

# UNIVERSIDAD AUTÓNOMA DE MÉXICO

## Facultad de Ciencias

Autores:

Fernanda Villafán Flores  
Fernando Alvarado Palacios  
Adrián Aguilera Moreno



Gráficas y Juegos

## Tarea 9 y 10

### Puntos Extra

1. (2 puntos) Sea  $G$  una gráfica conexa y  $e$  una arista de  $G$  que no sea un lazo. Exhiba una biyección entre el conjunto de árboles generadores de  $G$  que contienen a  $e$  y el conjunto de árboles generadores de  $G/e$ .

**Demostración:** Sea  $G$  un conjunto de árboles generadores y  $T \subset G$  tal que  $e \in T$  y  $T' \subset G$  tal que  $T' = G \setminus e$ .

Definamos la función  $f: T \rightarrow T'$  tal que si  $g \in T \implies f(g) = g \setminus e$ .

Observemos que como  $g \in T$  ( $g$  es un árbol generador)  $\implies f(g)$  también será un árbol generador  $\implies$  la función  $f$  preserva conexidad.

Pd)  $f$  es inyectiva

Sea  $g_1, g_2 \implies$  si  $f(g_1) = f(g_2) \implies$  el subconjunto  $A$  y  $A'$  de aristas en  $G$  que induce a  $f(g_1)$  y  $f(g_2)$  respectivamente son iguales  $\implies g_1 = A \setminus e = A' \setminus e = g_2 \implies g_1 = g_2$ .

Por lo tanto  $f$  es inyectiva

Pd)  $f$  es suprayectiva

Sea  $b \in T' \implies$  por definición  $b = b \setminus e \implies$  sea  $w$  un vértice que une a las aristas  $a_1$  y  $a_2 \implies$  "Partimos a  $w$  en dos vértices  $u, v$ " y en medio de  $u, v$  colocamos a  $e$  y denotamos a este nuevo árbol como  $g_3 \implies$  existe  $g_3 \in T$  tal que  $f(g_3) = b$ .

Por lo tanto  $f$  es sobreyectiva.

Por lo tanto  $f$  es un biyección.

□

2. (2 puntos) Sea  $T$  un árbol de DFS de una gráfica conexa no trivial  $G$ , y sea  $v$  la raíz de un bloque  $B$  de  $G$ . Demuestre que el grado de  $v$  en  $T \cap B$  es uno.

**Demostración:** Observemos el caso particular para  $k_2$ , este caso siempre se cumple sin importar en qué vértice se ejecute DFS, nuestro árbol siempre será de tipo  $k_2$ , donde los dos vértices tendrán grado 1.

Sea  $G$  una gráfica conexa con al menos 3 vértices y  $v \in V \implies$  ejecutando DFS en  $v$  y s.p.g supongamos  $v$  es la raíz de algún bloque  $T$  de  $G$ .  $\implies$  si  $w$  es el primer vecino del mismo bloque de  $v$  que visitamos.

Sabemos que dado 3  $x, y, z$  vértices en un bloque, entonces existe una trayectoria  $T_\alpha$  que une a  $x, y$  (S.P.G.) además  $T_\alpha$  no pasa por  $z$ , por tanto el siguiente vértice que visitemos después de  $w$  no saldrá de  $v$ . Además DFS ya no agrega a la pila los vértices previamente visitados. Así, en el árbol DFS de  $G$ ,  $G \cap T$  sólo tendrá los vértices de  $T$  y ahí la raíz tendrá  $d(v) = 1$ .

□

3. (2 puntos) Si  $f$  es la función de tiempo de entrada del algoritmo DFS, defina  $f^*: V \rightarrow \mathbb{N}$  de la siguiente forma. Si algún ancestro propio de  $v$  puede ser alcanzado desde  $v$  mediante una trayectoria dirigida que consista de flechas del árbol (posiblemente ninguna) seguida de una flecha que no está en el árbol (que va hacia arriba),  $f^*(v)$  se define como el menor valor de  $f$  de un ancestro de este tipo; si no,  $f^*(v) = f(v)$ . Observe que un vértice  $v$  es la raíz de un bloque si y sólo si tiene un hijo  $w$  tal que  $f^*(w) \geq f(v)$ . Modifique el algoritmo DFS para que regrese los vértices de corte y los bloques de una gráfica conexa.

Primero describiremos un algoritmo que encuentra los vértices de corte de una gráfica.

