Actividad Evaluativa Grafos

Antes de empezar a describir los distintos conceptos sobre los grafos primero tenemos que describir que es un grafo. Los grafos son estructuras matemáticas con una composición de un conjunto de objetos conocidos como nodos que se relacionan con otros nodos a través de un conjunto de conexiones conocidas como aristas. Se usan ampliamente en distintos ámbitos como la informática para representar problemas como rutas, relaciones, flujos y estructuras jerárquicas.

1. Algoritmos y trayectorias especiales

1.1 Algoritmo de Prim

El algoritmo de Prim es un algoritmo que nos permite encontrar el árbol recubridor mínimo en grafos no dirigidos con aristas ponderadas. Funciona añadiendo iterativamente la arista más barata que conecta un vértice en el árbol con un vértice fuera de él.

El algoritmo de Prim es un algoritmo voraz, esto quiere decir que es un algoritmo que encuentra una solución globalmente óptima a un problema a base de hacer elecciones localmente óptimas. Es decir: el algoritmo siempre hace lo que "parece" mejor en cada momento, sin tener nunca que reconsiderar sus decisiones, y acaba llegando directamente a la mejor solución posible.

Este algoritmo ayuda a solucionar problemas frecuentes de los diseñadores de juegos en línea y los proveedores de radio por Internet, que quieren transferir eficientemente una pieza de información a todos y cada uno de los que puedan estar escuchando. Esto es importante en los videojuegos para que todos los jugadores conozcan la posición más reciente de cada uno de los otros jugadores. Es importante también en la radio por Internet para que todos los oyentes que estén sintonizados estén recibiendo todos los datos que necesitan para reconstruir la canción que estén escuchando. La figura 1 ilustra el funcionamiento del algoritmo.

