Proyecto Algoritmos y Estructuras de Datos

Vera Usman Juan David (A00293816)

Introducción:

Eres miembro del Ferrocarril subterráneo y tendrás que guiar a un grupo de esclavos a su libertad. El camino estará lleno de peligros y deberás ser cauteloso para no perder a ninguno de tu grupo. Tu objetivo es llevarlos a un estado seguro, libre del control del sur, pasando por puntos seguros o de descanso.

Definición del problema:

El jugador empezará con un número determinado de esclavos a ayudar a cruzar las fronteras con los estados del sur, tendrán que desplazarse por diferentes caminos para llegar a puntos seguros o de descanso y evitar ser capturados o asesinados.

Si el jugador pasa por una zona de peligro perderá esclavos, si pierde a todos los esclavos y el también muere perderá el juego. La partida podrá contar como fallida si el jugador no logra terminar con un número mínimo de esclavos rescatados.

El juego finalizara cuando el jugador haya cruzado la frontera.

Requerimientos

Requerimiento funcional 1	
Nombre:	Cargar Mapa
Descripción:	Carga una matriz de adyacencia de un mapa a jugar.
Entrada:	El nivel que el jugador desee jugar
Salida:	El juego iniciara en el nivel elegido

Requerimiento funcional 2	
Nombre:	Calcular Dificultad
Descripción:	Se calculará la dificultad del mapa desde la posición de inicio elegida por el jugador
Entrada:	Índice del nodo
Salida:	

Requerimiento funcional 3	
Nombre:	Asignar Dificultad
Descripción:	Asigna las dificultades a las aristas
Entrada:	Una lista de las dificultades entre los vértices

Salida:	

Requerimiento funcional 4	
Nombre:	Puntaje mínimo
Descripción:	Calcula el puntaje que el jugador debe batir
Entrada:	Vértice desde donde inicia el jugador
Salida:	Puntaje mínimo

Requerimiento funcional 5	
Nombre:	Mover jugador
Descripción:	Permite mover al jugador de un vértice a otro
Entrada:	Posición final del jugador
Salida:	Valor booleano que indica si se puedo mover o no

Requerimiento funcional 6	
Nombre:	Eliminar esclavos
Descripción:	Resta la cantidad de esclavos con la que se desplaza el jugador
Entrada:	Número de esclavos a diezmar
Salida:	

Requerimiento no funcional 1	
Nombre:	Cambio de modelo
Descripción:	Cambia la representación del grafo que modela el mapa, pasa de ser una matriz ponderada a una lista de adyacencia que guarda los pesos.
Entrada:	Una instrucción para el cambio
Salida:	

Recopilación de información necesaria

- Nivel:

Un nivel, mapa, área, etapa, mundo, rack, tablero, pantalla o zona en un videojuego es el espacio total disponible para el jugador a la hora de completar un objetivo específico. Los niveles de los videojuegos suelen tener una dificultad que aumenta progresivamente para atraer la atención de jugadores de todos los tipos. Cada nivel muestra nuevo contenido y desafíos para mantener alto el interés de los jugadores.

A menudo un nivel en modos de juegos basados en objetivos es en el cual la mayor parte de la acción toma lugar con todos metidos en un área o escenario, y la meta más importante del jugador es simplemente completar todos los objetivos en progresión central.

Ref1.

- Grafos:

En matemáticas y ciencias de la computación, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. Son objeto de estudio de la teoría de grafos.

Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas).

Un **grafo no dirigido** es un grafo $G = (V, E)$ donde:

- $V \neq \emptyset$
- $E \subseteq \{x \in \mathcal{P}(V) : |x| = 2\}$ es un conjunto de *pares no ordenados* de elementos de V.

Un **par no ordenado** es un conjunto de la forma $\{a, b\}$, de manera que $\{a, b\} = \{b, a\}$.

Ilustración 1: Extracción de Wikipedia, explicación de un grafo no dirigido.

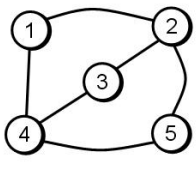
Ilustración 2: representación de un grafo no dirigido

Adyacencia: dos aristas son adyacentes si tienen un vértice en común, y dos vértices son adyacentes si una arista los une.

Ponderación: corresponde a una función que a cada arista le asocia un valor (costo, peso, longitud, etc.), para aumentar la expresividad del modelo.

