**ENGINEERING METHOD**

**Problematic context**

Se debe poder desarrollar un juego de uno o mas jugadores y este debe ser modelado utilizando grafos, debe tener un mínimo de 50 veritices y 50 aristas. Se debe también usar al menos dos de los algoritmos que estudiamos en el curso, ya sean BFS, DFS, Dijkstra, Floyd-Warshall, Prim o Kruskal.

**Step 1: Problem Identification**

Identification of needs and symptoms:

- Debemos desarrollar un juego para uno o mas jugadores que se modele utilizando grafos

- Debemos tener un mínimo de 50 vertices y de 50 aristas

- Se deben aplicar por lo menos dos algoritmos de grafos

*(Requirements specification is in another document called "requirementsIT1")*

**Step 2: Information Gathering**

Search for definitions of terms to be implemented in the problem:

*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.​

*BFS ‘Breadth First Search’*

Una búsqueda en anchura (BFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

La búsqueda por anchura se usa para aquellos algoritmos en donde resulta crítico elegir el mejor camino posible en cada momento del recorrido.

*DFS ‘Depth First Search’*

Una búsqueda en profundidad (DFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda.

La búsqueda en profundidad se usa cuando queremos probar si una solución entre varias posibles cumple con ciertos requisitos; como sucede en el problema del camino que debe recorrer un caballo en un tablero de ajedrez para pasar por las 64 casillas del tablero.

*Dijkstra*

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo).

*Floyd-Warshall*

El algoritmo de Floyd-Warshall es la opción utilizada cuando se desea determinar el camino mínimo entre todos los pares de vértices de un grafo, comparando todos los posibles caminos logra mejorar paulatinamente la estimación hasta llegar a la más óptima. Esto puede presentarse de manera más clara a través de un ejemplo de implementación. Pero antes, revisaremos el análisis del tiempo de ejecución para este caso.

*Árbol de Recubrimiento Mínimo -MST*

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.

*Prim*

El algoritmo de Prim encuentra un árbol de peso total mínimo conectando nodos o vértices con arcos de peso mínimo del grafo sin formar ciclos. Consiste en dividir los nodos de un grafo en dos conjuntos: procesados y no procesados. Al principio, hay un nodo en el conjunto procesado que corresponde a el equipo central; en cada interacción se incrementa el grafo de procesados en un nodo (cuyo arco de conexión es mínimo) hasta llegar a establecer la conexión de todos los nodos del grafo a procesar.

*Kruskal*

El algoritmo de Kruskal consiste en la obtención de un árbol cuyas aristas al ser sumados sus pesos el valor obtenido es el mínimo, tomando el nombre de algoritmo voraz.

El algoritmo está basado en una propiedad de los árboles que permite estar seguros de sí un arco debe pertenecer al árbol o no, y usar esta propiedad para seleccionar cada arco. Se debe tener en cuenta en el algoritmo que siempre que se añade una arista, ésta será siempre la conexión más corta (menor coste) alcanzable desde el nodo que se parte al resto del grafo. Así que por definición éste deberá ser parte del árbol. Este algoritmo es de tipo greedy, ya que a cada paso, selecciona la arista de menor valor y lo añade al sub-grafo (Villalobos, 2003).

Se debe tomar en cuenta que puede existir más de una solución que tenga el mismo valor

mínimo en especial cuando los valores de las aristas se repiten (Alonso Revenga, 2008).

**Step 3: Creative Solutions Search**

en este caso tenemos pensado a modo de boceto, implementar:

**Algoritmo de flujo optimo:** para determinar la solucion mas eficiente entre dos puntos, esto podria tener en cuenta factores como el costo de construccion de las tuberias y la capacidad de “flujo”.

Por otro lado, podemos optar por un algoritmo de:

**Propagacion de cambio**, ya que esto simularia como los cambios en una parte del grafo afectan a otras partes, asi podriamos ajustar de mejor manera y de formas mas dinamica las conexiones de las tuberias. pensamos tambien implementar un:

**algoritmo de optimizacion de redes dinamicas**: asi ajustariamos la red de tuberias en funcion de lo que se este pidiendo en el momento y basado en las condiciones del juego.

para simular la propagacion de flujo a traves de las tuberias pensamos usar un **algoritmo de exploracion de tuberias**, de esta manera se puede determinar la conectividad y la disponibilidad de flujo desde un punto inicial a lo largo de las tuberias, considerando los diferentes tipos y su capacidad. pensamos tambien diseñar un **algoritmo que simule la construccion de la red de tuberias de manera dinamica**. El DFS podría ser utilizado para explorar posibles conexiones y construir segmentos de tubería a medida que avanza, adaptándose a las condiciones del juego.

por ultimo pensamos implementar un **algoritmo que optimice la red de tuberias** para minimizar costos o maximizar la eficiencia del flujo. Podría considerar factores como la longitud total de las tuberías o la capacidad de flujo para determinar las conexiones más eficientes.

