Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI2692 - Laboratorio de Algoritmos y Estructuras II
Trimestre: Enero-Marzo 2024

Un algoritmo Divide-and-Conquer para el problema de la ruta de máximo beneficio

Integrantes:

19-10040 → Revete, Jose
1910109 → De Vincenzo, Alejandro

Diseño de la solución:

La solución para el algoritmo se divide en distintas partes, en primera instancia se define la función ***main*** la cual le dará lugar a la ejecución del resto de funciones. Esta función ***main*** tendrá como primera tarea la lectura y recolección de los datos del archivo recibido en la ejecución del script de bash. Luego se observan todas las primeras y segundas coordenadas para definir el tamaño del mapa sobre el cual se estará trabajando. Por consiguiente se dará lugar a la ejecución de la función ***divideAndConquerPRMB*** la cual tendrá como primera labor el sumar la cantidad de ciudad en las cuales su primera coordenada (eje X) se repite más de 3 veces. Luego ***divideAndConquerPRMB*** verifica los casos:

1. Si la cantidad de ciudades está entre 1 y 3, entonces ejecuta la función ***resolverMiniPRMB***, la cual se encargará de hacer una prueba manual de todas casos presentes (rutas), dependiendo de la cantidad de ciudades ingresadas, ejemplo: si son 2 ciudades, deberá comprobar la recompensa en cada ciudad y la ganancia que trae el ir de una ciudad a otra, esto lo realiza llamando a las funciones ***gananciaRuta*** y está a la función ***costo*** (está calcula el costo que implica ir de una ciudad a otra). Luego de obtener la ruta con mayor recompensa, esta función llamará a la función ***unirRutasCasos***, la cual se encargará de combinar las coordenadas y strings de rutas obtenidas, para así registrar los datos necesarios de las ciudades y llamar a otra función. Esta nueva función se llama ***escogerVecino***, este algoritmo prueba cada elemento de la ruta obtenida al unir la ruta izquierda y derecha, define una coordenada para calcular la distancia entre esta ciudad y las demás para al final escoger la que implica menor costo, esto se realiza mediante las funciones ***costoRuta*** y ***distancias***, luego al finalizar de recorrer los elementos de la ruta se ejecuta la función ***todasVisitadas***, para verificar que todas la ciudades han sido visitadas. Luego como todas las rutas y ganancias se iban comparando mientras se ejecutaba el algoritmo, entonces obtenemos la ruta con mejor ganancia y dicha ganancia.
2. Si la cantidad de ciudades es 0, no se hace nada.
3. Si la cantidad de ciudades es mayor que 3 entonces tenemos dos casos:
    - Si la suma realizada al comienzo de las primeras coordenadas de las ciudades es mayor que 3, entonces dividimos el plano por la mitad en el eje Y y aplicamos recursión para evaluar estos dos planos que hemos conseguido (superior e inferior). Posteriormente, si ahora la cantidad de ciudades es menor o igual que 3, entonces se llama a la función ***resolverMiniPRMB*** y se desarrolla el caso expuesto en 1., pero si la cantidad de ciudades es mayor que 3, entonces se sigue llamando recursivamente a ***divideAndConquerPRMB*** hasta obtener una división del plano tal que hayan 3 o menos ciudades en el.
    - Si la suma realizada al comienzo de las primeras coordenadas de las ciudades es menor o igual que 3, entonces dividimos el plano por la mitad en el eje X y se aplica lo mismo que en el caso anterior.

Procedimiento Divide-and-Conquer(ciudades, coordenadas, maxX, maxY, desdeX, desdeY)

```
1       si ciudades <= 3 entonces
2               devolver adhoc(ciudades,coordenadas);
3
4       si ciudades = 0 entonces // No hay ciudades, no se hace nada
5       en otro caso
```

```
6               suma ← Crear nueva variable;
7               suma ← 0;
8               para i ← 1 a ciudades hacer
9                       para i ←j a ciudades hacer
10                              si coordenadas[i][0] = coordenadas[j][0] && (i != j) entonces
11                                      suma ← suma + 1;
12              divisionEjeY ←Crear nueva variable;
13              si suma > 3 entonces
14                      divisionEjeY ← Verdadero;
15              en otro caso
16                      divisionEjeY ← Falso;
17
18      si divisionEjeY = Verdadero entonces
19              mitadY ← Crear nueva variable;
20              mitadY ← (maxY - desdeY) / 2
21              ciudadesEnMitadYabajo ← Crear nueva variable;
22              ciudadesEnMitadYabajo ← 0;
23              ciudadesEnMitadYarriba ← Crear nueva variable;
24              ciudadesEnMitadYarriba ← 0;
25              para i ←0 a ciudades hacer
26                      si coordenadas[i][1] <= mitadY entonces
27                              ciudadesEnMitadYabajo ←ciudadesEnMitadYabajo+1;
28                      en otro caso
29                              ciudadesEnMitadYarriba ←ciudadesEnMitadYarriba+1;
30              abajoY[0..mitadY] ←Crear un nuevo arreglo;
31              arribaY[mitadY+1..maxY] ←Crear un nuevo arreglo;
32
33              rutaArriba ← Divide-and-Conquer(ciudadesEnMitadYarriba, arribaY, maxX, maxY,
desdeX, (mitadY+1));
34              rutaAbajo ← Divide-and-Conquer(ciudadesEnMitadYabajo, abajoY, maxX, mitadY,
desdeX, desdeY);
35              ruta ← Crear un arreglo nuevo;
36              ruta ← unirRutas(rutaAbajo, rutaArriba); // Algoritmo de unirRutas
37      en otro caso
38              mitadX ← Crear nueva variable;
39              mitadX ← (maxX - desdeX) / 2
40              ciudadesEnMitadXizquierda ← Crear nueva variable;
41              ciudadesEnMitadXizquierda ← 0;
42              ciudadesEnMitadXderecha ← Crear nueva variable;
43              ciudadesEnMitadXderecha ← 0;
44              para i ←0 a ciudades hacer
45                      si coordenadas[i][1] <= mitadX entonces
46                              ciudadesEnMitadXizquierda ←ciudadesEnMitadXizquierda+1;
47                      en otro caso
48                              ciudadesEnMitadXderecha ←ciudadesEnMitadXderecha+1;
49              izquierdaX[0..mitadX] ←Crear un nuevo arreglo;
50              derechaX[mitadX+1..maxX] ←Crear un nuevo arreglo;
```

```
51                  rutaDerecha ← Divide-and-Conquer(ciudadesEnMitadXderecha, derechaX, maxX,
maxY, (mitadX+1), desdeY);
52                  rutaIzquierda ← Divide-and-Conquer(ciudadesEnMitadXizquierda, izquierdaX,
mitadX, maxY, desdeX, desdeY);
53                  ruta ← Crear un arreglo nuevo;
54                  ruta ← unirRutas(rutaIzquierda, rutaDerecha); // Algoritmo de unirRutas
55    devolver ruta;
```

