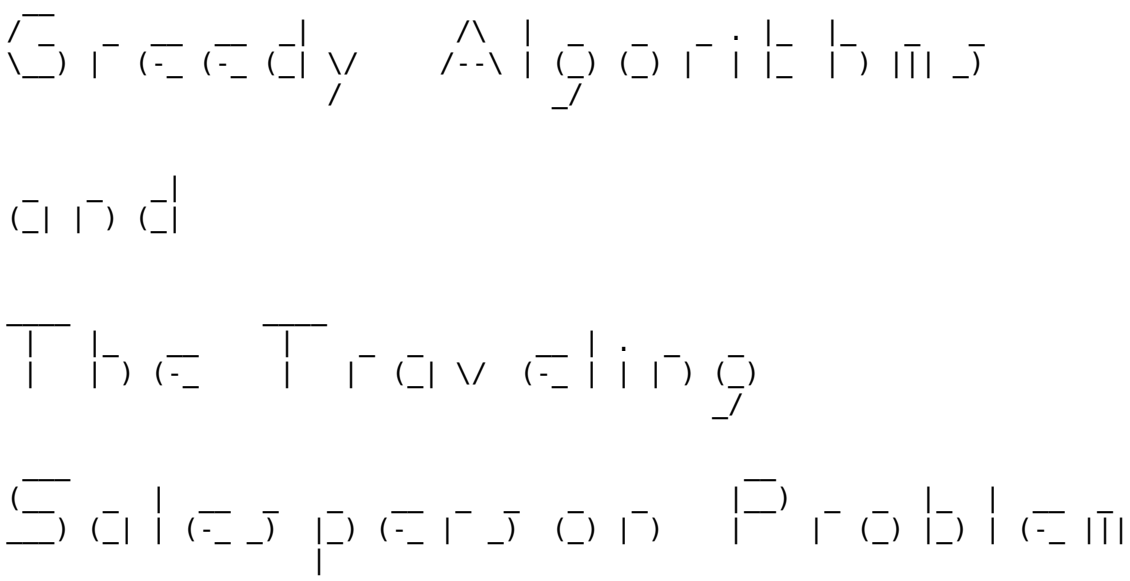
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬ 1. Introducción.

El problema del “traveling salesperson problem”, conocido por sus siglas TSP, es un problema clásico y difícil de resolver.

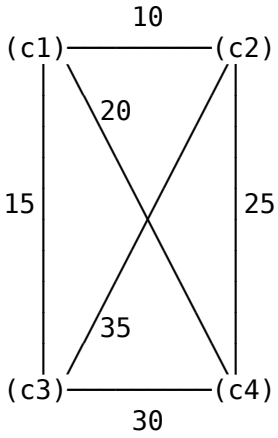
▬▬▬ 2. Descripción del Problema.

Tenemos un conjunto de ciudades, y las distancias que conectan cada una de estas ciudades. El problema consiste en encontrar la ruta más corta que visita cada ciudad una única vez y regresa al punto de inicio.

El grafo está completamente conectado, para cada ciudad existe una ruta hacia las demás ciudades. Existe lo que se conoce como un “ciclo de Hamilton” en el grafo. Tiene que existir un recorrido completo que pase por todas las ciudades. De los posibles recorridos, deseamos encontrar el de menor distancia.

▬▬▬ 3. Estructura del Problema.

Supongamos que tenemos el siguiente grafo no dirigido.



El grafo suele representarse en diferentes libros de texo mediante una matriz. Observe que se pone un valor muy alto, denominado “M” en la diagonal, pues no deseamos que la ruta contenga la misma ciudad más de una vez.

	c1	c2	c3	c4
c1	M	10	15	20
c2	10	M	35	25
c3	15	35	M	30
c4	20	25	30	M

De la misma forma, podemos representar el problema por medio de una lista de adyacencias:

```
grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
```

[c2, [[c1,10], [c3,35], [c4,25]],
[c3, [[c1,15], [c2,35], [c4,30]],
[c4, [[c1,20], [c2,25], [c3,30]]
]

▬▬▬▬ 4. Solución de Forma Exhaustiva.

Para resolver el problema de forma exhaustiva, se deben enumerar todos los posibles recorridos. Como el recorrido es circular, se inicia en una ciudad específica, recorro las demás ciudades y vuelvo a esa ciudad. Sin que el problema se vea afectado podemos suponer que siempre iniciamos en la ciudad “c1”.

