# Metodologías de la programación y algoritmia
# Reparto a domicilio
# 2021/2022


Autores:
Vicente Candela Pérez
Samuel Lajara Agulló
Toni Bailén Sevilla

**Índice**

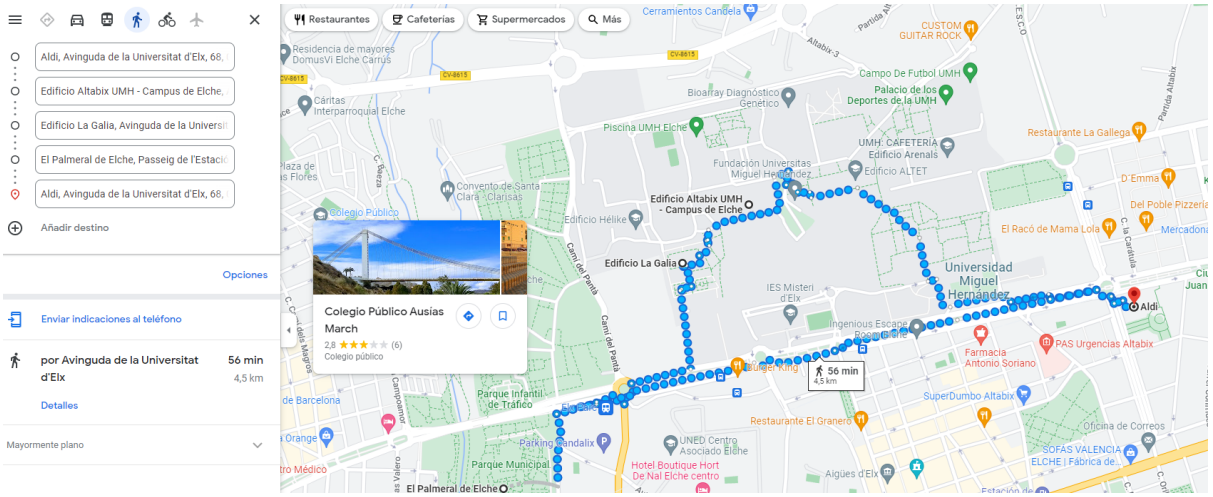
# Reparto a domicilio

## 1.- Algoritmo para obtener un listado de los posibles recorridos.

### 1.1.- Descripción y tipificación.

Para esta práctica vamos a usar un programa el cual busca todas las posibles soluciones recorriendo todos los caminos posibles, los cuales tienen que acabar en el punto de salida.

Por ejemplo vamos a trabajar con el reparto a domicilio de un supermercado, el repartidor tiene que ir a la casa de 3 clientes y volver a su puesto de trabajo, así que nuestro programa busca los posibles caminos y los muestra para que podamos escoger uno, además también nos dice cual es el camino más rápido para que el repartidor haga el recorrido completo lo más rápido posible.




#### 1.1.1. Tipificación

**n** es el número de clientes a visitar.

**V** es un vector de tamaño n que contiene los números asignados de los clientes en el orden en el cual se recorren. Nótese que, como se va a obtener un ciclo que recorra todos los clientes, es indiferente el vértice desde el que se empiecen, y el asignar un vértice inicial disminuye el tiempo de ejecución del algoritmo. El algoritmo se establece como vértice inicial el número 1 ($V_1 \leftarrow 1$).

**Coste** es la matriz de adyacencia.

**costeOptimo** es una variable que tiene como valor $+\infty$ porque tratamos de minimizar el coste.

**Voptimo** es el vector donde se guarda el orden de las posiciones del recorrido óptimo.

### 1.1.2. Función objetivo

La función objetivo que vemos a continuación trata de minimizar el coste del recorrido, devolviendonos el recorrido más óptimo, pero nuestro algoritmo no realiza solo eso, también se encarga de devolvernos todos los caminos posibles.

