UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 12 - Unidad 8

Algoritmos de Recorrido.

ESTRUCTURAS DE DATOS (271)

Clase N. 12 Unidad 8.

Clase 12: AGENDA

AGENDA

- **Temario:**
  - Algoritmos de recorrido DFS y BFS.
  - Árbol generador DFS: en grafos dirigidos y no dirigidos.
  - Determinación de componentes conexas y fuertemente conexas.
  - Análisis del tiempo de ejecución de los algoritmos mencionados.

- **Ejemplos en Lenguajes Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

Algoritmo de recorrido DFS

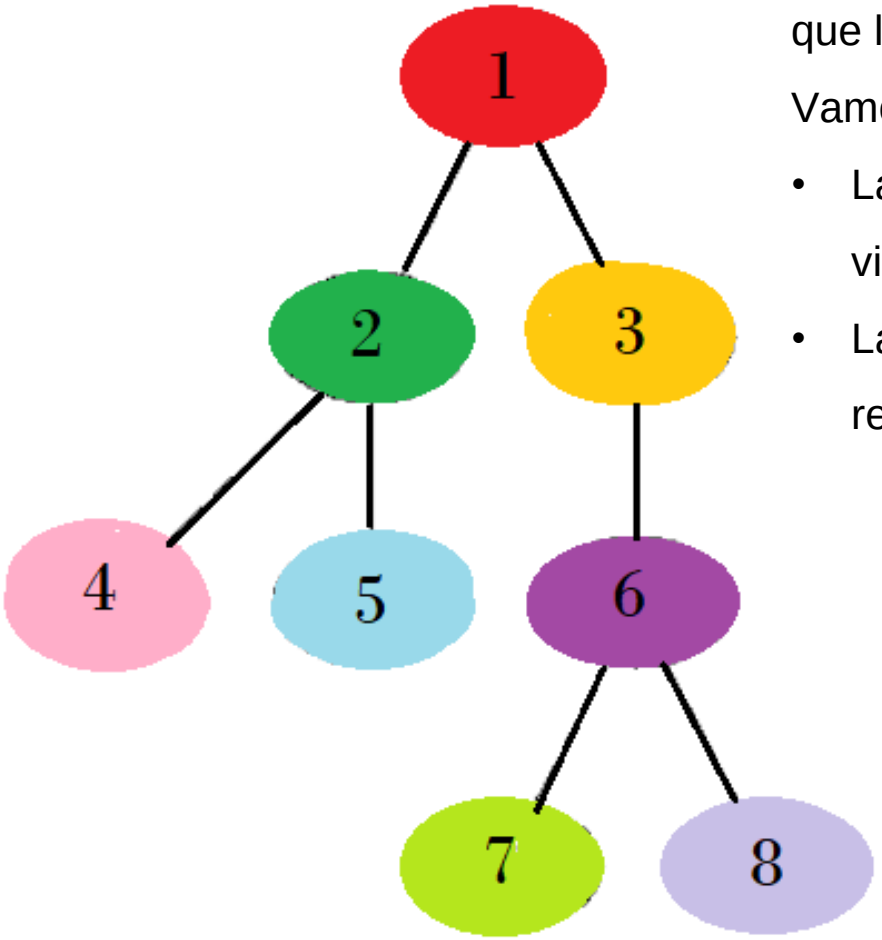
DFS (Depth-First Search, “búsqueda en profundidad”) es una técnica de recorrido de grafos.

La técnica consiste en recorrer el grafo, guardando una lista de “nodos que tenemos que visitar”. Esta lista tiene la particularidad de que el último nodo que agregamos a la lista es el primero que vamos a mirar (por eso “en profundidad”). Vamos a ir agotando los caminos del grafo.

El algoritmo consiste en primero, meter en la lista un nodo inicial. A partir de ahí, agarrar el último nodo que agregamos a la lista, y agregar todos sus vecinos. Vamos a ir marcando los nodos una vez que hayamos procesado a todos sus hijos. Esto es porque supongamos que empezamos desde A, y agregamos sus vecinos C y B. Ahora vamos a sacar de la lista B, y agotar todos los caminos desde éste. Y quizás hay un camino de B a C, y en ese caso queremos recorrer este camino.

