PROYECTO FINAL

SEGUNDA ENTREGA

LUIS FELIPE GÓMEZ ANDRADE

ELÍAS CALEB ESTUPIÑAN TOLOSA

OSCAN IVAN RIASCOS GUEVARA

ANDRÉS ARISTIZÁBAL

ALGORITMOS Y ESTRUCTURAS DE DATOS

2019-2

MÉTODO DE LA INGENIERÍA

Paso 1: Identificación del problema

Definición del problema

Implementar una solución creativa y dinámica de un juego de laberintos.

Justificación

Sam es un apasionado aventurero que, desde niño, le gusta afrontar desafíos explorando ruinas de antiguas culturas, en donde ninguna persona o, muy pocas, se han atrevido a entrar para investigar los secretos que allí se ocultan. Para suerte de él, vive en una misteriosa isla en alguna parte del planeta, la cual está llena de misteriosos laberintos construidos por una civilización ya extinta.

De esta manera, con el fin de seguir su sueño de convertirse en el mejor explorador del mundo, Sam ha tomado la decisión de estudiar aquellos misteriosos laberintos mencionados anteriormente. Así mismo, dado que no existe información sobre cómo llegar a la salida de cada laberinto (Solo se conocen las entradas), nuestro aventurero ha llegado a la conclusión de que primero se deberá realizar o encontrar un mapa de cada laberinto, con el fin de que, tiempo después, se estudien a fondo.

Su tarea, si decide aceptarla, es construir un aplicativo que sirva para guiar a Sam durante su peligrosa expedición y mapear cada laberinto, para que, tiempo después, se calcule la ruta más rápida desde la entrada hasta la salida, por si otro explorador quisiera estudiarlo y/o recorrerlo.

Dado que cada laberinto está bajo tierra, éste se encuentra totalmente a oscuras, por lo tanto, Sam poseerá una linterna que lo ayudará a guiarse. Adicionalmente, esta linterna, única en su tipo, tiene una funcionalidad de que, cada 15 segundos, permite alcanzar una zona de iluminación 3 veces más grande que la que posee por defecto, permitiéndole, a Sam, mapear o visualizar una zona más grande.

En adición a lo anterior, se tiene conocimiento de que algunas zonas del laberinto poseen puntos de “iluminación” las cuales conservan su “luz” cuando el explorador pasa por ellas y las mapea o visualiza. Por lo tanto, se debe de agregar una funcionalidad al aplicativo de encontrar un camino desde el punto donde se encuentra Sam hasta otro punto de “iluminación”, lo cual servirá para evitar caminar en círculos y perderse aún más en las oscuras y misteriosas ruinas. Así mismo, el aplicativo debe permitir el guardar el progreso llevado por Sam en el laberinto, para que, si decide salirse de la expedición, cuando vuelva contará con sus datos previamente conseguidos. Esta opción, solo se deberá permitir cuando Sam se encuentre en uno de los puntos de “iluminación”, ya que, el volver a estos sitios es relativamente más fácil que las otras partes oscuras y tenebrosas de los laberintos.

Por último, una vez terminado un laberinto, se debe de mostrar un mensaje donde al guía (el jugador o usuario) se le indique que ha terminado de dirigir a Sam en este laberinto. También, se debe de preguntar si se desea continuar la expedición o si está ya ha terminado. Todo lo anterior para que si, por un lado, se selecciona la primera opción (continuar expedición) se debe de trasladar a Sam hasta la entrada de otro laberinto para realizar el mismo procedimiento descrito con anterioridad. Por otro lado, si se selecciona la segunda opción (terminar expedición) se debe de llevar al guía a la pantalla principal del aplicativo.

Requerimientos Funcionales

1. El programa debe de estar en la capacidad de encontrar o calcular la ruta más corta, desde el punto de partida hasta el punto de salida del laberinto. Para esto, se debe de haber mapeado por completo el laberinto con anterioridad.
2. El programa debe de estar en la capacidad de encontrar o calcular un camino, desde el punto actual donde se encuentra Sam, hasta un punto de iluminación seleccionado en el laberinto. Para esto, se debe de haber mapeado el punto al cual se quiere llegar nuevamente.
3. El programa debe de permitirle a Sam estar en la capacidad de mapear una zona 3 veces más grande de la que puede mapear por defecto. Para esto, solamente se puede utilizar cada 15 segundos y, si Sam no utiliza la habilidad, no se reiniciará el conteo de los 15 segundos hasta que la utilice.
4. El programa debe de estar en la capacidad de que, una vez escogida la opción de continuar expedición, cuando se acaba de mapear un laberinto, lleve al jugador a la entrada de otro laberinto para realizarle el correspondiente mapeado.
5. El programa debe de estar en capacidad de que, una vez escogida la opción de terminar expedición, cuando se acaba de mapear un laberinto, lleve al guía o jugador al menú principal.
6. El programa debe de estar en capacidad de guardar la partida en curso cuando el personaje (Sam) se encuentre sobre un punto de luz. Para esto, se debió haber generado y pintado un mapa en pantalla.
7. El programa debe estar en capacidad de cargar una partida, guardada con anterioridad. Para esto, se debe de poseer un apartida guardada anteriormente.

