Tarea 1 de Programación 3

9 de setiembre de 2024

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Problema vértices obligatorios

Decimos que un vértice v de un grafo G es un vértice *obligatorio* si existen al menos dos vértices, s y t, de G, ambos distintos de v, tales que v pertenece a cada camino s-t de G.

Nos proponemos diseñar un algorimo que encuentre todos los vértices obligatorios de un grafo conexo dado.

Sean G un grafo conexo de al menos tres vértices, y T el árbol resultado de realizar una recorrida `DFS` sobre G comenzando en un vértice r cualquiera. El árbol T se considera orientado y con raíz en r. En general, para cualquier vértice x de G llamamos T_x al subárbol de T que tiene a x como raíz.

(a) Demuestre que si en T la raíz r tiene al menos dos hijos, x_1 y x_2, y s y t son vértices de T_{x_1} y T_{x_2} respectivamente, entonces r pertenece a cualquier camino s-t de G.

(b) Recordemos que la profundidad de un vértice v en un árbol con raíz r es la longitud (cantidad de aristas) del camino que empieza en r y termina en el vértice v.

Sea v un vértice de T que es distinto de r (la raíz de T) y no es una hoja, y sea x un hijo de v. Supongamos que ninguna arista (u, w) de G cumple que u pertenece a T_x y w tiene profundidad menor a v.

Demuestre que si s pertenece a T_x y t es distinto de v y no pertenece a T_x entonces v pertenece a cualquier camino s-t de G.

(c) Diseñe un algoritmo que encuentre todos los vértices obligatorios de un grafo conexo $G = (V, E)$.

El algoritmo debe tener tiempo de ejecución $O(|E|)$, y el espacio auxiliar usado debe ser $O(|V|)$. El grafo se representa mediante listas de adyacencia.

(d) Demuestre que el algoritmo es correcto.

(e) Demuestre que el tiempo de ejecución del algoritmo es $O(|E|)$.

Solución:

No se deben usar argumentos que, aunque sean ciertos, y parezcan intuivamente obvios, no están basados en razonamientos rigurosos.

Por ejemplo, si hay un camino entre s y t, y en una recorrida `DFS` se visita s antes que t, entonces t será descendiente de s en el árbol `DFS`. Esto es cierto, pero no está demostrado en el libro de referencia ni en otro material del curso.

Se puede usar resultados conocidos haciendo referencia a ellos.

Utilizaremos la propiedad (3.7) del libro de referencia, que aquí repetimos, y agregamos un par de corolarios para demostrar las dos primeras partes.

(3.7). *Sea T un árbol resultado de una recorrida `DFS`, sean x e y nodos en T, y sea (x, y) una arista de G que no es arista de T. Entonces uno de x o y es un ancestro del otro.*

Notamos que si (x, y) es una arista de T uno de x o y es el padre del otro. Como consecuencia de la propiedad (3.7), en cada arista de G se cumple que uno de los vértices es ancestro del otro en T.

Veamos un corolario de esta propiedad.

Sean v y x vértices del grafo, con x hijo de v en T. Llamamos *aristas de salida del subárbol T_x* a una arista (u, w), con u perteneciente a T_x, y w no perteneciente a T_x. Estas aristas o bien no son de árbol, o bien es la arista de árbol que une la raíz de T_x con su padre, (x, v).

(1). *Sean v un vértice genérico del grafo (que puede ser la raíz), x un hijo de v en el árbol T y (u, w) una arista de salida de T_X. Entonces w es v o un ancestro de v.*

Demostración. Por la consecuencia notada de la propiedad (3.7), uno de u o w es ancestro del otro. Como w no pertenece a T_x no puede ser descendiente de u, de donde se concluye que w tiene que ser ancestro de u, y como no pertenece a T_x, es v o un ancestro de v. □

Las condiciones de la propiedad se cumplen de manera explícita en la parte (b). También, considendo a x_1 como el vértice x de la propiedad, se cumplen en la parte (a), porque t pertenece a T_{x_2} que es disjunto con T_{x_1}. Por la definición de camino, y como s pertenece a T_x y t no, el camino s-t incluye al menos una arista (u, w) de salida de T_x, y por lo tanto un vértice que es ancestro propio de x.

(a) El vértice v de la propiedad (1), es la raíz del árbol. Cualquier camino s-t incluye un vértice w que es r o un ancestro de r, y como r no tiene ancestros, entonces r pertenece a s-t.

> Se debe ser consistente en el uso de la nomenclatura. Por ejemplo, en la propiedad (3.7) se le da nombres x e y a un par de vértices. En el contexto en que se aplique esta propiedad, esos nombres pueden no estar definidos, o incluso asignados a otras entidades.
>
> Si a una entidad ya se le dio un nombre, por ejemplo en la letra, no se le debe asigna un nuevo nombre.

(b) Por hipótesis, en las aristas (u, w) de salida de T_x la profundidad de w debe ser mayor o igual a la de v, por lo cual w no puede ser ancestro propio de v. Entonces, por la propiedad (1), w debe ser v. Como cualquier camino s-t debe incluir una arista de salida se concluye que v pertenece a s-t.

