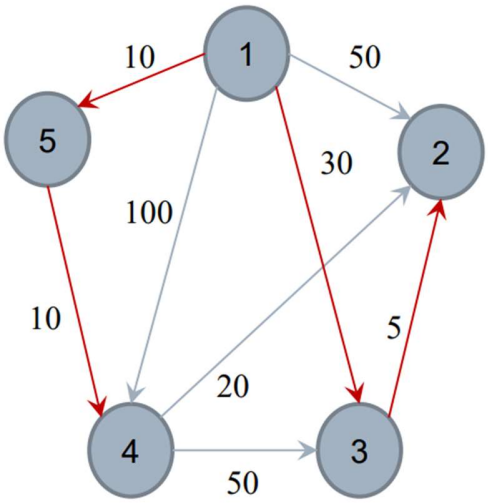
# Parte 2 Análisis de algoritmos Voraces

## El Problema del camino mínimo

El Problema del camino mínimo es uno de los problemas de optimización combinatoria más conocidos. Dadas una serie de ciudades, el objetivo consiste en encontrar los caminos mínimos entre una ciudad concreta y todas las demás. Es decir, partiendo de una ciudad concreta, ver cómo podemos unirla con todas las demás de forma que la distancia recorrida entre la ciudad de origen y cada una de las ciudades destino sea la mínima posible.

La solución voraz a este problema consiste en ir eligiendo en cada paso un nuevo camino siguiendo como criterio voraz la ciudad más cercana a la ciudad de origen (algoritmo de Dijkstra).

A modo de ejemplo, en el siguiente grafo (los círculos representan ciudades y los arcos la distancia entre las ciudades) tenemos que calcular cuales son los caminos mínimos entre la ciudad 1 y el resto de ciudades.




Como hay 5 ciudades entonces tenemos que calcular como podemos unir la ciudad 1 con las 4 ciudades restantes (hay que hallar 4 rutas o caminos) de modo que la distancia para ir desde la ciudad 1 a las demás sea la mínima posible.

La solución la hemos resaltado en color rojo, siendo la distancia mínima desde el nodo 1 a los 4 nodos restantes la siguiente:

**Solución D = {35,30,20,10}** ≡ (distancia desde el nodo 1 a cada nodo del grafo)

**Nota:**
El algoritmo voraz de Dijkstra viene explicado en el Tema 3 de teoría. Los algoritmos voraces se explican a partir de la transparencia 46 y el problema del camino mínimo de un grafo se expone en la transparencia 84, describiéndose el algoritmo de Dijkstra que lo soluciona en las transparencias 85 y 86.

## Desarrollo de la práctica

El alumno deberá desarrollar un proyecto en lenguaje JAVA que **resuelva el problema del camino mínimo siguiendo la estrategia voraz comentada**.

El programa deberá permitir seleccionar como entrada un conjunto de datos generados de forma aleatoria, o bien permitir cargar cualquier fichero de la biblioteca TSPLIB[1] (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/). Para ello el programa debe proporcionar una opción de menú o una ventana de diálogo que permita al usuario indicar la ruta del fichero que desea abrir.

El programa deberá calcular cual es el coste de la ruta calculada (suma del coste de cada uno de los caminos mínimos desde la ciudad origen al resto de ciudades) y deberá mostrar por pantalla el recorrido de dicha ruta, permitiendo guardar en disco un fichero con la solución obtenida.

El resultado del proceso de búsqueda se debe mostrar por pantalla en modo texto, indicando para cada ciudad destino la distancia mínima que hay que recorrer desde la ciudad origen junto con el camino que se debe seguir (desde la ciudad origen a la ciudad destino). La aplicación también debe permitir guardar la solución encontrada en un fichero en disco.

El formato del fichero generado por la aplicación debe ser el siguiente (se muestra el fichero solución para el grafo de ejemplo de la página anterior):

```
NAME : huelva4.opt.tour
TYPE : TOUR
DIMENSION : 5
SOLUTION: 95
TOUR_SECTION
35 - 1,3,2
30 - 1,3
20 - 1,5,4
10 - 1,5
-1
EOF
```

[1] El formato de los ficheros de entrada es el mismo que el utilizado en la parte 1 de la práctica.

El alumno deberá entregar un estudio teórico realizado sobre el pseudocódigo simplificado del algoritmo. También deberá indicar el resultado obtenido sobre los siguientes conjuntos de datos:

• **berlin52:** Tamaño 52 ciudades. Coste de la solución óptima: *a rellenar por el alumno*
• **ch130:** Tamaño 130 ciudades. Coste de la solución óptima: *a rellenar por el alumno*
• **ch150:** Tamaño 150 ciudades. Coste de la solución óptima: *a rellenar por el alumno*

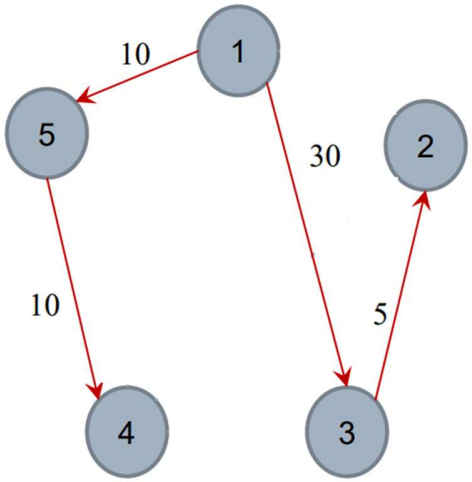
Además, se debe comparar los resultados esperados del estudio teórico con los experimentales. Para ello se elegirán grupos aleatorios de (como mínimo) los siguientes tamaños: 200, 500, 1500, 5000, realizándose para cada una de las tallas indicadas **10 ejecuciones** diferentes. Si se considera útil para la representación gráfica se pueden representar tallas adicionales.