Paso 2: Recopilación de la Información

Sobre un laberinto

Un laberinto es un pasatiempo gráfico consistente en trazar una línea desde un punto de origen situado en el exterior de un laberinto a uno de destino situado generalmente en el centro o bien en el lado opuesto. La dificultad consiste en encontrar un camino directo hasta el lugar deseado. El laberinto, por su propia configuración, contiene diferentes vías sin salida (de mayor o menor longitud) y solo un recorrido correcto. Puede adoptar diferentes formas: cuadrado, ovalado, redondo, cuadrangular, etc.

Al ir destinado mayoritariamente a niños, a veces, se propone como una prueba en la que, por ejemplo, un ratón debe alcanzar su queso, representándose el primero en exterior y el queso en el centro del laberinto.

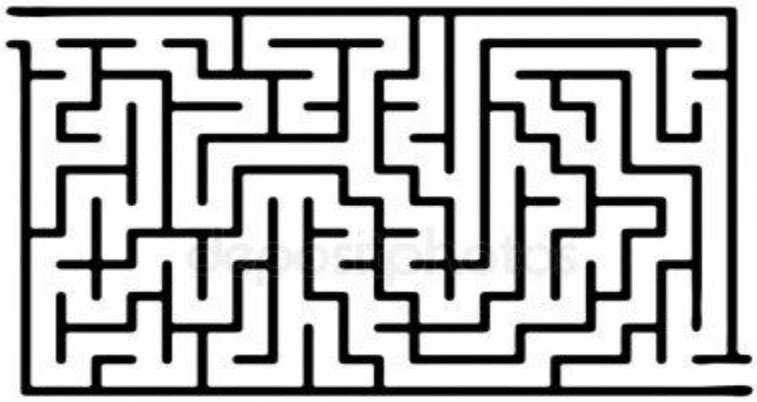

Figura 1: Laberinto estándar

Grafo

El gráfico es una estructura de datos que consiste en seguir dos componentes: 1) Un conjunto finito de vértices también llamados nodos. 2) Un conjunto finito de pares ordenados de la forma (u, v) llamada como borde. El par se ordena porque (u, v) no es igual que (v, u) en el caso de un gráfico dirigido (directed-graph). El par de la forma (u, v) indica que hay un borde desde el vértice u hasta el vértice v. Los bordes pueden contener peso/valor/coste.

Los gráficos se utilizan para representar muchas aplicaciones de la vida real: Los gráficos se utilizan para representar redes. Las redes pueden incluir trayectos en una red urbana o telefónica o en una red de circuitos. Los gráficos también se utilizan en redes sociales como LinkedIn, Facebook. Por ejemplo, en Facebook, cada persona está representada con un vértice (o nodo). Cada nodo es una estructura y contiene información como la identificación de la persona, el nombre, el género y la localización. Vea esto para más aplicaciones del gráfico.

A continuación dos son las representaciones más utilizadas de un gráfico.

1. Matriz de Adyacencia
2. Lista de Adyacencia

Hay otras representaciones también como, Matriz de Incidencias y Lista de Incidencias. La elección de la representación gráfica es específica de la situación. Depende totalmente del tipo de operaciones a realizar y de la facilidad de uso.

Tipos de Grafos

Existen principalmente cuatro tipos de grafos. Veamos:

- Grafo Simple: Un grafo simple G = (V, A) costa de un conjunto no vacío de vértices V y de un conjunto A de pares no ordenados de elementos distintos de V, a estos pares se les llama aristas. En otras palabras un grafo simple es un grafo en

los que existe a lo más una arista que une dos vértices distintos.

- Multígrafo: Un multígrafo G = (V, A) consta de un conjunto V de vértices, un conjunto A de aristas y una función f de A hacia {{u, v} | u, v∈V, u≠v}. Se dice que las aristas a1 y a2 son aristas múltiples o paralelas si f (a1) = f (a2).

