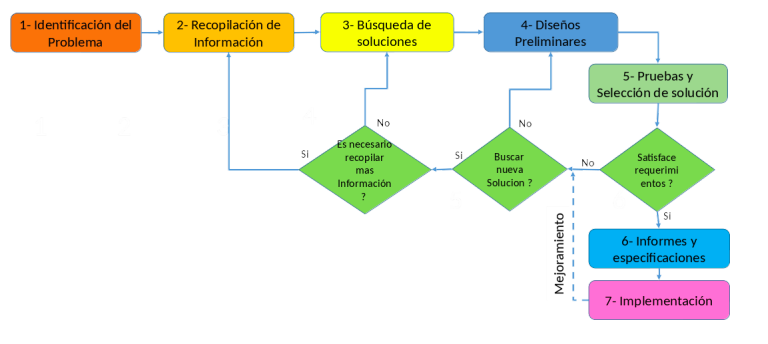
**Collin Gonzalez Benitez - A00382429**

**Samuel Gutierrez - A00381035**

**Método de la ingeniería**



**Enunciado del problema:**

Collin Gonzalez un famoso streamer colombiano de juegos de terror quiere pasarse en directo mediante su canal de twitch el nuevo juego de la saga de Resident Evil en un speed run para ser el primer streamer en acabar el juego. Para ello Collin investigó a fondo acerca de los diferentes jefes y caminos que brinda el juego ya que es común en esta saga tener diferentes caminos para terminar el juego, además de que cada decisión podría traerle un jefe con mayor o menor dificultad mientras avanza, adicional a esto en esta nueva secuela es conocido que el juego cuenta con 2 finales diferentes.Collin encontró toda esta información en un foro de fanáticos de la saga además de un grafo que muestra todos los posibles caminos que el jugador puede tomar para terminar en alguno de los 2 finales del juego.

Los vértices del grafo representan un jefe de la historia que se debe vencer antes de tomar el siguiente camino, por eso mismo las aristas ponderadas que conectan cada vértice con sus adyacentes representan las posibles opciones de caminos a elegir por el jugador para el siguiente jefe, y la dificultad del siguiente jefe se ve reflejada en la ponderación de la arista que conduce hacia el nuevo vértice. El grafo deja claro que el inicio de la aventura empieza desde el vértice 0 y muestra sus 2 finales (Jefes finales) en los vértices 49 y 50.

Collin cuenta con un amigo llamado Samuel estudiante de ingeniería en sistemas de la universidad Icesi que hace poco le comentó a Collin que estaba un poco frustrado por el último tema que estaba viendo en Computación y estructuras discretas 1 que era exactamente la teoría de grafos. Sabiendo que el juego salía al día siguiente Collin le pidió el favor a Samuel para que diseñara un código en Java con interfaz para poder buscar el camino con menor dificultad para pasarse el juego y así ser el primer streamer en completar el juego.

Además, Collin deseaba compartir este programa en el Foro de fanáticos ya que con otro par de opciones podría ser de bastante utilidad para otros jugadores que quisieran pasar el juego.

Esta opción agregada permite que cualquier jugador del juega al indicar mediante la interfaz el vértice del jefe en quien se encuentra en medio del juego, pueda saber si el camino por el que va tiene salida hacia el final del juego o si debe volver a uno de sus checkpoints para poder terminar el juego. Esto lo propuso Collin al saber que en esta nueva saga asignaron varios caminos sin salida en el juego para que la dificultad de llegar al final de este sea mayor.

1. **Identificación del problema**

El streamer requiere un programa que sea capaz de calcular la mejor ruta a seguir para completar el juego Resident Evil en la menor dificultad posible. Para obtener una correcta solución se requiere tener los datos pertinentes y el correcto y selección de los distintos algoritmos e implementaciones de la teoría de Grafos. La propuesta final debe cumplir con los requerimientos estipulados por parte del Streamer para resolver el problema.

**Definición del Problema**

El creador de contenido necesita un programa capaz de calcular de manera eficaz y efectiva el camino con la dificultad mínima con el fin de completar el juego en el menor tiempo posible.

**Especificación de Requerimientos**

**Requerimientos Funcionales:**

**R1:** El sistema debe contar con una interfaz para poder comunicarse con el usuario acerca de las diferentes funcionalidades del sistema :

(Buscar el camino con menos dificultad al final del juego, Verificar si el jugador puede terminar el juego desde el jefe donde se encuentra).

**R2:** El sistema debe permitir añadir un grafo mediante una selección de archivo de tipo JSON, para que este sea el grafo que el programa visualiza para ejecutar los algoritmos sobre el grafo.

**R3:** El sistema debe permitir mostrar al usuario el camino de distancia mínima entre un vértice y otro que el seleccione.

**R4:** El sistema debe poder ejecutarse con 2 diferentes implementaciones de grafos, lista de adyacencia y matriz de adyacencia para el cálculo de los recorridos sobre el grafo y los caminos de distancia mínima.

