

UNIVERSIDAD AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Autores:

Fernanda Villafán Flores

Fernando Alvarado Palacios

Adrián Aguilera Moreno



Gráficas y Juegos

Tarea 8

1. Sean G una gráfica conexa y $e \in E$. Demuestre que
 - (a) e está en cada árbol generador de G si y sólo si e es un puente de G ;
 - (b) e no está en árbol generador alguno de G si y sólo si e es un lazo.

2. Modifique el algoritmo BFS para que regrese una bipartición de la gráfica (si la gráfica es bipartita) o un ciclo impar (si la gráfica no es bipartita).

Solución: Para este ejercicio, emplearemos las siguientes estructuras de datos:

-) Colas: Estas fungen de la misma manera que originalmente en **BFS**.
-) Listas: Estas nos servirán para insertar listas binarias en un tiempo constante en una posición conocida.
-) Conjuntos: Esta estructura realmente se puede cambiar por una lista, arreglo (o la estructura que más les guste), sin embargo, se emplean conjuntos para preservar, en la medida de lo posible, el concepto de parte de una bipartición y aun más porque, en este caso, necesitamos una bipartición de vértices.

Los ciclos de longitud impar son obstrucción mínima de las gráficas bipartitas. Este resultado se menciona en las notas de clase, además, esto se sigue por la caracterización de gráficas bipartitas (demostrado en clases), esto es,

Teorema. Sea G una gráfica. Son equivalentes:

- (a) G es bipartita.
- (b) G no contiene ciclos impares.
- (c) G no contiene ciclos impares inducidos.

En esta ocasión, empecemos mostrando que el algoritmo termina. Para esto sólo nos debemos fijar en el **While** de la línea 25¹. Nótese que x e y no son el nodo raíz, así x e y son descendientes de r , por lo que ocasionalmente

$$\mathcal{P}(\mathcal{P}(\dots \mathcal{P}(x))) = r = \mathcal{P}(\mathcal{P}(\dots \mathcal{P}(y)))^2$$

como la cantidad de vértices en G es finita, entonces lo anterior pasa en una cantidad finita de iteraciones, por tanto la instrucción iterativa de la línea 25 termina.

Para terminar de mostrar la correctez del algoritmo sólo falta ver que realmente hace lo que se pide, así propongamos dos invariantes de ciclo:

Prop. 1: [Invariante de ciclo, caso bipartita]. El conjunto devuelto por el algoritmo `OddCycleOrBipartition`, cuando G no contenga ciclos impares, es una bipartición de V_G .

Dem. Para esto veamos que:

-) $X \cup Y = V_G$. Por la línea 5, tenemos que el vértice raíz (que es el primero en entrar a la cola) se pinta de negro, por tanto sus hijos tienen un padre coloreado (y en consecuencia, estos serán coloreados de blanco), luego por el **IF/ELSE** de la línea 12, el resto de los vértices se pintan de negro o blanco, como todo el resto de vértices es descendiente de r , entonces todos estos tienen un padre coloreado, pues su ancestro inmediato (no necesariamente r) está coloreado. Luego, todos los vértices de G son coloreados (esto por la demostración de **BFS**) e introducidos en X o Y , así $X \cup Y = V_G$.
-) $X \cap Y = \emptyset$. Observemos que r es coloreado de negro, así los hijos de r son coloreados de blanco. Luego, para algún vértice x distinto de r se tiene que si x está coloreado de negro, entonces sus hijos serán coloreados de blanco (análogo el caso cuando x es de color blanco). Si suponemos que existe

$$z \in V_G : z \in X \wedge z \in Y$$

¹El análisis del resto del algoritmo se realizó en clase y las modificaciones incluidas no alteran las condiciones iniciales de **BFS**.

entonces z debe tener 2 padres distintos (uno de color negro y otro blanco), pero esto contradice a la función de parentesco, que como vimos cuando se mostró **BFS**, a cada vértice le asocia un único padre. Por tanto no existe

$$z \in V_G : z \in X \wedge z \in Y$$

y concluimos que $X \cap Y = \emptyset$.

⊥

1: OddCycleOrBipartition($\langle G, r \rangle; L/C$)

Input: Una gráfica conexa G con un vértice distinguido r .

