**Algoritmo de optimización de Grafos**

**Jaime de Castro 14708**: preparación del código de los algoritmos tanto recursivo como dijkstra

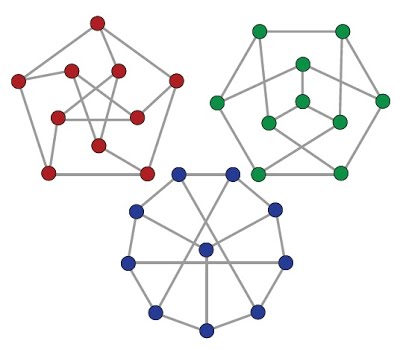
**Guillermo Moreno 16289**: elaboración y organización

**Alejandro Pérez Vicente 16335**: Toma de datos y análisis

Introducción:

La teoría de grafos se basa en calcular el camino optimo entre dos nodos de un grafo, con múltiples nodos organizados con distancias diferentes.

Es muy usada en las matemáticas y las ciencias de la computación que estudia las propiedades de los grafos.



Los grafos pueden ser de cualquier dimensión y geometría, por tanto, suelen ser bastante aleatorios.

En nuestro caso práctico nos limitaremos a la obtención del camino óptimo de un punto del grafo a otro mediante el uso de dos algoritmos: unos recursivo y otro basado en Dijkstra (similar a iterativo).

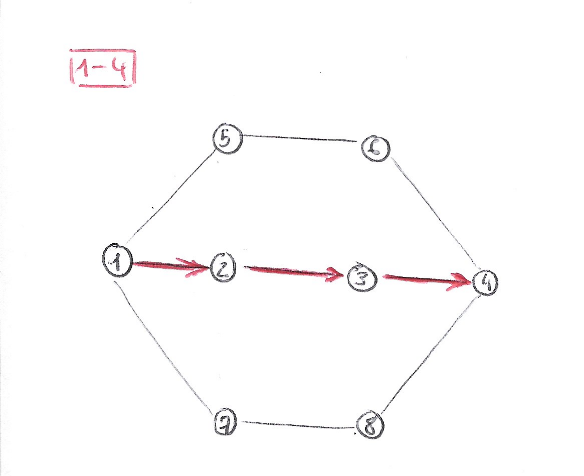
Además, tendremos en consideración el tamaño de los datos que utilicemos en los algoritmos, es decir, dimensión del grafo, distancia mínima entre nodos, numero de iteraciones, etc.

La herramienta principal que vamos a usar para la evaluación de experimentos es el tiempo de procesamiento que se empleará para la resolución del algoritmo.

Hay que tener en cuenta que los resultados son únicamente de carácter comparativo, ya que el código no está extensamente optimizado (se mide el tiempo de ejecución del algoritmo completo, incluyendo su inicialización)

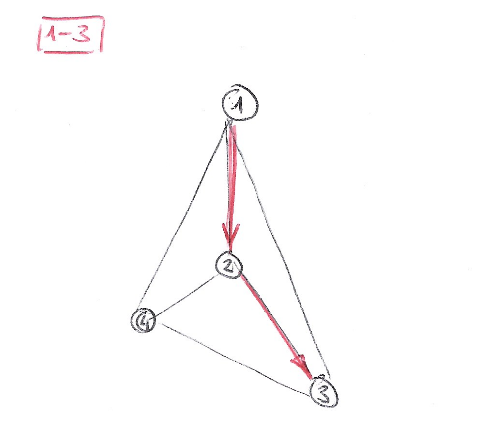
**Análisis del experimento:**

Inicialmente, haremos una evaluación de la resolución de grafos de forma experimental en ausencia de un algoritmo programado. Para ello planteamos diversos posibles ejemplos, los cuales observaremos su estructura e intentaremos resolverlos para sacar hipótesis.

