

Método de la Ingeniería

FASE 1: IDENTIFICACIÓN DEL PROBLEMA

Descripción del contexto problemático (causas y síntomas).

Un agujero de gusano tiene por lo menos dos extremos conectados a una única garganta. En cada uno de los extremos se encuentra un sistema de estrellas y este puede tener más de un punto de finalización de agujero de gusano dentro de sus límites. No hay agujeros de gusanos con ambos puntos de fin en el mismo sistema de estrellas, entre cualquier par de sistemas de estrellas existe máximo un agujero de gusano en cada dirección y puede tener más de un punto de finalización de agujero dentro de sus límites. Así mismo -por alguna razón desconocida- si se empieza desde nuestro sistema sola siempre es posible finalizar en cualquier sistema de estrellas siguiendo una secuencia de agujeros de gusano, por lo tanto, una de las hipótesis es que tal vez la tierra es el centro del universo.

Por otro lado, se debe tener en cuenta que -una de las hipótesis de los científicos- es la posibilidad de viajar en el tiempo si se cruza alguno de estos agujeros de gusano. Sin embargo, el tiempo que se podría viajar es incierto. Así que, un agujero de gusano en específico puede causar que una persona viaje 15 años en el futuro y otro puede hacerla viajar 42 años en el pasado.

Identificación del problema.

Una física brillante quiere usar los agujeros de gusano para estudiar el Big Bang que ocurrió hace tanto tiempo (el llamado inicio del todo) y dado que “Warp Drive” (forma teórica de propulsión superlumínica. Este empuje permitiría propulsar una nave espacial a una velocidad equivalente a varios múltiplos de la velocidad de la luz, mientras se evitan los problemas asociados con la dilatación relativista del tiempo. Este tipo de propulsión se basa en curvar o distorsionar el espacio-tiempo, de tal manera que permita a la nave acercarse al punto de destino) no ha sido inventado, entonces, no es posible para la física viajar de un sistema de estrellas a otro directamente. Sin embargo, esto puede ocurrir usando los agujeros de gusano.

La científica desea descubrir si existe un ciclo de agujero de gusanos en el universo que le permitan viajar en el pasado. De este modo, viajando tantas veces por este ciclo de agujeros de gusano la científica sería capaz de ir al pasado tan lejos como sea necesario para llegar al principio del universo y observar el Big Bang con sus propios ojos. Nuestra tarea es encontrar este ciclo de agujeros de gusanos ayudando a la científica lograr su objetivo.

Requerimientos funcionales

| | |
|------------------|---|
| Nombre | RF#1 Iniciar aplicación |
| Resumen | El sistema inicia por primera vez con los campos vacíos para ingresar datos |
| Entrada | |
| Resultado | Inicio del programa |

| | |
|------------------|---|
| Nombre | RF#2 Seleccionar tipo de algoritmo a usar |
| Resumen | El sistema permite definir el algoritmos -Dijkstra o Bellman Ford- con el cual se va a resolver el problema |
| Entrada | Tipo de algoritmo a usar |
| Resultado | Se ha seleccionado el algoritmo satisfactoriamente |

| | |
|------------------|---|
| Nombre | RF#3 Seleccionar versión del grafo a usar |
| Resumen | El sistema permite definir uno de los tres tipos de representación de grafo con el cuál se va a trabajar en este problema |
| Entrada | Tipo de representación de grafo: Matriz de adyacencia Lista de adyacencia Arreglo de listas |
| Resultado | Representación de grafo seleccionada satisfactoriamente |

| | |
|------------------|---|
| Nombre | RF#4 Cargar archivo de entrada |
| Resumen | El sistema permite cargar un archivo de texto -generado con anterioridad- el cual va a contener los casos de prueba |
| Entrada | Archivo de entrada con los casos de prueba |
| Resultado | Se cargo el archivo satisfactoriamente |

| | |
|------------------|---|
| Nombre | RF#5 Visualizar respuestas de casos de pruebas |
| Resumen | El sistema permite visualizar las soluciones encontradas a los casos de pruebas cargados o introducidos por consola |
| Entrada | |
| Resultado | Visualización de la salida obtenida satisfactoriamente |

| | |
|------------------|---|
| Nombre | RF#6 Generar salida de casos de prueba |
| Resumen | El sistema permite generar un archivo de texto de acuerdo con las respuestas obtenidas dado a los casos de prueba |
| Entrada | Solución de los casos de prueba |
| Resultado | Archivo de texto generado |

| Nombre | RF#7 Generar casos de prueba |
|-----------|---|
| Resumen | El sistema permite generar casos de prueba utilizando un generador que no esta relaciona con la solución implementada |
| Entrada | Número de casos de prueba |
| Resultado | Archivo de texto con los casos generados satisfactoriamente |

FASE 2: RECOPIACION DE LA INFORMACION NECESARIA

Para llevar a cabo la solución del problema presentado se debe tener conocimiento acerca de algunos tipos de grafos y sus algoritmos. Además, también se deben tener en cuenta sus recorridos.

-Tipos de grafos

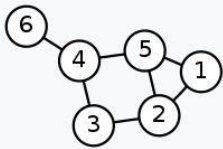
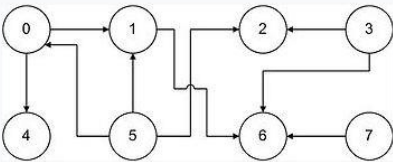
Matriz de adyacencia

Es una matriz cuadrada que se utiliza como una forma de representar relaciones binarias. Para construir una matriz a partir de un grafo se debe hacer lo siguiente:

- 1- Se crea una matriz cero, cuyas columnas y filas representan los nodos (vértices) del grafo.
- 2- Por cada arista que una a dos nodos, se suma 1 al valor que hay actualmente en la ubicación correspondiente de la matriz. Si tal arista es un bucle y el grafo es no dirigido, entonces se suma 2 en vez de 1.
- 3- Finalmente, se obtiene una matriz que representa el número de aristas (relaciones) entre cada par de nodos (vértices-elementos).