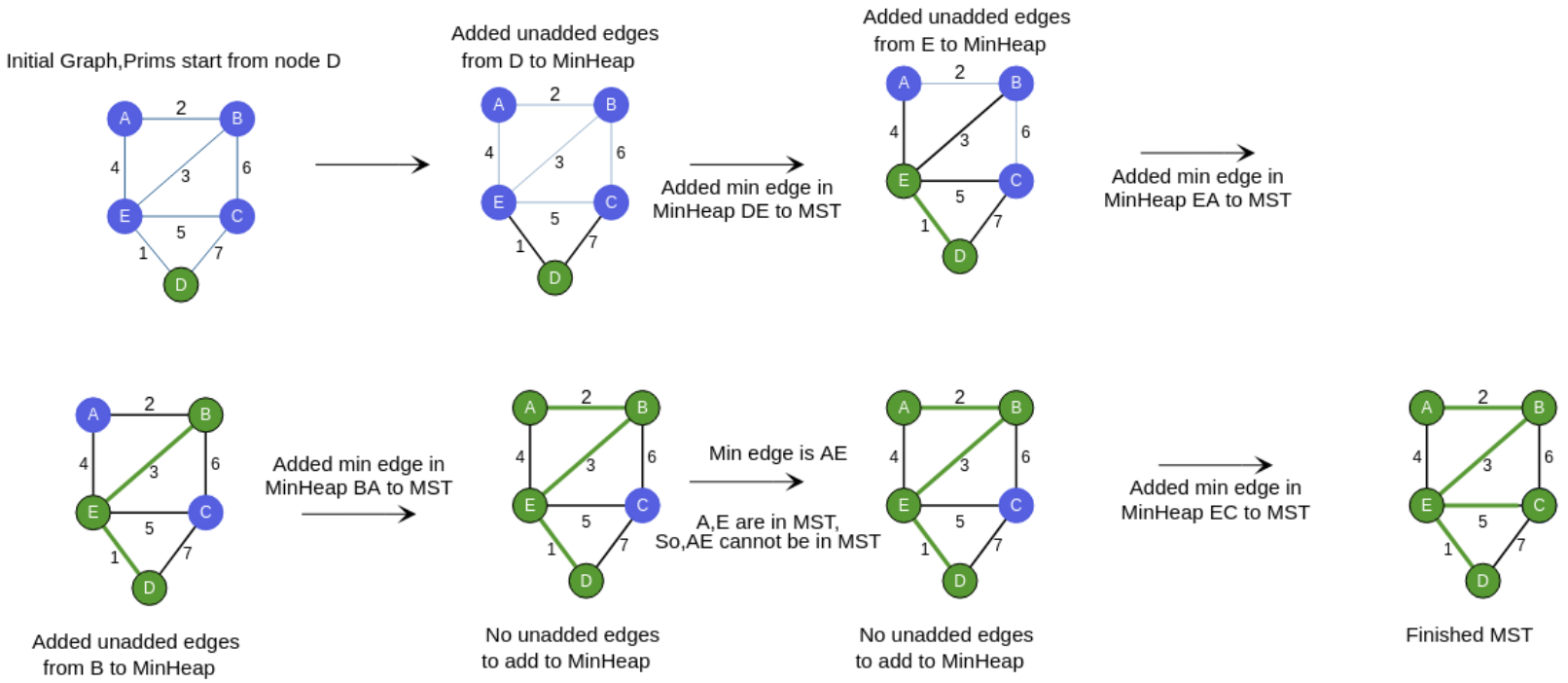

Figura 1: Funcionamiento del Algoritmo de Prim

MST significa Árbol de Expansión Mínima (*Minimum Spanning Tree*) en inglés.

Prim comienza en el nodo D. Se agregan las aristas DE (1) y DC (7) a la MinHeap. Se elige DE y se añade al MST. Desde E se agregan EA (4), EC (5) y EB (3). Se añade EA (4). Desde A se agrega AB (2) y AE (4), pero AE se descarta porque ambos nodos ya están en el MST. Se añade AB (2). Desde B no se agregan nuevas aristas útiles. La siguiente arista mínima es EC (5), que se añade. El MST está completo.

1.2 Algoritmo de Ford-Fulkerson

El algoritmo de Ford-Fulkerson los utilizamos para encontrar el flujo máximo posible en una red de flujo con capacidades asignadas a cada arista. Esta red se modela como un grafo dirigido donde cada arista tiene una capacidad máxima de flujo. La idea principal de este algoritmo es buscar caminos aumentantes desde el nodo fuente al nodo sumidero por donde aún pueda pasar flujo, y continuar aumentando el flujo total por estos caminos hasta que no se pueda encontrar ningún camino adicional.
Este método garantiza encontrar el flujo máximo siempre y cuando todas las capacidades sean números enteros.

La siguiente figura, la figura 2, resume el comportamiento del algoritmo.