Matriz de adyacencia (**MA**): Se utiliza una matriz de tamaño n × n donde las filas y las columnas hacen referencia a los vértices para almacenar en cada casilla un valor que indica la adyacencia o bien la ponderación (convirtiendo se así en una matriz de pesos) entre cada par de vértices del grafo. Comúnmente si una celda **MA**[i, j] almacena un valor infinito o igual a cero significa que no existe arista entre esos vértices.

Lista de adyacencia (**LA**): Se utiliza un vector de tamaño n (un elemento por cada vértice) donde **LA**[i] almacena la referencia a una lista de los vértices adyacentes a i. En una red esta lista almacenará también la longitud de la arista que va desde i al vértice adyacente.



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Ilustración 3: Matriz de adyacencia

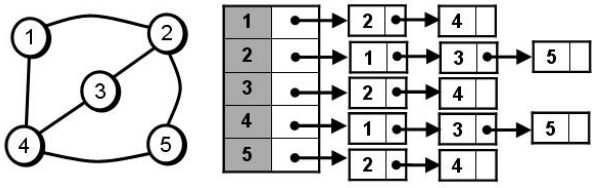


Ilustración 4: Lista de adyacencia

Ref2.

- Algoritmos empleados en grafos:

El recorrido de un grafo significa visitar cada vértice y arista exactamente una vez en un orden bien definido. Al usar ciertos algoritmos de grafos, debe asegurarse de que cada vértice del grafo se visite exactamente una vez. El orden en que se visitan los vértices es importante y puede depender del algoritmo o pregunta que esté resolviendo.

Breadth First Search (BFS)

BFS es un algoritmo de recorrido en el que debe comenzar a recorrer desde un vértice seleccionado (origen o nodo de inicio) y recorrer el grafo en capas para explorar los vértices vecinos (nodos que están directamente conectados al nodo de origen). Luego debe moverse hacia los vértices vecinos del siguiente nivel.

Ref6.

Depth First Search (DFS)

El algoritmo DFS es un algoritmo recursivo que utiliza la idea de retroceder. Implica búsquedas exhaustivas de todos los vértices avanzando, si es posible, de lo contrario, retrocediendo.

Aquí, la palabra retroceso significa que cuando avanza y no hay más nodos a lo largo de la ruta actual, retrocede en la misma ruta para encontrar vértices para atravesar. Se visitarán todos los vértices en la ruta actual hasta que se hayan recorrido todos los vértices no visitados, después de lo cual se seleccionará la siguiente ruta.

Ref8.

El problema de la ruta más corta se trata de encontrar una ruta entre vértices en un gráfico de modo que la suma total de los pesos de los bordes sea mínima.

Algoritmo de Dijkstra

El algoritmo de Dijkstra tiene muchas variantes, pero la más común es encontrar las rutas más cortas desde el vértice de origen a todos los demás vértices en el grafo.

Floyd – Algoritmo de Warshall

El algoritmo de Floyd-Warshall se usa para encontrar los caminos más cortos entre todos los pares de vértices en un grafo, donde cada borde en el grafo tiene un peso que es positivo o negativo.

Ref7.

Árbol de expansión mínima

El costo del árbol de expansión es la suma de los pesos de todas las aristas del árbol. Puede haber muchos árboles de expansión. El árbol de expansión mínimo es el árbol de expansión donde el costo es mínimo entre todos los árboles de expansión. También puede haber muchos árboles de expansión mínima. Los algoritmos más comunes que generan un árbol de expansión mínima son Kruskal y Prim.

Ref9.

Búsqueda de soluciones creativas

Para la búsqueda de soluciones creativas me base en una lluvia de ideas para analizar y proponer todo lo posible y dar solución a los diferentes aspectos que tiene el problema.

Una vez propuestas las ideas las analice para ver si eran buenas, posibles y las agregaba a una lista que sería analizada más a fondo después.

La lista de ideas que salieron fue la siguiente:

Para el mapa:

1. Representar el mapa como un grafo ponderado no direccionado, en donde cada vértice un punto seguro o de descanso y cada arista el camino con un numero de esclavos a perder para ir de un punto seguro a otro.

2. Representar el mapa como un grafo ponderado direccionado, en donde cada vértice un punto seguro o de descanso y cada arista el camino con un numero de esclavos a perder para ir de un punto seguro a otro.

3. Representar el mapa como un grafo ponderado no direccionado, en donde cada vértice un punto seguro o de descanso y cada arista el camino para ir de un punto seguro a otro.