Existe una matriz de adyacencia única para cada grafo (sin considerar las permutaciones de filas o columnas), y viceversa.

La siguiente tabla muestra dos grafos y su respectiva matriz de adyacencia. Se evidencia que, en el primero caso, como se trata de un grafo no dirigido, la matriz obtenida es simétrica:

| Grafo no dirigido | Matriz de adyacencia |
|---|--|
|  | $A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$ |
| Grafo dirigido | Matriz de adyacencia |
|  | $A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ |

Propiedades:

- 1- Para un grafo no dirigido la matriz de adyacencia es simétrica.

- 2- El número de camino $C_{i,j}(k)$, atravesando k aristas desde el nodo i al nodo j , viene dado por un elemento de la potencia k -ésima de la matriz de adyacencia

$$C_{i,j}(k) = [A^k]_{ij}$$

Lista de adyacencia

En teoría de grafos, una lista de adyacencia es una representación de todas las aristas o arcos de un grafo mediante una lista.

Si el grafo es no dirigido, cada entrada es un conjunto o multiconjunto de dos vértices conteniendo los dos extremos de la arista correspondiente. Si el grafo es dirigido, cada entrada es una tupla de dos nodos, uno denotando el nodo fuente y el otro denotando el nodo destino del arco correspondiente.

Típicamente, las listas de adyacentes no son ordenadas.



-Recorridos de grafos

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

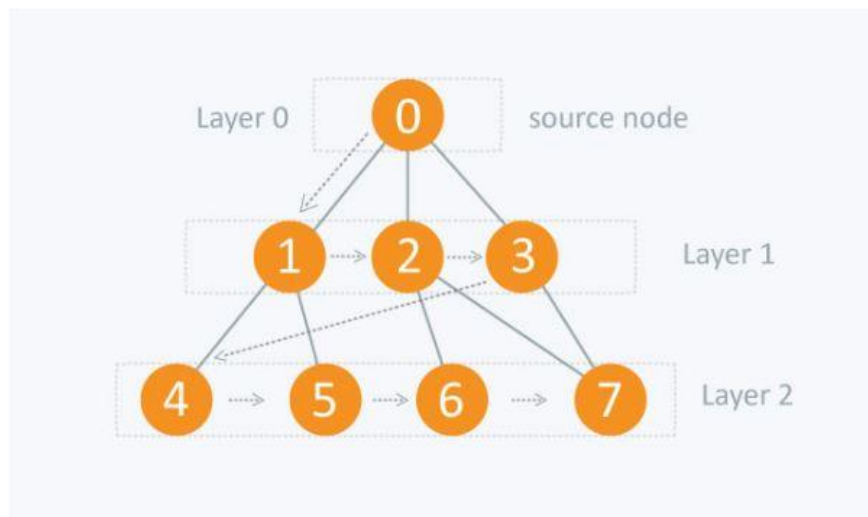
Durante un recorrido, es importante que realice un seguimiento de los vértices que se han visitado. La forma más común de rastrear vértices es marcarlos.

Primera búsqueda en amplitud (BFS)

Es el enfoque más utilizado. Este es un algoritmo de desplazamiento en el que se debe comenzar desde un nodo seleccionado (origen o nodo de inicio) y recorrer el grafo explorando los nodos vecinos (nodos que están conectados directamente al nodo de origen).

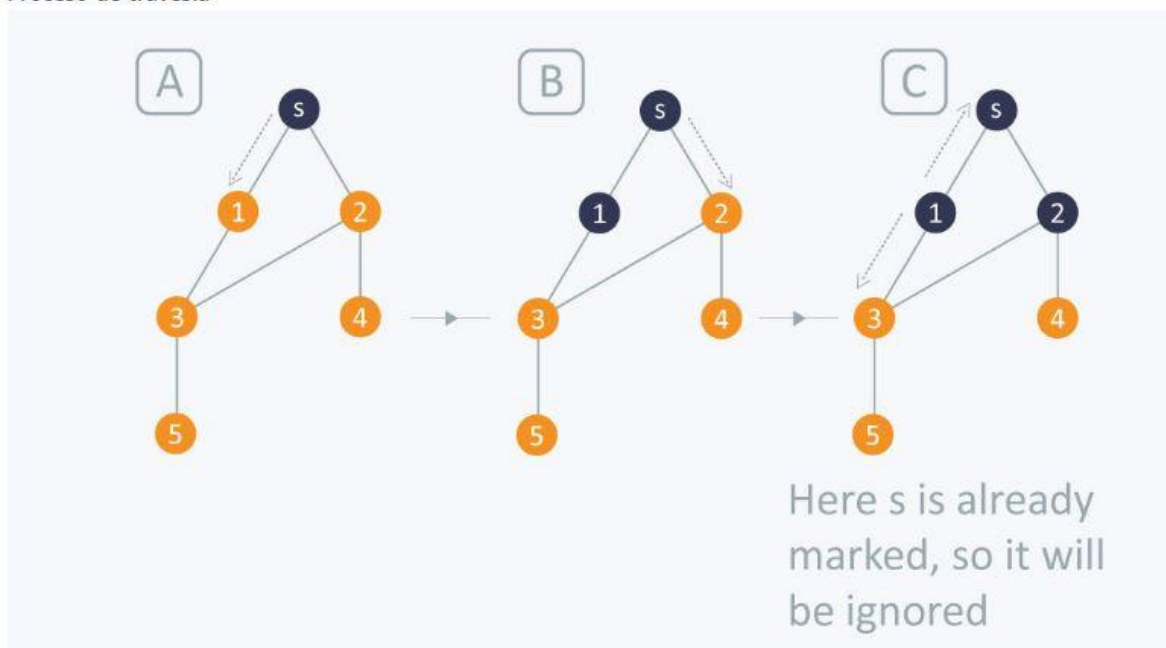
Luego se debe mover hacia los nodos vecinos del siguiente nivel.