- Pseudografo: Un Pseudografo G = (V, A) consta de un conjunto V de vértices, un conjunto A de aristas y una función f de A hacia {{u, v} | u, v∈V}. Una arista a es un bucle o lazo, si f(a) = {u; u} = {u} para algún u∈V.

- Dirigido: Un grafo dirigido G, también llamado dígrafo, es lo mismo que un multígrafo, solo que cada arista e de G tiene una dirección asignada o, en otras palabras, cada arista e está identificada por un par ordenado (u, v) de nodos G en vez del par desordenado [u. v]. Un grafo dirigido (V, A) consta de un conjunto V de vértices y de un conjunto A de aristas, que son pares ordenados de elementos de V. Utilizamos el par ordenado ⟨u, v⟩ para indicar que es una arista dirigida del vértice u al vértice v.

- Multígrafo Dirigido: Un multígrafo dirigido G = (V, A) consta de un conjunto V de vértices, un conjunto A de aristas y una función f de A hacia {⟨u, v⟩ | u, v V}. Se dice que las aristas a1 y a2 son aristas múltiples o paralelas si f (a1) = f (a2).

Recorridos de grafos

Para hacer recorridos sobre grafos se utilizan dos algoritmos muy conocidos, el primero hace un recorrido por amplitud y el segundo en profundidad. Veamos.

- **BFS** (Breath First Search) es el enfoque más comúnmente utilizado. Es un algoritmo de travesía en el que se debe empezar a atravesar desde un nodo seleccionado (fuente o nodo de inicio) y atravesar el gráfico por capas, explorando así los nodos vecinos (nodos que están conectados directamente con el nodo de origen). A continuación, debe desplazarse hacia los nodos vecinos del siguiente nivel.

  Como el nombre BFS sugiere, se requiere que usted recorra la gráfica a lo ancho de la siguiente manera:

  - Primero mueva horizontalmente y visite todos los nodos de la capa actual
  - Moverse a la siguiente capa

- **DFS** (Depth First Search) comienza con el nodo inicial del gráfico G, y luego va más y más profundo hasta que encontramos el nodo objetivo o el nodo que no tiene hijos. El algoritmo, entonces, retrocede desde el callejón sin salida hacia el nodo más reciente que aún no ha sido explorado por completo.

  La estructura de datos que se está utilizando en DFS es apilada. El proceso es similar al algoritmo BFS. En DFS, los bordes que conducen a un nodo no visitado se denominan bordes de descubrimiento, mientras que los bordes que conducen a un nodo ya visitado se denominan bordes de bloque.

Árboles de recubrimiento mínimo

Dado un grafo conexo y no dirigido, un árbol recubridor de ese grafo es un subgrafo que tiene que ser un árbol y contener todos los vértices del grafo inicial. Cada arista tiene asignado un peso proporcional entre ellos, que es un número representativo de algún objeto, distancia, etc.; y se usa para asignar un peso total al árbol recubridor mínimo computando la suma de todos los pesos de las aristas del árbol en cuestión. Un árbol recubridor mínimo o un árbol expandido mínimo es un árbol recubridor que pesa menos o igual que otros árboles recubridores. Todo grafo tiene un bosque recubridor mínimo.

Presentaremos dos algoritmos que nos permiten encontrar arboles de recubrimiento. Veamos:

- **Prim:** El Algoritmo de Prim se usa para encontrar el árbol de extensión mínima de un gráfico. El algoritmo de Prim encuentra el subconjunto de bordes que incluye cada vértice del gráfico de tal manera que la suma de los pesos de los bordes puede ser minimizada.

  El algoritmo de Prim comienza con el único nodo y explora todos los nodos adyacentes con todos los bordes de conexión en cada paso. Se seleccionaron los bordes con los pesos mínimos que no causaban ningún ciclo en el gráfico.

- **Kruskal:** El algoritmo de Kruskal se utiliza para encontrar el árbol de expansión mínima para un grafo ponderada conectada. El objetivo principal del algoritmo es encontrar el subconjunto de bordes mediante el cual, podemos atravesar cada vértice del gráfico. El algoritmo de Kruskal sigue un enfoque codicioso que encuentra una solución óptima en cada etapa en lugar de centrarse en un óptimo global.

Recorrido de camino mínimo

Un dígrafo ponderado por borde es un dígrafo donde asociamos pesos o costos con cada borde. Un camino más corto de vértice s a vértice t es un camino dirigido de s a t con la propiedad de que ningún otro camino tiene un peso menor.

