

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

Proposiciones a usar

Prop. 1) Corolario 2.2.3 Si G es una gráfica conexa, entonces G contiene un árbol generador (Demostración en la notas página 32).

Obs. 1) Por definición podemos concluir que una arista es un puente sii no está contenida en ningún ciclo.

- (a) e está en cada árbol generador de G si y sólo si e es un puente de G ;

\Rightarrow) Dem (Ad. abs.). Sea e que pertenece a todo árbol generador de G tal que e no sea un puente \rightarrow como G es conexa si quitamos a e , $G-e$ seguirá siendo conexa \rightarrow por Prop. 1 $G-e$ tendrá un árbol generador, que lo denotaremos como T_e , como todos los vértices de $G-e$ están también en $G \rightarrow$ que T_e también es un árbol generador de G pero e no está en T_e ! ya que por hipótesis e pertenece a todo árbol generador de G . Por lo tanto e es un puente.

\Leftarrow) Dem (Ad. abs.) Sea e un puente de G y T_e un árbol generador de G tal que e no pertenece a las aristas de T_e , donde e es una arista que une a u con $v \rightarrow$ como T_e es conexa \rightarrow existe una trayectoria en T_e que une a u con v , pero esta misma trayectoria también debe estar en $G \rightarrow$ existen al menos 2 trayectorias que unen a u con v ! pero esto es una contradicción ya que e es un puente.

- (b) e no está en árbol generador alguno de G si y sólo si e es un lazo.

\Rightarrow) Sea e que no está en ningún árbol generador de $G \rightarrow$ por contrapositiva del inciso a) $\rightarrow e$ no es un puente \rightarrow por contrapositiva de la Obs. 1) e está contenida en un ciclo, pero esto solo puede pasar si e es un lazo, ya que si esto no pasa podríamos dar un árbol generador de G tal que e pertenezca a este.

\Leftarrow) Sea e un lazo $\rightarrow e$ pertenece a un ciclo (por definición) \rightarrow por contrapositiva de la Obs. 1) e no es un puente \rightarrow por contrapositiva del inciso a) e no está en ningún árbol generador de G .

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.

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.

Prop.

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

²En otro caso, x e y tienen un ancestro común distinto de r y se cumple la misma condición.

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 la primer  $x$  y  $y$  en  $L$ ;
29      end
30      insertar  $\mathcal{P}(x)$  entre la primer  $x$  y  $y$  en  $L$ ;
31      return  $L$ ;
32    end
33    eliminar  $x$  de  $Q$ ;
34  end
35 end
36  $C \leftarrow [X, Y];$ 
37 return  $C$ ;
```

□

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

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

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

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

```

1  $Q \leftarrow []; i \leftarrow 0;$ 
2  $L \leftarrow [];$ 
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       $L \leftarrow [x, y, x];$ 
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 la primer  $x$  y  $y$  en  $L$ ;
22      end
23      insertar  $\mathcal{P}(x)$  entre la primer  $x$  y  $y$  en  $L$ ;
24      return  $L$ ;
25    end
26    eliminar  $x$  de  $Q$ ;
27  end
28 end
29 return  $L$ ;
```

□

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.

5. Describa un algoritmo para encontrar un bosque generador en una gráfica arbitraria (no necesariamente conexa).

Para encontrar un bosque generador en una gráfica G , llamaremos BFS a cada componente conexa de G (en caso que G , sea inconexa).

Algorithm 3: BosqueGenerador

```

1 Input:
2 Output:
3  $F \leftarrow \emptyset$  (1)
4  $P \leftarrow \emptyset$  (2)
5 for  $v$  en  $V_G$  do
6   | if  $v$  no está coloreado then
7   |   | encolar en  $P$  la función  $p$  que nos regresa  $BFS(G, v)$ 
8   |   end
9   end
10 for  $p$  en  $P$  do
11   | añadir  $p$  a  $F$  (3)
12 end
13 return  $F$ 

```

Comentarios:

- (1) Será la colección de árboles de G
 - (2) Colección de funciones de parentesco
 - (3) Nos servirá para obtener el árbol o árboles generados en la colección
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.