Laboratorio 8

José Alejandro Guzmán Zamora

18 de octubre de 2018

- Grafos

  1. Teniendo una lista de adyacencia representando un grafo dirigido, cuanto me toma computar las "$out - degrees$" y las "$in - degrees$" de cada vertice?

  Si se está utilizando una lista de adyacencia, encontrar el "$out - degree$" de cada vértice es relativamente simple. Lo que se tiene que hacer es contar la cantidad de elementos en la lista del vértice que se quiere contabilizar, por supuesto, sin tomar en cuenta el primer elemento de la lista encadenada en el caso que este represente el vértice per se. Por ejemplo, en el siguiente grafo dirigido:

  $$1 \to 2 \to 4/$$
  $$2 \to 4/$$
  $$3 \to 2 \to 7/$$
  $$4 \to 5 \to 3 \to 7/$$
  $$5 \to 1 \to 6/$$
  $$6 \to 4/$$
  $$7 \to 6/$$

  Si se desea encontrar el "$out - degree$" del vértice 4, lo que se hace es verificar la cantidad de elementos en la lista Adj[4] (si los índices comienzan en 1). En este caso el "$out - degree$" del vértice 4 es 3. En cuanto al tiempo que toma computarlo, se toma de manera general el vértice $u \in V$, encontrar este vértice desde el arreglo del grafo toma tiempo constante. Posteriormente se tomaría en cuenta el tamaño de la lista correspondiente. En conclusión toma tiempo lineal $O(n)$ en el que n se representa como el tamaño de Adj[u]. En el caso de que se desee encontrar el "$in - degree$" de un vértice, el procedimiento es un poco más complicado. Lo que se tendría que hacer es recorrer todas las listas encadenadas e incrementar un contador cada vez que se encuentre una referencia al vértice respectivo. Utilizando el mismo grafo que en el ejemplo anterior, para encontrar el "$in - degree$" del vértice 4, se debe recorrer cada lista y se descubre que el cuatro está en la lista de los vértices 1, 2 y 6. Respecto al tiempo que toma computarlo, en el peor de los casos el vértice podría estar en todas las listas encadenadas, hasta el final. Esto quiere decir que se tendría que recorrer la suma de todas las listas $(2 * |E|)$. Sin embargo, también hay que

considerar que hay que recorrer de manera central, el arreglo que representa al grafo y tiene un tamaño de V. En conclusión el tiempo para encontrar el $in - degree$" de un vértice $u \in V$ es $O(V + 2E) \in O(V + E)$.

2. Describa si utilizaría BFS o DFS para resolver los problemas a continuación.

Encontrar más nodos en una red de blockchain.

Para encontrar más nodos en una red de blockchain, puede o no que sea necesario recorrer todos los nodos conocidos. Como se recomienda que se haga el recorrido es a partir de DFS, la principal razón es que recorrer nodos conocidos representa una pérdida de tiempo, mientras más rápido se salga del área de nodos conocidos es mejor. Si se usa BFS, se debe de empezar recorriendo necesariamente todos los nodos conocidos, posteriormente se recorren los nodos de cada nodo ya recorrido. Por el otro lado, al usar DFS, se recorre sólamente un nodo conocido y de una vez se pasa a otro nodo que puede ser nuevo. Si tomamos como ejemplo Bitcoin, y se conoce a 200 $ledgers$, es mejor solamente ir con uno y empezar a recorrer su subred, a diferencia de recorrer primero a los 200 y avanzar capas de manera lenta.

Desarrollar un crawler para un motor de búsqueda.

Un crawler generalmente guarda todas las direcciones url de una página, las visita y realiza lo mismo sobre cada una. En este caso se recomienda que se utilice BFS. Esta recomendación se da basada en la variedad de temas y grupos que debe de cubrir un motor de búsqueda. Si se opta por utilizar DFS, al principio solo se descubrirían páginas relacionadas con el nodo inicial. En cambio si se usa BFS habría una ámplia variedad de contenido.

Encontrar la salida a un laberinto.

A pesar de que suene infantil, la manera en que se puede resolver este problema con grafos es primero recordar la manera en que se resuelven a mano. Cuando una persona tiene un laberinto como problema y desea dibujar una línea desde el principio hasta la salida, generalmente se va por un camino ignorando otros caminos. En ningún momento suena eficiente empezar a dibujar la línea y parar a medio camino para dibujar otra en diferente camino para todos los caminos disponibles. En conclusión se recomienda utilizar DFS, no es muy conveniente utilizar BFS porque en promedio se van a considerar más caminos antes de encontrar la salida.