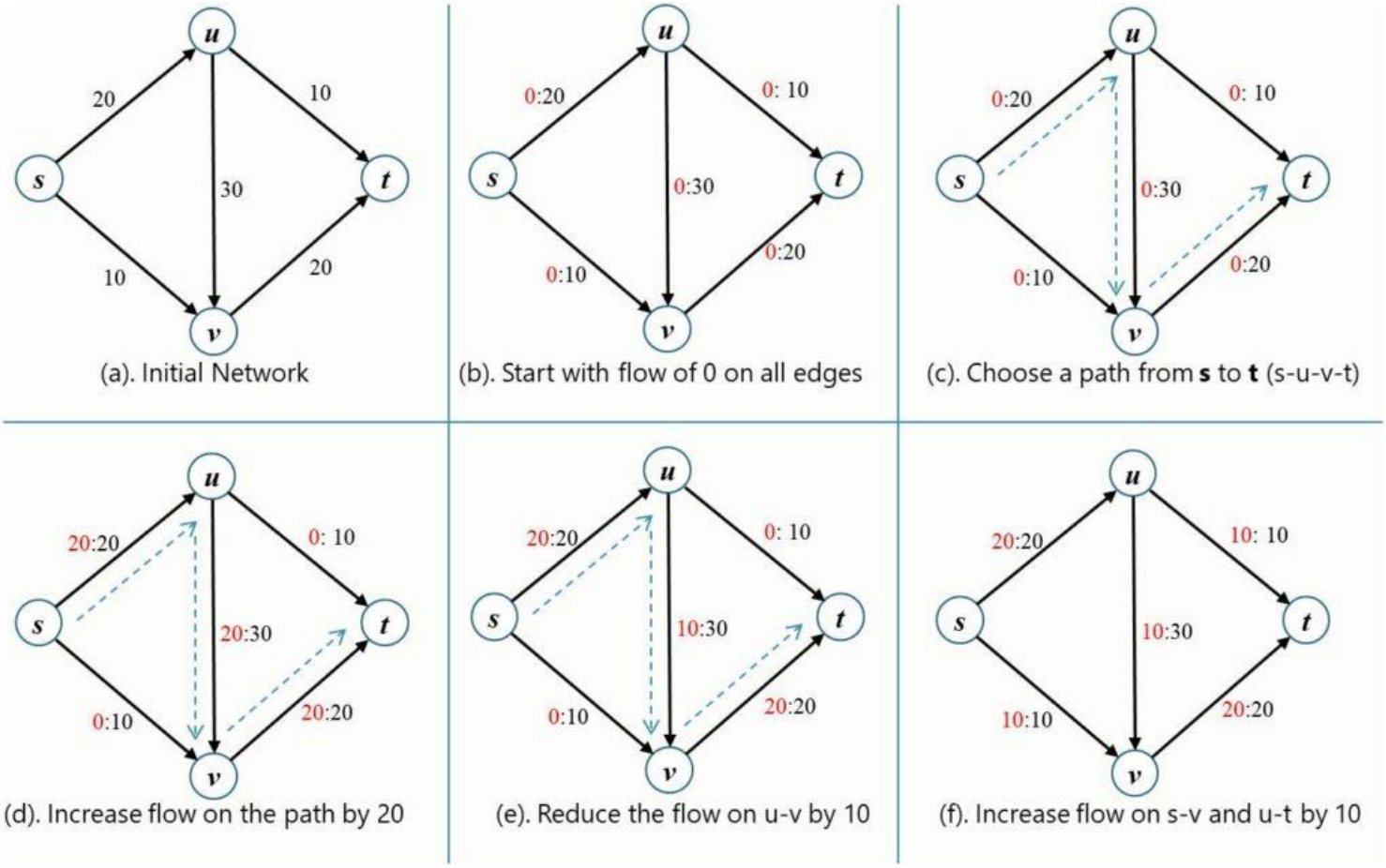

Figura 2: Funcionamiento del Algoritmo Ford-Fulkerson

En la figura 2, Se presenta una red de flujo con capacidades. Se inicia con flujo 0 en todas las aristas. Se busca un camino desde el nodo origen s hasta el nodo destino t con capacidad disponible (s-u-v-t). Se aumenta el flujo en ese camino según la mínima capacidad residual (20 unidades). Luego, se ajustan los flujos, reduciendo el flujo de retorno en u-v por 10. Se encuentra un nuevo camino aumentante (s-v-t) y se incrementa el flujo en 10 unidades. Este proceso se repite hasta que no existan más caminos con capacidad disponible desde s hasta t, momento en el cual se alcanza el flujo máximo.

1.3 Algoritmo de Kruskal

El algoritmo de Kruskal es un algoritmo voraz que permite encontrar el árbol de expansión mínima (MST) en un grafo no dirigido y ponderado. Su estrategia se basa en ordenar todas las aristas del grafo según su peso en orden ascendente y luego ir agregando las aristas más ligeras al árbol generado, siempre y cuando no formen ciclos con las aristas ya seleccionadas.

Para evitar la formación de ciclos, el algoritmo de Kruskal hace uso de una estructura de datos conocida como conjunto disjunto, que permite mantener control sobre los subconjuntos de vértices conectados. Si una nueva arista conecta dos vértices que pertenecen a diferentes conjuntos, entonces se agrega al MST y se fusionan los conjuntos

La Figura 3 ilustra el comportamiento del algoritmo.

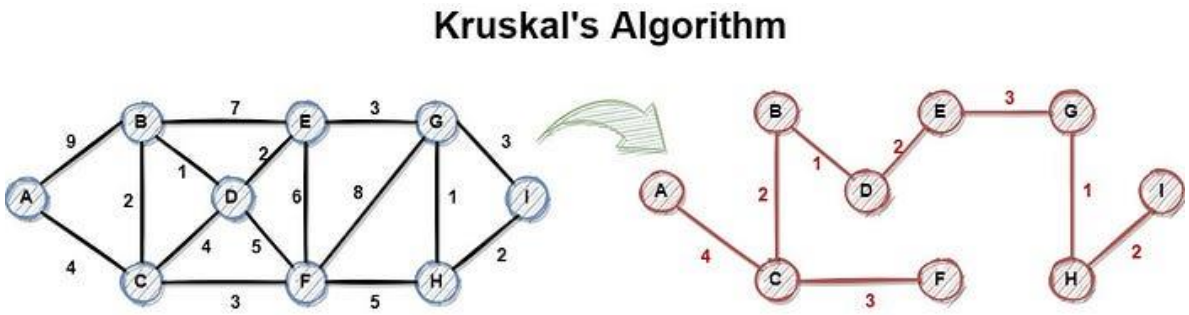

Figura 3: Funcionamiento del Algoritmo Kruskal

La imagen muestra el funcionamiento del algoritmo de Kruskal aplicado a un grafo ponderado. Primero ordena las aristas por peso y selecciona las de menor costo que no formen ciclos. Este proceso continúa hasta conectar todos los nodos del grafo.
El resultado es un árbol de expansión mínima con el menor peso total posible.

1.4 Algoritmo de Warshall para pesos mínimos

El algoritmo de Warshall, también conocido como algoritmo de Floyd-Warshall, nos permite encontrar las distancias más cortas entre todos los pares de vértices en un grafo ponderado dirigido o no dirigido. A diferencia de otros algoritmos que resuelven el problema de un solo origen, este encuentra las distancias mínimas entre todos los nodos del grafo.

