**MA3705. Algoritmos Combinatoriales. 2014.**
**Profesor:** José Soto
**Escriba(s):** Enzo Aljovin, Christopher Cabezas y Valentina Toro.
**Fecha:** 04 de Agosto 2014 .

# Cátedra 3

## 1. Previo

En la clase pasada se planteó el siguiente problema:

(**Árbol cobertor de costo mínimo**) Dado un grafo $G = (V, E)$ y una función peso $c : E \to \mathbb{R}_+$, encontrar un subgrafo generador de costo mínimo.

Una primera manera de abordarlo fue considerando las hipótesis extras de que no existía una función de costos (o equivalentemente, una función de costo que asigna a todas las aristas el mismo peso), y que el grafo $G$ era conexo. Para este caso, se mostroó un algoritmo genérico y se dieron dos casos especiales, que analizaremos durante el curso. Éstos son:

**Algoritmo 1:** BFS (Breath First Search o Búsqueda en amplitud)

```
Elegir r ∈ V;
U ← {r}; // Nodos visitados
F ← ∅; // Aristas de solución
Aux ← ∅;
Agregar al final de Aux todas las aristas de δ(r);
while Aux ≠ ∅ do
    Extraer PRIMERA arista e = vw de Aux;
    if e contiene un extremo no visitado, digamos
    w ∉ U then
        U ← U + w;
        F ← F + e;
        Agregar al final de Aux todas las aristas de
        δ(w);
    end
end
return T = (U, F)
```

**Algoritmo 2:** DFS (Depth First Search o Búsqueda en amplitud)

```
Elegir r ∈ V;
U ← {r}; // Nodos visitados
F ← ∅; // Aristas de solución
Aux ← ∅;
Agregar al final de Aux todas las aristas de δ(r);
while Aux ≠ ∅ do
    Extraer ÚLTIMA arista e = vw de Aux;
    if e contiene un extremo no visitado, digamos
    w ∉ U then
        U ← U + w;
        F ← F + e;
        Agregar al final de Aux todas las aristas de
        δ(w);
    end
end
return T = (U, F)
```

**Observación 1.**

- *En el árbol obtenido a partir de* BFS*, cada vértice está unido a la raíz por un camino de largo mínimo. Esto fue probado en la primera clase auxiliar.*
- *El algoritmo* DFS *trata de llegar a una hoja lo más rápido posible.*

Una manera de familiarizarse con dichos algoritmos, es visualizar paso a paso la ejecución de éstos con un grafo en particular. Para este propósito, en la sección **Anexo** se desarrollan ambos algoritmos utilizando el grafo mostrado en la figura 1 (Ejemplo de **BFS** - figura 2; ejemplo de **DFS** - figura 3).

En general, la decisión de optar por uno de estos algoritmos vendrá dada por el problema en que se quiera aplicar y/o resolver, o las características que

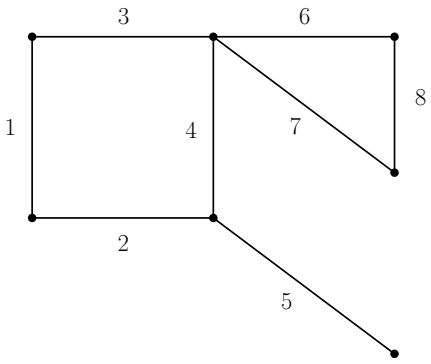



Figura 1: Grafo de ejemplo

busquemos en el árbol resultante. Es importante mencionar que en términos
de complejidad ambos algoritmos son prácticamente iguales. Pero donde sí hay que tener cuidado es en el tipo de estructura de dato con que se implementa cada algoritmo, puesto que esto podría causar una diferencia no menor en el análisis de complejidad. Sin embargo, este análisis queda para un curso de estructura de datos y/o complejidad computacional y escapa a los contenidos del curso.

## 2. Complejidad de un Algoritmo

Al momento de analizar y comparar algoritmos, surgen interrogantes naturales como: ¿Qué algoritmo es más rápido? ¿Cuál realiza menor cantidad de operaciones? Son estas mismas preguntas, las que motivan estudiar la complejidad de un algoritmo y, de esta manera, poder establecer cuan "rápido" es un algoritmo.

**Definición 1** (Tiempo(ALG, ENTRADA))**.** Corresponde al número de operaciones básicas que realiza ALG en ENTRADA, antes de terminar.