Output: Una lista que contenga un ciclo impar o un conjunto que contenga una bipartición entre los vértices.

```

1   $Q \leftarrow []; i \leftarrow 0;$ 
2   $L \leftarrow []; C \leftarrow \emptyset;$ 
3   $X \leftarrow \emptyset; Y \leftarrow \emptyset;$ 
4   $i \leftarrow i + 1; X \leftarrow r;$ 
5  colorear a  $r$  de negro;
6  añadir a  $r$  al final de  $Q$ ;
7   $t(r) \leftarrow i, \mathcal{P}(r) \leftarrow \emptyset, \ell(r) \leftarrow 0;$ 
8  while  $Q \neq []$  do
9      elegir a la cabeza  $x$  de  $Q$ ;
10     if  $x$  tiene un vecino  $y$  sin colorear then
11          $i \leftarrow i + 1;$ 
12         if  $x$  es color negro then
13              $Y \leftarrow y;$ 
14             colorear a  $y$  de blanco;
15         else
16              $X \leftarrow y;$ 
17             colorear a  $y$  de negro;
18         end
19         añadir  $y$  al final de  $Q$ ;
20          $t(r) \leftarrow i, \mathcal{P}(y) \leftarrow x, \ell(y) \leftarrow \ell(x) + 1;$ 
21     else
22         if  $x$  tiene un vecino  $y$  coloreado &  $\ell(x) = \ell(y)$  then
23              $L \leftarrow [x, y, x];$ 
24              $\text{temp} \leftarrow [];$ 
25             while  $\mathcal{P}(x) \neq \mathcal{P}(y)$  do
26                  $x \leftarrow \mathcal{P}(x); y \leftarrow \mathcal{P}(y);$ 
27                  $\text{temp} \leftarrow [x, y];$ 
28                 insertar  $\text{temp}$  entre los  $x$  e  $y$  más centrales en  $L$ ;
29             end
30             insertar  $\mathcal{P}(x)$  entre los  $x$  e  $y$  más centrales en  $L$ ;
31             return  $L$ ;
32         end
33         eliminar  $x$  de  $Q$ ;
34     end
35 end
36  $C \leftarrow [X, Y];$ 
37 return  $C$ ;
```

Prop. 2: [Invariante de ciclo, en caso de haber ciclo impar]. La lista devuelta por el algoritmo OddCycleOrBipartition, cuando G no es bipartita, es un ciclo impar.

Dem. Por el teorema anunciado con anterioridad (mostrado en clase), tenemos que nuestra gráfica no es bipartita.

Obsérvese que por la línea 22 tenemos un ciclo, pues estamos antes $x, y \in V_G$ que ya han sido explorados con anterioridad, sin embargo existe una xy -arista, por tanto hay dos trayectorias con algún vértice en común (*e.g.*, r) que terminan en x e y respectivamente, luego la xy -arista cierra un ciclo. De esta manera sólo nos falta verificar que el ciclo es de longitud impar, para esto veamos que, como x tiene el mismo nivel que y , entonces para el ancestro común a x e y más lejano de r , llamémosle $z \in V_G$, tenemos que $d(z, x) = d(z, y)$, esto se sigue de

$$\begin{aligned} d(z, x) &= \ell(x) - \ell(z) \\ &= \ell(y) - \ell(z) \\ &= d(z, y) \end{aligned}$$

así, $[d(z, x) + d(z, y)] \stackrel{2}{\equiv} 0$, y si ha esto le sumamos la xy -arista, entonces tenemos que

$$[d(z, x) + d(z, y) + |xy|] \stackrel{2}{\equiv} 1$$

de lo anterior, hemos encontrado un ciclo impar. El ciclo While de la 25 termina cuando se llega a un ancestro en común, que en el peor de los casos este sería r , así la línea 28 inserta listas binarias al centro de L y permite tener consistencia con el orden del ciclo, finalmente la línea 30 inserta al ancestro, de x e y , común más lejano de r a la lista L . Por tanto, con L se puede construir un ciclo de longitud impar. \dashv

Finalmente, observemos que por el teorema enunciado con anterioridad (que fue mostrado en clase), G contiene un ciclo impar o una bipartición en V_G , pero no ambas. Así, por la **Prop. 1** tenemos que el algoritmo `OddCycleOrBipartition` devuelve una bipartición cuando no contiene un ciclo impar, y por **Prop.2** tenemos que `OddCycleOrBipartition` regresa un ciclo impar cuando G no es bipartita.