Las representaciones gráficas del modelo teórico y práctico se superpondrán en una sola gráfica de líneas.

**Opcional (3 puntos):**

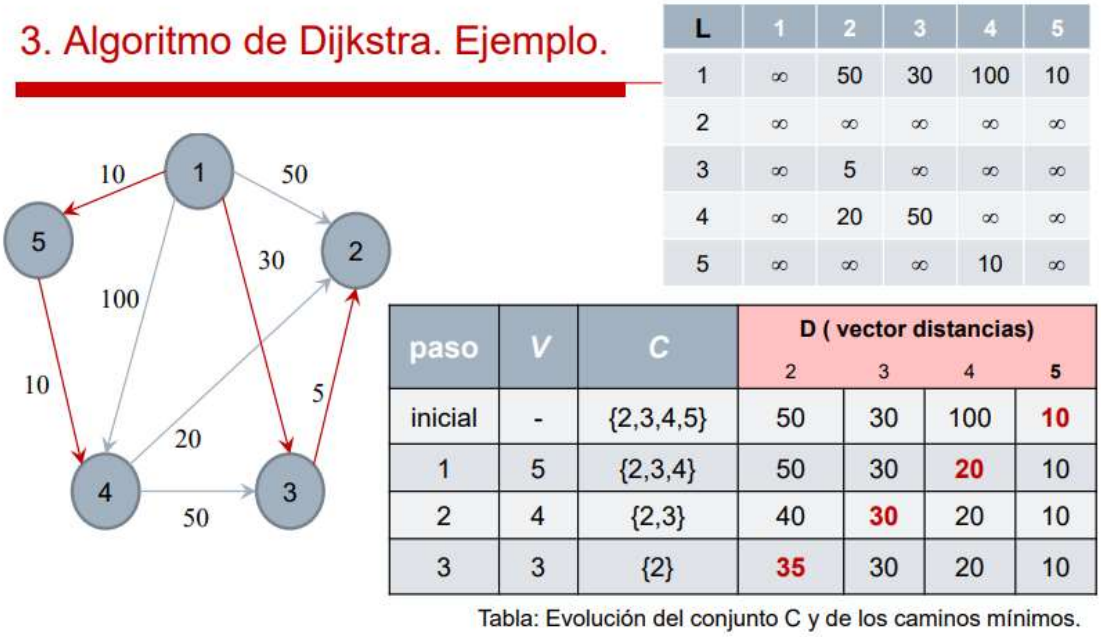
**El programa deberá mostrar también la solución en formato gráfico** por medio de un panel en el que se representen los puntos del dataset y los enlaces que forman cada uno de los caminos mínimos encontrados.

A modo de ejemplo, se muestra la solución en formato gráfico del grafo de ejemplo de la página 6.



**Opcional (2 puntos):**

**Hacer que el programa muestre, para conjunto de datos de menos de 10 puntos** (generados aleatoriamente o cargados desde ficheros) **la solución detallada (paso a paso)** generada por el algoritmo, mostrando por pantalla tanto la solución gráfica como el relleno de la tabla que se va generando en cada paso del algoritmo y que detalla la forma en la que el algoritmo de Dijkstra va formando y calculando la solución, tal como aparece en las transparencias que explica el algoritmo en los apuntes de teoría.




| L | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | ∞ | 50 | 30 | 100 | 10 |
| 2 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 3 | ∞ | 5 | ∞ | ∞ | ∞ |
| 4 | ∞ | 20 | 50 | ∞ | ∞ |
| 5 | ∞ | ∞ | ∞ | 10 | ∞ |

| paso | V | C | D ( vector distancias) 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| inicial | - | {2,3,4,5} | 50 | 30 | 100 | **10** |
| 1 | 5 | {2,3,4} | 50 | 30 | **20** | 10 |
| 2 | 4 | {2,3} | 40 | **30** | 20 | 10 |
| 3 | 3 | {2} | **35** | 30 | 20 | 10 |

Tabla: Evolución del conjunto C y de los caminos mínimos.

**Nota:**
El algoritmo voraz de Dijkstra necesita conocer las aristas que conectan los diferentes puntos que forman el grafo, así como los pesos asociados a cada arista. Al facilitarle sólo el conjunto de puntos que forman el grafo, el programa deberá generar todas las posibles aristas (conectar cada punto con todos los demás) y asociar a cada arista un peso.

Para la práctica tomar como peso de cada arista la siguiente:

$$\boldsymbol{peso\ de\ cada\ arista} = [(\boldsymbol{distancia\ entre\ los\ 2\ puntos\ que\ une\ x}\ \mathbf{100})\ \%\ \mathbf{100}] + \mathbf{1}$$

Es decir, debemos calcular la distancia euclídea, multiplicarla por 100, convertir dicho valor a entero, quedarnos con el resto de dividirlo entre 100 y sumarle 1 a la cantidad resultante (de esta forma el peso de cada arista será un valor comprendido entre 1 y 100)