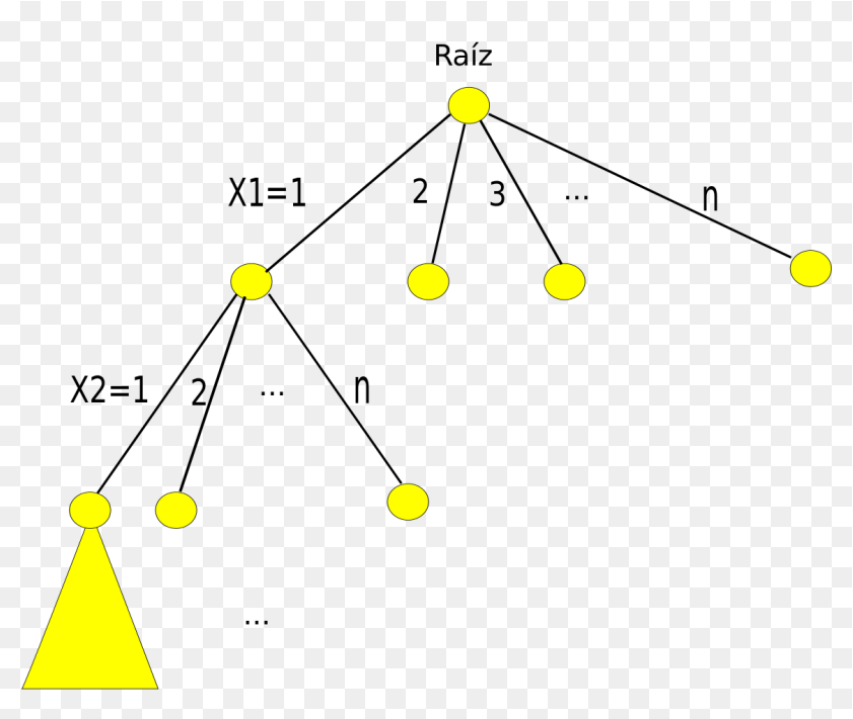
$$\text{Min}\left\{\sum_{i=1}^{n-1} Coste(v_i, v_{i+1}) + Coste(v_n, v_1)\right\}$$

### 1.1.3. Restricciones

- Debe existir una arista de un vértice al siguiente.
- Debe existir una arista del último vértice del camino al primero.
- No deben existir ciclos excepto al final del camino.
- La solución debe haber pasado por todos los clientes.

## 1.2.- Estrategia de programación.

El algoritmo que vamos a utilizar es el de vuelta a atrás ya que mediante esta técnica se pueden obtener todas las soluciones posibles de un problema, una solución o la solución óptima. Este algoritmo consiste en realizar una búsqueda exhaustiva y sistemática de la solución a un problema por medio de un método de prueba y error.



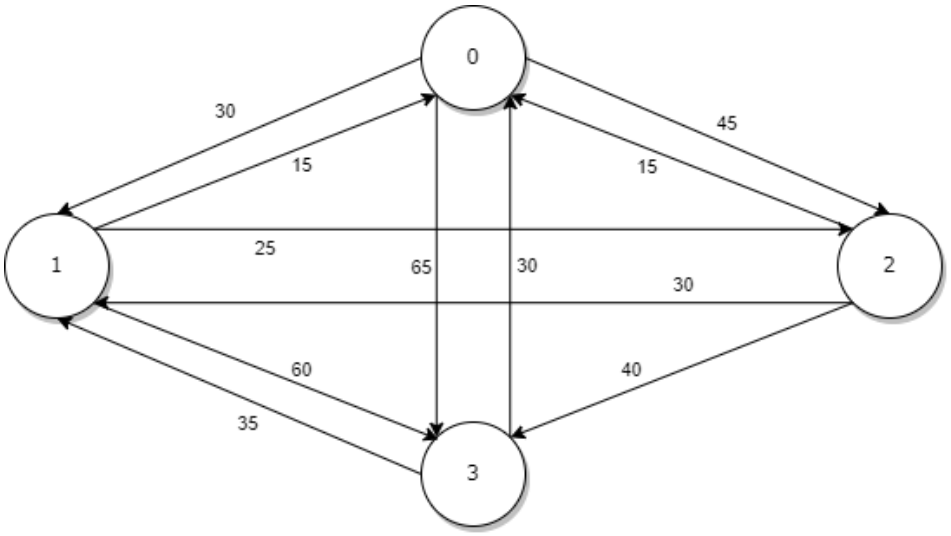
## 1.3.- Pseudocódigo.

```
costeÓptimo ← +∞
para i ← 0 hasta n
        V_i ← i
        Voptimo_i ← 0
fpara
k ← 1

función Viajante (Coste:real+[n,n], V:&natural[n], Vóptimo:&natural[n], costeÓptimo:&real,
k:entero)
        costeV: real
        V_k ← 0
        mientras V_k ≠ n hacer
                V_k ← V_k + 1
                si Coste_{Vk-1, Vk} ≠ ∞ y Cliclos (V) = FALSO
                        si k = n
                                si Coste_{Vk, V0} ≠ ∞
                                        costeV ← CalcularCoste(Coste, V)
                                        si costeV < costeÓptimo
                                                Vóptimo ← V
                                                costeÓptimo ← costeV
                                        fsi
                                fsi
                        si no
                                Viajante(Coste, V, Vóptimo, costeÓptimo, k+1)
                                V_{k+1} ← -1
                        fsi
                fsi
        fmientras
ffunción

función CalcularCoste(Coste:real+[n,n], V:&natural[n]):real
        suma: real
        i: natural
        k: natural
        suma ← 0
        k ← 0
        para i ← 1 hasta n
                suma ← suma + Coste_{k,Vi}
                k ← V_i
                si i = n
                        suma ← suma + Coste_{Vi,0}
                fsi
        fpara
        devolver suma
ffunción
```

