Nombre: Cristhian Daniel Parra Trepowski
Nro. De Cédula : 2.045.856
Fecha : 11 de junio de 2005

TRABAJO PRÁCTICO 2

ÍNDICE

1.1. Introducción

Este trabajo pretende proporcionar información adecuada y propuestas de solución sobre los dos problemas planteados en el mismo. Uno de ellos es un problema clásico y el otro no lo es tanto. De hecho, sobre este segundo no existen casi referencias en libros de algoritmia ni en papers, mientras que sobre el primero hay referencias en por lo menos (es una estimación) el 80% de los Libros de Algoritmia y de Matemática Discretas. Ambos problemas son problemas complejos y no existen soluciones eficientes para los mismos. Cuando me refiero a eficientes quiero decir soluciones que sean lineales o subcuadráticas.

1.2. Problema 1: *Identificar (imprimir) un ciclo simple con origen en w que incluya a todos los vértices de G y cuya suma (del ciclo) sea mínima.*

Este es un problema clásico de la programación. Se trata del famoso problema del viejo y conocido "Viajante de Comercio".

Viejo, porque fue planteado, en su versión inicial, ya a mediados del siglo XIX [2] (antes de las computadoras) por Sir William Rowan Hamilton (1805 – 1865), un eminente matemático inglés de la época.

Lo que hizo Hamilton fue proponer un juego con la forma de un dodecaedro. Cada esquina de dicho polígono tenía el nombre de un ciudad famosa de Europa y se pedía que, empezando por una determinada ciudad, se recorran todos los poblados, visitándolos solo una vez, retornando finalmente al punto de partida. Más adelante, en honor a hamilton, una generalización de este problema fue bautizado con su apellido. Hablo del problema de hallar un ciclo hamiltoniano en un grafo.

Se puede notar una leve relación con el problema del ciclo de Euler, sin embargo, a diferencia de los ciclos eulerianos, no existe ningún teorema o mecanismo que sirva para demostrar si un Grafo es o no Hamiltoniano. He ahí la esencia de su dificultad.

Pero nuestro problema en realidad, es una extensión de este propuesto por Hamilton, pues le agregamos la restricción de que el ciclo hallado, tenga el mínimo costo posible. Y aunque este agregado nos complique la vida, este problema es un problema fundamental de todo lo que tenga que ver con optimización y se pueden encontrar aplicaciones en muchísimos campos, tanto de la ingeniería como de otras ramas de la ciencia.

Es conocido, porque aparece en todos los libros y documentos sobre Algoritmos y Estructuras de Datos que encontré, además de aparecer también en libros de Matemática Discreta.

1.2.1.Soluciones. Algoritmos y Estructuras de Datos utilizados.

Es un poco complicado clasificar al Problema del Viajante entre las categorías de complejidad que tenemos. Si bien, Brassard [1] lo define como un problema *"NP",* yo creo que puede ser considerado como un *"NP-Completo",* puesto que una vez hayada una solución (que no puede ser hallada en tiempo polinómico), la única verificación posible en un tiempo polinómico es la de si es o no un ciclo hamiltoniano. Pero si quisiéramos verificar si es "mínimo", esto no se puede hacer tiempo polinómico. De esta manera, si lo que tenemos es una solución aproximada, no hay forma de saber si es la solución óptima, más que realizando el algoritmo de fuerza bruta. Ahora, si lo que tenemos es la solución óptima, obtenida con un algoritmo de fuerza bruta, esta si se puede verificar en tiempo polinómico, puesto que se supone que elegimos entre todas las posibilidades la que tiene menor costo. En síntesis, clasificar este problema me generó ciertas dudas.

Ahora que ya sabemos que tipo de problema es, hablemos de las soluciones.

1.2.1.1.ESTRUCTURAS DE DATOS UTILIZADOS. FUENTES.

Quiero describir ya este asunto para que en adelante me pueda referir a ellos sin problema al hablar de las soluciones.