Sistema de GPS para encontrar caminos de A a B.

Si se piensa este problema desde una perspectiva diferente, este problema es muy similar al de un laberinto. En este caso el inicio del laberinto es A y el final es B. Sin embargo, en este caso lo que se necesita hacer es encontrar la mayor cantidad de caminos para llegar a B. A causa de este detalle agregado, todo cambia y resulta más conveniente utilizar BFS. Si se utilizara DFS, puede ser que el sistema se mantenga mucho tiempo recorriendo un camino que no llega a B. En cambio, si se usa BFS, siempre se están considerando todos los caminos posibles y en momentos se completaran diferentes trayectorias sin desperdiciar tanto tiempo en caminos que no llegan a B.

Detectar un cliclo dentro de un grafo.

Si se desea encontrar un cliclo dentro de un grafo, se recomienda utilizar DFS. Esto es simplemente por el hecho que al ir directamente a los nodos a los que apunta el primer nodo, se puede encontrar al mismo sin necesidad de considerar otros. En otras palabras, no tiene sentido utilizar BFS porque en el peor de los casos el ciclo está hasta el final de la lista encadenada, para llegar en lugar de simplemente recorrerla, antes tendría que recorrer las listas encadenadas correspondientes a cada valor.

- BFS

  1. Cuál es el running time de BFS si se representa al grafo como una matriz? Cuál es el running time si se representa como una lista?

  En el caso que se represente al grafo como un adjacency list primero hay que considerar la instrucción de ENQUEUE() y DEQUEUE(), si estas operaciones toman tiempo constante, entonces el tiempo total que utilizan es de $O(V)$. Esto es porque cada vértice debe de entrar a la cola obligatoriamente una vez, esto resulta en V, sin embargo también hay que sacar a cada vértice de la cola, como resultado tenemos $O(2V) \in O(V)$. Después de considerar las operaciones de la cola, también hay que tomar en cuenta el recorrido que se hace sobre las listas encadenadas, este se hace para cada valor que entra a la cola, con un total de $O(2E) \in O(E)$ operaciones. Sumamos los resultados y tenemos una complejidad de $O(V + E)$. Si se representa al grafo como un adjacency matrix, también se debe de hacer el mismo análisis de las operaciones de la cola, resultando en $O(V)$. Las otras operaciones son un poco menos eficientes. En este caso, para cada valor de la cola no solo debe recorrer la lista encadenada, sino que tiene que recorrer todos los vértices para encontrar las aristas. Si tenemos un grafo de 8 vértices, al final se tendrían que realizar 64 operaciones, porque para cada vértice tiene que recorrer toda la columna correspondiente dentro de la matriz (columna de tamaño V). En conclusión el running time es la suma $O(V^2 + V) \in O(V^2)$.

- DFS

  1. Re escriba el algoritmo DFS utilizando un stack para eliminar la recursión.

DFS(G):

```
1       for each vertex u ∈ G.V:
2           u.color = white
3           u.π = NIL
4       time = 0
5       s = initialize global stack
6       for each vertex u ∈ G.V:
7           if u.color == white:
```

```
8               s.Push(u)
9               DFS-VISIT(G)
```

DFS-VISIT(G):

```
1       While s.length != 0:
2           u = s.pop()
3           u.d = time
4           u.color = gray
5           for each v ∈ G.Adj[u]:
6               if v.color == white:
7                   v.π = u
8                   s.Push(v)
9           u.color = black
9           time = time + 1
9           u.f = time
```

---

2. Explique como un vértice u de un grafo dirigido puede terminar dentro del "$depth - first - tree$" que contenga solo u. Aunque u tenga aristas saliendo y entrando en el grafo G.

Una manera intuitiva en la que un vértice u termina solo dentro del árbol de profundidad es que primero se recorran todos los vértices a los que apunta v y después recorrer v. Esto causa que al momento que toque recorrer v, no hay necesidad de analizarlo porque todo a lo que apunta ya fue recorrido, por lo tanto se queda solo en su árbol de profundidad.