Reporte: Algoritmo TSP (Dinámico)

Osvaldo de Luna

2 de noviembre de 2019

Problema TSP

Un hombre de negocios debe recorrer n ciudades las cuales están conectadas por un camino con distancia no necesariamente igual, es decir, desde una ciudad puedes llegar a cualquier otra. El problema es que no quiere recorrer una ciudad dos veces y quiere recorrer todas y cada una de ellas de la manera más óptima (es decir, con la menor distancia recorrida) hasta regresar a la ciudad donde inició su viaje. La única información que tiene esta persona es la ciudad de donde comenzará su recorrido y la distancia entre cada ciudad.

Solución propuesta

Con la ayuda de una clase creada y nombrada 'Map', se podrá operar el proceso dinámico para llegar a la solucón. Esta clase nos otorga dos valores: un String nombre y un Int valor. Lo que se pretende es almacenar el recorrido (nombre) y la distancia recorrida en él (valor), esto con la idea de poder manipular posteriormente el nombre ya existente para generar las nuevas combinaciones de recorridos y con el valor del recorrido anterior calcular el nuevo recorrido.
Para fines de ilustrar mejor el proceso del algoritmo, se estará estudiando el ejemplo con la siguiente matriz de caminos entre 4 ciudades:

Ciudades	**0**	**1**	**2**	**3**
0	0	10	8	3
1	10	0	1	9
2	8	1	0	7
3	3	9	7	0

Cuadro 1: Caminos

La idea es crear una nueva tabla por cada nueva ciudad agregada, es decir, se obtendrá una solución para menos ciudades, esto para que a partir de recorridos ya generados se pueda llegar a aquél en el que es visitan todas las ciudades.
Por ello, se generará una nueva tabla con el caso base de caminos con una sola ciudad x de inicio y un recorrido de 0, lo que en la clase 'Map' sería un nombre=x y valor=0, para cada ciudad existente.

Recorrido		**Distancia**
"0"	=	0
"1"	=	0
"2"	=	0
"3"	=	0

Cuadro 2: Recorridos con 1 ciudad

A partir de la tabla ya creada se generará una nueva que contenga una ciudad más en los recorridos, creando cada posible movimiento a partir de la ciudad presente.

Nombre		**Valor**
"01"	=	10
"02"	=	8
"12"	=	1
"13"	=	9
"20"	=	8
"21"	=	1
"23"	=	7
"30"	=	3
"31"	=	9
"32"	=	7

Cuadro 3: Recorridos con 2 ciudades

Se seguirá con ese procedimiento de manera sucesiva. Cabe destacar que para calcular el valor del nuevo recorrido, se toma el valor del recorrido anterior y acorde a la ciudad agregada se le suma el valor en la matriz de

caminos, esto con la última ciudad visitada en el camino anterior y la ciudad agregada.

Nombre		Valor
"012"	=	11
"013"	=	19
"021"	=	9
"023"	=	15
"031"	=	12
"032"	=	10
"102"	=	18
"103"	=	13
"120"	=	9
"123"	=	8
"130"	=	12
"132"	=	16
"201"	=	18
"203"	=	21
"210"	=	11
"213"	=	10
"230"	=	10
"231"	=	16
"301"	=	13
"302"	=	11
"310"	=	19
"312"	=	10
"320"	=	15
"321"	=	8

Cuadro 4: Recorridos con 3 ciudades

Se seguirá así hasta obtener las n ciudades, en este caso solo se ilustrará la tabla final con inicio en la ciudad 0 y 1.

Para concluir con los recorridos, el algoritmo realiza un regreso a ca-

Nombre		Valor
"0123"	=	18
"0132"	=	26
"0213"	=	18
"0231"	=	24
"0312"	=	13
"0321"	=	12
"1023"	=	25
"1032"	=	20
"1203"	=	12
"1230"	=	11
"1302"	=	20
"1320"	=	24

Cuadro 5: Recorridos con 4 ciudades, solo con inicio en las ciudades 0 y 1

sa, donde a partir de todos los recorridos con todas las ciudades ahora hace el regreso a la ciudad inicial y hace el respectivo cálculo al valor del recorrido.

Finalmente, se busca el recorrido con la menor distancia recorrida acorde a la ciudad de inicio solicitada, y así es como se llega a la solución del problema.

Solución con ciudad inicial 0: Recorrido "01230" Valor 21
Solución con ciudad inicial 1: Recorrido "10321" Valor 21

Nombre		Valor
"01230"	=	21
"01320"	=	34
"02130"	=	21
"02310"	=	34
"03120"	=	21
"03210"	=	21
"10231"	=	34
"10321"	=	21
"12031"	=	21
"12301"	=	21
"13021"	=	21
"13201"	=	34

Cuadro 6: Recorridos completos, solo con inicio en las ciudades 0 y 1

Análisis

Para el ejemplo anterior, tenemos

```
//PROBLEMA TSP
    int [][] caminos = new int[][]{
        {0,     10,     8,      3},
        {10,    0,      1,      9},
        {8,     1,      0,      7},
        {3,     9,      7,      0}
    };
    TSP tsp = new TSP(caminos, 1);
    tsp.buscarSolucion();
```

Figura 1: Código del ejemplo

```
compile-single:
run-single:
Camino: 01230, valor=21

BUILD SUCCESSFUL (total time: 5 seconds)
```

Figura 2: Compilación del ejemplo, con ciudad inicial 0

```
compile-single:
run-single:
Camino: 10321, valor=21

BUILD SUCCESSFUL (total time: 3 seconds)
```

Figura 3: Compilación del ejemplo, con ciudad inicial 1

Podemos ver que los resultados son los esperados. Ahora veamos un ejemplo visiblemente sencillo aunque con mayor número de ciudades para asegurar el funcionamiento del algoritmo.

```
//PROBLEMA TSP
    int [][] caminos = new int[][]{
        {0,         1,     100,       100,       100},
        {100,       0,     100,       100,       1},
        {100,     100,       0,         1,       100},
        {1,       100,     100,         0,       100},
        {100,     100,       1,       100,         0}
    };
    TSP tsp = new TSP(caminos, 0);
    tsp.buscarSolucion();
```

Figura 4: Camino de la prueba sencilla de intuir

```
compile-single:
run-single:
Camino: 014230, valor=5

BUILD SUCCESSFUL (total time: 2 seconds)
```

Figura 5: Compilación de la prueba sencilla de intuir, con ciudad inicial 0

```
compile-single:
run-single:
Camino: 230142, valor=5

BUILD SUCCESSFUL (total time: 1 second)
```

Figura 6: Compilación de la prueba sencilla de intuir, con ciudad inicial 2