**R5**: El sistema debe permitir mostrar al usuario si desde el vértice(Jefe que se encuentra) tiene la opción de llegar al final del juego o si este debe devolverse porque es un camino sin salida.

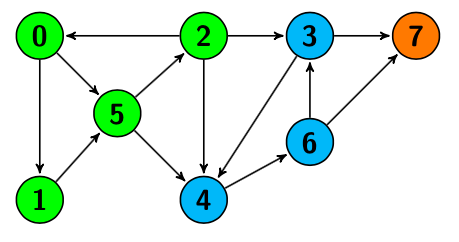
Requerimientos de Proceso:

RP1: El sistema debe desarrollarse por no más de 3 personas en el grupo.

**2. Recopilación de Información**

**Grafo:**

En la teoría de grafos, un grafo es un conjunto no vacío y estructura discreta que consta de objetos llamados vértices o nodos y las conexiones entre estos llamados aristas. Es comúnmente representado de la siguiente manera:



Un grafo en esencia es una red, debido a esto su funcionalidad es implementada en distintos ámbitos en los que se trabaje con redes, como por ejemplo:

* Red de personas
* Red de carreteras
* Red eléctrica
* Mapas
* Torneos

Existen distintos tipos de grafos:

* Grafo simple
* Multigrafo
* Grafo dirigido
* Pseudografo
* Multigrado dirigido

**Grafo dirigido:**

Un grafo dirigido a diferencia de los grafos no dirigidos tiene como característica que las aristas tienen una dirección específica, cosa que cambia la referencia de adyacencia (listas y matrices).

**Listas de adyacencia:**

Cada grafo tiene una lista de adyacencia de sus vértices, consiste en una lista en el que cada vértice del grafo tiene adjunto sus vecinos (vértices con los que conecta), esto puede variar dependiendo del tipo de grafo, pues por ejemplo en los grafos no dirigidos habría redundancia en la lista de adyacentes.

Tanto la lista de adyacencia como la matriz de adyacencia son ideas usadas para la implementación de la estructura de grafos en el ámbito de la programación, destacando su gran importancia.

Fuentes:

*Teoría de grafos*. (s. f.). Unipamplona. Recuperado 8 de noviembre de 2022, de https://www.unipamplona.edu.co/unipamplona/portalIG/home\_23/recursos/general/11072012/grafo3.pdf

Menéndez Velázquez, A. (1998, junio). *Una breve introducción a la teoría de grafos*. Refined. https://redined.educacion.gob.es/xmlui/bitstream/handle/11162/13526/011-026.pdf?sequence=1

**3. Búsqueda de Soluciones Creativas**

**Alternativa 1.**

**BFS y Dijkstra:**

Según la petición del usuario un programa elaborado en código facilita la solución de la problemática, debido a esto, esta alternativa consta de dos algoritmos relacionados con el uso de grafos.

El BFS es uno de los algoritmos de búsqueda y recorrido en teoría de grafos más simples y empleados en todo el mundo. Su funcionamiento es de la siguiente manera, el BFS es una búsqueda por anchura o amplitud, esto quiere decir que se comienza en un nodo y se exploran todos sus vecinos, posteriormente se hace lo mismo desde cada uno de los vecinos hasta que se recorra todo el grafo o no se pueda avanzar debido a las adyacencias. En código toda esta explicación del funcionamiento es más elaborada, pues se usan más parámetros como colores.

Dijkstra es un algoritmo que permite calcular el camino de longitud mínima entre dos vértices en un grafo ponderado(aristas con peso), cabe aclarar que este algoritmo a diferencia de otros solo funciona con grafos que tengan pesos positivos.

El algoritmo empieza encontrando el camino de longitud mínima entre un primer vértice y otro adyacente(*a*), después la longitud mínima entre *a* y otro vértice adyacente, así sucesivamente hasta determinar cuál es el camino de longitud mínima entre los dos vértices dados.

Estos dos algoritmos servirían como alternativa de solución al problema a tratar pues cumplen con la mayoría de los requerimientos y más importante aún, logran dar solución al problema.

**Alternativa 2.**

**DFS y Floyd-Warshall**

Estos son dos algoritmos que al igual que en la alternativa 1 se emplean en grafos pero tienen ciertas diferencias esenciales.

El algoritmo Floyd.Warshall al igual que Dijkstra es empleado para hallar el camino de longitud mínima entre dos vértices de un grafo ponderado. En este caso el valor de las ponderaciones de las aristas pueden ser tanto positivas como negativas. El funcionamiento de este algoritmo es el siguiente, usando una matriz de distancias que es establecida de acuerdo a las aristas entre vértices y su ponderación. Se comparan las filas y las columnas para actualizarla.

DFS al igual que el BFS es un algoritmo de búsqueda y recorrido en profundidad, expande un nodo hasta que no se pueda más. Según el código, en el caso del DFS a diferencia de BFS, no hay casos en que algún vértice se quede sin visitar, pues su funcionamiento recursivo está diseñado de esta manera.