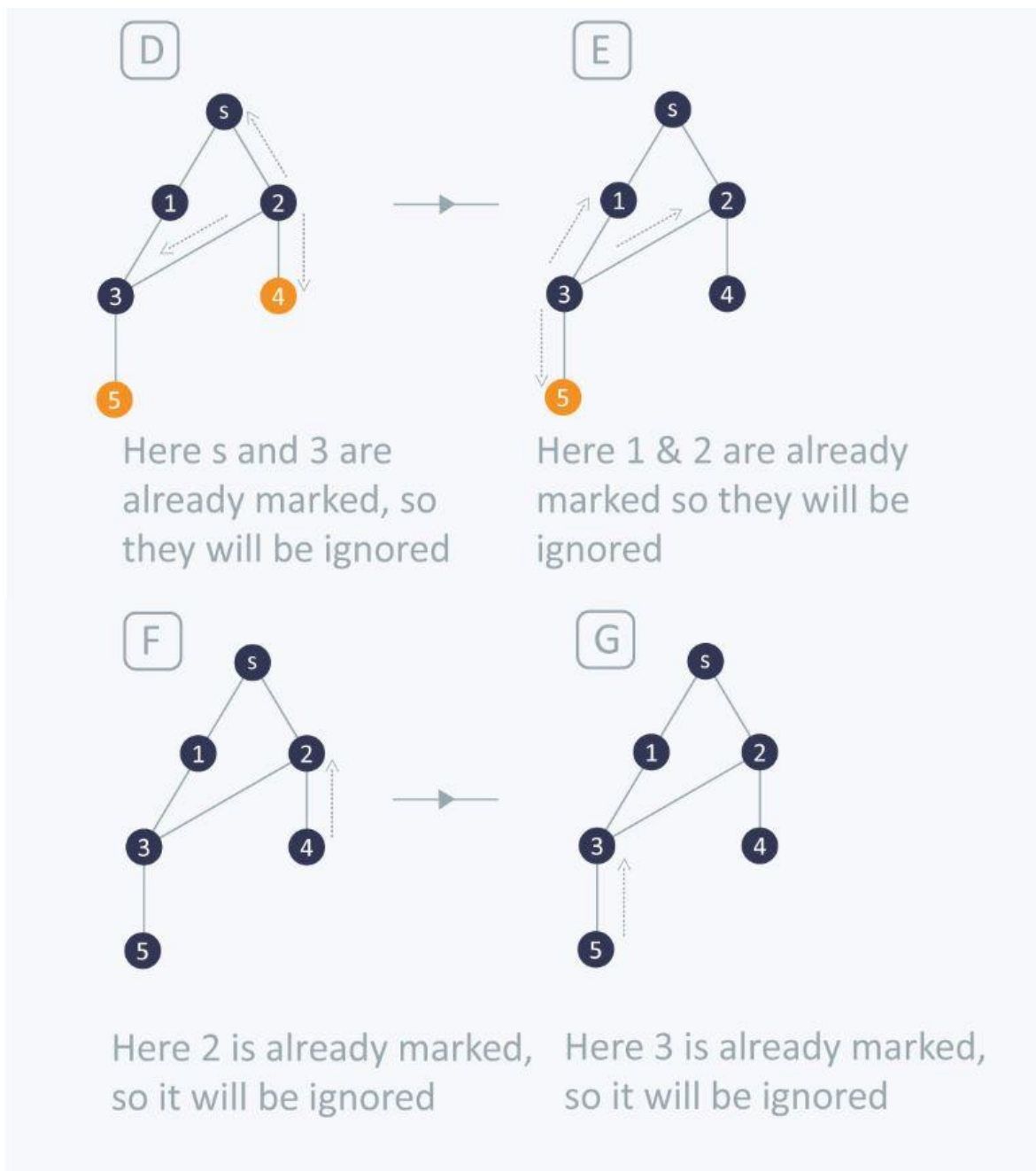
- 1- Primero se mueve horizontalmente y visita todos los nodos del nivel actual
- 2- Moverse al siguiente nivel



La distancia entre los nodos del nivel 1 es comparativamente menor que la distancia entre los nodos del nivel 2. Por lo tanto, en BFS, se debe atravesar todos los nodos en el nivel 1 antes de moverse a los nodos del nivel 2.