El algoritmo se basa en una técnica de programación dinámica y utiliza una matriz de adyacencia que representa los pesos de las aristas. Iterativamente, va actualizando las distancias mínimas al considerar cada nodo como un vértice intermedio, buscando mejorar las rutas existentes. Es especialmente útil cuando se desea conocer todas las posibles rutas mínimas, como en sistemas de navegación o en análisis de redes.

La figura 4 ejemplifica un caso de uso del algoritmo

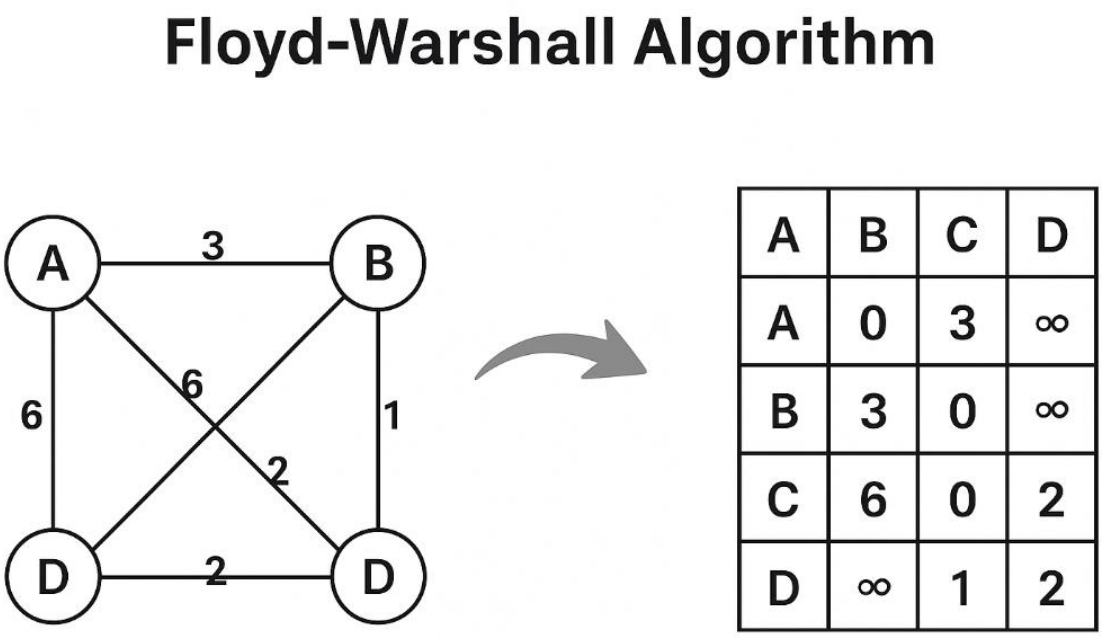

Figura 4: Funcionamiento del Algoritmo de Floyd-Warshall

La Figura ejemplifica correctamente el uso del algoritmo al mostrar un grafo con pesos y su correspondiente matriz de adyacencia inicial. Esta matriz representa las distancias mínimas entre nodos, donde se observan los valores directos entre pares conectados, ceros en la diagonal (distancia de un nodo consigo mismo) e infinitos para nodos no conectados, lo cual es la base sobre la que el algoritmo empieza a calcular las rutas más cortas entre todos los pares de nodos.

1.5 Calcular conectividad

La conectividad en un grafo se refiere a la existencia de caminos entre los distintos vértices. En un grafo no dirigido, se dice que es conexo si existe al menos un camino entre cualquier par de vértices. En grafos dirigidos, el concepto se divide entre conectividad fuerte (todos los vértices se alcanzan entre sí) y conectividad débil (al ignorar las direcciones, el grafo es conexo).

Para calcular la conectividad de un grafo se utilizan algoritmos de búsqueda como DFS (Depth-First Search o búsqueda en profundidad) o BFS (Breadth-First Search o búsqueda en anchura). Estos nos permiten determinar si todos los nodos están alcanzables desde cualquier punto.

1.6 Modelo de redes

Los modelos de redes utilizan grafos para representar relaciones, flujos o rutas en sistemas complejos. Cada nodo representa una entidad (como una ciudad, servidor o estación), y cada arista representa la conexión entre ellas (como caminos, cables o rutas de transporte).

Estos modelos permiten simular comportamientos del mundo real, optimizar recursos y planificar operaciones. Por lo general en los cursos de matemáticas discretas o de estructuras de datos en la UIS el ejercicio por excelencia que vemos que ilustra los modelos de redes es el de un mapa con las rutas de transporte de los estudiantes para llegar desde sus casas a la universidad, o para transportarse desde los distintos edificios en la universidad.