Para resolver el Problema, utilizamos, como era restricción del trabajo, un grafo no dirigido y ponderado, usando una representación de listas de adyacencias. Con esta representación del grafo, creo que se pierde eficiencia y sobre todo en estos algoritmos que acceden constantemente a las aristas del Grafo. Con una matriz de adyacencias, estos accesos tendrían un costo en O(1), mientras que con las listas de adyacencias dichos accesos están en O(n).

La clase "Hamilton.java", contiene todos los métodos que resuelven el problema. "TestHamilton.java", es tan solo un programa de prueba que lee un grafo usando la "GrafoIO.java", como lo hacíamos en clase. La clase se llama "Hamilton.java" en honor al primer hombre que hizo referencia a este problema, lo cual ya fue mencionado.

"GrafoNoDirigido.java", es la implementación del de grafo no dirigido sobre el cual trabajan mis algoritmos.

De ahora en más, siempre que nos refiramos a un método en esta sección, será algún método de "Hamilton.java".

Hago notar describiré con mayor profundidad dos alternativas de solución, y ambas están implementadas. El motivo por el cual elegí estas alternativas es que son las más generales y sencillas de entender. Al final también presento una breve explicación de otra alternativa menos general.

1.2.1.2FUERZA BRUTA.

Ya mencioné más arriba el algoritmo de ***fuerza bruta.*** Sucede que para este problema, este es el único algoritmo que obtiene una solución óptima. Pero por supuesto, su eficiencia es terrible. Una buena implementación la provee [6] en su capítulo 6.

El algoritmo consiste básicamente en generar todas las permutaciones posibles de las (n-1) ciudades que quedan excluyendo el inicio, y verificar para cada una si es un ciclo hamiltoniano. Al final, se elige de entre todas estas permutaciones, la que tiene menor costo.

Como generamos (n-1) permutaciones, podemos decir que en General esta solución está en O((n-1)!).

El método ***"HamiltonFuerzaBruta"***, recibe el número del vértice de inicio y calcula usando Vuelta Atrás, el ciclo simple de costo mínimo. Este algoritmo es una adaptación del algoritmo dado por [6] en su página 238 del capítulo 6.

Analizar asintóticamente esta implementación, es harto complicada. Tanto, que todavía no estoy seguro de como hacer. Básicamente, siendo "n" el número de vértices, tomamos un vector de n + 1 componentes y en la primera y en la última posición ponemos el inicio. Luego, para cada Posición, empezando por la segunda, investigamos todos posibles ciclos que se podrían dar permutando los vértices en esa posición. Cuando no hay

más posibilidades, volvemos atrás a la posición anterior. Puesto que para cada posición hay n posibilidades, y el algoritmo se efectúa sobre (n-1) posiciones. El costo es efectivamente O((n-1)!).

1.2.1.3. VECINO MÁS PRÓXIMO.

Por la dificultad de este problema, muchas veces se opta por una solución aproximada (heurística). El Algoritmo del Vecino más próximo es el más general y extremadamente sencillo que existe para este acometido. Es un algoritmo voraz de optimización local.

El algoritmo consiste básicamente en realizar los siguientes pasos.

1. Tomar los Vértices del Grafo y crear un lista C cuyo primer elemento es el inicio del ciclo.
2. Mientras hayan vértices sin visitar:
   2.1. Tomar el último vértice añadido a C y buscar en V el vértice más cercano a él que no haya sido visitado.
   2.2. Insertar el vértice encontrado y marcarlo como visitado.
3. Fin.

Como podemos ver, básicamente se realizan n-1 iteraciones en total, y para cada iteración "i", hay "n-i" posibles candidatos. Así tenemos que el costo está en O(n*(n-1)/2) puesto que los términos n-i forman un regresión aritmética. De esta manera concluimos que el algoritmo general está en $O(n^2)$.

Este algoritmo solo produce un aproximado, sin embargo [3] ofrece un teorema interesante en su página 160, con su respectiva demostración que no voy a tratar. El teorema dice que para un grafo de "n" vértices, siendo "d" la distancia de un ciclo hamiltoniano obtenida utilizando este método y "d_0" el costo del ciclo hamiltoniano mínimo, se da la siguiente relación:

$$(d / d_0) = (½) * [\lg n] + (½)$$

Obs: [ ] representa al entero más pequeño mayor o igual de lg n, o sea, es redondeo hacia arriba. lg n representa $\lg_2 n$.

