INGENIERIA INFORMATICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

Algoritmia

Práctica 2

David Teófilo Garitagoitia Romero
Daniel Cerrato Sánchez

Pareja 08 Grupo 1261
10/17/2022

Índice de Contenidos

1. Introducción

En esta práctica se asentarán y practicarán conceptos relacionados con las estructuras y algoritmos ya mostrados en clase, en concreto, se indagará en los conjuntos disjuntos y el problema del viajero.

2. TAD Conjunto Disjunto y Componentes Conexas

2.1. Cuestiones sobre CDs y CCs

Sin darnos cuenta, en nuestro algoritmo de encontrar CCs podemos pasar listas con ramas repetidas o donde los vértices coinciden con los de una que ya está, aunque en orden inverso. ¿Afectará esto al resultado del algoritmo? ¿Por qué?

No, no afectará al resultado porque los conjuntos disjuntos que se forman no guardan la dirección de las ramas, solamente si dos vértices están unidos o no. Si el algoritmo encuentra una rama que une dos vértices que ya sabe que están unidos, no hace nada, pasa a la siguiente rama.

Argumentar que nuestro algoritmo de encontrar componentes conexas es correcto, esto es, que a su final en los distintos subconjuntos disjuntos se encuentran los vértices de las distintas componentes del grafo dado.

Es correcto porque construye las componentes conexas desde cero; es decir, parte de la base de que todos los vértices están separados y va uniendo vértices según se indica en la lista de ramas. Estas uniones se hacen sobre conjuntos disjuntos, lo que determina qué vértices acaban juntos, aunque no asegura que los conjuntos guarden la topología exacta del grafo, ya que se realiza compresión de caminos.

El tamaño de un grafo no dirigido viene determinado por el número n de nodos y la longitud de la lista l de ramas. Estimar razonadamente en función de ambos el coste del algoritmo de encontrar las componentes conexas mediante conjuntos disjuntos.

El tamaño de un grafo no dirigido viene determinado por el numero 'n' de nodos y la longitud de la lista 'l' de ramas. Estimar razonadamente en función de ambos el coste del algoritmo de encontrar las componentes conexas mediante conjuntos disjuntos.

1. init_cd(n) -> O('n')

2. for branch in l -> O('l') en peor caso

    2.1. find1 -> O(lg('n'/2)) en peor caso

2.2. find2 -> O(lg('n'/2)) en peor caso

2.3. unión -> Cada vez más posibilidades de tardar más. O(4) en peor caso

3. cd_2_dict -> O(3'n') en caso peor

Si juntamos todo quedaría como O('n') + O('l' * [2 * lg('n'/2) + 4]) + O(3'n').

Si recalculamos un poco queda O('n') + O('l'2lg('n'/2) + 4'l') + O(3'n')

Si nos ceñimos al cálculo básico del algoritmo, hemos de quitar la inicialización del array de conjuntos disjuntos y la creación del diccionario final. Por lo que realmente tiene un coste de

O('l'2lg('n'/2) + 4'l').

Explicación de los factores de coste:

'l' viene dado por el bucle que recorre la lista de ramas

2*lg('n'/2) viene dado por el peor caso de los dos find que se realizan para encontrar los representantes de ambos vértices, donde solo hay dos árboles binarios donde buscar

y cada uno contiene la mitad de la lista sin haber hecho compresión de caminos. 4 es el número de operaciones que se realizan en la función unión en el peor caso.

Extendido es O('l'2[lg('n') - 1] + 4'l') = O('l'2lg('n') - 2'l' + 4'l') = O('l'2lg('n') + 2'l') = O('l'2lg(2'n'))

3. El problema del viajante de comercio

3.1.Cuestiones sobre la solución greedy de TSP

Estimar razonadamente en función del número de nodos del grafo el coste codicioso de resolver el TSP. ¿Cuál sería el coste de aplicar la función exhaustive_tsp? ¿Y el de aplicar la función repeated_greedy_tsp?

Supongamos N número de nodos.

greedy_tsp:

1. Bucle con N-1 iteraciones para visitar todas las ciudades

1.1. Ordenar los N costes desde la última ciudad visitada. Usa quickSort por defecto.

En el caso peor es N^2

1.2. Recorrer los N costes para escoger el menor hacia una ciudad no visitada

En el peor caso, este coste va aumentando de 1 a N, puesto que cada vez hay mas ciudades visitadas.

Agrupando queda como $O(N-1 * [N^2 + N(N+1)/2]) = O(N^3 - N^2 + (N^3 - N)/2) = O(3N^3/2 - N^2 - N/2)$

Así que el coste en el peor caso es $O(3N^3/2) = O(N^3)$ repeated_greedy_tsp:

1. Bucle con N iteraciones para iniciar los circuitos desde cada nodo

    1.1. Llamada a greedy_tsp. Peor coste $O(N^3)$

    1.2. Llamada a len_circuit. Coste O(N)

Agrupando queda como $O(N * [N^3 + N]) = O(N^4 + N^2)$.

Asi que el coste en el peor caso es $O(N^4)$

exhaustive_greedy_tsp:

1. Bucle que recorre las N! permutaciones

    1.1. Bucle que recorre los N caminos entre ciudades

El coste de la función es O(N * N!)

A partir del código desarrollado en la práctica, encontrar algún ejemplo de grafo para el que la solución greedy del problema TSP no sea óptima

```
   A   B   C   D
A [ 0  624 506 357]
B [624  0  995 350]
C [506 995  0  653]
D [357 350 653  0 ]
```

En este grafo, empezando en el nodo A por ejemplo, no se encuentra el circuito óptimo.

[FINAL DE DOCUMENTO]