Laboratorio 8

Juan Diego Sique Martínez

Octubre, 2018

1. Primera parte

1.1. Teniendo una lista de adyacencia representando un grafo dirigido, ¿cuánto me toma computar las «out-degrees» y las «in-degrees» de cada vértice?

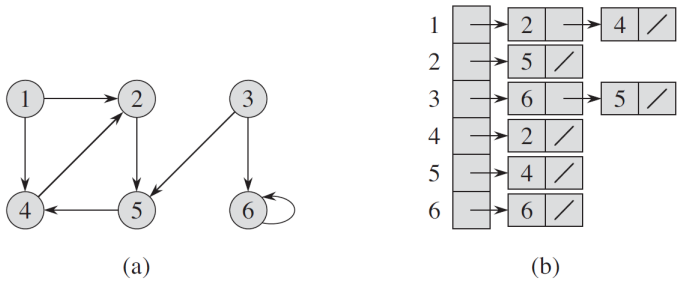

Figura 1: En el apartado (a) se muestra el grafo dirigido. En (b) se muestra su respectiva lista de adyacencia.

1.1.1. «Out-degreees»

En una lista de adyacencia obtener los nodos con característica «out-degree» o «saliente» es de $O(n)$. La razón es que la lista de adyacencia posee por cada nodo una enumeración encadenada de los nodos condectados por una arista saliente del vértice. Basta con recorrerla para computar los «out-degreees».

Otra forma de análisis, es usando también los vértices al momento de plantear su crecimento asintótico. Para encontrar los salientes basta con recorrer cada vértice de la lista y luego ver qué porción de vértices le toca (aristas) . Al final de recorrer la lista de adyacencia se habrán recorrido todos los vértices y las aristas por lo que su orden es $O(V + A)$

1.1.2. «In-degrees»

Para computar los «in-degrees» se debe de revisar cada uno de los elementos en la lista de adyacencia y verificar que exista el elemento en su enumeración encadenada. El enunciado anterior nos dice que esta búsqueda respeta un orden $O(n^2)$.

Al igual que los «out-degree», puede aplicársele un análisis aplicando los vértices y aristas. Para encontrar qué elementos son «in-degree» debe de recorrerse toda la estructura de datos que toma $O(V + A)$, por lo que se concluye que orden es idéntico al «out-degree».

1.2. Problemas para aplicar BFS o DFS

1.2.1. Encontrar más nodos en una red de «Blockchain»

Desconozco a profundidad cómo funcionan las redes de Blockchain, pero suponiendo que mantienen el principio de una red de telecomunicaciones podemos decir que hay dos acepciones a encontrar un nodo. Si es por medio de «broadcast» se usa un método BFS si lo que deseo es descubrir un nodo en específico para hacerle alguna solicitud usaría DFS.

1.2.2. Desarrollar un «crawler» para un motor de búsqueda

Usaría BFS, ya que van a una página y escanean su contenido para obtener enlaces a evaluar.

1.2.3. Encontrar la salida a un laberinto

Usaría DFS. Ya que lo que estoy buscando es un nodo específico, necesito evaluar la primera solución.

1.2.4. Sistema de GPS para encontrar caminos de A a B

Hay dos acepciones. Usaría DFS si lo que necesito es en contrar un camino de A hacia B con prontitud. Usaría BFS si lo que busco es una solución eficiente o que cumpla con algún requerimiento, o si probablmentne necesito varias soluciones para su comparación.

1.2.5. Detectar un ciclo dentro de un grafo

Usaría DFS, si hay un ciclo, tal y como establece el libro por medio de colores, o debido a que exmina rama por rama a lo hondo los identificará con mayor prontitud.

2. Segunda parte

2.1. ¿Cuál es el «running time» de BFS se representamos el grafo como una matriz? ¿Cuál es el «running time» si la representación es una lista?

2.1.1. Lista de adyacencia

Esisten dos maneras de hacer este análisis.

La primera es haciéndola con respecto a n. La primera parte(declaratoria) del algoritmo toma al menos n, ya que va ítem por ítem para poner sus propiedades por defecto. La segunda parte consite en analizar las conexiones o aristas relacionadas a cada elemento e introducirlos a una cola. Luego de ésto se repite el proceso dejando un crecimiento de $O(n^2)$.

La segunda es preferida por ser más precisa. Usaremos como variables generales V el número de vértices en el grafo y A el número de aristas. La parte primera o declaratoria toma al menos $O(V)$. La segunda parte (operativa) tiene dos ciclos a analizar. El primer ciclo («while») tomará al menos V ejecuciones, ya que no se introduce a nuestra cola dos veces el mismo elemento. El segundo ciclo («for») toma en sumatoria A ejecuciones. En conclusión la parte operativa al ser dominante le dará una complejidad de $O(V + A)$ a nuestro algoritmo.

2.1.2. Matriz

Al igual que en la lista de adyacencia existen dos maneras de aplciar el análisis.

La primera es hacerla con respecto a n. La primera parte toma n ejecuciones. La segunda parte posee dos facultades. La primera facultad es la cola, la que lleva $O(n)$ operaciones. La segunda parte es la inspección de vértices «out-degree». Ésta parte en una matriz lleva al menos n operaciones implicadas. Pero al encontrarse dentro de otro ciclo que respeta a n nos deja una complejidad del orden $O(n^2)$.

El segundo análisis usa como constantes de trabajo los vértices **V** y las aristas o conexiones **A**. La parte declaratoria dle algoritmo toma $O(V)$. La parte operativa posee como base una estructura de datos (cola) que utiliza $O(V)$ operaciones y $O(V)$ elementos. Dentro de un ciclo que evalúa la longitud de nuestra estructura de datos (cola), existe otro que verifica las conexiones de el vértice en cuestión que le tomará *operaciones*. Por lo tanto la parte operativa tiene una complejidad de $O(V^2)$. En conclusión la complejidad resultante es de $O(V^2)$.

3. Tercera parte

3.1. Agoritmo no recursivo DFS

```
Data: Un grafo G
Result: Recorrer un grafo utilizando DFS
for v in G.Vértices do
    v.color = WHITE;
    v.π = NULL;
end
time = 0;
for v in G.Vértices do
    if v.color = WHITE then
        DFS-VISIT(G, v)
    end
end
```

Algorithm 1: DFS declaratorio e introductorio

Ahora la modifcación del algoritmo a una forma iterativa-condicional.

```
Data: Un grafo G y un vértice v
Result: Recorrer caminos generados a partir del vértice v
S = new STACK;
PUSH(S, v);
while STACK is not empty do
    u = POP(S);
    if u.color = WHITE then
        time = time +1 ;
        u.d = time;
        u.color = GRAY;
        for vértice w in G.Ady(u) do
            if w.color == WHITE then
                w.pi = u;
                PUSH(w)
            end
        end
    else
        u.color = BLACK;
        time = time + 1 u.f = time
    end
end
```

Algorithm 2: DFS-VISIT iterativo-condicional

3.2. Expliquen cómo un vértice «u» de un grafo dirigido puede terminar dentro del «depth-first tree» que contenga sólo «u». Aunque «u» tenga aristas saliendo y entrando en el grafo G