Si este teorema es cierto, entonces este algoritmo obtiene soluciones muy aproximadas que permiten decidirse por él cuando la exactitud no es tan crítica.

Los método ***"VecinoMasProx"*** y ***"VecinoMasProxVAtras",*** implementan esta alternativa de solución. El primero es exactamente el mismo algoritmo descripto arriba. En realidad es una adaptación de lo que propone [6] en su capítulo 4, página 161. El costo está efectivamente O (n^2) para el primero. El problema de este algoritmo, es que si el grafo no es completo, puede que no encuentre solución aún habiendo por lo menos una. Esto es porque se sigue un solo camino de solución y si sucede que justo por este camino no existe un ciclo hamiltoniano, tan pronto como se encuentre si más aristas que tomar, el algoritmo termina.

Es por ello que traté de adaptar el método para que encuentre al menos la primera solución cuando el grafo no es completo. De eso se trata la segunda opción. Utilizamos un esquema de vuelta atrás para que investigue las posibilidades que quedaron más atrás si se llegó a un punto sin más aristas posibles que seguir. En este caso, lo que hacemos es algo parecido al algoritmo de fuerza bruta solo que tomando como inicio el vértice más cercano al inicial. Entonces tenemos aparentemente un algoritmo en O((n-2)!), pero la verdad es que por cada vuelta atrás que hacemos, es porque tenemos una arista menos que en el grafo completo, por lo tanto, a medida que se vuelve atrás, las posibilidades también se reducen y el costo sería mucho menor.

1.2.1.4. OTRA SOLUCIÓN.

Si el grafo es completo, se puede usar el algoritmo heurístico que consiste en generar un árbol de recubrimiento de costo mínimo del grafo, recorrerlo por niveles hasta el último nivel, y luego subir por la rama directa hasta la raíz. Así se genera una sucesión de los vértices del grafo. De esta sucesión, si eliminamos los vértices repetidos, tendríamos un ciclo hamiltoniano de costo mínimo.

Esta solución en particular no la quise implementar porque no hay forma de que funcione en Grafos no completos.

Lo interesante de esta solución es el resultado que arroja, y que menciona Brassard en su libro, que dice que la longitud del ciclo hamiltoniano mínimo, debe ser mayor o igual al costo del Árbol de recubrimiento de costo mínimo del grafo en cuestión.

1.2.2. Comentarios bibliográficos para este punto.

Las fichas de los libros que referencio aquí se encuentran en el apartado de referencias bibliográficas.

[1] ***G. Brassard:*** Este libro presenta el problema del viajante en su simétrica y proporciona las explicaciones del algoritmo que utiliza el árbol de recubrimiento de costo mínimo para resolver el problema. Además explica bastante bien los fundamentos teóricos del problema. Todo esto en la sección 13.2, página 529.

[2] ***Richard Johnsonbaugh:*** Aquí encontré la historia del Problema, quien lo formuló y varios ejemplos y ejercicios que me sirvieron que entender mejor el problema. Ver el Capítulo 4 en su sección 4.1.

[3] ***C. L. Liu:*** De este libro es el teorema que relaciona los resultados aproximados con el óptimo, así como su demostración. Toda la sección 5.7 está dedicada a los ciclos hamiltonianos, y la sección 5.8 al problema del viajante.

[6] Del capítulo 6, página 238 saqué la implementación en la cual me basé para el algoritmo de fuerza bruta. Del capítulo 4, página 161, la implementación para el algoritmo del vecino más próximo.

1.3. Problema 2: *Imprimir todos los árboles de Cobertura Mínima de un Grafo no Dirigido.*

Al contrario que el primer problema, este no es un problema conocido. O mejor dicho, no es un problema ***popular.*** No encontré un solo libro ni página de internet que lo mencione en su forma original. Solo encontré dos papers ([8] y [9]) que hablan de este problema pero en grafos bipartidos. Esto me obligó a pensar en una solución por cuenta propia. Al comienzo parecía un problema muy complicado, pero al final terminó siendo bastante trivial, si es que lo que creo es correcto.

