MA3705. Algoritmos Combinatoriales. 2014.
Profesor: José Soto
Escriba(s): Matías Alvarado, Felipe Arbulú y Cristóbal Parraguez.
Fecha: 01 de Agosto 2014 .

Cátedra 2

Dado el grafo $G = (V, E)$, se definió previamente:

- *Paseo:* Secuencia de vértices $v_1v_2 \dots v_k$ con $v_iv_{i+1} \in E$, $\forall i \in \{1, 2, \dots, k-1\}$.
- *Paseo Arista-Simple:* Paseo $v_1v_2 \dots v_k$ donde todas las aristas $e_i = v_iv_{i+1}$, $\forall i \in \{1, 2, \dots, k-1\}$ son distintas.
- *Camino:* Es un paseo que no repite vértices. Notar que como no repite vértices, no repite aristas. Luego todo camino es un paseo arista-simple.
- *Ciclo:* Es un paseo $v_1v_2 \dots v_k$ que no repite vértices excepto que $v_1 = v_k$.
- *Grafo Conexo:* Diremos que G es conexo si $\forall u, v \in V$ existe un u-v camino.
- *Grado de un vértice:* Definimos el grado del vértice $v \in V$ como $d(v) = |\delta(v)|$.
- *Vértice Aislado:* $v \in V$ se dice aislado si $d(v) = 0$ i.e., no existen aristas de G incidentes a v.

Hasta ahora cada paseo lo hemos denotado por sus vértices. También podemos denotar cada paseo (y por ende, cada camino y ciclo) por la secuencia de aristas como lo muestra el siguiente ejemplo:

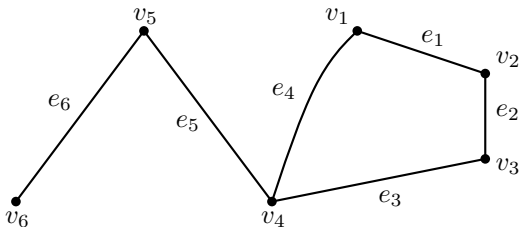

Figura 1: $(\{v_1, v_2, v_3, v_4\}, \{e_1, e_2, e_3, e_4\})$ es un ciclo.

Proposición 1. *Si u-v es un paseo en G, entonces existe un u-v camino en G.*

Demostración. Como sabemos que el grafo es finito y existe un u-v paseo, elijamos el u-v paseo más corto. Este paseo debe ser claramente un u-v camino (¡Verificar!). □

Definición 1. Si $G = (V, E)$, en V se define la relación $\sim_E$ de «estar conectados» como

$$u \sim_E v \iff \text{existe un } u\text{-}v \text{ camino con aristas en } E$$

Se puede verificar que esta relación es de equivalencia.

Definición 2 (Componentes Conexas)**.** Sea $G = (V, E)$ un grafo. Las *componentes conexas* son las clases de equivalencia de la relación $\sim_E$.

Por ejemplo, el siguiente grafo tiene exactamente tres componentes conexas

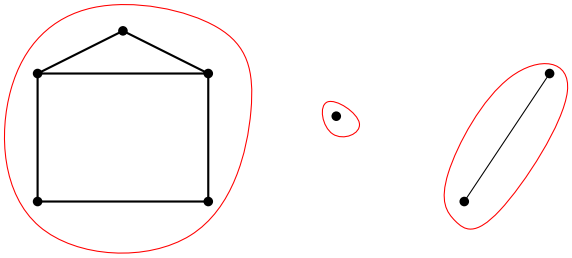

Figura 2: Grafo con tres componentes conexas

Alternativamente: W es componente conexa si $W \subseteq V$ es un conjunto maximal de vértices con $G[W]$ conexo. (¡Verificar!)

Definición 3. Dado $G = (V, E)$ un grafo, definimos $\text{cc}(G)$ como el número de componentes conexas de G.
Si G es conexo, entonces claramente $\text{cc}(G) = 1$.

Lema 1. *Sea $G = (V, E)$ un grafo. Si G es acíclico (i.e., no hay ciclos en G) entonces* $\text{cc}(G) = |V| - |E|$.

Demostración. Por inducción sobre $|E|$.
Si $|E| = 0$, entonces $\text{cc}(G) = |V|$ (cada vértice es componente conexa) y se cumple la proposición.
Si $|E| \geq 1$, sea $e = uv$ una arista y K la componente conexa que contiene a u y v. Al borrar e, K debe dividirse en dos componentes conexas, pues de lo contrario en $G - e = (V, E - e)$ existe un u-v camino P, pero $P + e$ es un ciclo, lo que es una contradicción (en rigor debieramos verificar que K no se divide en mas de dos componentes conexas, pero esto es directo de que todo vertice sigue conectado a u o a v). De esto se deduce que $\text{cc}(G) = cc(G - e) - 1$, y por la hipótesis inductiva, $\text{cc}(G) = |V| - (|E| - 1) - 1 = |V| - |E|$.
Por inducción, hemos demostrado lo que se quería. □

Corolario 1. *Si G es un grafo acíclico y conexo, tenemos* $1 = \text{cc}(G) = |V| - |E| \Rightarrow |E| = |V| - 1$.