Proceso de travesía





La complejidad temporal de BFS es $O(V + E)$, donde V es el número de nodos y E es el número de bordes

Primera búsqueda en profundidad (DFS)

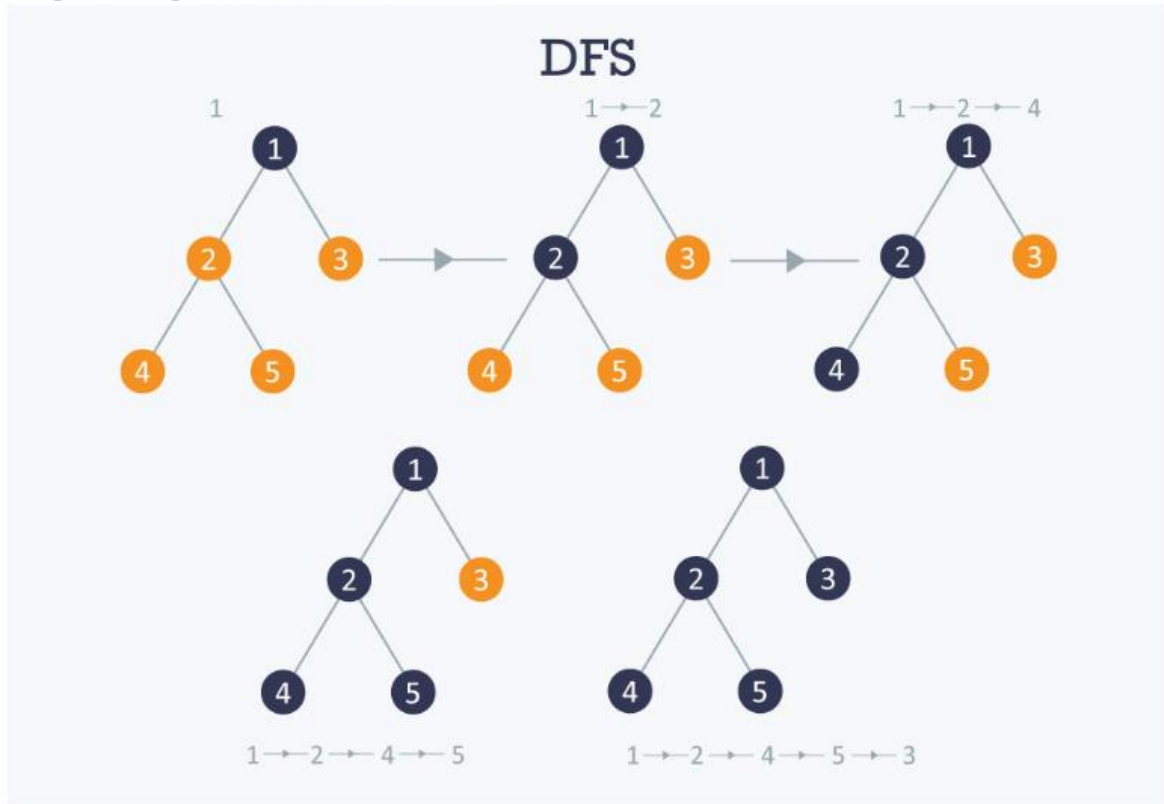
El algoritmo DFS es un algoritmo recursivo que utiliza la idea de dar marcha atrás. Implica búsquedas exhaustivas de todos los nodos al avanzar, si es posible, o al retroceder.

Aquí, la palabra retroceder significa que cuando se está moviendo hacia adelante y no hay más nodos a lo largo de la ruta actual, se mueve hacia atrás en la misma ruta para encontrar nodos para atravesar. Todos los nodos serán visitados en la ruta actual hasta que todos los nodos no visitados hayan sido atravesados, después de lo cual se seleccionará la siguiente ruta.

Esta naturaleza recursiva de DFS se puede implementar utilizando pilas. La idea básica es la siguiente:

- 1- Elija un nodo de inicio y empuje todos sus nodos adyacentes en una pila.
- 2- Pop un nodo de la pila para seleccionar el siguiente nodo para visitar y empujar todos sus nodos adyacentes en una pila.
- 3- Se debe repetir este proceso hasta que la pila esté vacía, asegurándose de que los nodos que se visitan estén marcados. Esto evitará que se visite el mismo nodo más de una vez. De lo contrario, visitar el mismo nodo más de una vez puede ocasionar que se termine en un bucle infinito.

La siguiente imagen muestra cómo funciona DFS.



La complejidad del DFS es $O(V + E)$, cuando se implementa utilizando una lista de adyacencia.

Algoritmos de camino más corto

El problema del camino más corto consiste en encontrar un camino entre 2 vértices en un grafo tal que la suma total de los pesos de las aristas sea la mínima.

Se debe tener en cuenta que este problema podría resolverse fácilmente usando **(BFS)** si todos los pesos de las aristas fueran **(1)**. Sin embargo, los pesos pueden tomar cualquier valor. Por lo tanto, existen tres algoritmos diferentes que resuelven esta dificultad, según sea el caso.

- **Algoritmo de Bellman Ford:**

El algoritmo de Bellman Ford se usa para encontrar las rutas más cortas desde el vértice de origen a todos los otros vértices en un grafo ponderado. Depende del siguiente concepto: la ruta más corta contiene como máximo **n-1** Bordes, porque el camino más corto no podría tener un ciclo.

A partir de esto surge una pregunta, ¿por qué el camino más corto no debería tener un ciclo? Y la respuesta a esto es que **NO** es necesario volver a pasar un vértice, ya que se podría encontrar la ruta más corta a todos los demás vértices sin la necesidad de una segunda visita para ninguno de ellos.

Paso del algoritmo:

- 1- El bucle exterior atraviesa desde 0: **n-1**.
- 2- Se deben recorrer todas las aristas verificando si la siguiente distancia del nodo es mayor a la distancia del nodo actual más el peso de la arista (**Distancia del nodo > Distancia del nodo actual + peso de la arista**). En este caso, se va a reemplazar la distancia por la distancia del nodo actual más el peso de la arista (**Distancia del nodo actual + peso de la arista**).