---

**Algorithm 1:** DFS-CutVertex

---

```

1 Input: Una gráfica  $G$ , un vértice distinguido  $r$  y el tiempo  $t$  de entrada de vértice en
   cuestión.
2 Output: Una colección  $C$  con los vértices de corte.
3  $i \leftarrow t$ ;
4  $i \leftarrow i + 1$ ;
5  $C \leftarrow []$ ;
6  $f(r) \leftarrow i$ ,  $f^*(r) \leftarrow f(r)$ 
7 while  $r$  tenga vecinos do
8   Sea  $v$  un vecino de  $r$ .
9   if  $v$  no está visitado then
10     $p(v) \leftarrow r$ 
11     $C \leftarrow \text{DFS-CutVertex}(G, v, i)$ ;
12    if  $r$  no tiene padre, y tiene mas de dos hijos then
13      añadir  $r$  a la colección  $C$ ;
14    end
15    else
16       $f^*(r) \leftarrow \min(f^*(r), f^*(v))$ ;
17      if  $f^*(v) \geq f(r)$  then
18        añadir  $r$  a la colección  $C$ .
19      end
20    end
21  end
22  else if  $v$  no es el padre de  $r$  y  $f(v) < f(r)$  then
23     $f^*(r) \leftarrow \min(f^*(r), f(v))$ 
24  end
25 end
26  $i \leftarrow i + 1$ ;
27  $l(r) \leftarrow i$ ;
28 return  $C$ ;

```

---

Nótese que el tiempo  $t$  en la primer ejecución es igual a 0, como el código es recursivo, eventualmente nos servirá introducir un nuevo tiempo que corresponda a la entrada del vértice en cuestión a la pila. Una vez teniendo el algoritmo para obtener los vértices de corte de una gráfica, podemos usarlo en un nuevo algoritmo para encontrar tanto los vértices de corte, junto con los bloques en una gráfica  $G$  conexa.

**Algorithm 2:** DFS-CutVertexBlock.

---

```

1 Input: Una gráfica  $G$ , un vértice distinguido  $r$  y el tiempo  $t$  de entrada de vértice en
   cuestión.
2 Output: Una colección  $C$  de vértices de corte y un conjunto  $B$  de bloques de la gráfica  $G$ .
3  $S \leftarrow []$ ;  $i \leftarrow t$ ;
4  $i \leftarrow i + 1$ ;
5  $C \leftarrow []$ ;  $B \leftarrow \emptyset$ ;
6 marcar a  $r$  como visitado.
7  $f(r) \leftarrow i$ ,  $f^*(r) \leftarrow f(r)$ ;
8 while  $r$  tenga vecinos do
9   Sea  $v$  un vecino de  $r$ .
10  if  $v$  no ha sido visitado then
11     $p(v) \leftarrow r$ ;
12    metemos a la arista  $(r, v)$  a  $S$  en el tope;
13     $C \leftarrow \text{DFS-CutVertex}(G, v, i)$ ;
14     $f^*(r) \leftarrow \min(f^*(r), f^*(v))$ ;
15    if  $f^*(v) \geq f(u)$  then
16      Sacamos el tope de la pila y lo insertamos en  $S$ , tantas veces hasta que saquemos
      a  $(u, v)$  y estás serán las aristas de una componente  $b$  y añadimos  $b$  a  $B$ .
17    end
18  end
19  else if  $v$  no es el padre de  $r$  y  $f(v) < f(r)$  then
20    Metemos a la arista  $(r, v)$  en  $S$ ;
21     $f^*(r) \leftarrow \min(f^*(r), f(v))$ ;
22  end
23 end
24  $i \leftarrow i + 1$ ;
25  $l(r) \leftarrow i$ ;
26 return  $C, B$ ;
```

---

Así obtenemos los vértices de corte y los bloques de una gráfica  $G$  conexa.

4. (2 puntos) Sea  $T$  un árbol óptimo en una gráfica conexa ponderada  $(G, w)$  (con pesos positivos), y sean  $x$  y  $y$  vértices adyacentes en  $T$ . Demuestre que la trayectoria  $xTy = xy$  es una  $xy$ -trayectoria de peso mínimo en  $G$ .