> En una posible demostración por casos, no es válido argumentar que uno de ellos es igual a la parte (a), porque en esa parte el vértice v es la raíz del árbol, no tiene ancestros. Es posible que los argumentos sean similares, pero en ese caso deberían repetirse teniendo en cuenta los detalles en que ambos casos son diferentes.

> En base exclusivamente a las hipótesis no se puede afirmar que no hay un camino desde el vértice s perteneciente a T_x hasta un vértice a profundidad menor a v que no tenga a v como vértice intermedio.
>
> Si existiera un vértice z no perteneciente a T_x y distinto de v con profundidad mayor o igual que v, y en G existiera la arista (u, z), con u perteneciente a T_x, entonces podría haber un camino, que no incluye a v, desde s hasta u, siguiendo con (u, z), y luego con un camino desde z hasta vértices menos profundos. Las hipótesis no excluyen la existencia de una arista (u, z) con esas propiedades, ni el siguiente camino desde z. La arista (u, z) no puede existir, pero eso lo podemos afirmar como consecuencia de la propiedad (3.7).

(c) El algoritmo se diseña teniendo en cuenta lo que se demostró en las partes anteriores. Para ello se enriquece una recorrida con `DFS` con atributos para verificar esas propiedades.

Para cumplir con la parte (a), el vértice raíz se lo incluye en el conjunto de obligatorios si y solo si tiene más de un hijo. Para verificarlo se agrega un contador de hijos para cada nodo (este contador solo es necesario para el vértice raíz, pero se lo incluye para todos los vértices por simplicidad en la descripción).

Llamamos *mínima profundidad de retorno* de un vértice x a la profundidad del vértice más cercano a la raíz al que se puede llegar desde el vértice recorriendo aristas de T_x y una arista que no es de árbol.

Para cumplir con la parte (b), un vértice v distinto de la raíz se lo incluye en el conjunto de obligatorios si y solo si tiene algún hijo x cuya mínima profundidad de retorno es mayor o igual a la profundidad de v. Para ello se mantiene para cada vértice su profundidad, y su mínima profundidad de retorno. Esa mínima profundidad de retorno es el mínimo entre su propia profundidad, el mínimo de las profundidades de los vértices destino de las aristas que no son de árbol que salen de v, y el mínimo de las mínimas profundidades de retorno de sus hijos.

Para cada vértice también se mantiene la información habitual de una recorrida `DFS`, esto es, el padre y si ya ha sido descubierto o no.

En las partes (a) y (b) se demostró que las propiedades en ellas descritas son condiciones suficientes para que un vértice sea obligatorio. No se demostraron los recíprocos, o sea, que para que la raíz sea obligatorio es necesario que tenga más de un hijo, y que para que un vértice v que no es la raíz es necesario que tenga un hijo con mínima profundidad de retorno mayor o igual a la profundidad de v.

Si no se demuestran los recíprocos sería posible que existan vértices que no cumplen las anteriores condiciones pero que de todas formas sean obligatorios.

Por lo tanto, lo que se podría afirmar es que los vértices que el algoritmo considera obligatorios, efectivamente lo son, pero no se podría afirmar que los encuentra todos. Como consecuencia, la demostración de corrección del algoritmo debe incluir la demostración de los recíprocos.

```
1  Algorithm Vértices obligatorios (G = (V, E))
       Data: arreglos indizados por los vértices para: descubierto, padre, cantidad de hijos,
             profundidad, minima profundidad de retorno
       Data: lista para obligatorios
2      Inicializar cada vértice como, no descubierto y con cantidad de hijos igual a 0 y la lista de
         obligatorios vacía; /* los otros atributos se asignan al ser descubiertos   */
3      Elegir un vértice cualquiera, r
4      y asignarle profundidad 0
       /* no hace falta asignar el padre ni la mínima profundidad de retorno      */
5      Invocar RecorridaDFS(r)
6      if la cantidad de hijos de r es mayor que 1 then
7          Agregar r a la lista de obligatorios;
8      return la lista obligatorios

9  Procedure RecorridaDFS(v)
10     Marcar que v fue descubierto
11     foreach x adyacente a v do
12         if x no fue descubierto then
13             Sumar 1 a la cantidad de hijos de v
14             Establecer que el padre de x es v
15             Establecer que la profundidad de x es uno más que la profundidad de su padre, v
16             Inicializar la minima profundidad de retorno de x igual a su profundidad
17             Invocar RecorridaDFS(x)
18             if la minima profundidad de retorno de x es mayor o igual que la profundidad de v then
19                 if v no es la raíz, r then
                       /* v no es raíz si su profundidad es mayor que 0              */
20                     Agregar v a la lista de obligatorios
21             if la minima profundidad de retorno de x es menor que mínima la profundidad de retorno de
                 v then
22                 Ajustar la mínima la profundidad de retorno de v
                   /* igulándola a la mínima profundidad de retorno de x          */
23         else if x no es el padre de v then
24             if la profundidad de x es menor que mínima la profundidad de retorno de v then
25                 Ajustar la mínima la profundidad de retorno de v
                   /* igulándola a la profundidad de x                            */
```

Figura 1: Algoritmo vértices obligatorios.