2. Aplicaciones Grafos

2.1 Juego de la Locura instantánea

El juego de locura instantánea es un rompecabezas/puzzle y una aplicación de la teoría de grafos que consiste en lo siguiente: Consta de cuatro cubos diferentes. Cada cara de cada cubo está pintada de uno de cuatro colores diferentes: azul, verde, rojo o amarillo. El objetivo del rompecabezas es alinear los cuatro cubos de modo que a lo largo de los cuatro lados largos (anterior, superior, posterior e inferior) cada uno de los cuatro colores aparezca exactamente una vez. La clave de encontrar la posición correcta de los cubos reside en su gran cantidad de posibles configuraciones, por lo que, si simplemente se busca una solución sin ser sistemático, es muy improbable que se encuentre.

Para abordar este problema de manera eficiente, la teoría de grafos ofrece una forma estructurada de representar y explorar todas las posibles configuraciones de los cubos. En este contexto, cada estado posible del rompecabezas puede representarse como un nodo en un grafo, donde el estado incluye el orden y la orientación de los cubos.

Las aristas del grafo representan movimientos válidos, como rotar un cubo o intercambiar su posición con otro. Así, el rompecabezas se convierte en un problema de búsqueda de caminos dentro de un grafo: el objetivo es encontrar un camino desde un estado inicial (una configuración aleatoria) hasta un nodo que represente una configuración válida (donde cada color aparece exactamente una vez en cada fila visible).

Utilizando algoritmos de búsqueda como BFS (búsqueda en anchura), se puede recorrer el grafo sistemáticamente para encontrar la solución en el menor número de pasos posibles. BFS es especialmente útil aquí porque garantiza encontrar la solución más corta (con menos movimientos), al explorar primero todas las configuraciones cercanas antes de profundizar más.

De esta forma, la teoría de grafos no solo permite representar el problema de manera clara, sino también aplicar estrategias computacionales eficientes para resolverlo.

Figura 5: Juego de la locura instantánea

2.2 Árboles de Juegos

Los arboles de juego son grafos que representan todas las posibles posiciones en un juego secuencial con información perfecta, un juego de información perfecta es un juego en que los jugadores conocen todo lo que podrían desear conocer acerca de lo que ha sucedido desde el principio del juego cuando tienen que realizar un movimiento hasta el momento, como lo son el ajedrez o tres en raya.

Representados en forma de grafos, cada nodo representa un estado del juego, y las aristas representan movimientos posibles. Se usa mucho en inteligencia artificial y teoría de juegos para determinar la mejor estrategia a seguir.

El siguiente diagrama muestra los primeros tres movimientos iniciales en un juego de tres en raya

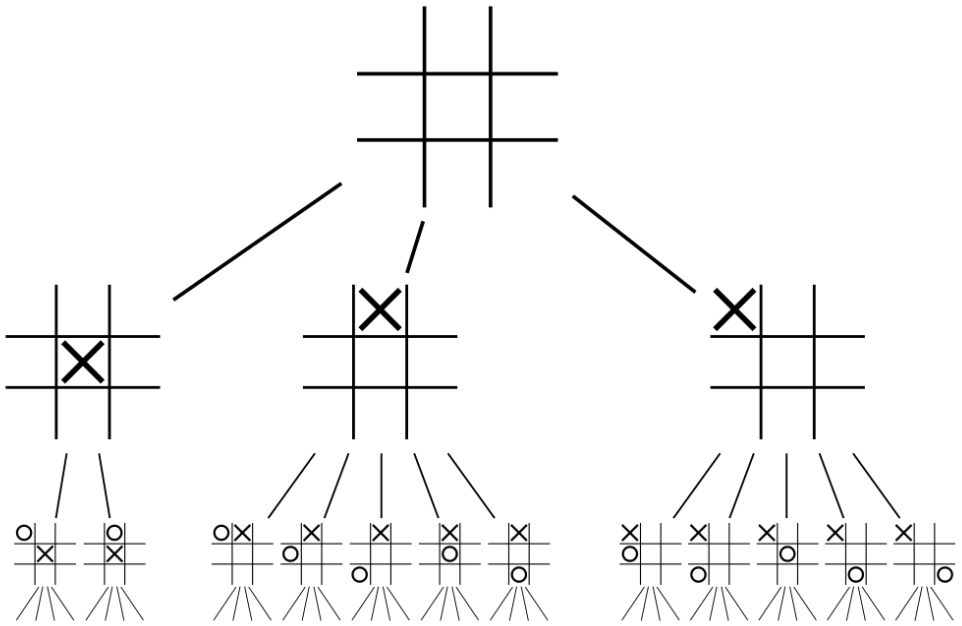

Figura 6: Juego de la locura instantánea

La figura 6 muestra las opciones iniciales que tiene el primer jugador, Las rotaciones y reflexiones de las posiciones son equivalentes, por lo que el primer jugador tiene tres opciones de movimiento: en el centro, en el borde o en la esquina.