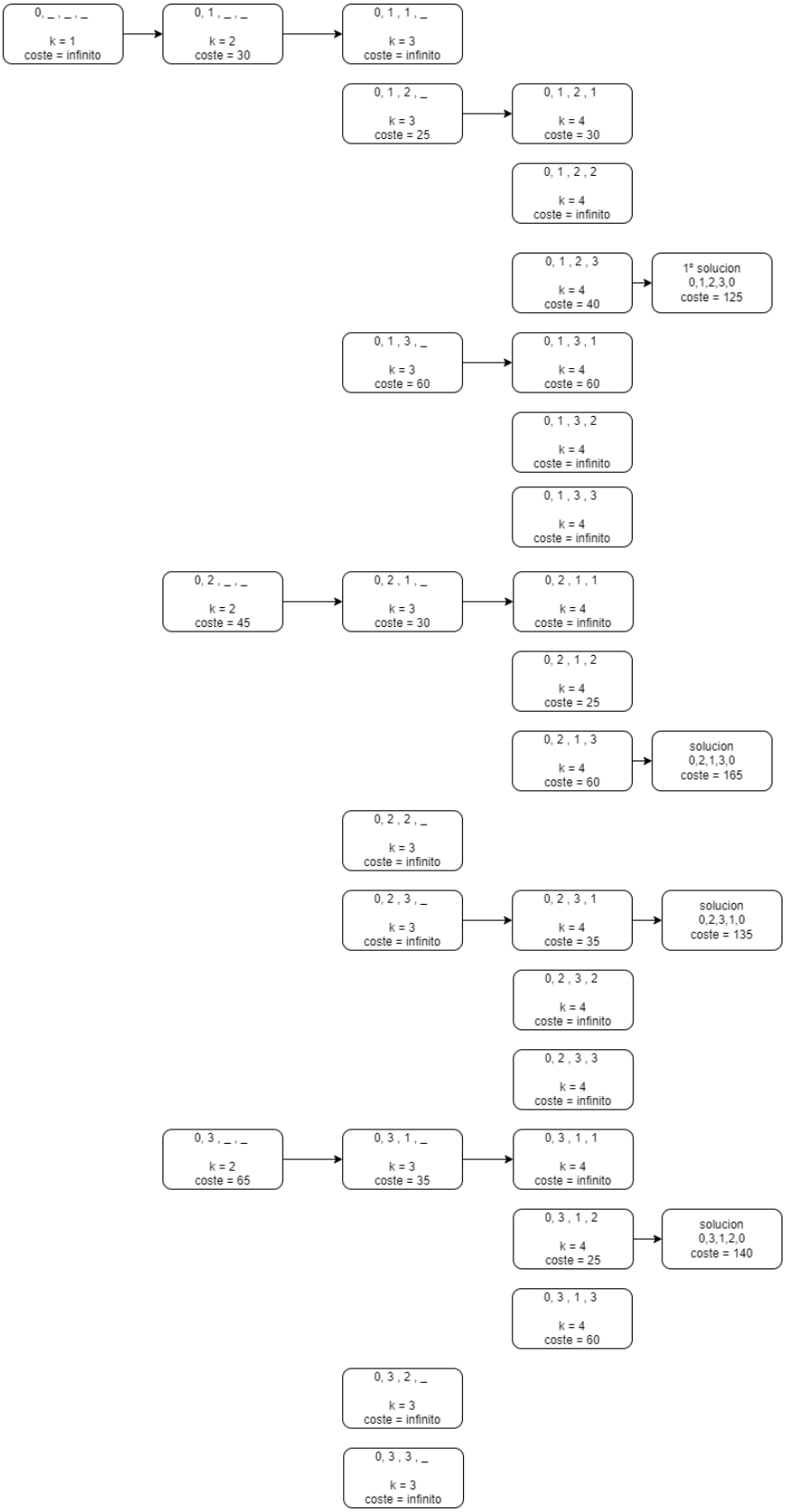
```
función Cliclos(V:&natural[n]):booleano
        i: natural
        j:natural
        para i ← 1 hasta n
                para j ← i+1 hasta n
                        si Vi ≠ -1
                                si Vi = Vj
                                        devolver verdadero
                                fsi
                        fsi
                fpara
        fpara
        devolver falso
ffunción
```