**Step 4: Transition from Ideas to Preliminary Designs**

aqui vamos a dividir en secciones y con base a esto eliminar las ideas menos factibles.

1. Algoritmo de Flujo Óptimo: Considera aspectos económicos y de eficiencia, adaptándose bien a la temática de construcción de tuberías aunque puede ser más complejo de implementar y requerir ajustes para equilibrar adecuadamente el flujo y los costos.

2. Algoritmo de Propagación de Cambio: Ofrece flexibilidad dinámica en la adaptación del sistema de tuberías a eventos o cambios en el juego. Pero podría requerir una gestión cuidadosa para evitar comportamientos impredecibles en el juego.

3. Algoritmo de Exploración de Tuberías: es relacionado directamente con la mecánica central del juego Pipe Mania. Puede volverse computacionalmente costoso en grafos grandes.

4. Algoritmo de Construcción de Red: Se alinea con la lógica del jugador en Pipe Mania, construyendo conexiones a medida que avanza la desventaja es que puede requerir ajustes para equilibrar la construcción y la complejidad del juego.

5. Algoritmo de Optimización de Red de Tuberías: Estratégico y se integra bien con la mecánica del juego.

Puede ser complejo y necesita una ponderación cuidadosa de factores para una optimización eficiente.

6. Recorrido en Amplitud (BFS): Es efectivo para encontrar la ruta más corta entre dos puntos en grafos no ponderados. Útil para la expansión controlada desde un punto de inicio. No es eficiente en grafos densos o con aristas de gran peso.

7. Recorrido en Profundidad (DFS): Explora en profundidad y puede ser eficiente para encontrar soluciones en grafos grandes. Útil para la búsqueda de caminos y componentes conectados.No garantiza encontrar la ruta más corta y puede caer en ciclos infinitos en grafos no acíclicos.

8. Algoritmo de Dijkstra: Encuentra la ruta más corta en grafos ponderados con pesos no negativos. Adecuado para redes de tuberías con distancias representativas. No maneja bien los pesos negativos y puede ser costoso en términos de tiempo en grafos grandes.

9. Algoritmo de Warshall: Encuentra todos los caminos más cortos entre todos los pares de vértices en grafos ponderados. Útil para determinar la conectividad entre puntos. Puede ser computacionalmente costoso en grafos grandes y no maneja bien los pesos negativos.

10. Árbol de Recubrimiento Mínimo (MST) - Prim: Encuentra un árbol de recubrimiento mínimo eficientemente. Adecuado para optimizar la conectividad en una red de tuberías. No maneja bien grafos con aristas de peso negativo.

11. Árbol de Recubrimiento Mínimo (MST) - Kruskal: Encuentra un árbol de recubrimiento mínimo eficientemente y maneja bien los grafos con aristas de peso negativo. Puede ser menos eficiente en grafos densos.

**Step 5: Evaluation and Selection of the Best Solution**

este paso se hara con el fin de encontrar la mejor solucion para implementar, crearemos un sistema de evaluacion numerico basandonos en los siguientes criterios:

1. eficiencia
2. usabilidad
3. mantenibilidad
4. escalabilidad

enumerando cada punto del 1 al 5, siendo 1 muy malo y 5 muy bueno. al final de cada caso se hara una suma y entre mas alto sea el valor mas conveniente de utilizar sera.

Recorrido en Amplitud (BFS):

1. 4 - Eficiente para encontrar la ruta más corta en grafos no ponderados.
2. 5 - Fácil de entender y usar.
3. 4 - Mantenible, pero puede requerir ajustes para casos específicos.
4. 3 - Escalabilidad moderada.

Recorrido en Profundidad (DFS):

1. 3 - Eficiente para explorar grafos grandes, pero no garantiza la ruta más corta.
2. 4 - Intuitivo, pero puede resultar complejo en algunos casos.
3. 4 - Mantenible, pero cuidado con ciclos en grafos no acíclicos.
4. 3 - Escalabilidad moderada.