Definición 4 (Árbol)**.** G se dice *árbol* si es acíclico y conexo.

La figura muestra el ejemplo de un árbol

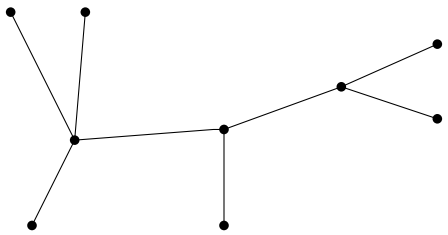

Figura 3: Árbol

Definición 5 (Bosque)**.** Si G es acíclico se dice *bosque*. Claramente un bosque debe componerse de uno o más árboles de vértices disjuntos entre sí.

Definición 6 (Hoja)**.** Un vértice v se dice *hoja* si $d(v) = 1$.

Lema 2. *Si T es árbol con $n = |V| \geq 2$ entonces T tiene al menos dos hojas.*

Demostración. Sea $uw_1w_2 \dots w_kv$ el camino maximal en T para la inclusión. Si $d(u) \geq 2$, entonces existe $w \in V$ adyacente a u distinto de w_1. Si w es alguno de $w_2, \dots w_k, v$ entonces existe un ciclo en T, lo que es una contradicción. Entonces $wuw_1w_2 \dots w_kv$ es un camino más largo que $uw_1w_2 \dots w_kv$, lo que contradice nuestro supuesto inicial. Luego $d(u) = 1$, análogamente $d(v) = 1$ y así u, v son hojas de T. □

Definición 7. Sea $G = (V, E)$ grafo, $e \in \binom{V}{2}$, $F \subseteq E$. Decimos que F «*genera e*» si existe un camino con aristas en F que conecte los extremos de e. También decimos que $G' = (V, F)$ genera e.

Por ejemplo, en la figura, F genera e.

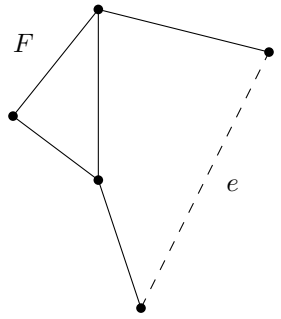

Figura 4: En este caso $e \notin E$ pero F genera e

Finalmente, si H es un subgrafo de G, decimos que H genera G, si H genera todas las aristas de G.

Problema 1 (Problema del subgrafo generador de costo mínimo)**.** Dado $G = (V, E)$ y una función de costo $c : E \to \mathbb{R}_+$, encontrar un subgrafo (V, F) de G que genere G a costo mínimo.

Lema 3. *$e = uv \in E$ es una arista de un ciclo C de $G \iff G - e$ genera e.*

Demostración.

- $\Rightarrow$ Sea $C = uw_1w_2 \cdots w_kvu$ un ciclo en G, tomando $P = uw_1w_2 \cdots w_kv = C - e$, P es un u-v camino en $G - e$; luego como $P \subseteq E - e \Rightarrow E - e$ genera e, en otras palabras $G - e$ genera e.
- $\Leftarrow$ Sea P un u-v camino en $G - e \Rightarrow P + e = C$ es un ciclo en G que contiene a e.

□

Corolario 2. *Si $G' \subseteq G$ es un subgrafo de G tal que genera G y e pertenece a un ciclo de $G' \Rightarrow G' - e$ genera G.*

Demostración. $G' - e$ genera G' y G' genera G, por lo tanto, $G' - e$ genera G.

□

Como consecuencia del hecho que $c \geq 0$ tenemos que el «*Problema del subgrafo generador de costo mínimo*» admite soluciones que son bosques. En el caso en que G es conexo, el problema también se conoce como árbol generador de costo minimo o árbol cubridor.

Definición 8 (Algoritmo)**.** Secuencia finita y ordenada de instrucciones para resolver un problema.

Por ejemplo, veamos un algoritmo genérico para encontrar un árbol generador.

Dado $G = (V, E)$, sin función de costo y conexo.

Algoritmo 1: Buscar árbol generador

```
Elegir r ∈ V;
U ← {r}; // Nodos visitados
F ← ∅; // Aristas de solución
while δ(U) ≠ ∅ do
    Elegir e ∈ δ(U), e = uv, u ∈ U, v ∉ U;
    U ← U + v;
    F ← F + e;
end
return (U, F)
```

Para analizar la correctitud del algoritmo probaremos:

1. El algoritmo termina.
2. En cada iteración $T = (U, F)$ es un árbol.
3. Al final T es generador.

Demostración.

