RAMIFICACIÓN Y PODA:
PROBLEMA DEL VIAJANTE

Paula Muñoz Lago

Grado Ingeniería Informática

1. DESCRIPCIÓN

El problema consiste en encontrar un camino mínimo para recorrer n ciudades, sin repetir ninguna y volviendo siempre al punto de partida.

2. OBJETIVOS

El objetivo de esta práctica es comprender el método de ramificación y poda, aplicando diferentes cotas para hacer la solución más eficiente.

3. ALGORITMO

El funcionamiento del programa es el siguiente: se carga del archivo “grafo.txt” todos los caminos y ciudades con sus costes en una matriz de adyacencia. A continuación llama al algoritmo, que se ha programado en la función *double viajante_rp(int **mat, int N, vector<int> &sol_mejor)*, ésta devuelve el coste del recorrido mínimo que debe hacer el viajante, así como la mejor solución, es decir, las ciudades que debe recorrer y su orden. Como se ha mencionado anteriormente, el algoritmo recibe el grafo, cuyos nodos serán las ciudades y las aristas el coste de ir de una a otra, así como el número de ciudades a recorrer.

El algoritmo funciona haciendo uso de una **cola de prioridad**, ésta ordena de menor a mayor el coste estimado de los nodos contenidos en ella, de tal forma que en la “cima” se encontrará siempre al nodo de menor coste estimado.

Para evitar repetir soluciones, es decir, no contemplar como soluciones distintas las siguientes:

Madrid - Toledo - Segovia - Madrid

Segovia - Madrid - Toledo - Segovia

se fija una primera ciudad de partida, desde la cual pueden estudiarse distintas soluciones pero siempre empezando por ese nodo.

Por ello, la primera ciudad se introduce en la cola de prioridad y, si el coste estimado de recorrer todas las ciudades desde ésa, es menor que el coste mejor, (inicialmente inicializado a infinito), estudiará sus hijos, es decir, explorará sus ciudades adyacentes.

El **coste estimado** de cada nodo, es la cota optimista, utilizada para realizar la poda de soluciones. Se distinguen dos cotas, la primera de ellas inicialmente guarda en un vector el coste mínimo de ir de cada nodo a cualquier otro, es decir, guarda el mínimo de cada fila de la matriz de adyacencia. De esta forma, cada vez que es necesario calcular el coste estimado de un nodo, se tendrá en cuenta la solución parcial de ese momento, para ver cuales son los nodos que faltan por recorrer y así sumar sus arista de coste mínimo, junto con el coste de llegar al nodo actual. La segunda cota resulta algo más costosa, pero a su vez mucho más precisa, busca el elemento mínimo de cada fila y resta ese valor a todos los elementos de dicha fila, y repitiendo el proceso con cada una y de la misma manera con las columnas. Finalmente la suma de elementos mínimos en las filas y después en las columnas generan el coste estimado. Además, va reduciendo la matriz de tal forma que el coste en sus nodos hijos será más bajo.

En el siguiente ejemplo se estudia el funcionamiento de esta segunda cota, con una matriz de tamaño 5*5, se calcula el coste estimado de realizar el recorrido desde el primer nodo.

Mínimo por fila

-1	20	30	10	11
15	-1	16	4	2
3	5	-1	2	4
19	16	18	-1	3
16	4	7	16	-1

10
2
2
3
4

⟹

-1	10	20	0	1
13	-1	4	2	0
1	3	-1	0	2
16	13	15	-1	0
12	0	3	12	0

1 0 3 0 0 Mínimo por columna

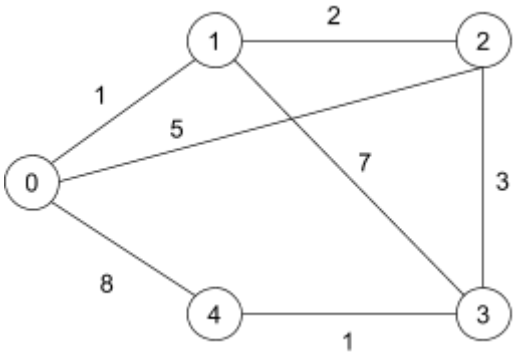
La matriz resultante será:

-1	10	17	0	1
12	-1	1	2	0
0	3	-1	0	2
15	13	12	-1	0
11	0	0	12	0

El coste estimado resultante es la suma de todos los mínimos, 25

Se repetirá el proceso para cada nodo explorado. Siendo x y z nodos tales que z es hijo de x, se cambiarán los elementos de la fila x, y los elementos de la columna z, por -1, para a continuación realizar el cálculo previamente explicado.

Finalmente, si el grafo tiene un circuito hamiltoniano, encontrará esa solución y guardará el coste de ésta como el mejor, para que, si los nodos que nos quedaban por explorar, pertenecientes a soluciones distintas, tuviesen un coste estimado mayor, poder descartar esas soluciones.
A continuación se muestra un **ejemplo de ejecución con la primera cota**, para el grafo:



costes_minimos:

0	1	2	3	4
1	1	2	1	1

En este caso el algoritmo explora un total de 11 nodos, el tiempo medio de exploración de cada nodo es 1 milisegundo y se ejecuta en un tiempo total de 23 ms.

```
1- Exploramos nodo: 0
2- Exploramos nodo: 1
3- Exploramos nodo: 2
4- Exploramos nodo: 3
5- Exploramos nodo: 2
6- Exploramos nodo: 1
7- Exploramos nodo: 3
8- Exploramos nodo: 3
9- Exploramos nodo: 4
10- Exploramos nodo: 4
11- Exploramos nodo: 2
En total hemos explorado 11 nodos
El tiempo medio de exploracion de cada nodo es 1
El coste de la mejor solucion tiene coste: 15
0 1 2 3 4
El tiempo total de ejecucion es 23
```