Este algoritmo depende del principio de relajación, donde la distancia más corta para todos los vértices se reemplaza gradualmente por valores más precisos hasta llegar a la solución óptima. Al principio, todos los vértices tienen una distancia de "Infinito", pero solo la distancia del vértice fuente, de inicio o principal es igual a 0. Después se actualizan todos los vértices conectados con las nuevas distancias (fuente, vértice, distancia + bordes de peso) y se aplica el mismo concepto para los nuevos vértices con nuevas distancias y así sucesivamente.

La complejidad del tiempo del algoritmo Bellman Ford es relativamente alta $O(V \times E)$, en el caso de que $E = V^2$, $O(E^3)$.

NOTA: Una aplicación muy importante del Bellman Ford es verificar si hay un ciclo negativo en el grafo.

• **Algoritmo Dijkstra**

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

Pasos del algoritmo:

- 1- Establecer todas las distancias de los vértices como infinito, excepto la del vértice de origen. Esta, se establece como 0.
- 2- Se empuja el vértice de origen en una cola de prioridad mínima en el formulario (distancia, vértice), ya que la comparación en la cola de prioridad mínima estará de acuerdo con las distancias de los vértices.
- 3- Se hace Pop al vértice con la distancia mínima de la cola de prioridad (en primera instancia el vértice es igual al de fuente o inicio).
- 4- Se actualizan las distancias de los vértices conectados al vértice emergente en caso de que la (**distancia de vértice actual + peso de la arista**) < **distancia del vértice siguiente**. Después, se empuja el vértice con la nueva distancia a la cola de prioridad.
- 5- Si el vértice resaltado es visitado anteriormente, simplemente se continua sin usarlo.
- 6- Se vuelve a aplicar el mismo algoritmo hasta que la cola de prioridad esté vacía.

La complejidad del tiempo del algoritmo de Dijkstra es $O(V^2)$ pero con la cola de prioridad mínima se reduce a $O(V + E \log(V))$.

Ahora bien, se debe tener en cuenta que, si se desea encontrar la ruta más corta entre todos los pares de vértices, ambos métodos -anteriormente mencionados- serían costosos en términos de tiempo. A continuación, se va a mencionar otro algoritmo diseñado para este caso.

- **Algoritmo de Floyd-Warshall**

El algoritmo se usa para encontrar las rutas más cortas entre todos los pares de vértices en un grafo, donde cada borde del grafo tiene un peso que es positivo o negativo. La mayor ventaja de usar este algoritmo es que todas las distancias más cortas entre cualquier par de vértices podrían ser calculados en $O(V^3)$, donde V es el número de vértices en un grafo.

Los pasos del algoritmo:

Para un grafo con N vértices

- 1- Se inicializan los caminos más cortos entre cualquier par de vértices con el infinito.
- 2- Encontrar todos los pares de caminos más cortos que use 0 vértices intermedios, luego se debe encontrar la tura más corta que use 1 vértice intermedio y así sucesivamente hasta que usar todos los N vértice como nodos intermedios.
- 3- Minimizar los caminos más cortos entre cualquier par de operaciones previas.
- 4- Para cualquier par de vértices (i, j) , uno deberá minimizar las distancias entre este par usando el primer K nodos, por lo que el camino más corto será: $\min(dist[i][k] + dist[k][j], dist[i][j])$

$dist[i][k]$ representa el camino más corto que solo usa los primeros K vértices, $dist[i][k]$ representa el camino más corto entre el par k, j . Como el camino más corto será una concatenación del camino más corto de i a k , luego de k a j .

La complejidad del tiempo del algoritmo es $O(V^3)$ donde V es el número de vértices en un grafo.

Árbol de expansión mínima

¿Qué es un árbol de expansión?

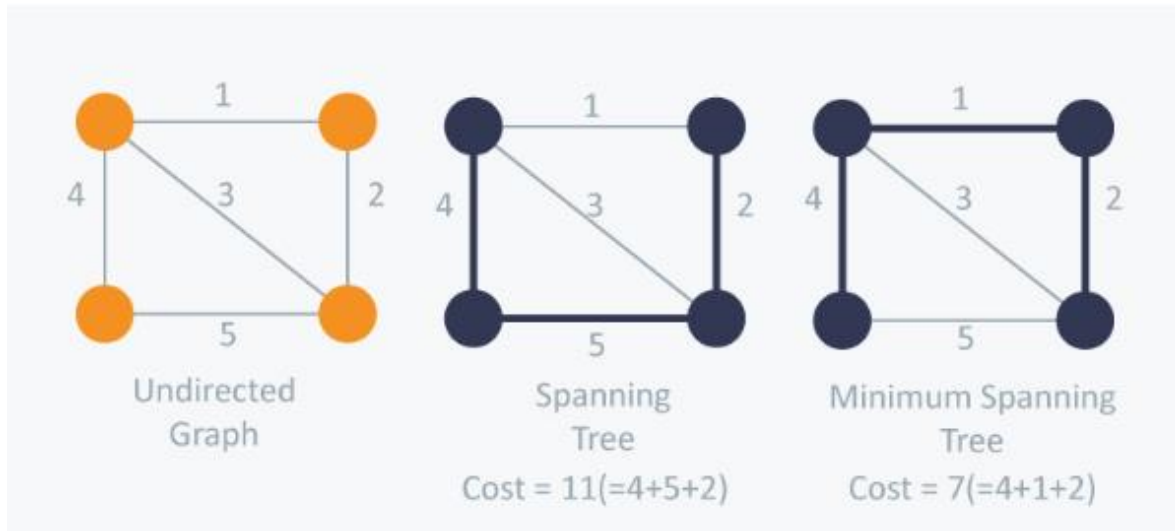
Dado un grafo no dirigido y conectado $G = (V, E)$, un árbol de expansión del grafo G es un árbol que se extiende G (es decir, incluye cada vértice de G) y es un subgrafo de G (Cada borde en el árbol pertenece a G).

