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 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.

```
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 la primer x y y en L;
            end
            insertar 𝒫(x) entre la primer x y y en L;
            return L;
        end
        eliminar x de Q;
    end
end
C ← [X, Y];
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.

```
Q ← []; i ← 0;
L ← [];
i ← i + 1;
colorear a r de negro;
añadir a r al final de Q;
t(r) ← i, P(r) ← ∅, ℓ(r) ← 0;
while Q ≠ [] do
    if x tiene un vecino y sin colorear then
        i ← i + 1;
        elegir a la cabeza x de Q;
        colorear a y de negro;
        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 la primer x y y en L;
            end
            insertar P(x) entre la primer x y y en L;
            return L;
        end
        eliminar x de Q;
    end
end
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).

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.