**Demostración:** (Demostración por reducción al absurdo)

Sea  $(G, w)$  una gráfica conexa ponderada,  $T$  un árbol óptimo y  $x, y \in V(G, W)$ , supongamos que  $xTy$  no es una trayectoria óptima en  $(G, w) \implies$  Existe  $t_0$  tal que  $t_0$  sea un árbol generador,  $t_0$  sea diferente a  $T \implies$  y la trayectoria óptima sea  $xt_0y \implies t_0$  tiene mayor peso que  $T$ , ya que la trayectoria  $xTy$  es mas pesada  $\implies$  que  $T$  no es un árbol óptimo (Lo que es una contradicción a nuestra hipótesis).

Por lo tanto  $xTy$  es una trayectoria óptima.

□

5. (2 puntos) Demuestre que si todos los pesos de una gráfica ponderada  $G$  son distintos, entonces  $G$  tiene un único árbol óptimo.

**Demostración:** Supongamos a  $G$  una gráfica ponderada con el costo de sus aristas distintos entre sí, sean  $T_1$  y  $T_2$  arboles tales que

$$T_1 \neq T_2$$

y ambos son arboles óptimos en  $G$  (P.D., la unicidad de  $T_1$ ), entonces consideremos a  $e$  una arista de peso mínimo que está contenida en  $T_1$  o  $T_2$  (esto por que las aristas tienen pesos distintos entonces solo habrá una con peso mínimo), llamémosla  $e_1$ , luego s.p.g. que  $e_1$  está contenida en  $T_1$ .

$$\Rightarrow e_1 \notin T_2$$

por lo que si unimos  $e_1 + T_2$  es claro que existirá al menos un ciclo  $C$  el cual contendrá al menos una arista digamos  $e_2$  que no está contenida en  $T_1$  (Esto por que si lo estuviera, entonces  $T_1$  formaría un ciclo con  $e_2$ , pero  $T_1$  es un árbol).

Entonces llegamos a que  $e_1$  y  $e_2$  contenidas en  $T_1$  o  $T_2$  por construcción de las mismas tenemos que

$$w(e_1) < w(e_2)$$

luego podemos notar que se si

$$(T_2 \cup e_1) - e_2$$

es claro que genera la construcción de  $T_1$  por la construcción de  $T_1$  y  $T_2$  implica que

$$W(T_1) < W(T_2)!!$$

$\therefore G$  es un árbol óptimo único.

□

6. (2 puntos) Modifique el algoritmo de Borůvka-Kruskal para que en cada iteración vértices en la misma componente del bosque  $F$  reciban el mismo color y vértices en componentes distintas reciban colores distintos.

A continuación se muestra la modificación del algoritmo de Borůvka-Kruskal:

---

**Algorithm 3:** Borůvka-Kruskal-Coloration

---

```

1 Input:  $G$  una gráfica ponderada
2 Output: Un bosque óptimo  $B = (F, V)$  de  $G$  donde cada árbol tiene una coloración
   distinta.
3 1.  $F \leftarrow \emptyset$ ,  $w(B) \leftarrow \emptyset$ ,  $i \leftarrow 0$ ;
4 2. for  $v \in V_G$  do
5   | 3. asignarle  $i++$  como color a  $v$ .
6 end
7 4. while exista una arista  $e \in E - F$  tal que  $F \cup \{e\}$  induce a un bosque do
8   | 5. Elegir  $e$  de peso mínimo con dicha característica
9   | 6. if los extremos de  $e$  tienen colores diferentes then
10    | 7. Asignamos como el color de ambas, al color mas mínimo entre las dos.
11    | 8.  $F \leftarrow F \cup \{e\}$ ,  $w(B) \leftarrow w(B) + w(e)$ 
12   end
13   .
14 end
15 return  $(F, w(T))$ ;
```

---

7. (2 puntos) Demuestre que el problema de encontrar un árbol generador de peso máximo en una gráfica conexa puede resolverse eligiendo iterativamente una arista de peso máximo, con la condición de que la subgráfica resultante siga siendo un bosque. (Proponga un algoritmo y demuestre que es correcto.)
8. (2 puntos) Escriba una versión del algoritmo BFS para digráficas. Utilice esta versión de BFS dirigida para describir un algoritmo que encuentre un ciclo dirigido de longitud mínima en una digráfica. Su versión dirigida de BFS debe de correr en tiempo  $\mathcal{O}(|V| + |E|)$ , y el algoritmo para encontrar el ciclo dirigido más corto debe correr en tiempo a lo más  $\mathcal{O}(|V|^2 + |V||E|)$ .