Al ver esta definición, nos preguntamos qué se considera una operación básica. En general, dependerá del modelo que se esté utilizando. Sin embargo, aquí consideraremos que las siguientes operaciones corresponden a operaciones básicas:

- Sumar y restar
- Multiplicar y dividir*
- Comparar
- Acceder
- Copiar

Nos interesaremos en operaciones que dependan de un parámetro, por ejemplo "$n$ sumas" o "$n^2$ sumas". Un caso que no consideraremos es "4 sumas".

Hay que considerar además que una **entrada** se puede codificar como cierto número de bits por registro.

**Definición 2** (Complejidad de un algoritmo)**.** Sea ALG un algoritmo. Consideramos su complejidad como la función

$$f(N) = \text{máx } \{Tiempo(ALG, ENT) \mid N(ENT) = N\}$$

donde $N(ENT)$ corresponde al tamaño de la entrada $ENT$.

**Observación 2.** *En general, dado un algoritmo, no nos preocuparemos de calcular de manera explícita su función complejidad, sino que nos interesará ver cómo crece y su comportamiento de manera asintótica.*

**Ejemplo 1.** *Supongamos que se tiene un algoritmo ALG, cuya función de complejidad es*

$$T(N) = 30N^5 + 12N + 10\lfloor \log N \rfloor.$$

*En este caso se tendrá que $T = \mathcal{O}(N^5)$. Esto significa que $T$ "no crece" más rápido que $N^5$.*

En el caso de algoritmos de grafos, se prefiere dejar la complejidad expresada en términos de parámetros del grafo (como por ejemplo, la cantidad de vértices y/o aristas).

## 2.1. Análisis de los algoritmos DFS/BFS

### 2.1.1. Representaciones de un grafo

Para realizar el análisis, primero veamos cómo se representa un grafo. En general, existen dos maneras clásicas de codificar un grafo.

- **Matriz de Adyacencia**
  En este caso, dado un grafo $G = (V, E)$, se ordenan los vértices y se construye la matriz cuadrada $A$ tal que si $v, w \in V$

$$A_{vw} = \begin{cases} 1 & \text{si} \quad vw \in E \\ 0 & \text{si} \quad vw \notin E \end{cases}$$

  Es fácil notar que la matriz $A$, en el caso de un grafo no dirigido,será simétrica.

- **Lista de Adyancencia**
  A cada vértice se le asocia una lista. Los elementos de esta lista pueden corresponder a los vecinos del vértice o a las aristas incidentes.

Ambas formas para escribir grafos poseen ventajas y desventajas. Por ejemplo, en la matriz de adyacencia se detecta inmediatamente si una arista existe o no, pero el precio que se paga por esto es el espacio utilizado (cantidad cuadrática con respecto al número de vértices). En este sentido, la lista de adyacencia es menos eficiente. Por otra parte, en la lista es más fácil ver los vecinos de un vértice.
Para este curso, consideraremos que manejamos ambas formas de representación.

### 2.1.2. Análisis de los algoritmos DFS/BFS

En el siguiente algoritmo aparece el orden de cada una de las acciones que se realizan, donde $|V| = n$ y $|E| = m$:

**Algoritmo 3:** DFS/BFS

```
Elegir r ∈ V;   O(1)
U ← {r};   O(1)
F ← ∅;   O(1)
Agregar al final de Aux todas las aristas de δ(r);   O(1)
while Aux ≠ ∅ do
    Extraer PRIMERA/ÚLTIMA arista e = vw de Aux;   O(1)
    if e contiene un extremo no visitado, digamos w ∉ U   O(1)
    then
        U ← U + w;
        F ← F + e;
        Agregar al final de Aux todas las aristas de δ(w);   O(n)
    end
end
return T = (U, F)
```

Realicemos un primer análisis, considerando que realizamos el "trabajo" dentro del **while** una vez por cada arista del grafo. Se obtiene que los algoritmos BFS/DFS son de orden $\mathcal{O}(mn)$. Si consideramos además que en la clase anterior se obtuvo un resultado para grafos conexos, que establecía que en un grafo conexo acíclico (i.e un árbol) $m = n - 1$, obtenemos para estos algoritmos una cota inferior de $m^2$ (en el caso del árbol). Por ejemplo, si tuviéramos un grafo con 20 nodos y 20 aristas, y luego se aumentara a 40 el número de aristas, el algoritmo se demoraría 4 veces más que en el anterior, pero en la realidad estos algoritmos son bastante más rápidos. De esta manera queda plasmado que al realizar estimaciones poco finas nos puede llevar a cotas no muy útiles ni cercanas a la realidad.