¿Qué es un árbol de expansión mínima?

El costo del árbol de expansión es la suma de los pesos de todos los bordes del árbol. Puede haber muchos árboles que se extienden. El árbol de expansión mínimo es el árbol de expansión en el que el costo es mínimo entre todos los árboles de expansión. También puede haber muchos árboles de expansión mínima.

El spanning tree mínimo tiene aplicación directa en el diseño de redes. Se utiliza en algoritmos que se aproximan al problema del vendedor ambulante, al problema de corte mínimo multiterminal y al ajuste perfecto ponderado de costo mínimo. Otras aplicaciones prácticas son:

1. Análisis de conglomerados
2. Reconocimiento de escritura a mano
3. Segmentación de imagen



Hay dos algoritmos famosos para encontrar el árbol de expansión mínima:

El algoritmo de Kruskal

El algoritmo de Kruskal construye el árbol de expansión agregando aristas uno por uno en un árbol de expansión en crecimiento. El algoritmo de Kruskal sigue un enfoque codicioso, ya que en cada iteración encuentra una arista que tiene menos peso y la agrega al creciente árbol de expansión.

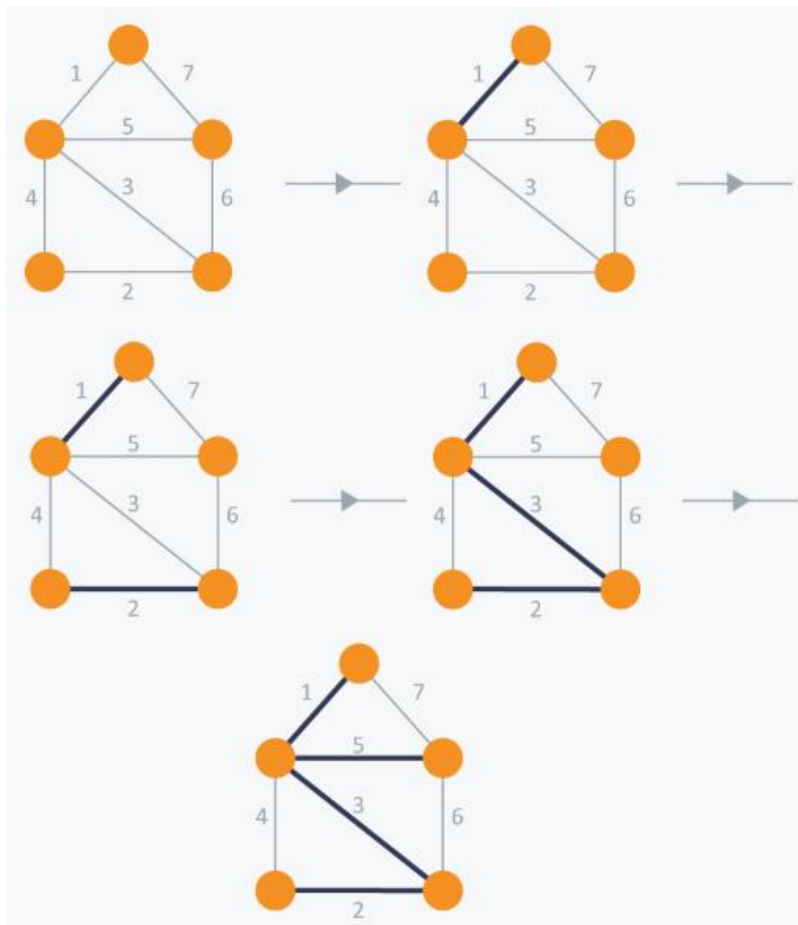
Pasos del algoritmo:

- Ordena las aristas del grafo con respecto a sus pesos.
- Comienza agregando bordes al MST desde la arista con el peso más pequeño hasta la arista del peso más grande.
- Solo agrega aristas que no formen un ciclo, aristas que solo conecten componentes desconectados.

¿Así que ahora la pregunta es cómo comprobar si vértices están conectados o no?

La respuesta es que se podría hacer usando DFS que comienza desde el primer vértice, luego verificar si el segundo vértice es visitado o no. Pero DFS hará que la complejidad del tiempo sea grande, ya que tiene un orden de $O(V+E)$ donde V es el número de vértices y E el número de aristas. Así que la mejor solución es "**Conjuntos disjuntos**": los conjuntos disjuntos son conjuntos cuya intersección es el conjunto vacío, lo que significa que no tienen ningún elemento en común.

Ejemplo:



En el algoritmo de Kruskal, la operación que consume más tiempo es la ordenación por lo que la complejidad total de las operaciones será $O(E \log(V))$, que es la complejidad de tiempo global del algoritmo.