Ahora, analicemos la complejidad del algoritmo. Sabemos que **BFS** tiene complejidad contenida en $\mathcal{O}(|E| + |V|)$, por tanto el algoritmo `OddCycleOrBipartition` tiene complejidad en

$$\mathcal{O}(|E| \cdot |V| + |V|^2) \approx \mathcal{O}(|V|^2)$$

esto por la anexión de la instrucción iterativa While de la línea 25, pues todas las demás alteraciones a **BFS** se realizan en un tiempo constante. \square

3. Describa un algoritmo basado en BFS para encontrar el ciclo impar más corto en una gráfica.

Solución: A continuación se muestra una propuesta para el algoritmo solicitado:

2: OptimalOddCycle($\langle G, r \rangle; L/C$)

Input: Una gráfica conexa G , que contenga al menos un ciclo impar, con un vértice distinguido r .

Output: Una lista que contenga al ciclo impar más corto.

```

1  $Q \leftarrow []; i \leftarrow 0;$ 
2  $L_1, L_2 \leftarrow [];$ 
3  $i \leftarrow i + 1;$ 
4 colorear a  $r$  de negro;
5 añadir a  $r$  al final de  $Q$ ;
6  $t(r) \leftarrow i, \mathcal{P}(r) \leftarrow \emptyset, \ell(r) \leftarrow 0;$ 
7 while  $Q \neq []$  do
8   if  $x$  tiene un vecino  $y$  sin colorear then
9      $i \leftarrow i + 1;$ 
10    elegir a la cabeza  $x$  de  $Q$ ;
11    colorear a  $y$  de negro;
12    añadir  $y$  al final de  $Q$ ;
13     $t(r) \leftarrow i, \mathcal{P}(y) \leftarrow x, \ell(y) \leftarrow \ell(x) + 1;$ 
14  else
15    if  $x$  tiene un vecino  $y$  coloreado &  $\ell(x) = \ell(y)$  then
16       $L_1 \leftarrow [x, y, x];$ 
17       $\text{temp} \leftarrow [];$ 
18      while  $\mathcal{P}(x) \neq \mathcal{P}(y)$  do
19         $x \leftarrow \mathcal{P}(x); y \leftarrow \mathcal{P}(y);$ 
20         $\text{temp} \leftarrow [x, y];$ 
21        insertar  $\text{temp}$  entre los  $x$  e  $y$  más centrales en  $L_1$ ;
22      end
23      insertar  $\mathcal{P}(x)$  entre los  $x$  e  $y$  más centrales en  $L_1$ ;
24      if  $|L_2| = []$  then
25         $L_2 \leftarrow L_1;$ 
26      else if  $|L_1| < |L_2|$  then
27         $L_2 \leftarrow L_1;$ 
28      end
29    end
30    eliminar  $x$  de  $Q$ ;
31  end
32 end
33 return  $L_2;$ 

```

Para este ejercicio emplearemos la estructura de datos listas, pues se puede insertar en alguna posición de la lista en tiempo constante.

Prop. [Invariante de ciclo]. La lista devuelta por el algoritmo OddCycleOrBipartition es un ciclo impar de longitud más pequeña en comparación con todos los posibles ciclos impares en G .

Dem. Por **Prop. 2** en el ejercicio 2 (pues se siguen las mismas líneas) sabemos que nuestro algoritmo ya devuelve un ciclo de longitud impar, falta mostrar que este es de longitud más corta. Así, analicemos el If de la línea 24, podemos notar que cuando se itera el algoritmo OptimalOddCycle y se encuentra por 1era vez un ciclo de longitud impar, tenemos que $L_1 = L_2$, y tanto L_1 como L_2 son la representación de ciclos de longitud impar.

En iteraciones posteriores a la 1era, ya no se entrará la línea 24, en cambio se seguirá directamente la instrucción de la línea 26, luego L_1 y L_2 son la representación de ciclos de longitud impar, no

necesariamente distintos, así:

-) Si $|L_1| < |L_2|$, entonces L_2 no es el ciclo de longitud impar más corto, por la línea 27, L_2 representará un ciclo impar de longitud menor al que anteriormente representaba.
-) En otro caso, L_2 es el más corto, pues L_1 es de mayor longitud, o bien, ambos son de la misma longitud y nos quedamos con el que ya teníamos.

cuando el algoritmo termine L_2 contendrá la representación de un ciclo impar con menor longitud comparado con los posibles ciclos impares en G . \dashv

Usando la proposición anterior, tenemos que L_2 es el ciclo impar de menor longitud en G , por tanto el algoritmo `OptimalOddCycle` hace lo que se pide.

