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 <u>While</u> 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.

   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 concecuencia, 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

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

[1] El análisis del resto del algoritmo se realizo 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 parentezco, 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 $X \cap Y = \emptyset$.

$\dashv$

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.

```
Q ← []; i ← 0;
L ← []; C ← ∅;
X ← ∅; Y ← ∅;
i ← i + 1; X ← r;
colorear a r de negro;
añadir a r al final de Q;
t(r) ← i, 𝒫(r) ← ∅, ℓ(r) ← 0;
while Q ≠ [] do
    elegir a la cabeza x de Q;
    if x tiene un vecino y sin colorear then
        i ← i + 1;
        if x es color negro then
            Y ← y;
            colorear a y de blanco;
        else
            X ← y;
            colorear a y de negro;
        end
        añadir y al final de Q;
        t(r) ← i, 𝒫(y) ← x, ℓ(y) ← ℓ(x) + 1;
    else
        if x tiene un vecino y coloreado & ℓ(x) = ℓ(y) then
            L ← [x,y,x];
            temp ← [];
            while 𝒫(x) ≠ 𝒫(y) do
                x ← 𝒫(x); y ← 𝒫(y);
                temp ← [x,y];
                insertar temp entre los x e y más centrales en L;
            end
            insertar 𝒫(x) entre los x e y más centrales en L;
            return L;
        end
        eliminar x de Q;
    end
end
C ← [X, Y];
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, llamemosle $z \in V_G$, 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, $[d(z,x) + d(z,y)] \overset{2}{\equiv} 0$, y si ha esto le sumamos la xy-arista, entonces tenemos que

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

de lo anterior, hemos encontrado un ciclo impar. El ciclo <u>While</u> 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 <u>While</u> 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($< G, r >; L/C$)

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

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

```
1  Q ← []; i ← 0;
2  L1, L2 ← [];
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             L1 ← [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 los x e y más centrales en L1;
22             end
23             insertar P(x) entre los x e y más centrales en L1;
24             if |L2| = [] then
25                 L2 ← L1;
26             else if |L1| < |L2| then
27                 L2 ← L1;
28             end
29         end
30         eliminar x de Q;
31     end
32 end
33 return L2;
```

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 reprecentación 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 teniamos.

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.

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.