1.3.1. Soluciones.

Lo primero que pensé fue en realiza Prim para cada vértice, pero luego un compañero me dio algunas explicaciones que me marearon y que hicieron que desconfiara de esta alternativa. Entonces me enfoqué analizar una solución basada en Kruskal y usando vuelta para atrás. El algoritmo sería de fuerza bruta era básicamente el siguiente:

1. Ordenar las aristas del grafo de menor a mayor.
2. Generar una las permutación que no forme ciclos para cada grupo de aristas que tienen el mismo peso siguiendo el orden establecido.
3. Agregar cada permutación generada a un bosque de MST's siempre que su costo sea el mínimo y no se encuentre ya en el bosque.
4. Repetir 2 y 3 para todas las permutaciones posibles.

Cuando me puse a pensar como implementar, la verdad que me desesperé puesto que tenía que tener cuenta muchos detalles e implementar estructuras de datos complejas que soporten los que estaba planteando. Por ejemplo, tenía que tener si o si alguna especie de conjunto para poder implementar el bosque de MST's.

Todo este asunto me hacía pensar que no iba a poder resolver este problema puesto que disponía de muy poco tiempo. Gracias a Dios, fue usted profesor el que me devolvió las esperanza cuando me dijo que tenía que analizar mejor la alternativa de Prim para cada vértice. Eso fue exactamente lo que hice y llegué a la conclusión que utilizando esta alternativa, se pueden generar todos los árboles de recubrimiento mínimo.

Finalmente me decidí por esta última alternativa. Por su simplicidad y porque asintóticamente su costo es mucho mayor. La primera alternativa tendría un costo factorial por su naturaleza combinatoria, sin embargo este último solo es polinomial. Esto me lleva a la conclusión de que es un problema de la clase P.

Si bien no pude llegar a hacer una demostración sistemática y absoluta de la correctitud de este algoritmo, puedo decir intuitivamente y empíricamente que es una solución correcta. Digo empíricamente porque para el conjunto de grafos que probé, me funcionó bastante bien. Solo un Grafo me generó problemas. El mismo se encuentra junto con los fuentes en el archivo G1AllMST.dat.

Para este grafo en particular, no fue la solución de este problema lo que no funcionó, sino mi implentación del algoritmo de Prim.

Resulta ser que para este grafo en particular, prima me generó varios árboles de recubrimiento mínimo con costos diferentes. Mucho analicé mi implementación de Prim, pero no pude llegar a encontrar el error. Me gustaría saber luego tu opinión profesor puesto que la verdad es que no sé que está mal en él.

El algoritmo finalmente elegido es el siguiente:

1. Desde el primer vértice del grafo hasta el último hacer:
    a. Instanciar un nuevo MST.
    b. Generar en el mismo el MST resultante de ejecutar Prim sobre el vértice actual.
    c. Si dicho MST no fue generado aún, agregar al bosque de MST's e imprimirlo.
    d. Si ya fue generado, continuar con el siguiente vértice.

1.3.1.1.ESTRUCTURAS DE DATOS UTILIZADOS. FUENTES.

De nuevo, el grafo es no dirigido e implementado mediante lista de adyacencias.

La clase “MST.java”, contiene el método para generar un árbol de recubrimiento de costo mínimo mediante el algoritmo de Prim. El MST se guarda en un vector que representa al árbol guardando quien es el nodo padre de cada vértice del grafo.

“AllMst.java” contiene el método que ejecuta Prim sobre cada vértice produciendo el bosque de MST's. El bosque de MST's es sencillamente una lista enlazada que contiene a todos los MST's generados.

1.3.2.Costo asintótico.