4. Mostrar un árbol de recubrimiento mínimo usando el algoritmo de Prim para ayudar a el jugador a llegar al destino perdiendo el menor número de esclavos posibles

5. Mostrar un árbol de recubrimiento mínimo usando el algoritmo de Kruskal para ayudar a el jugador a llegar al destino perdiendo el menor número de esclavos posibles

Cálculo de niveles:

6. Calcular los niveles de peligro mediante BFS para llegar de un punto seguro a otro.

7. Calcular los niveles de peligro mediante DFS para llegar de un punto seguro a otro.

Mecánica del puntaje:

8. Se propuso usar el algoritmo de Dijkstra para generar un puntaje el cual el jugador deberá superar tomando el camino más corto desde donde el jugador inicio hasta la frontera restando este valor por un número.

9. Se propuso usar el algoritmo de Floyd-Warshall para generar un puntaje el cual el jugador deberá superar.

Transición de la formulación de ideas a los diseños preliminares

Las siguientes propuestas fueron las que se descartaron por distintas razones, principalmente por complicar la solución o no moldearse a lo que se busca:

Para el mapa:

- Se descarta la idea numero 2 porque no es necesario el uso de un grafo dirigido, el jugador debería poder elegir a cuál nodo moverse que sea adyacente al que el este parado.
- Se descarta la idea 3 porque necesito los valores que le van a restar puntos al jugador para poder dar un puntaje mínimo
- Las ideas 4 y 5 se descartan por que le estaría mostrando posibles caminos que debería tomar y el juego perdería dificultad.

Cálculo de niveles:

- Se descarta la idea 7 debido a que el grafo tendrá muchos ciclos y muchas aristas por lo que el calculo de nivel seria engorroso y podría no tener mucho sentido. La idea es que el nivel aumente progresivamente y se tenga más control sobre él.

Mecánica del puntaje:

- Se descarta Floyd-Warshall debido que hace cálculos que no requiero, dependiendo de donde el jugador arranque deberé calcular niveles y de ahí tendrá que llegar a un punto en especifico perdiendo la menor cantidad de esclavos posibles. El resultado que busco me lo da Dijkstra.

Las ideas restantes fueron las elegidas para el desarrollo del juego, ya de una u otra manera, permitirán el debido funcionamiento de este, además de que facilitan la implementación del programa.

TAD del grafo:

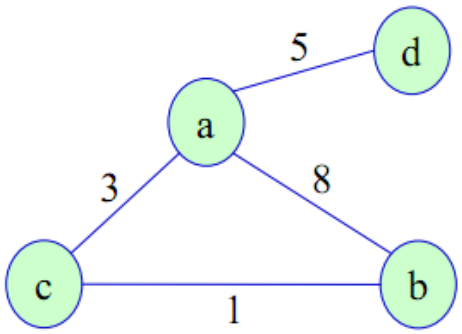
TAD Mapa

Lista de Adyacencia con pesos

Vértices	Adyacentes
a	(b,5), (c,3), (d,8)
b	(a,5), (c,1)
c	(a,3), (b,1)
d	(a,5)

Matriz de Pesos

	a	b	c	d
a	0	8	3	5
b	8	0	1	∞
c	3	1	0	∞
d	5	∞	∞	0



inv:

- El grafo representa el mapa en el cual se desarrollará el juego.
- Los vértices simbolizan lugares seguros o de descanso
- Las aristas representan el peligro de desplazarse de una zona segura a otra
- Una arista con valor infinito en la matriz de pesos representa no conexión entre nodos

Operaciones Primitivas:

Mapa:	Mapa X int[][] X boolean	=> Mapa
calcularDificultad:	Mapa X int	=> Mapa
asignarDificultad:	Mapa X int[]	=> Mapa
puntajeMinimo:	Mapa X int	=> int
estaConectado:	Mapa X int X int	=> int
cambiarModelo:	Mapa	=> Mapa

Mapa(int[][] nivel, boolean tipoModelo)

"carga un mapa que el jugador va a jugar, crea los dos tipos de representaciones del modelo del grafo dado un modelo base"

{pre: el jugador elegio un mapa para jugar}
{post: deja como asignado la lista de adyacencia con pesos o matriz de pesos para jugar y realizar las operaciones pertinentes}

calcularDificultad(int verticeInicial)

"hace un bfs para conocer el nivel de los vértices y calcular la dificultad de llegar a un vértice"

{pre: el vértice elegido existe y ha sido el cual el jugador decidió empezar la partida}
{post: se tendrá una lista con los niveles de dificultad asociados a los nodos, se procede a asignar dificultad}

asignarDificultad(int[] dificultadDelosVertices)

