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 pagina 32).

   Obs. 1) Por definicion podemos concluir que una arista es un puente sii no esta contenida en ningun ciclo.

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

   $\Longrightarrow$) 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 seguira siendo conexa $\rightarrow$ por Prop. 1 G-e tendra un arbol generador, que lo denotaremos como $T_e$, como todos los vértices de G-e están tambien en G $\rightarrow$ que $T_e$ tambien es un arbol generador de G pero $e$ no esta en $T_e$! ya que por hipóteis $e$ pertenece a todo árbol generador de G. Por lo tanto $e$ es un puente.

   $\Longleftarrow$) Dem (Ad. abs.) Sea $e$ un puente de G y $T_e$ un árbol generador de G tal que e no pertenece a las arista 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 tambien debe estar en G $\rightarrow$ existen al menos 2 trayectorias que unen a $u$ con $v$! pero esto es una contradiccion ya que $e$ es un puente.

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

   $\Longrightarrow$) Sea $e$ que no esta en ningun arbol 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 podriamos dar un árbol generador de G tal que $e$ pertenezca a este.

   $\Longleftarrow$) 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 esta en ningun arbol 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 insertas 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 25[1]. Nótese que $x$ e $y$ no son el nodo raíz, así $x$ e $y$ son descendientes de $r$, por lo que ocasionalmente $\mathcal{P}(\mathcal{P}(\cdots\mathcal{P}(x))) = r = \mathcal{P}(\mathcal{P}(\cdots\mathcal{P}(y)))$[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.**

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

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

**1:** OddCycleOrBipartition($< G, r >; L/C$)

**Input:** Una gráfica conexa $G$ con un vértice distingido $r$.
**Output:** Una lista que contenga un ciclo impar o un conjunto que contenga una bipartición entre los vértices.

```
1  Q ← []; i ← 0;
2  L ← []; C ← ∅;
3  X ← ∅; Y ← ∅;
4  i ← i + 1; X ← r;
5  colorear a r de negro;
6  añadir a r al final de Q;
7  t(r) ← i, P(r) ← ∅, ℓ(r) ← 0;
8  while Q ≠ [] do
9      elegir a la cabeza x de Q;
10     if x tiene un vecino y sin colorear then
11         i ← i + 1;
12         if x es color negro then
13             Y ← y;
14             colorear a y de blanco;
15         else
16             X ← y;
17             colorear a y de negro;
18         end
19         añadir y al final de Q;
20         t(r) ← i, P(y) ← x, ℓ(y) ← ℓ(x) + 1;
21     else
22         if x tiene un vecino y coloreado & ℓ(x) = ℓ(y) then
23             L ← [x,y,x];
24             temp ← [];
25             while P(x) ≠ P(y) do
26                 x ← P(x); y ← P(y);
27                 temp ← [x, y];
28                 insertar temp entre la primer x y y en L;
29             end
30             insertar 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 ← [X, Y];
37 return C;
```

□

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

**2:** OptimalOddCycle($< G, r >; L/C$)

**Input:** Una gráfica conexa $G$ con un vértice distingido $r$.
**Output:** Una lista que contenga al ciclo impar más corto.

```
1  Q ← []; i ← 0;
2  L ← [];
3  i ← i + 1;
4  colorear a r de negro;
5  añadir a r al final de Q;
6  t(r) ← i, P(r) ← ∅, ℓ(r) ← 0;
7  while Q ≠ [] do
8      if x tiene un vecino y sin colorear then
9          i ← 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) ← i, P(y) ← x, ℓ(y) ← ℓ(x) + 1;
14     else
15         if x tiene un vecino y coloreado & ℓ(x) = ℓ(y) then
16             L ← [x,y,x];
17             temp ← [];
18             while P(x) ≠ P(y) do
19                 x ← P(x); y ← P(y);
20                 temp ← [x, y];
21                 insertar temp entre la primer x y y en L;
22             end
23             insertar 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

```
Input:
Output:
F ← Ø (1)
P ← Ø (2)
for v en V_G do
    if v no está coloreado then
        encolar en P la función p que nos regresa BFS(G, r)
    end
end
for p en P do
    añadir p a F (3)
end
return F
```

Comentarios:

(1) Será la collecion de árboles de G

(2) Collecion de funciones de parentesco

(3) Nos servirá para obtener el árbol o árboles generados en la colleción

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.