El método que resuelve el problema se encuentra en AllMst.java y se llama ***“AllMSTPrim”.***

El análisis asintótico en este caso es muy sencillo. Básicamente, si tenemos “n” vértices. Nuestro costo es n * (Costo de Prim). La implementación de Prim que utilizo usa una cola de prioridad para las aristas. El costo de Prim podemos distribuirlo así. Consideremos |E| la cantidad de aristas y |V| la cantidad d vértices. Además asumimos que el grafo es completo para tener una cota superior, así para cada vértice, hay |V -1| aristas.

– Inicializar el vector distancias	|V|
– Construir el Heap de Aristas	|V| * log |V-1]
– Construir el árbol.	
extraer el mínimo el Heap	log |E|.
ciclo interno	|V-1| log |E|.

Así podemo decir que Prim está en O(|V|+|V| * log |V-1]+|V|*(log |E| + |V-1| log |E|)), que luego de simplificaciones tendríamos algo así:

$$O((|V| * \log |V-1]+|V|*|V| \log |E|)) = O((|V|*|V| \log |E|))$$

Considerando el caso peor del Grafo completo esto sería:

$$O(|E| \log |E|).$$

Y como $|E| = |V|*|V| \rightarrow O(|E| \log |V|^2) \rightarrow O(2 * |E| \log |V|) \rightarrow O(|E| \log |V|)$

Luego, recordando que |V| es "n" y que efectuamos n veces Prim, ***"AllMSTPrim"*** es un algoritmo casi cúbico que está en:

$$O(n * n * n * \log n) \rightarrow \underline{\mathbf{O(n^3 * \log n)}}$$

Observación: Interiormente, la verificación de si el árbol está o no es en el bosque está básicamente en $O(n^2)$, pero como n^3 es un término más significativo, no lo contamos en el costos total.

1.3.3. Comentarios bibliográficos para este punto.

[5] ***Aho:*** Este me parece que es el que mejor explica como funcionan los algoritmos de Prim, Kruskal y Dijkstra inclusive. El capítulo 7 de este libro me dio el soporte teórico para realizar este ejercicio.

[8] y [9] son papers que encontré en internet (sé que no son válidos como bibliografía) y que me parecieron muy interesantes aunque no pude ahondar en ellos. Los incluyo aquí solo como propuesta de lecturas adicionales.

REFERENCIAS BIBLIOGRÁFICAS:

Libros Consultados:

[1] ***Fundamentos de Algoritmia.*** *G. Brassard, P. Bradley. Edición en Español. Editorial Prentice Hall, Madrid 1997*

[2] ***Matemáticas Discretas.*** *Richard Johnsonbaug. Grupo Editorial Iberoamericana, México D.F., 1988.*

[3] ***Elementos de Matemáticas Discretas.*** *C. L. Liu. Segunda Edición. McGraw Hill. 1995*

[4] ***Teoría y Problemas de Matemática Discreta.*** *Seymour Lipschutz. McGraw Hill. 1976.*
** Solo me sirvió de soporte Teórico. Me ayudó a recordar algunos asuntos.*

[5] ***Data Structures and Algorithms.*** *Alfred Aho. Addisson Wesley.*
Prim, Kruskal, Dikjstra. Algoritmos en General.

[6] ***Técnicas d e Diseño de Algorimos.*** *1997. (No Conozco el Autor).*
Implementaciones de "VecinoMasProximo" y "Fuerza Bruta" del Viajante.

Lecturas Adicionales Recomendadas:

[7] ***The Traveling Salesman Problem: A Case Study in Local Optimization.*** *David S. Johnson1 Lyle A. McGeoch.* Es un versión preliminardel capítulo correspondiente que aparece en el Libro "Búsqueda Local en Optimización Combinatoria" de E. H. L Aarts y J. K. Lenstra.

[8] ***Representing all Minimum Spanning Trees with Applications to Counting and Generation.*** *David Eppstein Department of Information and Computer Science University of California, Irvine. December 15, 1995*

[9] ***Finding All Minimum Cost Perfect Matchings in Bipartite Graph.*** *Komei FUKUDA, Tomomi MATSUI. Diciembre de 1988. Revisado en Diciembre de 1991.*