Algoritmo de Dijkstra:

1. 4 - Eficiente para encontrar la ruta más corta en grafos ponderados no negativos.
2. 4 - Requiere conocimiento de programación y gráficos.
3. 4 - Mantenible, pero puede ser costoso en términos de tiempo.
4. 3 - Escalabilidad moderada.

Algoritmo de Warshall:

1. 3 - Computacionalmente costoso, pero eficiente para la conectividad entre puntos.
2. 3 - Puede ser complejo de entender.
3. 3 - Mantenible, pero cuidado con la complejidad.
4. 2 - Escalabilidad limitada en grafos grandes.

Árbol de Recubrimiento Mínimo (MST) - Prim:

1. 5 - Eficiente para encontrar un árbol de recubrimiento mínimo.
2. 4 - Requiere un buen entendimiento de gráficos y programación.
3. 4 - Mantenible y relativamente claro.
4. 4 - Escalabilidad razonable.

Suma de Evaluaciones:

1. 19
2. 20
3. 19
4. 15

Ranking de Eficiencia:

Recorrido en Amplitud (BFS)

Recorrido en Profundidad (DFS)

Algoritmo de Dijkstra

Árbol de Recubrimiento Mínimo (MST) - Prim

Algoritmo de Warshall

6. Algoritmo de Flujo Óptimo:

1. 4 - Eficiente al considerar costo y capacidad de flujo.
2. 3 - Requiere un entendimiento más profundo de la economía de las tuberías.
3. 4 - Mantenible, pero la optimización puede ser compleja.
4. 3 - Escalabilidad moderada.

7. Algoritmo de Propagación de Cambio:

1. 3 - Eficiente para ajustar dinámicamente las conexiones.
2. 4 - Intuitivo, pero puede requerir ajustes para evitar comportamientos impredecibles.
3. 3 - Mantenible, pero cuidado con la complejidad en cambios dinámicos.
4. 3 - Escalabilidad moderada.

8. Algoritmo de Exploración de Tuberías:

1. 3 - Eficiente pero puede ser costoso en grafos grandes.
2. 4 - Relacionado directamente con la mecánica central del juego.
3. 3 - Mantenible, pero puede requerir optimizaciones para grandes volúmenes de tuberías.
4. 3 - Escalabilidad moderada.

9. Algoritmo de Construcción de Red:

1. 4 - Eficiente para construir conexiones dinámicamente.
2. 4 - Se alinea con la lógica del jugador en Pipe Mania.
3. 4 - Mantenible y adaptable a cambios en la construcción.
4. 3 - Escalabilidad moderada.

10. Algoritmo de Optimización de Red de Tuberías:

1. 4 - Eficiente para optimizar la red de tuberías.
2. 4 - Estratégico y se integra bien con la mecánica del juego.
3. 4 - Mantenible con una ponderación cuidadosa de factores.
4. 3 - Escalabilidad razonable.

Suma de Evaluaciones:

1. 18
2. 19
3. 18
4. 15

por lo que ya vimos, podemos concluir que utilizaremos los que obtuvieron este puntaje:

1. BFS 19
2. DFS 20
3. Dijskstra 19
4. Warshall 18
5. MST- prim 15

**Step 6. Preparation of Reports and Specifications**

**Step 7. Implementation of the Design**

**Bibliografia**

<https://es.wikipedia.org/wiki/Grafo>

<https://www.encora.com/es/blog/dfs-vs-bfs>

<http://atlas.uned.es/algoritmos/voraces/dijkstra.html#:~:text=El%20algoritmo%20de%20Dijkstra%2C%20tambi%C3%A9n,con%20pesos%20en%20cada%20arista>.

<https://medium.com/algoritmo-floyd-warshall/algoritmo-de-floyd-warshall-e1fd1a900d8>

<https://prezi.com/hg8iod-dutfx/arboles-recubridores-minimales-algoritmos-de-kruskal-y-prim/#:~:text=Un%20%C3%A1rbol%20recubridor%20m%C3%ADnimo%20o,tiene%20un%20bosque%20recubridor%20m%C3%ADnimo>.

<https://arodrigu.webs.upv.es/grafos/doku.php?id=algoritmo_prim>

<http://repositorio.puce.edu.ec/bitstream/handle/22000/12141/Tesis-Algoritmo%20de%20Kruskal%20v1%20(1).pdf?sequence=1&isAllowed=y#:~:text=El%20algoritmo%20de%20Kruskal%20es,varios%20nodos%20comunicados%20entre%20s%C3%AD>.