## 1.4.- Ejemplo.

Disponemos de 3 clientes a los cuales hay que llevarles su pedido desde el supermercado, cada uno está en una ubicación con una distancia diferente. Pretendemos hacer el recorrido lo más rápido posible saliendo desde el supermercado y regresando a él, para ello vamos a dibujar un grafo con su traza correspondiente:



**Traza realizada:**



## 1.5.- Complejidad asintótica.

| | NV1 | NV2 | NV3 | NV4 | NV5 |
|---|---|---|---|---|---|
| (1) costeÓptimo ← +∞ | O(1) | | | | |
| (2) para i ← 0 hasta n | O(n) | | | | |
| (3) $V_i$ ← i | | O(1) | | | |
| (4) $Voptimo_i$ ← 0 | | O(1) | | | |
| (5) fpara | | | | | |
| (6) k ← 1 | O(1) | | | | |
| (7) función **Viajante** (Coste:real+[n,n], V:&natural[n], Vóptimo:&natural[n], costeÓptimo:&real, (8) k:entero) | | | | | |
| (9) costeV: real | | | | | |
| (10) $V_k$ ← 0 | $t_{(1)} = 2$ | | | | |
| (11) mientras $V_k \neq n$ hacer | $t_{(2)} = T(n^3)$ | | | | |
| (12) $V_k$ ← $V_k$ + 1 | | $t_{(3)} = 4$ | | | |
| (13) si $Coste_{Vk-1, Vk} \neq \infty$ y Cliclos (V) = FALSO | | $t_{(4)} = T(n^2)$ | | | |
| (14) si k = n | | | $t_{(5)} = 1$ | | |
| (15) si $Coste_{Vk, V0} \neq \infty$ | | | | $t_{(6)} = 1$ | |
| (16) costeV ← CalcularCoste(Coste, V) | | | | | $t_{(7)} = T(n)$ |
| (17) si costeV < costeÓptimo | | | | | $t_{(8)} = 1$ |
| (18) Vóptimo ← V | | | | | $t_{(9)} = 3$ |
| (19) costeÓptimo ← costeV | | | | | $t_{(10)} = 3$ |
| (20) fsi | | | | | |
| (21) fsi | | | | | |
| (22) si no | | | | | |
| (23) Viajante(Coste, V, Vóptimo, costeÓptimo, k+1) | $t_{(14)} = ""$ | | | | |
| (24) $V_{k+1}$ ← -1 | | | | $t_{(15)} = 2$ | |
| (25) fsi | | | | | |
| (26) fsi | | | | | |
| (27) fmientras | | | | | |
| (28) ffunción | | | | | |
| | | | | | |
| (30) función **CalcularCoste**(Coste:real+[n,n], V:&natural[n]):real O(n) | | | | | |
| (31) suma: real | | | | | |
| (32) i: natural | | | | | |
| (33) k: natural | | | | | |
| (34) suma ← 0 | O(1) | | | | |
| (35) k ← 0 | O(1) | | | | |
| (36) para i ← 1 hasta n | O(n) | | | | |
| (37) suma ← suma + $Coste_{k,Vi}$ | | O(1) | | | |
| (38) k ← $V_i$ | | O(1) | | | |
| (39) si i = n | | O(1) | | | |
| (40) suma ← suma + $Coste_{Vi,0}$ | | | O(1) | | |
| (41) fsi | | | | | |
| (42) fpara | | | | | |
| (43) devolver suma | | | | | |
| (44) ffunción | | | | | |

```
(46) función Cliclos(V:&natural[n]):booleano                    O(n^2)
(47)     i: natural
(48)     j:natural
(49)     para i ← 1 hasta n                                     O(n)
(50)             para j ← i+1 hasta n                                    O(n)
(51)                     si V_i ≠ -1                                             O(1)
(52)                             si V_i = V_j                                            O(1)
(53)                                     devolver verdadero                                      O(1)
(54)                             fsi
(55)                     fsi
(56)             fpara
(57)     fpara
(58)     devolver falso                                         O(1)
(59) ffunción
```

Tras calcular la complejidad asintótica de la función Calcular Coste y la función Ciclos, comenzamos a calcular la complejidad asintótica de la función viajante, la cuál es una función recursiva y no podemos calcular el orden de forma sencilla.