"asigna valores de dificultad a las aristas dentro de un rango asociado a la dificultad de llegar de un vértice a otro"

{pre: los niveles están calculados}
{post: la matriz de pesos y lista de adyacencia con pesos tendrán los valores de dificultad correspondientes}

puntajeMinimo(int verticeInicial)

"Sera la diferencia entre 100 (el número de esclavos a pasar) y la suma del valor del camino más corto y el piso del 30% del valor del camino más corto desde donde inicio el jugador hasta la frontera mediante el uso de Dijkstra"

{pre: el vértice elegido existe y ha sido el cual el jugador decidió empezar la partida } {post: se dará el valor del camino mínimo para llegar a la frontera desde donde inicio el jugador}
estaConectado(int a, int b) "se revisará si “b” es adyacente a “a” y se retornara el valor de la arista que los une o -1 si no están unidos" {pre: los vértices “a y b” existen en el grafo, el jugador se encuentra en “a” y se desea desplazar a “b”} {post: el valor de la arista que los conecta o -1}
cambiarmodelo() "Cambia la representación del grafo con el que se está trabajando, se pasa a usar una u otra dependiendo de quién estaba en uso actual" {pre: había un modelo (lista de adyacencia con pesos o matriz de pesos) activo} {post: se pasaran a hacer cálculos y consultas con el nuevo modelo asignado}

Evaluación y Selección de la Mejor Solución

Como las propuestas no viables ya fueron descartadas por diferentes motivos, quedaron unas pocas que son las necesarias para el desarrollo del juego, por lo tanto, al ser cada una únicas en lo que desarrollan, este paso se va a saltar, ya que no hay necesidad de ver cuál es mejor que otra según un criterio. Las siguientes son las que sobraron y las que se van a usar:

Para el mapa:

- Representar el mapa como un grafo no direccionado ponderado, en donde cada vértice sea un punto seguro o de descanso y cada arista el camino para ir de un punto seguro a otro. El jugador se desplazará por los caminos sin conocer el peligro.

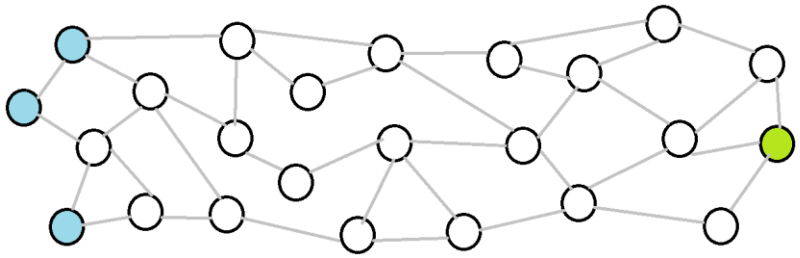


Ilustración 5 Representa como se vería el Nivel a ojos del jugador.

Los puntos azules indican los posibles puntos de partida, el verde el punto de destino.

Para las pantallas:

- Calcular los niveles de peligro mediante BFS para llegar de un punto seguro a otro. Esto hará que el nivel de dificultad aumente gradualmente y fluya de una manera más justa para el

jugador. Se tomarán solo 3 niveles de dificultad cuando se pase el nivel de dificultad 3 se reiniciará a 1 y volverá a aumentar.

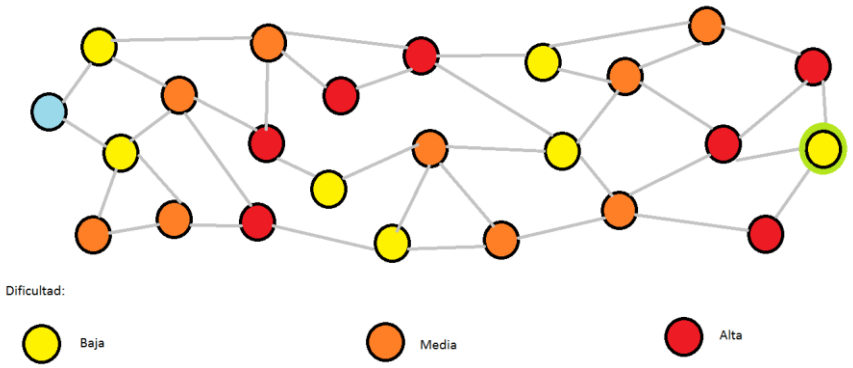


Ilustración 6 Muestra cómo se asignarían los niveles gracias al BFS desde el punto que el jugador eligió.