1. El algoritmo termina a lo mas en n iteraciones.
2. Inicialmente $(U, F) = (\{r\}, \emptyset)$ que es un árbol. Sea (U, F) el árbol al principio de una iteración.
   En dicha iteración T cambia a $T' = (U + v, F + uv)$. T' es conexo pues $v \sim_{F+e} u$ y $u \sim_F w$; $\forall w \in U$.
   Si T' tuviera un ciclo, este ciclo C debe usar e, pero $C - e \subseteq T$ es un camino de u a v, lo que es una contradicción, pues T no contiene caminos de u a v. Luego T es un árbol.

   Para probar que G es generador usaremos el siguiente teorema:

   Teorema 1 (Conexidad)**.** *$G = (V, E)$ es conexo* $\iff \forall U \subseteq V; U \neq \emptyset; U \neq V : \delta(U) \neq \emptyset$.

   Demostración.

   $\Rightarrow$ Sea $U \neq \emptyset$ con $U \subset V$, consideremos $u \in U$ y $v \notin U$; como G es conexo existe un u-v camino en G, digamos P.

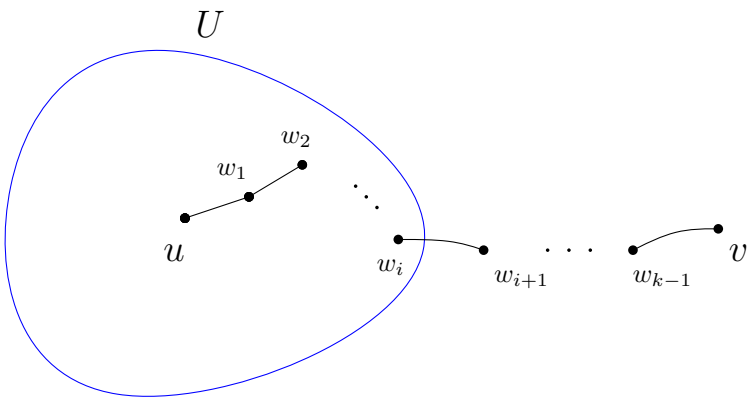

Figura 5: Camino P.

   $P = uw_1...w_{k-1}v$, con $w_0 = u \in U$ y $w_k = v \notin U$. Sea w_i el último vértice tal que $w_0w_1...w_i$ estan todos en U. Luego $w_{i+1} \notin U$; de donde $w_iw_{i+1} \in \delta(U)$.

   $\Leftarrow$ Por contradicción. Si G no es conexo, sea U una componente conexa de G, luego $\emptyset \neq U \neq V$, entonces $\delta(U) = \emptyset$, lo que es una contradicción.

□

3. Al final del algoritmo, sabemos que $T = (U, F)$ es un árbol y se sale del **while**, luego $\delta(U) = \emptyset$; como G es conexo, debe ser $U = \emptyset$ o $U = V$. Pero $U \neq \emptyset$ pues $r \in U$. Se concluye que $U = V$. Asi T es un árbol que contiene todos los vértices de V. Luego, hay un camino entre cada par de vértices de G. Es decir, T genera G

□

Casos especiales del algoritmo genérico

Algoritmo 2: BFS (Breadth First Search o Búsqueda en amplitud)

Elegir $r \in V$;
$U \leftarrow \{r\}$; `// Nodos visitados`
$F \leftarrow \varnothing$;
$Aux \leftarrow \varnothing$;
Agregar al final de Aux todas las aristas de $\delta(r)$;
while $Aux \neq \varnothing$ **do**
  Extraer PRIMERA arista $e = vw$ de Aux;
  if *e contiene un extremo no visitado, digamos* $w \notin U$ **then**
    $U \leftarrow U + w$;
    $F \leftarrow F + e$;
    Agregar al final de Aux todas las aristas de $\delta(w)$;
  end
end
return $T = (U, F)$

Algoritmo 3: DFS (Depth First Search o Búsqueda en profundidad)

Elegir $r \in V$;
$U \leftarrow \{r\}$; `// Nodos visitados`
$F \leftarrow \varnothing$;
$Aux \leftarrow \varnothing$;
Agregar al final de Aux todas las aristas de $\delta(r)$;
while $Aux \neq \varnothing$ **do**
  Extraer ÚLTIMA arista $e = vw$ de Aux;
  if *e contiene un extremo no visitado, digamos* $w \notin U$ **then**
    $U \leftarrow U + w$;
    $F \leftarrow F + e$;
    Agregar al final de Aux todas las aristas de $\delta(w)$;
  end
end
return $T = (U, F)$