Procedimiento adhoc(ciudades,coordenadas)

```
1         ganancia ← Crear nueva variable;
2         ruta ← Crear un arreglo nuevo;
3         si ciudades = 1 entonces
4               ganancia ← coordenadas[0][2];
5               ruta ← Se guarda la ciudad;
6         si ciudades = 2 entonces
7                   ganancia1 ← Crear nueva variable;
8                   ganancia2 ← Crear nueva variable;
9                   ganancia1 ← coordenadas[0][2];
10                  ganancia2 ← coordenadas[1][2];
11                  ganancia12 ← Crear nueva variable;
12                  ganancia12 ←(ganancia1 + ganancia2) - ((coordenadas[1][0] -
coordenadas[0][0])²+(coordenadas[1][1] - coordenadas[0][1])²)**(½)
13                  si ganancia1 > ganancia2 entonces
14                        si ganancia1 > ganancia12 entonces
15                              ganancia ← ganancia1;
16                              ruta ←Se guarda la ciudad de ganancia1;
17                        en otro caso
18                              ganancia ← ganancia12;
19                              ruta ←Se guarda las ciudades de ganancia12;
20                  en otro caso
21                        si ganancia2 > ganancia12 entonces
22                              ganancia ← ganancia2;
23                              ruta ←Se guarda la ciudad de ganancia2;
24                        en otro caso
25                              ganancia ← ganancia12;
26                              ruta ←Se guarda las ciudades de ganancia12;
27        si ciudades = 3 entonces
28                  ganancia1 ← Crear nueva variable;
29                  ganancia2 ← Crear nueva variable;
30                  ganancia3 ← Crear nueva variable;
31                  ganancia1 ← coordenadas[0][2];
32                  ganancia2 ← coordenadas[1][2];
33                  ganancia3 ← coordenadas[2][2];
34                  ganancia12 ← Crear nueva variable;
35                  ganancia12 ←(ganancia1 + ganancia2) - ((coordenadas[1][0] -
coordenadas[0][0])²+(coordenadas[1][1] - coordenadas[0][1])²)**(½)
```

```
36              ganancia13 ← Crear nueva variable;
37              ganancia13 ←(ganancia1 + ganancia3) - ((coordenadas[2][0] -
coordenadas[0][0])²+(coordenadas[2][1] - coordenadas[0][1])²)**(½)
38              ganancia23 ← Crear nueva variable;
39              ganancia23 ←(ganancia3 + ganancia2) - ((coordenadas[1][0] -
coordenadas[2][0])²+(coordenadas[1][1] - coordenadas[2][1])²)**(½)
40              ganancia123 ← Crear nueva variable;
41              ganancia123 ← ganancia12 + ganancia23 - ganancia2;
42              ganancia132 ← Crear nueva variable;
43              ganancia132 ← ganancia13 + ganancia23 - ganancia3;
44              ganancia213 ← Crear nueva variable;
45              ganancia123 ← ganancia12 + ganancia13 - ganancia1;
46              ganancia ← // Se guardara el mas grande entre ganancia1, ganancia2, ganancia3,
ganancia12, ganancia13, ganancia23, ganancia123, ganancia132 y ganancia213
47              ruta ←// Se guardará la(s) ciudad(es) que correspondan a ganancia
48      devolver ruta;
```

Detalles de la implementación:

La realización del algoritmo fue una prueba y corrección constante. Luego de pensar en varias formas de llevar a cabo el proyecto, la mejor implementación del algoritmo que conseguimos es la que se puede apreciar en el Proyecto. Este funciona bien hasta cierto límite, se probó con 400 ciudades y tarda un aproximado de 4 min en concluir la prueba. En pocas ciudades el algoritmo funciona bien y el tiempo de respuesta es de menos de 1 min.

Lecciones aprendidas:

Durante la realización del proyecto la mayor lección aprendida fue la de descansar y no presionar de más para completar algo, esto en varias ocasiones resultó contraproducente en cuanto a la intención del algoritmo. Además, investigar, intercambiar de ideas y opiniones fue vital, dado que sin el apoyo mutuo, muy probablemente no pudiéramos haber concluido este proyecto. Por último, no tener miedo a probar mil metodos distintos nos ayudó a avanzar o nos enseñó errores que luego no llegamos a cometer. Otro aprendizaje es que ambos preferiamos usar arreglos que listas (simplemente porque ya sabíamos manejarlas), pero la utilidades de estas son gigantes, creemos que esto nos ayudó mucho en el resultado final.