Para el juego:

- Se propuso usar el algoritmo de Dijkstra para generar un puntaje el cual el jugador deberá superar tomando el camino más corto desde donde el jugador inicio hasta la frontera restando este valor por un número.
  Como conozco el punto de destino cuando tenga el punto desde el cual jugador decidió iniciar podre tener el camino mínimo para llegar a la zona segura, Dijkstra me dará el valor sin problemas.

Preparación de informe y especificaciones

Diagrama de Clases:

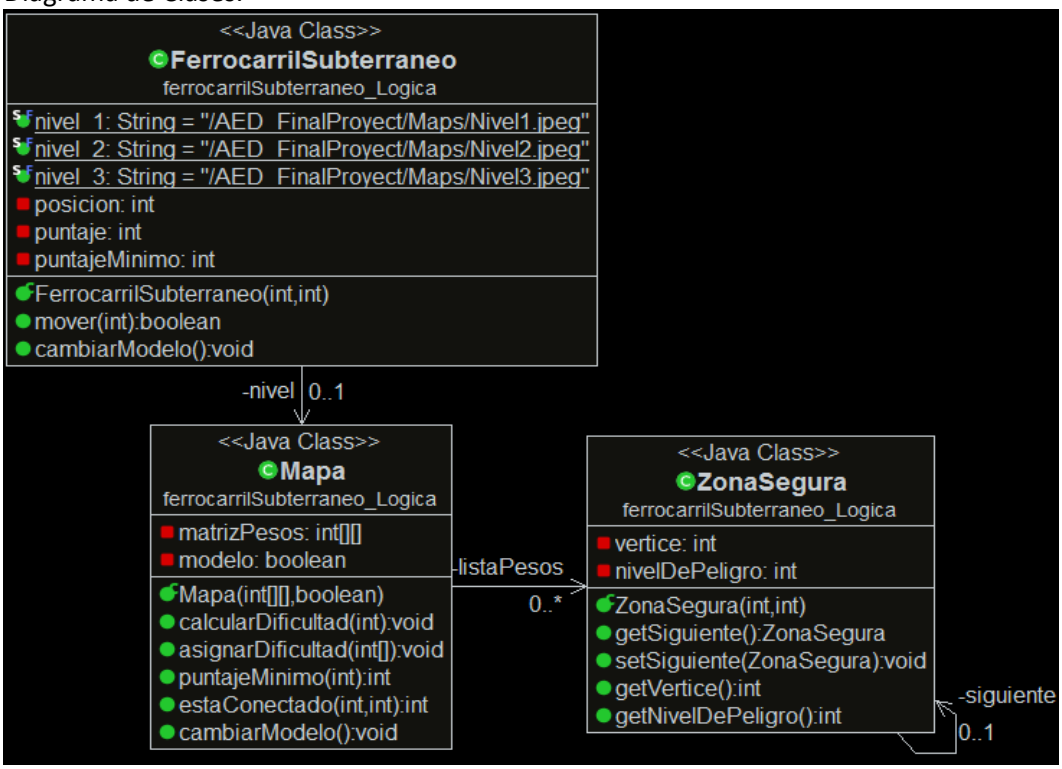


Diagrama de objetos:

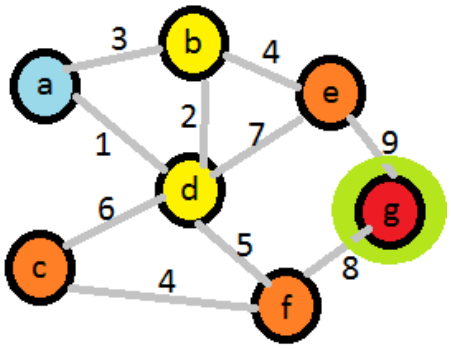


Ilustración 7: Nivel de Prueba, representa la clase Mapa

La ilustración numero 7 muestra como seria el cálculo de nivel y asignación de peligro para desplazarse de una zona a otra a partir del punto elegido por la persona para llegar a la frontera.

Para llegar a una zona segura con un nivel de peligro bajo se pueden perder entre 1 y 3 esclavos; a una de peligro medio entre 4 y 7 esclavos; por último, a una de peligro alto entre 8 y 10 esclavos.

En la ilustración 6, por ejemplo, el costo para llegar a la frontera hubiese estado entre 1 y 3.