Siendo un poco más precavido y fino a la hora de analizar, notamos que donde se realiza "trabajo" es cada vez que una arista entra o sale de $Aux$. Por otra parte, se tiene que cada una de ellas puede entrar o salir a lo más 2 veces. Por lo

tanto, se realiza una cantidad constante de trabajo cada vez que una arista entra o sale de $Aux$. Esto nos lleva a que en el peor de los casos se tendrá que el algoritmo estará dominado por $m$.

En estricto rigor, se debería considerar en el algoritmo el hecho que al recibir una entrada se debe chequear que en efecto corresponde a un grafo. Dado esto, se tendrá que al menos se deberá "leer" una vez cada vértice del grafo. Esto nos lleva a incluir otro término en la complejidad del algoritmo. En virtud de lo antes mencionado se tiene que los algoritmos BFS/DFS son $\mathcal{O}(n+m)$, donde el término $n$ viene del hecho de leer la entrada y el término $m$ del análisis antes hecho.

## 2.2. Algoritmo de PRIM

**Algoritmo 4:** Algoritmo de Prim (Prim 1957 - Jarník 1930)

Elegir $r \in V$;
$U \leftarrow \{r\}$
$F \leftarrow \varnothing$
**while** $\delta(U) \neq \varnothing$ **do**
  Sea $e = uv \in \delta(U),\ u \in U,\ v \notin U$ tal que $e$ es la arista de menor peso en $\delta(U)$
  $U \leftarrow U + w$;
  $F \leftarrow F + e$;
**end**
**return** $T = (U, F)$

Para analizar la correctitud del algoritmo recién expuesto, dado que ya se ha verificado que este algoritmo devuelve un árbol, la demostración se centrará en ver que en efecto este árbol corresponde a uno de costo mínimo. Para ello, se introducirán definiciones y lemas previos, los cuales nos permitirán concluir.

**Definición 3** (Conjunto de aristas extendible)**.** Sea $G = (V, E)$ un grafo conexo y $w : E \to \mathbb{R}_+$ una función peso. Se dirá que un conjunto de aristas $E' \subseteq E$ es *extendible*, si existe un árbol generador de peso mínimo $T = (V, F)$, con $E' \subseteq F$.

**Lema 1.** *Sea $G = (V, E)$ un grafo conexo, $E'$ extendible y $e \in E \setminus E'$, entonces*

$$E' + e \text{ es extendible} \iff \exists U \subseteq V, \text{ con } \varnothing \subsetneq U \subsetneq V : \quad E' \cap \delta(U) = \varnothing, \text{ pero } e \text{ es de peso mínimo en } \delta(U)$$

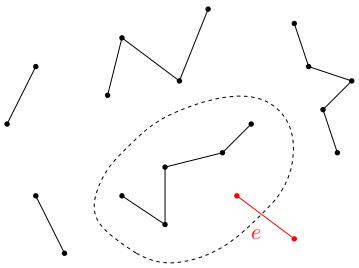
**Demostración** Veamos la doble implicancia.

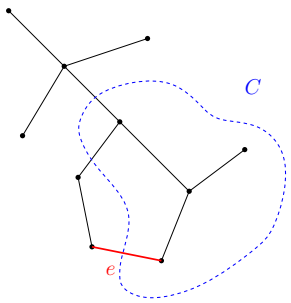
($\Longleftarrow$) Se tiene $E'$ extensible, $e \in E \setminus E'$ y sea $U \subseteq V$ tal que $\varnothing \subsetneq U \subsetneq V$.

Como $E'$ es extendible, se tiene que existe un árbol óptimo $T = (V, F)$ con $E' \subseteq F$. Analicemos dos casos

- $e \in F$:
  En este caso, se tiene directo que $E' + e \subseteq F$ y por tanto, $E' + e$ es extensible.

- $e \notin F$:
  En el caso que $e \notin F$, se tendrá que $F + e$ tendrá un único ciclo $\mathcal{C}$. Como $F$ es conexo (por la propiedad demostrada en la cátedra 2): $\delta(U) \cap F \neq \varnothing$. Sea $f \in \delta(U) \cap \mathcal{C},\ f \neq e$. Considerando $e = uv$ con $u \in U, v \in V \setminus U$ (ya que $e \notin F$). Sea $P$ el camino que une $u$ con $v$ en $F$ (notar que dicho camino es único), de esto sigue que $\mathcal{C} = P + e$. Luego, $P$ contiene arista $f \in \delta(U)$ y además $E' + e \subseteq F$.



