matriz de adyacencia que contiene las siguientes habitaciones:</p>  <pre> graph TD     1((1)) -- 5 --&gt; 2((2))     1((1)) -- 9 --&gt; 3((3))     2((2)) -- 3 --&gt; 4((4))     3((3)) -- 2 --&gt; 4((4))     2((2)) -- 1 --&gt; 3((3)) </pre> <p>Donde sus nombres son números y 3 tiene como nombre MainExit y es la salida.</p> | Room: 2 | El camino más corto es -2-Main exit (exit)<br>El peso del camino es 1   |
| Mansion  | +shortestWayOut (String): List<Room> | El mismo que el anterior pero implementado bajo matriz de adyacencia.   | Room: 2 | El camino más corto es -2-Main exit (exit)<br>El peso del camino es 1   |
| Mansion  | +shortestWayOut (String): List<Room> | El mismo que en el caso 1   | Room: 1 | El camino más corto es -1-2-Main exit (exit)<br>El peso del camino es 6 |
| Mansion  | +shortestWayOut (String): List<Room> | El mismo que en el caso 2   | Room: 1 | El camino más corto es -1-2-Main exit (exit)<br>El peso del camino es 6 |
| Mansion  | +shortestWayOut (String): List<Room> | El mismo que en el caso 1<br>Pero la salida principal es 1  | Room: 4 | La distancia es infinita, no hay camino de salida                       |



|         |  |  |         |   |
|---------|--|--|---------|---|
|         |  |  |         |   |
| Mansion | +shortestWayOut<br>(String):<br>List<Room> | El mismo que en el caso 2<br>Pero la salida principal es 1 | Room: 4 | La distancia es infinita, no hay camino de salida |
| Mansion | +shortestWayOut<br>(String):<br>List<Room> | El mismo que en el caso 1                                  | Room: 6 | Lanza la excepción<br>NotFoundException           |
| Mansion | +shortestWayOut<br>(String):<br>List<Room> | El mismo que en el caso 2                                  | Room: 6 | Lanza la excepción<br>NotFoundException           |

| Prueba 6: Verifica que el método shortestPath encuentra el camino más corto entre una habitación y otra. |   |  |                  |  |
|--|---|--|------------------|--|
| Clase  | Método                                      | Escenario  | Entrada          | Resultado  |
| Mansion  | +shortesPath(String,<br>String): List<Room> | Se tiene el siguiente grafo implementado bajo lista de adyacencia:<br> <p>Donde las aristas son dirigidas en sentido derecha-izquierda.</p> | From: u<br>To: w | El camino más corto es u-t-w y toma 2 habitaciones |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que el anterior pero implementado bajo matriz de adyacencia.  | From: u<br>To: w | El camino más corto es u-t-w y toma 2 habitaciones |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 1 Pero se le añade una habitación a conectada de a a r   | From: u<br>To: a | No existe un camino entre esas dos habitaciones    |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 2 Pero se le añade una habitación a conectada de a a r   | From: u<br>To: a | No existe un camino entre esas dos habitaciones    |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 1  | From: u<br>To: a | Lanza la excepción<br>NotFoundException            |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 2  | From: u<br>To: a | Lanza la excepción<br>NotFoundException            |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 1  | From: a<br>To: u | Lanza la excepción<br>NotFoundException            |
| Mansion  | +shortesPath(String,<br>String): List<Room> | El mismo que en el caso 2  | From: a<br>To: u | Lanza la excepción<br>NotFoundException            |

| Prueba 7: Verifica que el método announceClosure transmite el mensaje de salida en el menor peso posible |                            |  |         |            |
|--|----------------------------|--|---------|------------|
| Clase  | Método                     | Escenario  | Entrada | Resultado  |
| Mansion  | +announceClosure(): double | Se tiene la siguiente mansión implementada bajo lista de adyacencia<br>       | Ninguna | Retorna 14 |
| Mansion  | +announceClosure(): double | El mismo que el anterior pero implementado bajo matriz de adyacencia   | Ninguna | Retorna 14 |
| Mansion  | +announceClosure(): double | Se tiene la siguiente mansión implementada bajo lista de adyacencia<br>4<br> | Ninguna | Retorna 43 |
| Mansion  | +announceClosure(): double | El mismo que el anterior pero implementado bajo matriz de adyacencia   | Ninguna | Retorna 43 |

| Prueba 8: Verifica que el método addTreasure añade correctamente un tesoro a la mansión. |  |  |   |   |
|--|--|--|---|---|
| Clase  | Método                                     | Escenario  | Entrada                                       | Resultado   |
| Mansion  | +addTreasure(String, String, double): void | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>-Room, false<br>-Kitchen, false | Room: Kitchen<br>Name: Treas1<br>Value: 30000 | La cocina tiene un Tesoro en su lista y es un Treas1 de valor 30000   |
| Mansion  | +shortesPath(String, String): List<Room>   | El mismo que el anterior pero implementado bajo matriz de adyacencia   | Room: Kitchen<br>Name: Treas1<br>Value: 30000 | La cocina tiene un Tesoro en su lista y es un Treas1 de valor 30000   |
| Mansion  | +addTreasure(String, String, double): void | El mismo que en el caso 1 pero tiene un Tesoro en su lista y es un Treas1 de valor 300000 que pertenece a Kitchen                                  | Room: Kitchen<br>Name: Knife<br>Value= 30000  | La cocina tiene dos Tesoros en su lista donde el primero es un Treas1 de valor 300000 y el segundo es un Knife de valor 30000 |