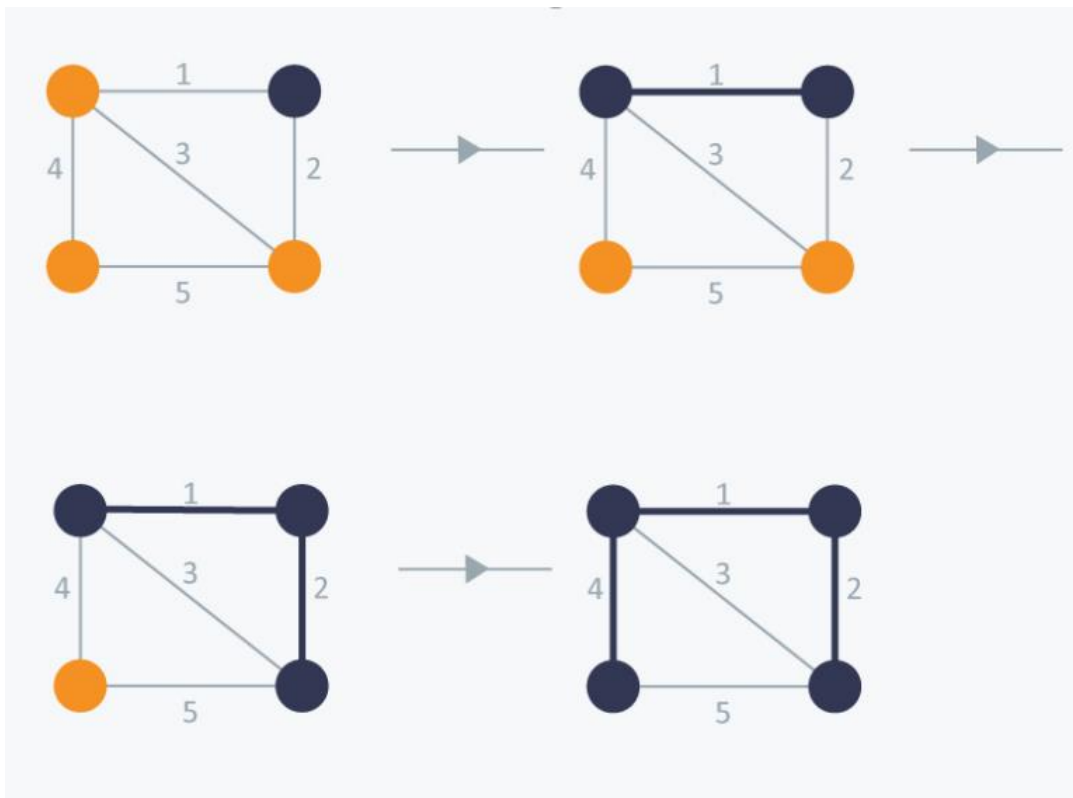
Algoritmo de Prim

El algoritmo de Prim también usa el enfoque codicioso para encontrar el árbol de expansión mínima. En el algoritmo de Prim, se crece el árbol de expansión desde una posición inicial. A diferencia de una **arista** en Kruskal, agregamos **vértice** al creciente árbol de expansión en Prim's.

Pasos del algoritmo:

- Mantener dos conjuntos desarticulados de vértices. Uno que contiene vértices que están en el árbol de expansión creciente y otro que no están en el árbol de expansión creciente.
- Seleccionar el vértice más barato que está conectado al árbol de expansión creciente y no está en el árbol de expansión creciente y agregarlo al árbol de expansión creciente. Esto se puede hacer usando colas de prioridad. Insertar los vértices, que están conectados al creciente árbol de expansión, en la Cola de prioridad.
- Comprobar si hay ciclos. Para hacerlo, marque los nodos que ya se han seleccionado e inserte solo los nodos en la Cola de prioridad que no están marcados.

Ejemplo:



La complejidad de tiempo del algoritmo de Prim es $O((V+E)\log(V))$ porque cada vértice se inserta en la cola de prioridad solo una vez y la inserción en la cola de prioridad lleva tiempo logarítmico.

FASE 3: BÚSQUEDA DE SOLUCIONES CREATIVAS

- Alternativa No. 001 (Depth First Search)

Una Búsqueda en profundidad es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol (teoría de grafos) de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa (Backtracking), de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado. Tomado de Wikipedia.

Se podría buscar si hay una arista que tenga peso negativo, y si la encuentra detiene el recorrido y devuelve un valor de verdad informando si se encontró una arista negativa la cual sirva para, en un ciclo, viajar al pasado.

- Alternativa No. 002 (Breadth First Search)

Búsqueda en anchura es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo (usado frecuentemente sobre árboles). Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran 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 árbol.

Formalmente, BFS es un algoritmo de búsqueda sin información, que expande y examina todos los nodos de un árbol sistemáticamente para buscar una solución. El algoritmo no usa ninguna estrategia heurística. Tomado de Wikipedia.

Se podría buscar si hay una arista que tenga peso negativo, y si la encuentra detiene el recorrido y devuelve un valor de verdad informando si se encontró una arista negativa la cual sirva para, en un ciclo, viajar al pasado.

- **Alternativa No. 003 (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, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo describió por primera vez en 1959. Tomado de Wikipedia. Si se visita un nodo más de una vez el cual tiene un peso de su arista negativa, es porque se encontró un ciclo y este es uno de los caminos que servirá para que la científica pueda viajar al pasado.

- **Alternativa No. 004 (Bellman Ford)**

El algoritmo de Bellman-Ford (algoritmo de Bell-End-Ford) genera el camino más corto en un grafo dirigido ponderado (en el que el peso de alguna de las aristas puede ser negativo). El algoritmo de Dijkstra resuelve este mismo problema en un tiempo menor, pero requiere que los pesos de las aristas no sean negativos, salvo que el grafo sea dirigido y sin ciclos. Por lo que el Algoritmo Bellman-Ford normalmente se utiliza cuando hay aristas con peso negativo. Este algoritmo fue desarrollado por Richard Bellman, Samuel End y Lester Ford. Tomado de Wikipedia.