A continuación se muestran los posibles recorridos

[c1,c2,c3,c4,c1],
[c1,c2,c4,c3,c1],
[c1,c3,c2,c4,c1],
[c1,c3,c4,c2,c1],
[c1,c4,c2,c3,c1],
[c1,c4,c3,c2,c1]

Posteriormente, se debe evaluar la distancia de cada una de las rutas. Por ejemplo para la ruta:

[c1,c2,c3,c4,c1]

Se tiene un recorrido de la forma:

c1 → c2 → c3 → c4 → c1

Cuya distancia estará dado por:

c1 → c2, tiene una distancia de 10
c2 → c3, tiene una distancia de 35
c3 → c4, tiene una distancia de 30
c4 → c1, tiene una distancia de 20

para un total de: 10 + 35 + 30 + 20 = 95

Aquí tenemos el cálculo de las distancias para las demás rutas:

[c1,c2,c3,c4,c1] → 10 + 35 + 30 + 20 = 95 unidades

[c1,c2,c4,c3,c1] → 10 + 25 + 30 + 15 = 80 unidades

[c1,c3,c2,c4,c1] → 15 + 35 + 25 + 20 = 95 unidades

[c1,c4,c2,c3,c1] → 20 + 25 + 35 + 15 = 95 unidades

[c1,c3,c4,c2,c1] → 15 + 30 + 25 + 10 = 80 unidades

[c1,c4,c3,c2,c1] → 20 + 25 + 35 + 10 = 95 unidades

Por la tanto la solución será uno de estos dos posibles recorridos:

[c1,c2,c4,c3,c1] → 10 + 25 + 30 + 15 = 80 unidades

[c1,c3,c4,c2,c1] → 15 + 30 + 25 + 10 = 80 unidades

Observemos que ambos son el mismo recorrido, pero en un sentido inverso.

▬▬▬ 5. Análisis de Complejidad Algoritmo Exhaustivo.

Observemos que para este caso de 4 ciudades, como iniciamos y terminamos en “c1”, el número de rutas intermedias está dado por:

De “c1” tengo 3 ciudades a donde ir.
Posteriormente 2 ciudades a donde ir.
Posteriormente 1 ciudad a donde ir.

Por lo tanto para 4 ciudades el número de opciones está dado por:

(4-1)! = 3! = 3*2*1 = 6 rutas

Para 5 ciudades tengo que revisar:

(5-1)! = 4! = 24 rutas

Observe como crece este problema:

Para 8 ciudades: (8-1)! = 7! = 5040 rutas

Para 10 ciudades: (10-1)! = 9! = 362880 rutas

Para 20 ciudades: (20-1! = 19! = 121645100408832000 rutas

En general, para un problema con “n” ciudades, tengo que revisar:

(n-1)! rutas

Por lo tanto, podemos decir que la complejidad del problema está dada por:

O(n!)

▬▬▬ 6. Solución con un Algoritmo Greedy.

La implementación de un algoritmo “greedy” para el TSP es la siguiente.

1. Se tiene una lista con los nodos recorridos.
   Inicialmente esta lista tiene el nodo inicial.

2. Desde el último nodo de la lista con los nodos recorridos
   se escoge el vecino con la ruta de menor valor.

3. Se agrega el nodo a la lista de nodos recorridos.

4. Se repite el proceso hasta tener todos los nodos en la lista.

Supongamos que tenemos el siguiente grafo:

```
grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
[ c2, [ [c1,10], [c3,35], [c4,25] ],
[ c3, [ [c1,15], [c2,35], [c4,30] ],
[ c4, [ [c1,20], [c2,25], [c3,30] ]
]
```

>>> Iniciamos el algoritmo:

```
grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
[ c2, [ [c1,10], [c3,35], [c4,25] ],
[ c3, [ [c1,15], [c2,35], [c4,30] ],
[ c4, [ [c1,20], [c2,25], [c3,30] ]
]
```

nodos: [c1]

arco con menor valor: [c2,10]

Valor Ruta: 10

>>> Continuamos el proceso:

```
grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
[ c2, [ [c1,10], [c3,35], [c4,25] ],
[ c3, [ [c1,15], [c2,35], [c4,30] ],
[ c4, [ [c1,20], [c2,25], [c3,30] ]
]
```

nodos:[c1,c2]

arco con menor valor: [c4,25]