($\Longrightarrow$) Si $E' + e$ es extendible, existe $F$ árbol óptimo tal que $E' + e \subseteq F$.

Notemos que $F - e$ desconecta el grafo en dos conjuntos. Sea $U$ uno de ellos. Veamos que $e \in \delta(U)$ es de peso mínimo. Si $\exists\ f \in \delta(U)$ con $w(f) < w(e)$. Luego, el árbol

$$F' = F - e + f$$

es tal que $w(F') < w(F)$, lo que contradice que $F$ sea un árbol óptimo.

■

# 3. Anexo

## 3.1. Ejemplo BFS

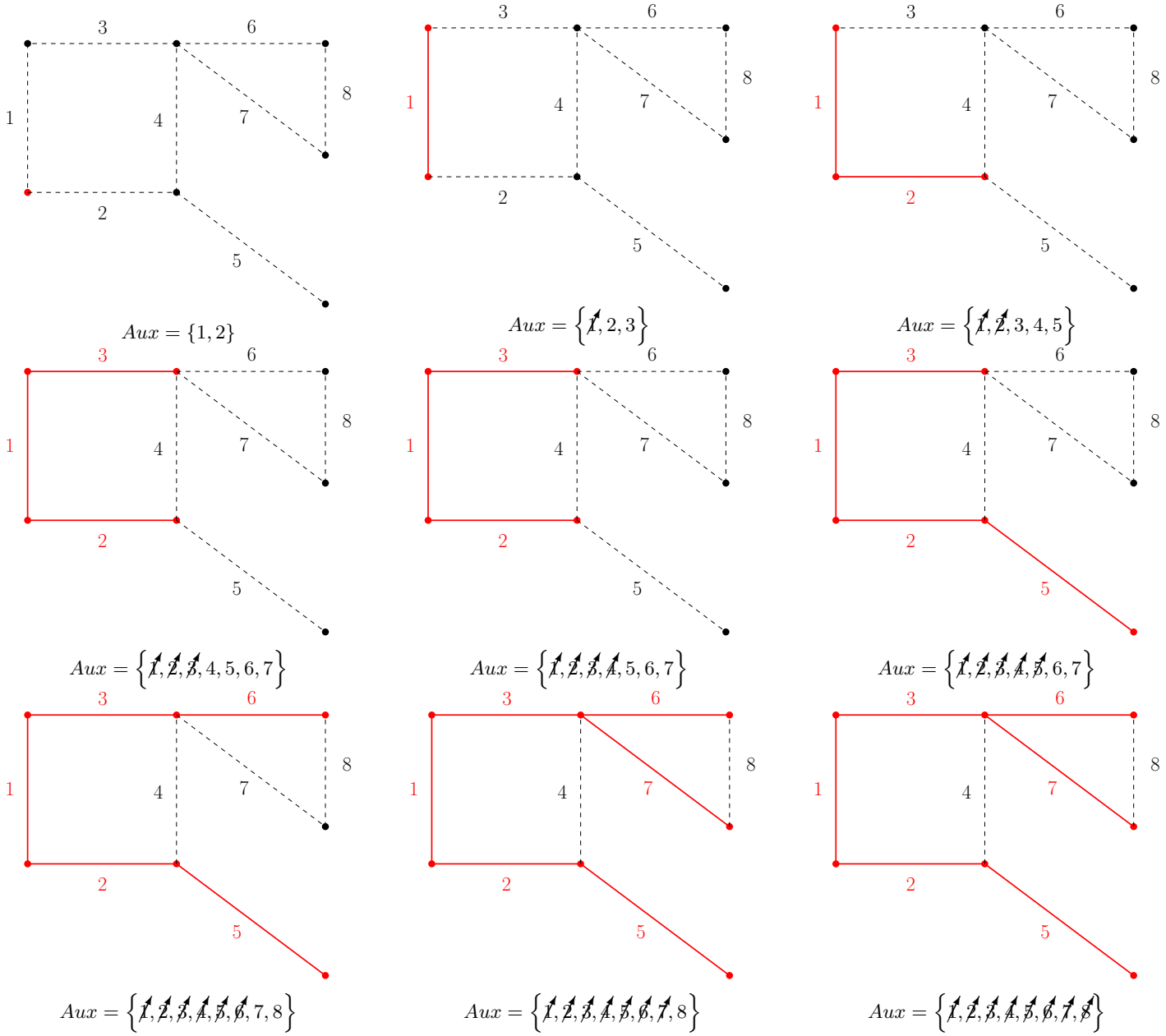



Figura 2: **EJEMPLO BFS**

### 3.2. Ejemplo DFS

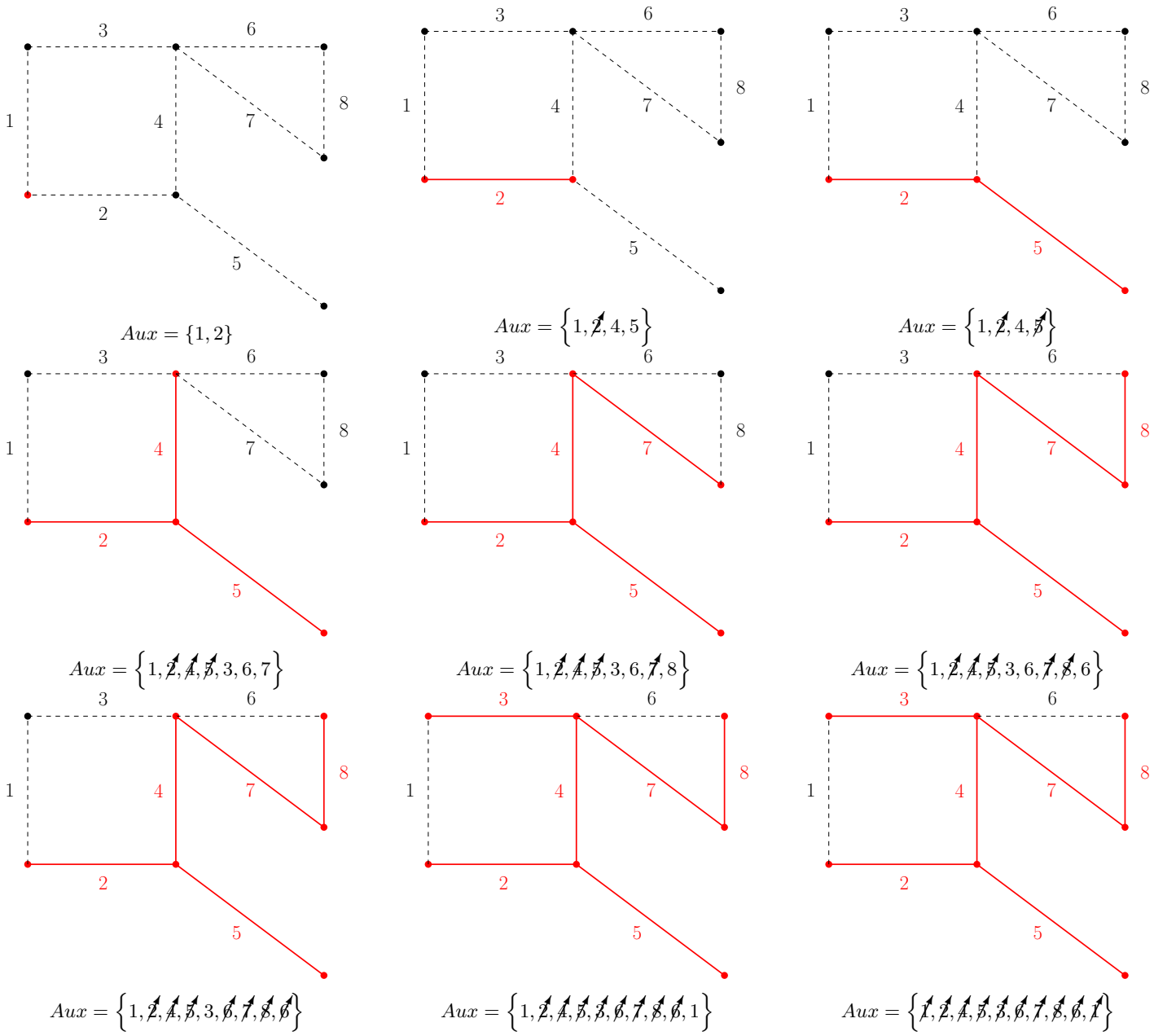



Figura 3: **EJEMPLO DFS**