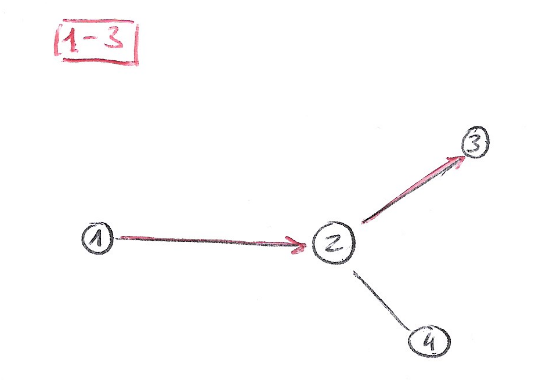


Ejemplo 1:

La forma intuitiva de resolverlo es colocarse en el nodo de origen y ver con que nodos de los de su alrededor está conectado. De todos ellos cogeremos el más cercano. Procederemos así hasta llegar al nodo final.

Ejemplo 2:

En este caso, siguiendo la idea anterior llegamos al nodo 2, pero después aunque la distancia al nodo 4 es menor, vemos que es más razonable elegir directamente el nodo 3 que es el de destino.

Ejemplo 3:

Si solo hay un nodo central que conecta todos los demás, entonces ese nodo siempre formara parte de la solución, y para determinar el camino mas corto solo hay que considerar las distancias.

Hipótesis:

1. Para buscar la solución, parto del nodo inicial y busco el mas cercano de los que se conectan a su alrededor
2. A mayor numero de conexiones, más probabilidad de una conexión directa entre nodos, es decir, el camino solución atravesará menos nodos
3. A mayor numero de nodos, es más probable que el camino atraviese un mayor número de nodos si los nodos origen y fin están muy alejados en la estructura.

Para simplificar el tratamiento de los datos, consideramos la representación del grafo en una matriz de conectividad. Dicha matriz se caracteriza por:

* Reflejar la distancia de un nodo a otro.
* Tener los elementos de la diagonal principal nulos, puesto que no tiene sentido que haya distancia de un nodo al mismo nodo.
* Ser simétrica, ya es razonable que la distancia del nodo1 al nodo2 (elemento[i][j]) es la misma que del nodo2 al nodo1 (elemento[j][i]).

El parámetro básico que se mide es, como hemos dicho antes, el tiempo total de ejecución del algoritmo. Para ello utilizamos la librería de c++ <chrono>, permitiéndonos tomar el intervalo de tiempo transcurrido durante la ejecución del algoritmo.

Las distancias entre nodos están representadas en la matriz mediante números del 0 al 9. Podemos suponer por ejemplo que ese número son los metros de un nodo a otro, pero también se podria haber supuesto que son kilómetros u otra medida distinta. Luego el significado de los valores de la matriz depende de cómo queramos definirlo. También podriamos aumentar fácilmente el límite de estos valores para que puedan ser superiores a 9.

El número de nodos está definido en DIM (variable solo modificable desde el editor de texto del programa).

Cuantos más ceros incluya la matriz (sin tener en cuenta los de la diagonal), más dispersa es. Esto implica que hay menos conexiones entre nodos, es decir, no todos los caminos son posibles. La dispersión de la matriz se puede modificar a partir de X (variable solo modificable desde el editor de texto del programa).

Como ya hemos visto, el número de nodos y el número de conexiones son los principales factores que pueden afectar a la resolución del grafo. Por tanto, probaremos los algoritmos desarrollados (el dijkstra y el recursivo) modificando dichas condiciones. El programa cuenta previamente con una generación de matrices aleatorias que cumplan con las caracteristicas citadas en la introducción, seguido de la solucion de esa matriz por ambos algoritmos. Esto permite realizar un experimento distinto a cada ejecución del programa.

La tabla con los experimentos probados y sus resultados se adjunta a continuación.

Complejidad Dijkstra O(V^2) / O(V\*logV +N)

Complejidad Recursivo O(V!)