|         |  |  |  |   |
|---------|--|--|--|---|
| Mansion | +addTreasure(String, String, double): void | El mismo que el anterior pero implementado bajo matriz de adyacencia | Room: Kitchen<br>Name: Knife<br>Value= 30000 | La cocina tiene dos Tesoros en su lista donde el primero es un Treas1 de valor 300000 y el segundo es un Knife de valor 30000 |
| Mansion | +addTreasure(String, String, double): void | El mismo que en el caso 1  | Room: Bathroom<br>Name: Ring<br>Value:40000  | Lanza la excepción NotFoundException  |
| Mansion | +addTreasure(String, String, double): void | El mismo que en el caso 2  | Room: Bathroom<br>Name: Ring<br>Value:40000  | Lanza la excepción NotFoundException  |

| Prueba 8: Verifica que el método getTreasures añade correctamente un tesoro a la mansión. |                               |  |         |   |
|---|-------------------------------|--|---------|---|
| Clase   | Método                        | Escenario  | Entrada | Resultado   |
| Mansion   | +getTreasures(): List<String> | Una mansión implementada bajo lista de adyacencia que contiene las siguientes habitaciones:<br>-Main exit, true<br>-Room, false<br>-Kitchen, false<br>Con los siguientes tesoros<br>-Kitchen, Treas1, 30000<br>-Room, Ring, 20000<br>-Kitchen, Knife 40000 | Ninguna | La lista tiene tamaño 3, la lista de los tesoros del museo es de tamaño 0 |
| Mansion   | +getTreasures(): List<String> | El mismo que el anterior   | Ninguna | La lista tiene tamaño 3, la lista de los tesoros del museo es de tamaño 0 |
| Mansion   | +getTreasures(): List<String> | El mismo que en el caso 1 pero se elimina luego de la implementación la Room   | Ninguna | La lista tiene tamaño 3, la lista de los tesoros del museo es de tamaño 1 |
| Mansion   | +getTreasures(): List<String> | El mismo que el anterior pero implementado bajo matriz de adyacencia   | Ninguna | La lista tiene tamaño 3, la lista de tesoros del museo es de tamaño 1     |
| Mansion   | +getTreasures(): List<String> | El mismo que el caso 1 pero no se añade el tesoro de la Room y se elimina la Room  | Ninguna | La lista tiene tamaño 2, la lista de tesoros del museo es de tamaño 0     |
| Mansion   | +getTreasures(): List<String> | El mismo que el caso 1 pero no se añade el tesoro de la Room y se elimina la Room  | Ninguna | La lista tiene tamaño 2, la lista de tesoros del museo es de tamaño 0     |

## Bibliografía

Anónimo. (Desconocido). “Algoritmo de Prim”. Tomado de: <https://sites.google.com/site/complejidadalgoritmicaes/prim>

Anonimo. (Desconocido). “Bellman-Ford Algorithm | DP-23”. Tomado de: <https://www.geeksforgeeks.org/bellman-ford-algorithm-dp-23/>

Anonimo. (Desconocido). “Breadth First Search or BFS for a Graph”. Tomado de: <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

Anonimo. (Desconocido). “Depth First Search or DFS for a Graph”. Tomado de: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

Anonimo. (Desconocido). “Dijkstra’s shortest path algorithm | Greedy Algo-7”. Tomado de: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/>

Anonimo. (Desconocido). “Floyd Warshall Algorithm | DP-16”. Tomado de: <https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/>

Anonimo. (Desconocido). “Graph and its representations”. Tomado de: <https://www.geeksforgeeks.org/graph-and-its-representations/>

Anonimo. (Desconocido). “Graph Data Structure and Algorithms”. Tomado de: <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

Anonimo. (Desconocido). “Kruska’s Minimum Spanning Tree Algorithm | Greedy Algo-2”. Tomado de: <https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

Anónimo. (Mayo 09 del 2018). “La tenebrosa historia de la casa Winchester (+Video)” Tomado de: <https://culturizando.com/la-tenebrosa-historia-de-la-casa/>

Anonimo.(Desconocido). “Prim’s Algorithm”. Tomado de: <https://www.programiz.com/dsa/prim-algorithm>

Anonimo.(Desconocido). “Prim’s Minimum Spanning Tree(MST) | Greedy Algo-5”. Tomado de: <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>

Anónimo. (Desconocido). “Sarah’s Story”. Tomado de: <https://winchestermysteryhouse.com/sarahs-story/>

Anónimo. (Desconocido). “Teoría de Grafos”. Tomado de: [http://www.ma.uva.es/~antonio/Industriales/Apuntes\\_05-06/LabM/B\\_T-Grafos.pdf](http://www.ma.uva.es/~antonio/Industriales/Apuntes_05-06/LabM/B_T-Grafos.pdf)

Miller, B. (2014, julio 2) “El tipo abstracto de datos grafo”. Tomado de: <http://interactivepython.org/runestone/static/pythoned/Graphs/ElTipoAbstractoDeDatosGrafo.html>

Pérez, L. (Desconocido). “¿Visitarías una mansión embrujada de 2 hectáreas? Conoce la historia de la Mansión Winchester”. Tomado de: <https://www.vix.com/es/mundo/190470/visitarias-una-mansion-embrujada-de-2-hectareas-conoce-la-historia-de-la-mansion-winchester>