BFS - DFS

Colectivo Estructuras de Datos y Algoritmos

Septiembre 2021

1. Explique por qué es posible afirmar que el recorrido *DFS*, aplicado a un grafo conexo y no dirigido G, define implícitamente un árbol libre en G.

2. Sea $G =< V, E >$ un grafo no dirigido representado por listas de adyacencia. Argumente por qué al realizar *BFS* sobre G, el valor de $d[u](u \in V)$ no depende del orden en que aparezca u en las listas de adyacencia. ¿Ocurriría lo mismo si se calculara $d[u]$ empleando el algoritmo *DFS*?

3. Implemente un algoritmo que dado un grafo no dirigido $G =< V, E >$, un vértice de inicio $s \in V$ y un entero k, devuelva en $O(|V| + |E|)$ los vértices de G que son alcanzables desde s pasando por a lo sumo k aristas.

Respuesta

El algoritmo **BFS** visto en conferencia a partir de un grafo $G =< V, E >$ y un nodo $s \in V(G)$ devuelve un array *pi* y un array *d*.
Donde $d[u]_{u \in V(G)}$ es la cardinalidad del camino de menor logitud que existe entre s y u en G (∞ en caso de no existir dicho camino).
Por lo anteriormente definido, este problema se reduce a saber en el array d, resultado de aplicar $BFS(G, s)$, todos los nodos $u \in V(G)$ tal que $d[u] \leq k$.

```
1  def Solve(G, s, k):
2      d, pi = BFS(G, s)
3      l = []
4      for u in V(G):
5          if(d[u] <= k):
6              l.push(u)
7      return l
```

La complejidad temporal del BFS es $O(|V| + |E|)$ y para calcular la solución es necesario hacer $O(V)$. Aplicando la regla de la suma la complejidad temporal del algoritmo es $O(|V| + |E|)$

4. Proponga un algoritmo que permita determinar si un grafo no dirigido tiene un ciclo o no en $O(|V| + |E|)$. Si el grafo tiene ciclos, su algoritmo debe devolver uno cualquiera. Explique por qué su algoritmo funciona correctamente y justifique el tiempo de ejecución del mismo.

Hint

a) Ver definición en la próxima conferencia de los tipos de arista que existen luego de un recorrido DFS.

b) Demostrar que G es acíclico $\Leftrightarrow$ el DFS no reporta aristas de *retroceso*.

5. Sea $G =< V, E >$ un grafo no dirigido y conexo. Implemente un algoritmo que permita determinar en $O(|V| + |E|)$ si G es o no bipartito y en caso positivo devuelva una partición bipartita de los vértices del grafo. Justifique la correctitud y el orden de su algoritmo.

Hint

a) Ver definición en la próxima conferencia de los tipos de arista que existen luego de un recorrido DFS.

b) Resolver el ejercicio 4.

c) Demostrar que G es *bipartito* $\Leftrightarrow \nexists$ ciclo de longitud impar.

6. Implemente un algoritmo que dado un grafo $G =< V, E >$ y un vértice de origen s determine para cada vértice $v \in V$ la cantidad de caminos de longitud mínima que hay de s a v. Su algoritmo debe funcionar en $O(|V|+|E|)$. Justifique el orden y la correctitud de su algoritmo.

Respuesta

Sea $d[u]$ la longitud de; camino de longitud mínima entre s y u.
Sea $c[u]$ la cantidad de caminos de longitud mínima entre s y u en G. $u \in V(G)$.