Para terminar de mostrar la correctez del algoritmo `OptimalOddCycle`, veamos que este termina. Para esto analicemos el `While` de la línea 18, que como ya vimos en el ejercicio 2, termina en un tiempo finito de iteraciones, por tanto nuestro algoritmo termina, pues las demás instrucciones que alteran a **BFS** son instrucciones no iterativas.

Ahora, analicemos la complejidad del algoritmo. Sabemos que **BFS** tiene complejidad contenida en $\mathcal{O}(|E| + |V|)$, por tanto el algoritmo `OptimalOddCycle` tiene complejidad en

$$\mathcal{O}(|E| \cdot |V| + |V|^2) \approx \mathcal{O}(|V|^2)$$

esto por la anexión de la instrucción iterativa `While` de la línea 18, pues todas las demás alteraciones a **BFS** se realizan en un tiempo constante. \square

4. Sea G una gráfica con conjunto de bloques B y conjunto de vértices de corte C . La *gráfica de bloques y cortes* de G , denotada por $B_C(G)$, esta definida por $V_{B_C(G)} = B \cup C$ y si $u, v \in V_{B_C(G)}$, entonces $uv \in E_{B_C(G)}$ si y sólo si $u \in B$, $v \in C$ y v es un vértice de u . Demuestre que $B_C(G)$ es un árbol.

Demostración: Probaremos que $B_C(G)$ es un árbol.

Para esto, tomaremos los siguientes casos:

- Si dos bloques de G son adyacentes.

Sean $B_1, B_2 \in B$.

Si B_1 es adyacente a B_2 , entonces tenemos que existe una arista e tal que:

$$e = (u, v)$$

, donde $u \in V_{B_1}$ y $v \in V_{B_2}$.

Ahora sabemos que por definición de $B_C(G)$, existe un vértice w tal que $w \in V_{B_1}$ y $w \in V_{B_2}$.

Por lo anterior, tenemos un ciclo entre los vértices u , v y w . Esto implica que $B_C(G)$ no es un árbol, ya que un árbol es conexo y acíclico.

Por lo tanto, B_1 y B_2 no pueden ser adyacentes.

- Si dos vértices de corte son adyacentes.

Sea $B_1 \in B$ y sean $c_1, c_2 \in C$ tales que:

$$c_1 \in V_{B_1} \text{ y } c_2 \in V_{B_1}$$

Si c_1 es adyacente a c_2 , entonces tenemos que existe una arista e tal que:

$$e = (c_1, c_2)$$

Por lo anterior, para cualquier vértice $w \in V_{B_1}$ tenemos que existe un ciclo entre c_1 , c_2 y w .

Esto implica que $B_C(G)$ no es un árbol, ya que un árbol es conexo y acíclico.

Por lo tanto, c_1 y c_2 no pueden ser adyacentes.

□

5. Describa un algoritmo para encontrar un bosque generador en una gráfica arbitraria (no necesariamente conexa).
6. Una *gráfica de Moore de diámetro d* es una gráfica regular de diámetro d y cuello $2d + 1$. Demuestre que si G es una gráfica de Moore, entonces todos los árboles de BFS de G son isomorfos.

Puntos Extra

1. Sea G una gráfica conexa en la que todo árbol de DFS es una trayectoria hamiltoniana (con la raíz en uno de los extremos). Demuestre que G es un ciclo, una gráfica completa, o una gráfica bipartita completa en la que ambas partes tienen el mismo número de vértices.
2. Modifique BFS para que sea recursivo en lugar de iterativo.
3. Modifique DFS para que sea recursivo en lugar de iterativo.
4. Modifique al algoritmo BFS para que:
 - (a) Reciba una gráfica no necesariamente conexa con dos vértices distinguidos r y t .
 - (b) El algoritmo empiece en r , y termine cuando encuentre al vértice t , en cuyo caso lo regresa, junto con una trayectoria de longitud mínima de r a t , o cuando decida que el vértice t no puede ser alcanzado desde r , en cuyo caso regresa el valor **false**.
 - (c) El primer paso dentro del loop de **while** sea **eliminar** la cabeza de la cola.