

RESEÑA DEL PROBLEMA

Problema	:	Cartero
Complejidad	:	Media
Tópico	:	Grafos
Conocimientos	:	Algoritmos de camino mínimo en grafos
Tamaño del Código	:	Corto
Codificación	:	Fácil
Estructuras de datos	:	Básicas
Propuesto por	:	Raúl Duque, Colombia

ANTECEDENTES

El problema de *cartero* es simplemente una variante del conocido problema de *camino mínimo en grafos* y trata evaluar su identificación, adaptación e implementación.

APROXIMACIONES

A grandes rasgos el programa podría analizar los caminos posibles entre un vértice de Inicio (C_i) y un vértice de destino (C_f), que para este problema en particular son únicos, con la característica de que al menos uno de sus vértices intermedios sea un vértice donde es posible almorzar. Luego, de todos estos caminos posibles se elige el de costo mínimo, donde el costo lo interpretamos como el número de “encuentros caninos” sufridos durante el trayecto del camino. Es posible implementar un sencillo algoritmo de *búsqueda exhaustiva* que cumpla con las restricciones antes mencionadas pero su desempeño (tiempo de ejecución) será pobre cuando se tenga un grafo densamente poblado. Este tipo de solución resolverá a lo sumo 3 de los 5 casos de prueba utilizados en la calificación.

SOLUCION

Identificada la necesidad de un algoritmo eficiente y su relación con *camino mínimo en grafos* es importante notar la necesidad de adaptar este algoritmo, independiente de la implementación que se realice (DIJKSTRA, FLOYD, BELLMAN, JOHNSON). Analicemos dos adaptaciones incorrectas:

- Si simplemente consideramos el camino mínimo entre los vértices C_i y C_f , puede ocurrir que este camino no contenga ningún vértice donde se pueda almorzar por lo cual no será una solución válida, pero se obtendrán soluciones correctas cuando coincida que este camino contiene vértices donde se pueda almorzar.
- Se toma como punto para almorzar A_p , el cual ofrece el camino mínimo entre C_i y cada uno de los A_j (vértices donde se puede almorzar) con $1 \leq j \leq m$. Después de elegir este vértice A_p se halla el camino mínimo de A_p a C_f . Aunque este algoritmo puede funcionar para algunos casos es incorrecto. Suponga el camino entre C_i y A_p tiene un tamaño p_1 , y que, el camino de A_p a C_f tiene tamaño p_2 . Suponga ahora que se tiene un camino que pasa por un vértice A_q que a su vez es un vértice donde se puede almorzar. De acuerdo con el algoritmo tenemos la certeza de que $p_1 \leq \text{costo}\{C_i..A_q\}$ para cualquier A_q , pero no tenemos ninguna certeza acerca de que $p_2 \leq \text{costo}\{A_q..C_f\}$, con lo cual puede ocurrir que $p_1 + p_2 > \text{costo}\{C_i..A_q\} + \text{costo}\{A_q..C_f\}$, con lo cual demostramos la invalidez de la solución $C_i - A_p - C_f$.

Analizado lo anterior concluimos que se deben considerar todos los posibles caminos $C_i - A_j - C_f$, donde A_j corresponde a vértices donde es posible almorzar y con $1 \leq j \leq m$, y asumiendo que los caminos entre $C_i - A_j$ y $A_j - C_f$ son mínimos (minimizar la expresión $\text{costo}\{C_i..A_j\} + \text{costo}\{A_j..C_f\}$).

Supongamos que tenemos como solución el camino $C_i-A_p-C_f$ y que existe otro camino con costo menor que también tiene como vértice para almorzar a A_p , esto contradice el hecho de que C_i-A_p y A_p-C_f sean caminos mínimos (si fuese así tendríamos un C_i-A_p y/o un A_p-C_f de menor costo entonces C_i-A_p y/o A_p-C_f no hubiesen sido considerados como mínimos). De otro lado, no es posible que con otro vértice A_q se logre un costo menor pues de hecho se han analizado todos los A_j (que incluye a A_q) y fue descartado como solución.

CODIGO

La solución expuesta esta compuesta de cuatro funciones, que en su orden son:

leer()

Esta función primero inicializa la matriz de adyacencia *mapa* con un valor *especial* para indicar la NO existencia de un arco entre *mapa[i][j]*. Luego lee del archivo de entrada los arcos y número de perros para cada arco (calle) y los actualiza en *mapa[i][j]* y en *mapa[j][i]* ya que el grafo es no dirigido. Por último lee la información correspondiente a los vértices donde es posible almorzar junto con la información adicional restante.

floyd()

Esta función simplemente aplica el algoritmo de FLOYD al grafo almacenado en *mapa* y almacena en *mapa[i][j].quien* el vértice que fue usado como intermediario para que sea más fácil la reconstrucción del camino.

minimo()

Esta función resuelve el problema a partir de la matriz de caminos mínimos calculada en el punto anterior. Esto lo logra realizando un ciclo que examina para cada A_j perteneciente al conjunto de vértices donde se puede almorzar, la existencia de un camino de menor costo entre C_i-A_j y A_j-C_f respecto al mínimo actual. Al terminar la función tenemos el costo del camino mínimo junto con el vértice que fue usado como lugar para almorzar (*pclave*).

camino()

Esta función reconstruye el camino solución a partir de la información almacenada en *mapa[i][j].quien*. Ya que hemos usado como punto intermedio el vértice *pclave*, el proceso se realiza en dos partes: primero se invoca la función con el punto inicial C_f y punto final *pclave*, y después con punto inicial *pclave* y punto final C_i , con esto hemos reconstruido totalmente el camino solución.

MEJORAS AL ALGORITMO

Debido a que la solución del problema se basa en un algoritmo FLOYD su complejidad es de orden $O(n^3)$, ya que se calcula el camino mínimo entre todos los pares de vértices. Es posible mejorar la velocidad de ejecución del programa sustituyendo el algoritmo de FLOYD por 2 llamadas al algoritmo de DIJKSTRA (que calcula el vector de costos mínimos entre un vértice y todos los demás vértices) una desde la oficina inicial a todas las esquinas y otra entre la oficina final a todas las demás esquinas, aunque su codificación se hace un poco más complicada. Este algoritmo tendrá una complejidad de $O(2*n^2)$, que claramente mejora la implementación por FLOYD a medida que n sea más grande. Para el tiempo asignado a este problema es suficiente la implementación por FLOYD.

BIBLIOGRAFIA

- CORMEN Thomas, Leiserson Charles and Rivest Ronald. *Introduction to Algorithms*. The MIT Press, 1990.
- PARBERRY Ian. *Problems on Algorithms*. Prentice Hall, 1995.

Raúl Andrés Duque M.
rduque@venus.uanarino.edu.co