Las complejidades asintóticas de un algoritmo de vuelta atrás son elevadas, por lo que estas complejidades pueden oscilar entre factorial y exponencial, por lo que vamos a suponer que nos encontramos en el peor de los casos y nos hallamos ante un problema de complejidad asintótica de $O(n^n)$.

# 2.- Algoritmo Para Obtener El Camino Más Corto.

## 2.1.- Descripción y tipificación.

El primer programa sabemos que tarda bastante tiempo en darte los resultados y tiene muchos más caminos con pocos clientes nuevos , pero a cambio puede encontrarnos un camino, todos los caminos o el camino más corto, dependiendo del tiempo que tengamos.

Entonces hemos decidido mejorar este programa para que tarde menos tiempo en encontrarnos un camino:

Lo primero que hace es buscar un camino existente que pase por todas las casas, el cual lleva un tiempo de ir de una casa a otra, una vez encontrado ese camino lo dejamos como el mejor camino hasta el momento y buscamos el siguiente, si este se puede realizar en menos tiempo, lo sustituimos por el anterior.

¿Pero cómo hace para ser tan rápido el programa? Pues la cualidad que tiene es que cuando un camino, el cual no ha pasado aún por todas las casas, pero la suma del tiempo de las primeras casas ya es igual al camino que tenemos guardado, directamente deja de buscar en ese camino y todos los posibles caminos existentes que comienzan pasando por las mismas casas que acabamos de pasar.

### 2.1.1. Tipificación

La tipificación es igual que la del primer algoritmo.

### 2.1.2. Función objetivo

En este algoritmo si buscamos en específico minimizar los costes, por lo cual esta es la función objetivo.

$$\text{Min}\left\{\sum_{i=1}^{n-1} Coste(v_i, v_{i+1}) + Coste(v_n, v_1)\right\}$$

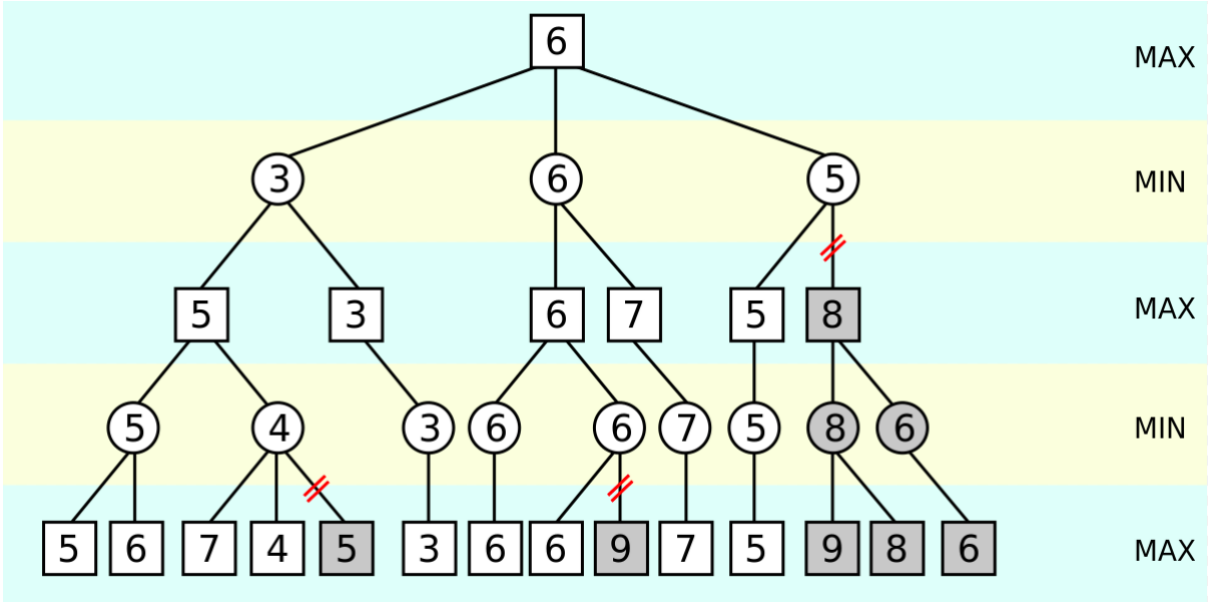
### 2.1.3. Restricciones

En la mejora se le añade una restricción en la que:

- costeOptimo debe de ser menor que costeV.