El algoritmo por defecto para este tipo de problemas que se requieren buscar ciclos, en este caso para viajar al pasado mediante un camino dado entre diferentes sistemas de estrellas.

- **Alternativa No. 005 (Floyd-Warshall)**

En informática, el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica. Tomado de Wikipedia.

Se podría buscar si hay una arista que tenga peso negativo, y si la encuentra detiene el recorrido y devuelve un valor de verdad informando si se encontró una arista negativa la cual sirva para, en un ciclo, viajar al pasado.

- **Alternativa No. 006 (Prim)**

El algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible, Árbol Recubridor Mínimo o Mínimo Árbol de Expansión (MST en inglés). Si el grafo no es conexo, entonces el algoritmo

encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo no conexo. Adaptado de Wikipedia.

En la búsqueda que hace para crear el mínimo árbol de expansión se tiene una verificación en caso de encontrar una arista negativa, y de encontrarla devuelve valor de verdad indicando si encontró una arista con peso negativo (distancia) para que la científica pueda viajar al pasado.

- **Alternativa No. 007 (Kruskal)**

El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). Tomado de Wikipedia.

En la búsqueda que hace para crear el mínimo árbol de expansión se tiene una verificación en caso de encontrar una arista negativa, y de encontrarla devuelve valor de verdad indicando si encontró una arista con peso negativo (distancia) para que la científica pueda viajar al pasado.

FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES

Después de exponer varias ideas y algoritmos con los cuales se puede abordar este problema y dar una solución, se deben descartar ideas no efectivas para así enfocarse en las que pueden ser mejores para solucionar el problema de la científica.

A continuación, se expondrán las ideas anteriores brindando razones por las cuales se descartan o no:

- **Alternativa No. 001 (Depth First Search)**

De manera que este algoritmo, DFS, recorre los nodos de forma lineal hasta el último y después se devuelve para seguir con los demás nodos adyacentes, no se puede asegurar si cuando encuentre una arista que tiene peso negativo, con esta, se pueda crear un ciclo y así poder lograr que la científica viaje al pasado.

- **Alternativa No. 002 (Breadth First Search)**

De manera que este algoritmo, BFS, solo puede usarse para buscar el camino más corto en grafos que no tienen peso (o distancias de tiempo en este caso), o solo para grafos con pesos constantes, se descarta de inmediato dado que este problema requiere de grafos con pesos diferentes (distancias tanto positivas como negativas).

- **Alternativa No. 003 (Dijkstra)**

Este algoritmo, aunque de manera directa no funcione, se le puede hacer una modificación (la cual aumentara su complejidad temporal) para que pueda trabajar con pesos negativos (distancias en este caso) y así podrá encontrar si existe un ciclo.

- **Alternativa No. 004 (Bellman Ford)**

Este algoritmo, es de los más indicados para este tipo de problemas dado que, trabaja con grafos ponderados donde una o más aristas pueden ser negativas. Este algoritmo es como una derivación de Dijkstra pero que funciona con grafos que tienen aristas negativas, aunque su complejidad temporal es mayor. Además, de implementarse con una lista de adyacencias, su complejidad sería menor que de implementarse con una matriz de adyacencias.

- **Alternativa No. 005 (Floyd-Warshall)**

Este algoritmo supone como precondition que en el grafo no hay ciclos negativos dado que el camino mínimo no está bien definido, el camino puede ser infinitamente pequeño. No obstante, algunos caminos pueden decrementarse, pero no garantiza el camino se reduzca si se vuelve a ejecutar. Por estas razones, además que si se ejecuta más de dos veces el algoritmo el tiempo de búsqueda aumenta, no se tomara en cuenta dado que puede provocar una respuesta equivocada y la científica se podría perder del Big Bang.

- **Alternativa No. 006 (Prim)**

Este algoritmo supone como precondition que el grafo sea no dirigido, entre otros, lo cual va en contra de la problemática que involucra los agujeros de gusano, estos tienen un punto x en el espacio a un punto y , con $x \neq y$.

- **Alternativa No. 007 (Kruskal)**

Este algoritmo supone como precondition que el grafo sea no dirigido, entre otros, lo cual va en contra de la problemática que involucra los agujeros de gusano, estos tienen un punto x en el espacio a un punto y , con $x \neq y$.

FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN

Teniendo en cuenta la fase anterior en la que se explica cuál es la funcionalidad fundamental de cada algoritmo aplicado a un grafo se llega a la conclusión de que las dos mejores soluciones son:

- **Alternativa No. 003 (Dijkstra)**

Algoritmo más efectivo que toma un tiempo de 0.1 segundos para encontrar la solución.

- **Alternativa No. 004 (Bellman Ford)**

Segundo algoritmo eficiente que toma un tiempo de 0.5, 0.9 y 1.1 segundos para encontrar la solución. De acuerdo con la implementación del algoritmo (3 versiones). Sin embargo, la segunda mejor opción es escoger el mejor tiempo que toma el Bellman Ford el cual es 0.5 segundos.

FASE 6: PREPARACIÓN DE INFORME Y ESPECIFICACIONES:

En la carpeta documents esta todos los archivos relacionados.

FASE 7: IMPLEMENTACIÓN

Proyecto en eclipse implementado.

Bibliografía:

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

<https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>

<https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/>

https://es.wikipedia.org/wiki/Matriz_de_adyacencia

https://es.wikipedia.org/wiki/Lista_de_adyacencia