Se debe notar que cuando en la línea 23 se procesan las aristas que no son de árbol, la comparaciónse debe haber con la profundidad, no con la mínima profundidad de retorno, de x. La mínima profudidad de retorno de x puede ser menor, debido a una arista de salida de algún otro subárbol de x. Pero esa arista no podría ser parte de un camino que evite que x sea obligatorio para v.

(d) Hay que demostrar que los vértices devueltos por el algoritmo son obligatorios y que los devuelve todos. Para esto último debemos demostrar los recíprocos de las partes (a) y (b).

(2). *La raíz r de T es un vértice obligatorio si y solo si tiene más de un hijo.*

Demostración. En la parte (a) se demostró que si r tiene más de un hijo existen pares de vértices s y t tales que todos los caminos s-t contienen a r; por lo tanto r es un vértice obligatorio.

Veamos ahora que si r tiene un solo hijo, x, no es vértice obligatorio. En este caso todo par de vértices s y t distintos de r pertenecen a T_x. Como T_x es conexo existe un camino entre cada par de sus vértices que contiene solo aristas de T_x, a las cuales no pertenece r. □

(3). *Sea v distinto de r. El vértice v es obligatorio si y solo si tiene un hijo x, tal que no hay arista (u, v) de salida de T_x, donde la profundidad de w es mayor o igual a la profundidad de v.*

Demostración. En la parte (b) se demostró que si esto se cumple entonces v es vértice obligatorio. Vamos a demostrar que si no se cumple entonces no es obligatrorio. Para mayor claridad vamos a considerar los casos posibles.

Ni s ni t son descendientes de v. Los caminos desde r hasta s y desde r hasta t no incluyen v. Sea x el último vértice de esos dos caminos que pertenece a ambos (el ancestro común más profundo de s y t). El vértice x existe porque r es común a ambos caminos. Entonces, la concatenación de los caminos s-x y x-t es un camino s-t que no contiene a v.

Uno de s o t es descendiente de v y el otro no. Sin pérdida de generalidad supongamos que es s el descendiente de v y que pertenece al subárbol T_x, con x hijo de v. Hay una arista (u, w) de salida de T_x, donde w es un ancestro propio de v. Como w no es descendiente de v hay, como en el caso anterior, un camino w-t que no contiene a v. Como s y u pertenecen a T_x hay un camino s-u que no contiene a v. Entonces la concatenación del camino s-u, la arista (u, w) y el camino w-t es un camino s-t que no contiene a v.

s y t son descendientes de v. Si s y t pertenecen al mismo subárbol de v, hay un camino entre ellos que contiene solo aristas del subárbol, a las cuales no pertenece v. Si s y t pertenecen a distintos subárboles de v, como en el caso anterior, hay caminos desde s hasta w_s y desde t hasta w_t que no contienen a v, donde w_s y w_t son ancestros propios de v, entre los que, como se vio en el primer caso, hay un camino que no contiene a v. Entonces, la concatenación de los caminos s-w_s, w_s-w_t y w_t-t es un camino s-t que no contiene a v.

□

Ahora demostramos que los vértices devueltos por el algoritmo son obligatorios y que los devuelve todos.

El algoritmo es una adaptación de `DFS` que se sabe que termina construyendo el árbol representado por el arreglo de padres. Se agrega para cada nodo su profundiad, el conteo de la cantidad hijos, y la mínima profundidad de retorno, y una lista para los vértices obligatorios. Las operaciones que resuleven esto son elementales, por lo que se mantiene que el algortimo termina.

La profundidad del vértice se establece al ser descubierto (uno más que la de su padre) y nunca cambia.

La cantidad de hijos de v se inicializa en 0 y se incrementa cada vez que se descubre un nuevo vértice x, al cual se le asigna v como padre.

La mínima profundidad de retorno es el mínimo entre su profundidad (línea 16), la mínima profundiad de retorno de cada uno de sus hijos (línea 22), y la mínima profundidad de las aristas que no son de árbol (línea 25).

La raíz del árbol se agrega a la colección de obligatorios si y solo si tiene más de un hijo (líneas 6 y 19). Un vértice v que no es raíz se agrega a la colección de obligatorios si y solo si tiene algún hijo cuya mínima profundidad de retorno no es menor que la profundiad de v (líneas 18 y 19).

(e) El espacio adicional consiste en arreglos de tamaño $|V|$ y una lista de tamaño menor o igual $|V|$, por lo tanto el espacio adicional es $\Theta(|V|)$.

El tiempo de ejecución de una recorrida `DFS` representada mediante listas de adyacencia se sabe que es $O(|V| + |E|)$. El algoritmo solo agrega operaciones $O(1)$. Como el grafo es conexo se cumple que $|V|$ es $O(|E|)$, por lo que el tiempo de ejecución es $O(|E|)$.