## 2.2.- Estrategia de programación.

El algoritmo que vamos a utilizar es el de vuelta a atrás con ramificación y poda. Vamos a usar este algoritmo a modo de mejora del que teníamos anteriormente, ya que mediante esta técnica se puede obtener una solución o la solución óptima, debido a la poda. Este algoritmo consiste en realizar una búsqueda exhaustiva y sistemática de la solución, encontrando un coste óptimo y podando todos los costes que sean peores que el coste óptimo.



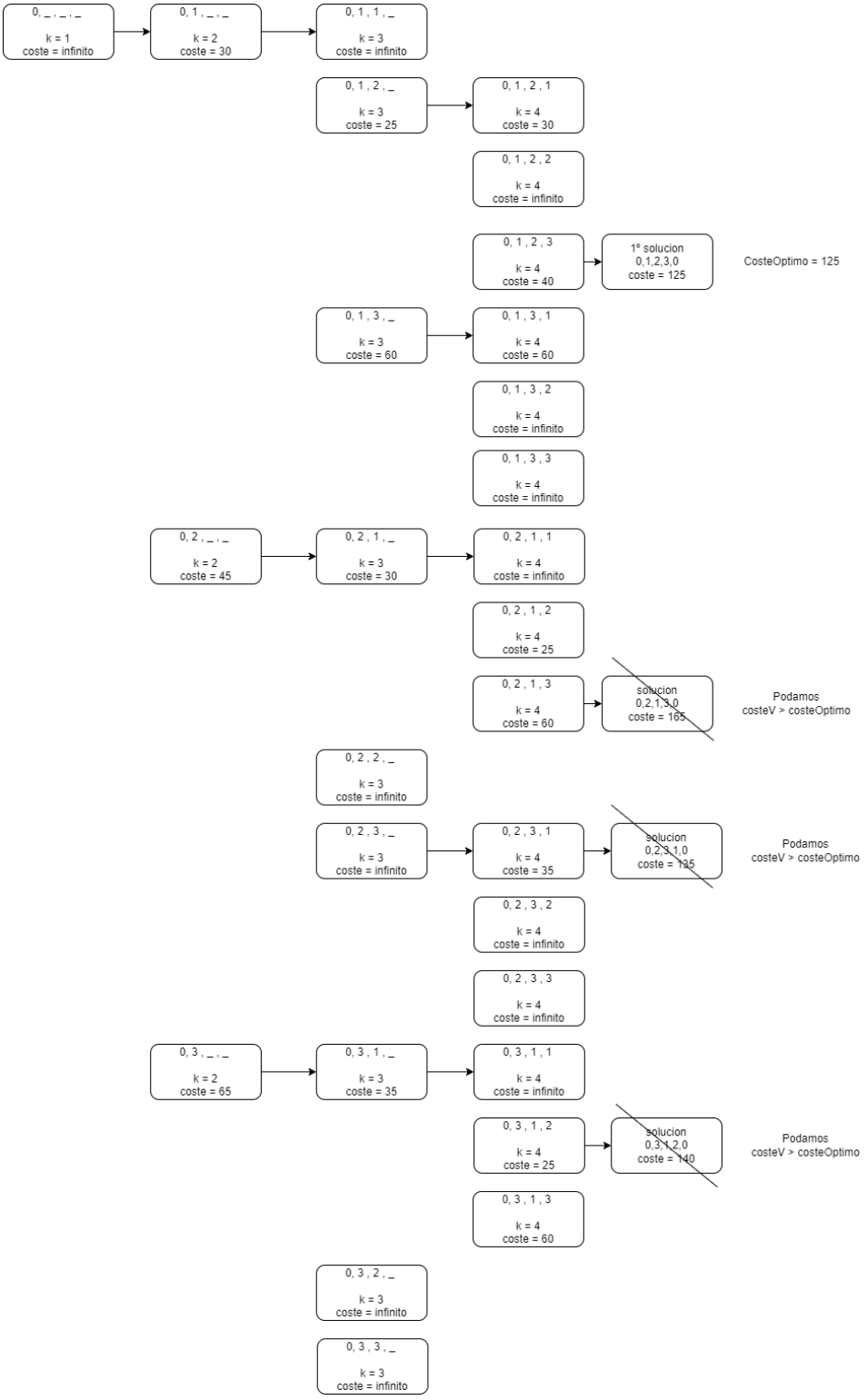
## 2.3.- Pseudocódigo.

```
costeÓptimo ← +∞
para i ← 0 hasta n
        V_i ← i
        Voptimo_i ← 0
fpara
k ← 1

función Viajante2 (Coste:real+[n,n], V:&natural[n], Vóptimo:&natural[n], costeÓptimo:&real,
k:entero)
        costeV: real
        V_k ← 0
        mientras V_k ≠ n hacer
                V_k ← V_k + 1
                si Coste_{Vk-1, Vk} ≠ ∞ y Cliclos (V) = FALSO
                        costeV ← CalcularCoste2(Coste, V, k, costeÓptimo)
                        si costeV ← ∞
                                si k = n
                                        si Coste_{Vk, V0} ≠ ∞
                                                si costeV < costeÓptimo
                                                        costeÓptimo ← costeV
                                                        Vóptimo ← V
                                                fsi
                                        fsi
                                si no
                                        Viajante2(Coste, V, Vóptimo, costeÓptimo, k+1)
                                        V_{k+1} ← -1
                                fsi
                        sí no
                                V_{k+1} ← -1
                        fsi
                fsi
        fmientras
ffunción
```

```
función CalcularCoste2(Coste:real+[n,n], V:&natural[n], k:entero, costeÓptimo:real):real
        suma: real
        i: natural
        x: natural
        suma ← 0
        x ← 0
        para i ← 1 hasta k
                si costeOptimo > suma + Coste_{x,Vi}
                        suma ← suma + Coste_{x,Vi}
                        x ← V_i
                        si i = k
                                suma ← suma + Coste_{Vi,0}
                        fsi
                sí no
                        devolver ∞
                fsi
        fpara
        devolver suma
ffunción

función Cliclos(V:&natural[n]):booleano
        i: natural
        j:natural
        para i ← 1 hasta n
                para j ← i+1 hasta n
                        si V_i ≠ -1
                                si V_i = V_j
                                        devolver verdadero
                                fsi
                        fsi
                fpara
        fpara
        devolver falso
ffunción
```