En cada camino de longitud mínima de s a u donde $u \neq s$, sea v el vértice anterior a u, en dicho camino se cumple lo siguiente:
$d[v] + 1 = d[u]$ (por definición de caminos de longitud mínima)
El conjunto de caminos de longitud mínima que van de s a u se pueden dividir en subconjuntos, donde a cada subconjuntos lo representa el último vértice en ese camino. De este razonamiento se puede definir la función $c[u]$ de la siguiente forma:

$$c[s] = 1$$

$$c[u] = \sum_{v \in V(G) \mid <u,v> \in E(G) \wedge d[v]+1=d[u]} c[v]$$

Para el correcto cálculo de esta función, por su naturaleza recursiva, es necesario que en el momento de calcular $c[u]$ todos los vértices $v \in V(G)$ tal que $d[v] < d[u]$ ya tengan calculados correctamente sus valores.
En este contexto el BFS es un algoritmo que no solo nos brinda el array d correctamente calculado, además los va calculando en orden de la longitud del camino (visto en conferencia).
El siguiente pseudocódigo brinda solución al problema apoyándose en los razonamientos anteriores:

```
1  def BFS(G, s):
2      d  = [oo for u in V(G)]
3      pi = [-1 for u in V(G)]
4      c  = [0  for u in V(G)]
5      q = queue(s)
6      c[s] = 1
7      d[s] = 0
8      while |q| > 0:
9          v = q.pop()
10         for u in G[v]:
11             if(d[v] + 1 < d[u]):
12                 c[u] = c[v]
13                 d[u] = d[v] + 1
14                 q.push(u)
15                 pi[u] = v
16             else if (d[v] + 1 == d[u]):
17                 c[u] += c[v]
18     return d, pi, c
```

7. Dado un grafo conexo y no dirigido desarrolle un algoritmo que en $O(|V|+|E|)$ retorne el menor ciclo existente en el grafo que contiene a un vértice s dado. Explique por qué su algoritmo funciona y justifique su costo

computacional.

8. Sea $G =< V, E >$ un grafo conexo y no dirigido. Sean $u, v \in V$. Plantee un algoritmo que en $O(|V| + |E|)$ halle un conjunto S de caminos simples de u a v que cumpla lo siguiente:

   a) Cualquier par de caminos en S solo tienen en común a los vértices u y v.

   b) Cualquier otro camino de u a v que se agregue a S violará la condición anterior.

   Justifique el orden y la correctitud de su algoritmo.

9. Sea B la matriz de adyacencia de un grafo no dirigido $G =< V, E >$. Si se define $A = BB^T$ donde B^T es la matriz transpuesta de B, ¿cuál es el significado del valor $A_{i,j}$?

Respuesta

$$B_{i,j} = \begin{cases} 1 & \text{si } < i, j > \in E(G) \\ 0 & \text{en otro caso} \end{cases}$$

$$B^T_{i,j} = \begin{cases} 1 & \text{si } < j, i > \in E(G) \\ 0 & \text{en otro caso} \end{cases}$$

$$A_{i,j} = \sum_{k \in V(G)} B_{i,k} * B^T_{k,j}$$

$$B_{i,k} * B^T_{k,j} = \begin{cases} 1 & \text{si } < i, k >, < k, j > \in E(G) \\ 0 & \text{en otro caso} \end{cases}$$

$B_{i,k} * B^T_{k,j}$ es 1 si existe un camino de longitud 2 entre i y j que pasa por k.
$\sum_{k \in V(G)} B_{i,k} * B^T_{k,j}$ es la cantidad de vértices k tal que existe un camino de longitud 2 entre i y j que pasa por k.
$A_{i,j}$ es la cantidad de caminos de longitud 2 que existen en G entre i y j.

10. El diámetro de un árbol $T =< V, E >$ se define como $\max_{u,v \in V} \delta(u, v)$, es decir, el mayor de todos los caminos de longitud mínima entre dos vértices del árbol. Proponga un algoritmo que calcula de forma eficiente el diámetro de un árbol. Justifique el orden del tiempo de ejecución de su algoritmo y demuestre la correctitud del mismo.

Hint

a) BFS

1) Demuestre que sea $s \in V(T)$ el nodo más alejado de un nodo $u \in V(T)$. s es extremo de un diámetro de T.

b) DFS

1) Sea un nodo $s \in V(T)$, y sea d un diámetro de T. Existen solo dos casos.

   1.1) $s \in d$ por tanto el diametro se consige luego de buscar los nodos más alejados de s.

   1.2) $s \notin d$ por tanto d esta en alguno de los subárboles resultantes luego de eliminar s de T