La representación del grafo anterior en el programa seria mediante una matriz de pesos o una lista de adyacencia con pesos (en esta se guarda el vértice adyacente y el peso).
Lo que está contenido en la lista { (vértice, peso) }, son objetos de la clase ZonaSegura.

Matriz de pesos para la ilustración 7

	a	b	c	d	e	f	g
a	0	3	∞	1	∞	∞	∞
b	3	0	∞	2	4	∞	∞
c	∞	∞	0	6	∞	4	∞
d	1	2	6	0	7	5	∞
e	∞	4	∞	7	0	∞	9
f	∞	∞	4	5	∞	0	8
g	∞	∞	∞	∞	9	8	0

Lista de adyacencia con pesos para la ilustración 7

a	(b,3)	(d,1)			
b	(a,3)	(d,2)	(e,4)		
c	(d,6)	(f,4)			
d	(a,1)	(b,2)	(c,6)	(e,7)	(f,5)
e	(b,4)	(d,7)	(g,9)		
f	(c,4)	(d,5)	(g,8)		
g	(e,9)	(f,8)			

La clase tren subterráneo tendría entonces además del grafo nivel:

- posicion: a
  Es la posición actual en este ejemplo son letras, pero en el programa los vértices serán números.
- puntaje: 100
  Es con el cual inicia.
- puntajeMinimo: 100 – (14 + ⌊14 * 30%⌋) = 100 – (14 + 4) = 82

Casos de pruebas:

Objetivo: cargar un mapa de un nivel elegido por el jugador				
Clase: Mapa		Mapa(int[][], boolean)		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida
1	Carga el mapa del nivel 1 y se verifica que la lista y la matriz tengan los valores correctos.	1	Nivel 1 y true para seleccionar la matriz.	La lista y la matriz creadas.

	Además, que quede seleccionada la que el usuario quizo.			

Objetivo: Mueve al jugador al punto deseado				
Clase: FerrocarrilSubterraneo		Mover(int)		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida
1	Mueve al jugador de un punto “a” a un punto “b”	2	Una posición valida de llegada	El jugador se desplazó al punto deseado
2	No mueve al jugador de un punto “a” a un punto “b”	2	Una posición no valida de llegada	El jugador no se desplazó al punto deseado

Objetivo: Cambiar el modelo con el que se esta interactuando				
Clase: FerrocarrilSubterraneo		cambiarModelo()		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida
1	Se cambia de matriz de pesos a lista de adyacencia con pesos	2		El modelo valor booleano del modelo cambio
2	Se cambia de lista de adyacencia con pesos a matriz de pesos	2		El modelo valor booleano del modelo cambio

Objetivo: Calcular la dificultad del grafo cargado				
Clase: Mapa		calcularDificultad(int)		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida
1	Realiza el BFS para calcular los niveles de peligro para llegar a un vértice	2	El vértice desde el cual se va a iniciar	La lista con los niveles del grafo.

Objetivo: Asigna los costos de llegar de un vértice a vértice				
Clase: Mapa		asignarDificultad(int[])		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida
1	Asigna a la matriz los pesos y a la lista le agrega los pesos también. Las dificultades se asignan dentro del rango.	3	Lista de niveles de peligro para llegar de un vértice a otro	La lista y la matriz creadas tiene los pesos aleatorios.

Objetivo: Calcular la dificultad del grafo cargado				
Clase: Mapa		estaConectado(int, int)		
caso	Descripción de la prueba	Esc	Valores de entrada	Salida

1	Revisa si dos nodos validos están conectados entre si	4	El vértice inicio y destino	El valor entre estos dos vértices
2	Revisa si dos no nodos validos están conectados entre si	4	El vértice inicio y destino	-1, no fue posible hallar arista
2	Revisa si un vértice valido y otro no valido están conectados entre si	4	El vértice inicio y destino	-1, no fue posible hallar arista

Referencias:

1. https://es.wikipedia.org/wiki/Nivel_(videojuegos)
2. https://es.wikipedia.org/wiki/Grafo
3. https://es.wikipedia.org/wiki/Grafo#/media/Archivo:Kaari_suuntaamaton_graafiteoria.png
4. https://es.wikipedia.org/wiki/Grafo#/media/Archivo:Matriz_de_adyacencia.jpg
5. https://es.wikipedia.org/wiki/Grafo#/media/Archivo:Listas_de_adyacencia.jpg
6. https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/
7. https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/
8. https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/
9. https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/