Valor Ruta: 10 + 25

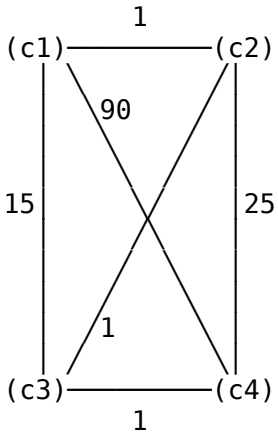
>>> Continuamos el proceso:

```
grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
[ c2, [ [c1,10], [c3,35], [c4,25] ],
[ c3, [ [c1,15], [c2,35], [c4,30] ],
[ c4, [ [c1,20], [c2,25], [c3,30] ]
]


nodos:[c1,c2,c4]

arco con menor valor: [c3,30]

Valor Ruta: 10 + 25 + 30



>>> Finalizamos.


grafo:
[
[ c1, [ [c2,10], [c3,15], [c4,20] ],
[ c2, [ [c1,10], [c3,35], [c4,25] ],
[ c3, [ [c1,15], [c2,35], [c4,30] ],
[ c4, [ [c1,20], [c2,25], [c3,30] ]
]

Cuando se tienen todos los nodos se vuelve al nodo inicial:

nodos: [c1,c2,c4,c3,c1]

Valor Ruta: 10 + 25 + 30 + 15



>>> Respuesta final:

Entonces la ruta está dada por:

c1 → c2 → c4 → c3 → c1


Con un valor de:

10 + 25 + 30 + 15 = 80
```

En este caso, el algoritmo “greedy” encontró la misma solución que el algoritmo exhaustivo. Esto no siempre es así. Como se ha dicho, los algoritmos “greedy” en muchas ocasiones dan soluciones subóptimas. Es decir, una solución aproximada a la solución óptima.

▬▬▬ 7. El Algoritmo Greedy es Aproximado.

Tal como se ha mencionado anteriormente, el algoritmo greedy no siempre da la mejor solución. Observe el siguiente ejemplo:



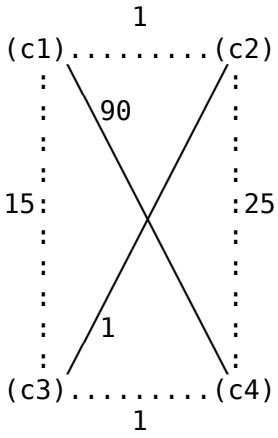
grafo:

```
[
[ c1, [ [c2, 1], [c3,15], [c4,90] ] ],
[ c2, [ [c1, 1], [c3, 1], [c4,25] ] ],
[ c3, [ [c1,15], [c2, 1], [c4, 1] ] ],
[ c4, [ [c1,90], [c2,25], [c3, 1] ] ]
];
```

Si se ejecuta el algoritmo exhaustivo se obtienen las siguientes soluciones:

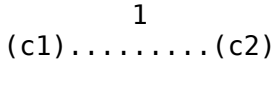
[c1,c2,c4,c3,c1] con un valor de 42

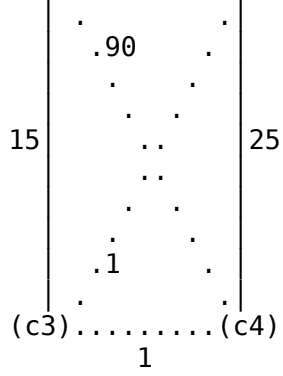
[c1,c3,c4,c2,c1] con un valor de 42



Si se ejecuta el algoritmo greedy, esta es la solución que produce:

[c1,c2,c3,c4,c1] con un valor de 93





▬▬▬ 8. Análisis de Complejidad del Algoritmo Greedy.

Se escoge un nodo en cada iteración del algoritmo. De esta forma en la primera iteración si partimos de un nodo particular tenemos:

(n-1) nodos para escoger.

En la segunda iteración tenemos:

(n-2) nodos para escoger y así sucesivamente.

Por lo tanto:

$$(n-1) + (n-2) + ... + 1$$

$$= 1 + 2 + ... + (n-2) + (n-1)$$

$$= \frac{(n-1)*n}{2}$$

$$= O(n^2)$$

▬▬▬ 9. Resumen.

• El TSP es un problema muy común que aparece en muchas aplicaciones de la vida diaria.

• Al resolver el TSP de forma exhaustiva, su complejidad es de O(n!)

• Al resolver el TSP con un algoritmo “greedy”, su complejidad puede llegar a ser de $O(n^2 * n*log(n))$

• El algoritmo “greedy” es de aproximación, lo que significa que no siempre da la solución óptima, en muchos casos da solamente una aproximación.

▬▬▬ 10. Ejercicios.

●● Ejercicio 1.

Construya un programa que realice el algoritmo exhaustivo del TSP.
Utilice los datos del ejemplo.

●● Ejercicio 2.

Construya un programa que realice el algoritmo greedy del TSP.
Utilice los datos del ejemplo.

●● Ejercicio 3.

Existe un algoritmo llamado el algoritmo rutas de Hamilton.
Investigue este algoritmo.
Tiene alguna relación con el TSP?

●● Ejercicio 4.

Existe un algoritmo llamado el algoritmo de “clique”.
Investigue este algoritmo.
Tiene alguna relación con el TSP?