2.3 Algoritmos:

2.3.1 Algoritmo del vecino más cercano

El Algoritmo del vecino más cercano , también conocido como K-Vecinos Más Cercanos o KNN (por su nombre en ingles, K-Nearest-Neighbor), es un algoritmo usado comúnmente para encontrar soluciones al Problema del Viajante, que consiste en encontrar el camino único más corto que, dada una lista de ciudades y las distancias entre ellas, visita todas las ciudades una sola vez y regresa a la ciudad de origen.

Este algoritmo es una implementación directa de la teoría de grafos, ya que cada ciudad se representa como un nodo y cada camino entre ciudades como una arista con un peso que indica la distancia.

Aunque tenemos que considerar que no siempre da la mejor solución posible porque toma decisiones basadas solo en lo que parece mejor en ese momento, sin considerar cómo afectarán esas decisiones al resultado final. Este comportamiento se debe a que es un algoritmo voraz (recordar este problema en la descripción del algoritmo de búsqueda A), por lo que en cada paso elige la opción que ofrece el beneficio inmediato más alto, en este caso, la ciudad más cercana. Sin embargo, esta elección local no garantiza el mejor recorrido total, ya que puede llevar a caminos más largos después o dejar ciudades aisladas al final.

El algoritmo KNN fue desarrollado por primera vez por Evelyn Fix y Joseph Hodges en 1951 en el contexto de una investigación realizada para el ejército de los Estados Unidos. Casi 36 años después, el algoritmo fue perfeccionado por James Keller, quien desarrolló un "KNN difuso" que produce tasas de error más bajas.

En la actualidad, el algoritmo de KNN es el algoritmo más usado, gracias a su adaptabilidad para la mayoría de los campos, desde la genética hasta las finanzas y el servicio al cliente.

2.3.2 Algoritmo de búsqueda en anchura

La búsqueda en anchura , también conocida por sus siglas en inglés como BFS es una estrategia para recorrer los elementos en un grafo. Comienza en un nodo raíz y explora todos sus vecinos antes de pasar a los vecinos de esos vecinos.

Es ideal para encontrar la ruta más corta en grafos no ponderados y para verificar si un grafo es conexo. También se utiliza en sistemas de recomendación, motores de búsqueda, y en redes sociales para detectar comunidades o encontrar niveles de cercanía entre usuarios.

El funcionamiento del algoritmo se ve ejemplificado en la siguiente figura, junto con el productor resultante o output del recorrido.