Indice	Nuevo nodo	Solución parcial	Coste estimado	Orden exploracion
1	0	0, 1	1 + (2 + 1 + 1) = 5	1
1	0	0, 2	5 + (1 + 1 + 1) = 8	5
2	1	0, 1, 2	3 + (1 + 1) = 5	2
2	1	0, 1, 3	8 + (2 + 1) = 10	7
3	2	0, 1, 2, 3	6 + 1 = 7	3
4	3	0, 1, 2, 3, 4	15	4
5	2	0, 2, 1	7 + (1 + 1) = 9	6
5	2	0, 2, 3	8 + (1 + 1) = 10	8
6	1	0, 2, 1, 3	14 + 1 = 15	descartado
7	3	0, 1, 3, 2	11 + 1 = 12	11
7	3	0, 1, 3, 4	9 + 1 = 10	9
8	3	0, 2, 3, 1	15 + 1 = 16	descartado
8	3	0, 2, 3, 4	9 + 1 = 10	10
9	4	0, 1, 3, 4, x		descartado
10	4	0, 2, 3, 4, x		descartado
11	2	0, 1, 3, 2, x		descartado

Las razones por las que se han descartado soluciones han sido porque, o bien el coste estimado resultaba mayor o igual al coste de la primera solución encontrada, o bien porque no era un circuito cerrado.

Se estudia la **ejecución de la segunda cota** para el mismo grafo (ver figura), que como puede verse en la imagen, poda más que la cota anterior ya que es más precisa, aunque más costosa en cuanto a tiempo, por lo que estudia menos posibilidades.

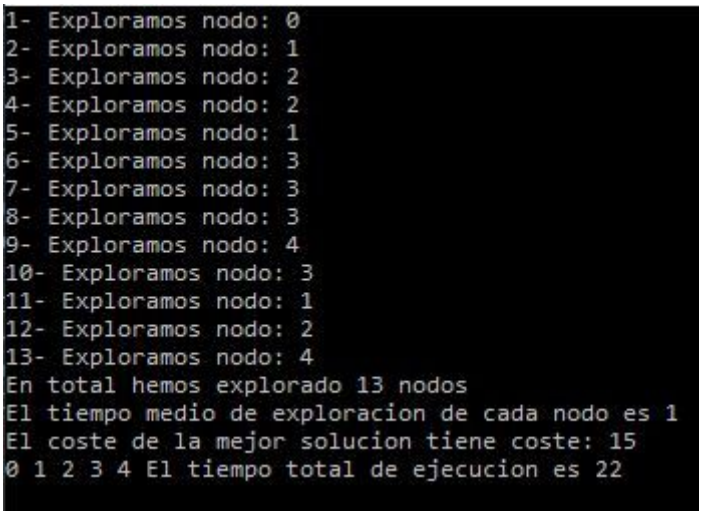
```
1- Exploramos nodo: 0
2- Exploramos nodo: 1
3- Exploramos nodo: 2
4- Exploramos nodo: 2
5- Exploramos nodo: 1
6- Exploramos nodo: 3
7- Exploramos nodo: 3
8- Exploramos nodo: 4
En total hemos explorado 8 nodos
El tiempo medio de exploracion de cada nodo es 2
El coste de la mejor solucion tiene coste: 15
0 1 2 3 4
El tiempo total de ejecucion es 28
```

Indice	Nuevo nodo	Solución parcial	Coste estimado	Orden exploracion
1	0	0, 1	1 + 2 = 3	1
1	0	0, 2	5 + 0 = 5	3
2	1	0, 1, 2	3 + 0= 3	2
2	1	0, 1, 3	8 + 17 = 25	10 descartado
3	2	0, 1, 2, 3	6 + 6 = 12	5 descartado
4	2	0, 2, 1	7 + 0 = 7	4
4	2	0, 2, 3	8 + 6 = 14	7
5	1	0, 2, 1, 3	14 + 6 = 20	9 descartado
6	3	0, 1, 2, 3, 4	15	6
7	3	0, 2, 3, 4	9 + 0 = 9	8 descartado

Como se ha comentado anteriormente, las razones por las que se descarta un nodo pueden ser, o bien porque el coste estimado resultante es mayor al de la mejor solución obtenida, o bien porque no cumple la condición de que sea un circuito hamiltoniano, es decir, el último nodo visitado tiene que tener una arista con el primero y no visitar el mismo nodo dos veces.

Tras estudiar la ejecución del algoritmo con ambas cotas, se ve a continuación cuál sería el resultado si no usase ninguna de ellas, por lo que es de esperar que el algoritmo tenga que trabajar más.

Explora un total de 13 nodos, aunque llega al mismo resultado, en un tiempo menor al de las dos últimas ejecuciones. Esto se debe a que, en este caso, se ahorra el cálculo de la cota, debido a que no poda en ningún momento más que la poda de factibilidad.



4. CONCLUSIONES

Cabe destacar la importancia de elegir una buena cota a la hora de realizar un problema de ramificación y poda, ya que es mejor una cota un poco más costosa, aunque más ajustada y que pode más, a una de cálculo rápido pero que no pode mucho, haciendo al algoritmo expandir nodos de forma innecesaria.