## 2.4.- Ejemplo.

Vamos a usar el ejemplo del apartado 1. El grafo será el mismo y lo único que variará será la traza.




En este caso como se puede observar solo se está podando el nodo hoja, pero no tiene por que podarse el nodo hoja, se podará en el momento en que la suma de los costes sea superior al coste óptimo.

## 2.5.- Complejidad asintótica.

| | NV1 | NV2 | NV3 | NV4 | NV5 |
|---|---|---|---|---|---|
| (1) costeÓptimo ← +∞ | O(1) | | | | |
| (2) para i ← 0 hasta n | O(n) | | | | |
| (3) $V_i$ ← i | | O(1) | | | |
| (4) $Voptimo_i$ ← 0 | | O(1) | | | |
| (5) fpara | | | | | |
| (6) k ← 1 | O(1) | | | | |

(7) función **Viajante2** (Coste:real+[n,n], V:&natural[n], Vóptimo:&natural[n], costeÓptimo:&real, k:entero)
(8) costeV: real
(9) $V_k$ ← 0 **$t_{(1)} = 2$**
(10) mientras $V_k$ ≠ n hacer **$t_{(2)} = T(n^3)$**
(11) $V_k$ ← $V_k$ + 1 **$t_{(3)} = 3$**
(12) si $Coste_{Vk-1, Vk}$ ≠ ∞ y Cliclos (V) = FALSO **$t_{(4)} = T(n^2)$**
(13) costeV ← CalcularCoste2(Coste, V, k, costeÓptimo) **$t_{(5)} = T(n)$**
(14) si costeV ← ∞ **$t_{(6)} = 2$**
(15) si k = n **$t_{(7)} = 1$**
(16) si $Coste_{Vk, V0}$ ≠ ∞ **$t_{(8)} = 1$**
(17) si costeV < costeÓptimo **$t_{(9)} = 1$**
(18) costeÓptimo ← costeV **$t_{(10)} = 3$**
(19) Vóptimo ← V **$t_{(11)} = 3$**
(20) fsi
(21) fsi
(22) si no
(23) Viajante2(Coste, V, Vóptimo, costeÓptimo, k+1)**$t_{(15)} =$ “”**
(24) $V_{k+1}$ ← -1 **$t_{(16)} = 2$**
(25) fsi
(26) sí no
(27) $V_{k+1}$ ← -1 **$t_{(17)} = 2$**
(28) fsi
(29) fsi
(30) fmientras
(31) ffunción