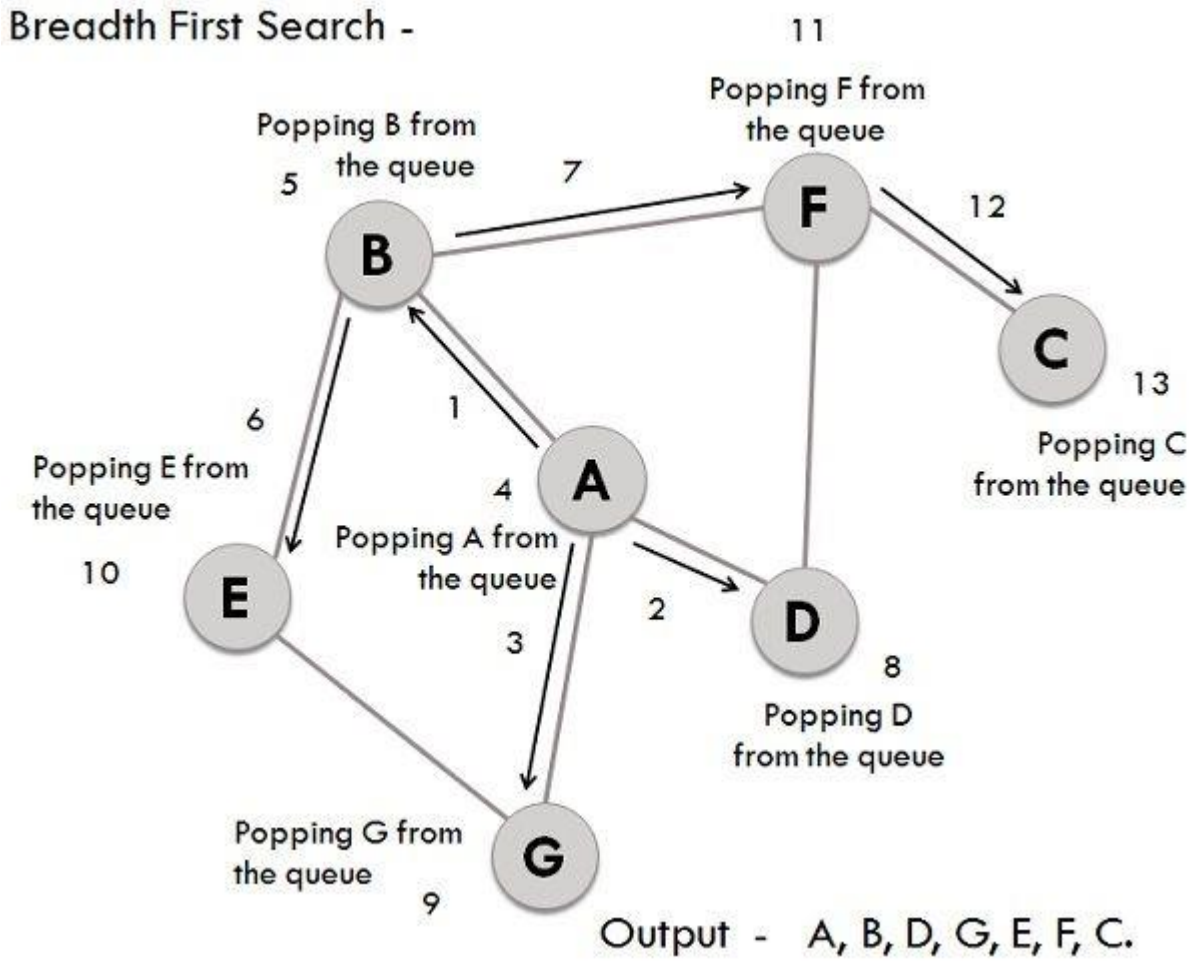

Figura 7: Algoritmo de búsqueda en anchura

2.3.3 Algoritmo de búsqueda en profundidad

El algoritmo de búsqueda en profundidad o por sus siglas en ingles DFS es un algoritmo explora tan lejos como sea posible a lo largo de cada rama antes de retroceder. Utiliza una pila (implícita mediante recursión o explícita) para ir visitando nodos. Es útil para detectar ciclos, encontrar componentes conexas, y resolver laberintos.

El algoritmo DFS es un método basado en aristas y funciona de forma recursiva, donde los vértices se exploran a lo largo de una ruta (arista). La exploración de un nodo se suspende al encontrar otro nodo inexplorado, y los nodos inexplorados más profundos se recorren primero.

El algoritmo de búsqueda se relaciona con la búsqueda en anchura por ser usados para el recorrido de grafos, pero se diferencian en que en profundidad explora un camino lo más profundo posible antes de retroceder, usando una pila o recursión. En cambio, BFS explora primero todos los vecinos del nodo actual antes de ir más lejos, utilizando una cola. DFS no garantiza encontrar el camino más corto, mientras que BFS sí lo hace en términos de cantidad de pasos. DFS es útil para detectar ciclos o explorar completamente un grafo, y BFS es ideal para encontrar rutas más cortas o niveles de conexión entre nodos.

El funcionamiento del algoritmo se ve ejemplificado en la siguiente figura, junto con el productor resultante o output del recorrido.