Esta alternativa junto a la alternativa número uno es una de las principales candidatas a ser la final, aunque si observamos la complejidad temporal de los algoritmos Dijkstra y Floyd-Warshall hay una diferencia (*-* respectivamente), de acuerdo al problema a tratar hay que decidir cuál sería la mejor opción.

**Alternativa 3.**

**Simulación.**

Una alternativa poco ortodoxa pero no deja de ser una alternativa a tener en cuenta, sería simular el juego y todos los caminos posibles para posteriormente compararlos. Para simular todas las partidas haría falta o tener muchos dispositivos o crear algún tipo de código que simula las partidas en solo uno y recoja la información de manera automática.

**4. Transición de las Ideas a los Diseños Preliminares.**

Evaluando la lista de alternativas propuestas como solución al problema a resolver, la alternativa ideal para solucionar el problema es la alternativa uno. A pesar de la similitud entre las dos primeras alternativas y saber que ambas podrían dar solución al problema, el factor diferencial fue la complejidad temporal del algoritmo Dijkstra a comparación de la complejidad temporal del algoritmo Floyd-Warshall *-* respectivamente. Después de analizar los requerimientos y la necesidad del usuario y la manera en que está estructurado el problema, es mejor usar la complejidad temporal menor del algoritmo Dijkstra.

**5. Evaluación y Selección de la Mejor solución.**

**Los criterios para evaluar y seleccionar la mejor solución serán:**

**Criterio A Eficiencia.** Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

[3] Muy eficiente

[2] Eficiente

[1] Poco eficiente

**Criterio B Complejidad.** Se preferiría la que tenga menor complejidad temporal, teniendo en cuenta el contexto del problema.

[3] Poca

[2] Compleja

[1] Mucha

**Criterio C.** Se preferirá la alternativa que no pueda ayudar a desarrollar todos los requerimientos del proyecto sin excepción. El rango será:

[2] Desarrolla todos los requerimientos

[1] Desarrolla la mayor parte de los requerimientos, sin embargo presenta problemas en algunos

|  | **Criterio A** | **Criterio B** | **Criterio C** | **Total** |
| --- | --- | --- | --- | --- |
| **Alternativa 1:**  **BFS y Dijkstra** | **3** | **2** | **2** | **7** |
| **Alternativa 2:**  **DFS y Floyd-Warshall** | **3** | **1** | **2** | **6** |
| **Alternativa 3:**  **Simulación** | **1** | **1** | **1** | **3** |

**TAD- Grafo**

**Diseño de pruebas unitarias del TAD del grafo:**

**Agregar vértices y aristas :**

**Vértice Aristas**

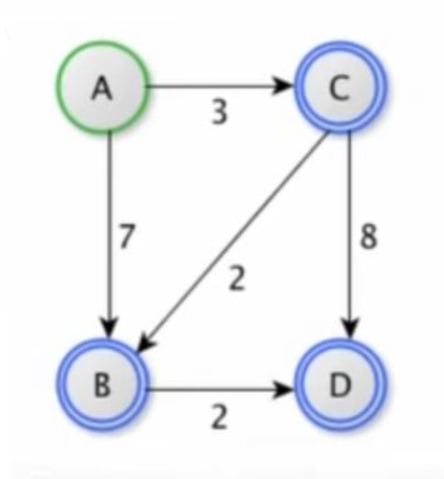
| **Nombre : A, Indice : 0** | **From : A, To : C, Peso : 3**  **From : A , To : B, Peso : 7** |
| --- | --- |
| **Nombre : B, Índice : 1** | **From : B, To : D, Peso : 2** |
| **Nombre : C, Índice : 2** | **From : C, To : D, Peso : 8**  **From : C , To : B, Peso : 2** |
| **Nombre : D, Índice : 3** | **No aristas** |

**Eliminar vértices y arista:**

**Vértice Aristas**

| **Índice : 0** | **from: 0, to: 2** |
| --- | --- |
| **Índice : 1** | **from: 2, to: 1** |
| **Índice : 2** | **from: 2, to: 3** |
| **Índice : 3** | **from: 1, to: 3** |

**Pruebas unitarias del Problema :**

****

**Pruebas de BFS**

**En base al grafo de arriba:**

**Valor A Iniciar Resultado esperado**

| **A** | **Desde A alcanza todos los vértices** |
| --- | --- |
| **C** | **Alcanza los vértices B y D** |
| **B** | **Alcanza sólo el vértice D** |

**Pruebas de Dijkstra**

**En base al grafo de arriba:**

**Valores a calcular menor distancia**

| **Desde: A, Hasta : D** | **Recorrido : {A,C,B,D} Peso: 7** |
| --- | --- |
| **Desde : A Hasta: B** | **Recorrido : {A,C,B} Peso: 5** |
| **Desde: B Hasta : C** | **Recorrido{nul} Peso : Infinito** |