(32) función **CalcularCoste2**(Coste:real+[n,n], V:&natural[n], k:entero, costeÓptimo:real):real O(n)

(33) suma: real
(34) i: natural
(35) x: natural
(36) suma ← 0 O(1)
(37) x ← 0 O(1)
(38) para i ← 1 hasta k O(n)
(39) si costeOptimo > suma + $Coste_{x,Vi}$ O(1)
(40) suma ← suma + $Coste_{x,Vi}$ O(1)
(41) x ← $V_i$ O(1)
(42) si i = k O(1)
(43) suma ← suma + $Coste_{Vi,0}$ O(1)
(44) fsi
(45) sí no
(46) devolver ∞ O(1)
(47) fsi
(48) fpara
(49) devolver suma O(1)
(50) ffunción

Esta función es la misma que la usada en el algoritmo anterior.

(51) función **Ciclos**(V:&natural[n]):booleano $O(n^2)$
(52) i: natural
(53) j:natural
(54) para i ← 1 hasta n O(n)
(55) para j ← i+1 hasta n O(n)
(56) si $V_i \neq -1$ O(1)
(57) si $V_i = V_j$ O(1)
(57) devolver verdadero O(1)
(58) fsi
(59) fsi
(60) fpara
(61) fpara
(62) devolver falso O(1)
(63) ffunción

Este algoritmo, el Viajante 2, reduce mucho los tiempos de ejecución por lo cual es más eficiente trás las pruebas empíricas, aun así la complejidad asintótica en este algoritmo es similar a la anterior, teóricamente y en el peor de los casos junto con la recursividad podemos llegar a tener una complejidad asintótica de $O(n^n)$.

## 2.6. - Evidencias.

Para la comparación de ambos algoritmos vamos usar el fichero llamado “10clientes.txt”. Esta matriz es una matriz de 11x11, es decir, 10 clientes más el supermercado. Haciendo una simulación lo más parecida posible a la vida real, hemos indicado que todos los clientes pueden viajar a todos los clientes, habiendo así el máximo de caminos posibles en la matriz. De este modo, habrá n! caminos en el primer algoritmo.

Ejecución del primer algoritmo ocultando todos los recorridos:

```
Listado de recorridos resaltando el mas corto

Tiempo de ejecucion: 4459ms
Recorridos obtenidos: 3628800
Recorrido mas corto: (Super, 2, 8, 7, 4, 3, 9, 6, 10, 1, 5, Super)
Coste: 125
```

Ejecución del algoritmo mejorado:

```
Listado de recorridos resaltando el mas corto mejorado

Tiempo de ejecucion: 19ms
Recorridos obtenidos: 230
Recorrido mas corto: (Super, 2, 8, 7, 4, 3, 9, 6, 10, 1, 5, Super)
Coste: 125
```

La mejora en el segundo algoritmo será más notable cuanto antes encuentre una solución óptima, ya que en el momento que la encuentre podará el resto de soluciones.

# 3.- Bibliografía.

Título: Tema 1: Análisis de algoritmos

Autor/-es: Yolanda Marhuenda

Año:  2021/2022

Título: Tema 2: Algoritmos Recursivos

Autor/-es: Yolanda Marhuenda

Año:  2021/2022

Título: Tema 6: Algoritmos de Vuelta Atrás

Autor/-es: Yolanda Marhuenda

Año:  2021/2022

Título: Tema 7: Ramificación y Poda

Autor/-es: Yolanda Marhuenda

Año:  2021/2022

Título: Práctica 6. Lectura de Ficheros

Autor/-es: Manuel Quesada y María Dolores Guillen

Año:  2021/2022

Título: Práctica 8. Vuelta Atrás

Autor/-es: Manuel Quesada y María Dolores Guillen

Año:  2021/2022