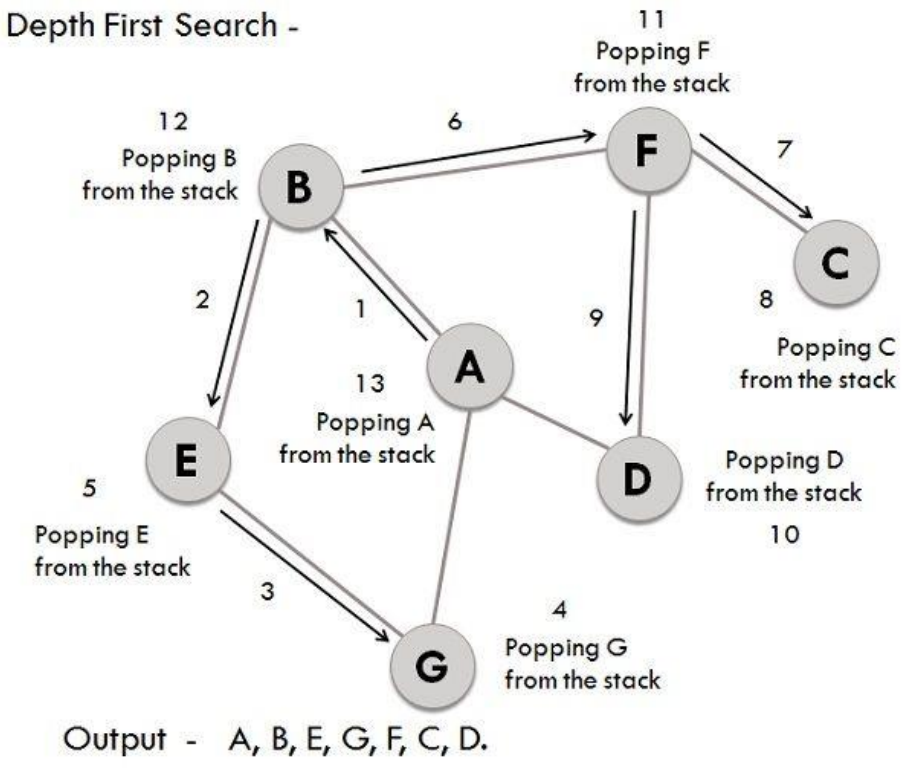

Figura 7: Algoritmo de búsqueda en profundidad

2.3.4 Algoritmo de búsqueda A

El algoritmo A* (A estrella) es un algoritmo de búsqueda que tiene como su motivo de origen el solucionar problemas en otros algoritmos de búsqueda en grafos como los algoritmos voraces, que no pueden indicar el camino de coste mas bajo, o por el coste real de desplazarse de un nodo a otro ,pudiéndose dar el caso de que sea necesario realizar un movimiento de coste mayor para alcanzar la solución. Es por que este algoritmo tiene en cuenta ambos factores, el valor heurístico de los nodos y el coste real del recorrido. Usando una función f(n) = g(n) + h(n), donde g(n) es el costo desde el inicio hasta el nodo n, y h(n) es la estimación del costo restante.
Es muy eficiente para encontrar el camino más corto en grafos grandes, especialmente cuando se tiene una buena heurística. Se utiliza en navegación GPS, planificación de rutas en videojuegos y en robótica.

2.3.5 Algoritmo de Dijsktra

El Algoritmo de Dijsktra es un algoritmo que nos ayuda a determinar el camino más corto de un grafo dado el nodo de origen, el algoritmo encuentra el camino más corto desde un nodo origen hacia todos los demás nodos en un grafo con pesos no negativos. Funciona expandiendo siempre el nodo con la menor distancia acumulada desde el origen, actualizando las distancias de sus vecinos si se encuentra un camino más corto.

Es fundamental en redes de telecomunicaciones, sistemas de navegación y planificación urbana. Es confiable y óptimo para grafos ponderados sin aristas negativas.

El algoritmo de Dijsktra es similar al algoritmo de Prim que revisamos al inicio del documento, porque ambos usan una cola de prioridad para seleccionar el siguiente vértice que se agregará al grafo en crecimiento.

El siguiente GIF muestra como se comporta el algoritmo

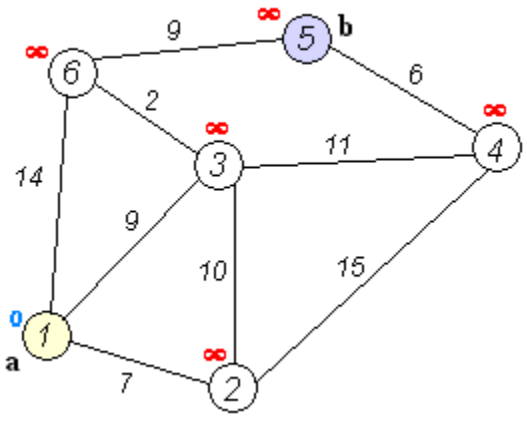

Figura 8: Algoritmo de Dijsktra

2.3.6 Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford también calcula el camino más corto desde un nodo origen hacia todos los demás, pero a diferencia de Dijkstra, puede manejar aristas con pesos negativos. Funciona iterando sobre todas las aristas varias veces para actualizar los costos mínimos conocidos hasta que se estabilicen. Las limitaciones del algoritmo de Dijkstra se deben a que Dijkstra asume que una vez que encuentra el camino más corto a un nodo, ya no puede encontrar uno mejor después, esto es válido solo si todas las aristas tienen pesos positivos o cero, pero si una arista tiene un peso negativo, puede aparecer más adelante un camino mejor que reduzca aún más la distancia total, pero Dijkstra ya habrá marcado ese nodo como “visitado” y no lo volverá a revisar, por lo tanto, no detectará ese mejor camino.

Además, detecta la existencia de ciclos de peso negativo, algo que otros algoritmos no hacen. Es útil en redes con costos que pueden cambiar (como en finanzas o transacciones), o en grafos donde se desea verificar la existencia de inconsistencias.