Veamos dos algoritmos que nos ayudan a encontrar caminos mínimos en grafos.

- **Dijkstra:** El algoritmo de Dijkstra encuentra la ruta menos costosa en un gráfico ponderado entre nuestro nodo de inicio y un nodo de destino, si existe tal ruta.

  Al final del algoritmo, cuando hemos llegado al nodo de destino, podemos imprimir la ruta de menor coste retrocediendo desde el nodo de destino hasta el nodo de inicio.

- **Floyd-Warshall:** El algoritmo de Floyd-Warshall se utiliza para encontrar todos los problemas del trayecto más corto de un par a partir de un gráfico ponderado dado. Como resultado de este algoritmo, se generará una matriz, que representará la distancia mínima entre cualquier nodo y todos los demás nodos del gráfico.

  Al principio, la matriz de salida es la misma que la matriz de costes del gráfico. Después de eso, la matriz de salida se actualizará con todos los vértices k como vértices intermedios.

Paso 3: Búsqueda de soluciones creativas

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

- Encontrar camino más corto desde la entrada hasta la salida.
- Encontrar camino desde el punto actual hasta un punto de iluminación.
- Mapear zona 3 veces más grande
- Continuar expedición
- Terminar expedición
- Guardar partida
- Cargar partida

Luego de indagar sobre qué estructuras nos permiten implementar funcionalidades antes mencionadas optamos por las siguientes:

- Grafo simple
- Multígrafo
- Pseudografo
- Grafo dirigido• Multígrafo dirigido
- Árbol n-ario
- Hash Table

Paso 4: Transición de ideas a diseños preliminares

Conociendo de antemano, que hay conexiones entre los distintos caminos que se pueden dar en el laberinto, impediría que se usaran árboles para su implementación porque significaría que habrá algún tipo de interrelación entre las raíces de los árboles que simulan los posibles caminos. Por otro lado, una Hash Table nos permitiría acceder de manera muy eficiente a cada punto de un camino o sendero específico, pero definición misma de la Hash Table nos impide modelar conexiones entre los demás vértices; por lo que procedemos a descartarlas.

Procedemos a analizar las restantes ideas alternativas, las cuales se describen someramente a continuación:

	Tipos de aristas	*¿Admite aristas múltiples?*	*¿Admite bucles?*
Grafo simple	*No dirigidas*	*No*	*No*
Multígrafo	*No dirigidas*	*Si*	*No*
Pseudografo	*No dirigidas*	*Si*	*Si*
Grafo dirigido	*Dirigidas*	*No*	*Si*
Multígrafo dirigido	*Dirigidas*	*Si*	*Si*

Paso 5: Evaluación y selección de la mejor solución

A continuación presentamos los criterios para hacer la selección de la mejor solución.

Criterio A: Conexión entre senderos. La conexión entre senderos o caminos debe permitir llegar de uno a otro.

- [2] Permite llegar de un sendero a otro, y viceversa.
- [1] Permite llegar de un sendero a otro, pero no indispensablemente de regreso.

Criterio B: Aristas múltiples. Dado que se puede usar el mismo sendero para devolverse al llegar a un punto ciego en el laberinto, no se necesitan aristas múltiples.

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

Criterio C: Conexión hacia sí mismo. No se necesita retornar al mismo sendero par devolverse en el laberinto.

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

	Criterio A	*Criterio B*	*Criterio C*	*Total*
Grafo simple	*2*	*2*	*2*	*6*
Multígrafo	*2*	*1*	*2*	*5*
Pseudografo	*2*	*1*	*1*	*4*
Grafo dirigido	*1*	*2*	*1*	*4*
Multígrafo dirigido	*1*	*1*	*1*	*3*

En conclusión, la solución elegida con la que se puede implementar un juego de laberinto es con un **grafo simple**, dado que permite conexiones reciprocas entre senderos, no se necesitan múltiples senderos, ni se requieren bucles.

Bibliografía

Geeksforgeeks

https://www.geeksforgeeks.org/graph-and-its-representations/

Hackerearth

https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

Stack Abuse

https://stackabuse.com/graphs-in-java-depth-first-search-dfs/

https://stackabuse.com/graphs-in-java-dijkstras-algorithm/

Javatpoint

https://www.javatpoint.com/prim-algorithm

https://www.javatpoint.com/kruskal-algorithm

Tutorialspoint

https://www.tutorialspoint.com/Floyd-Warshall-Algorithm