Clase 12: Algoritmo DFS

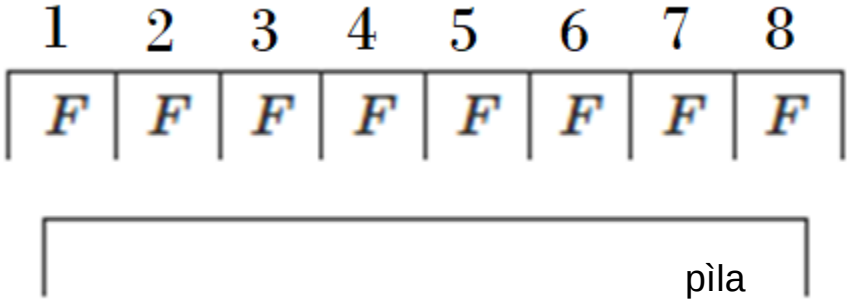
Ejemplo del recorrido:



Supongamos que tenemos el siguiente grafo y que lo vamos a empezar a recorrer desde el **nodo 1**

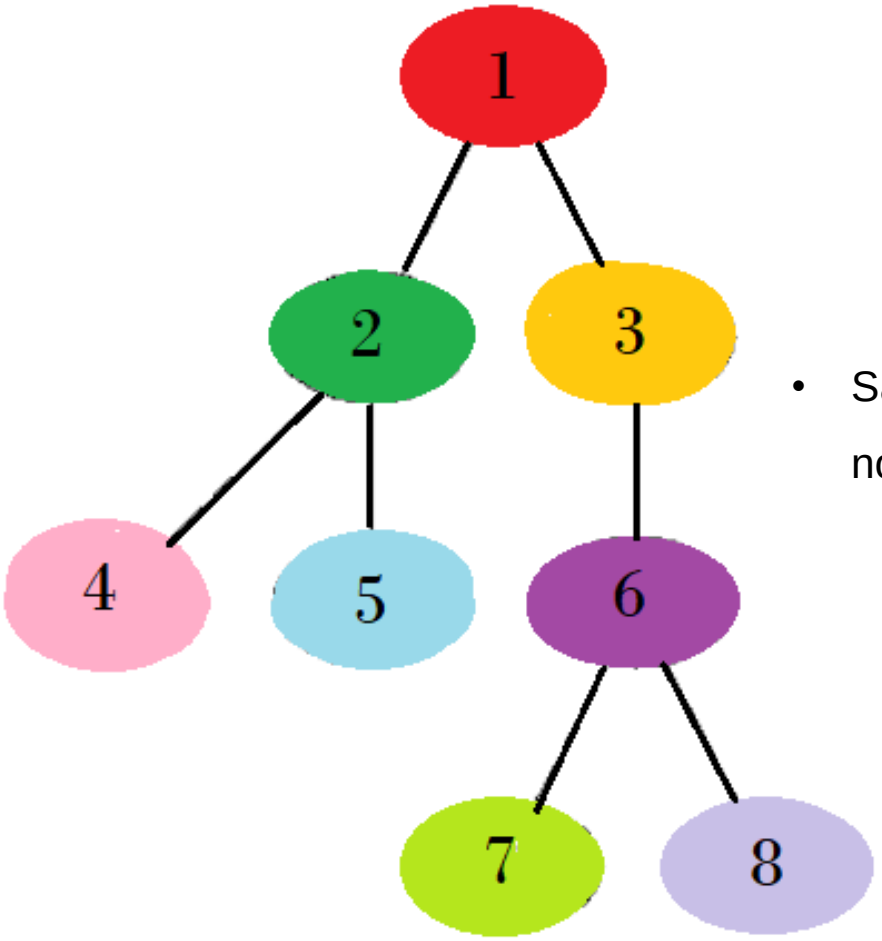
Vamos a usar dos estructuras:

- La primera para representar todos los nodos visitados (marcamos todos en False)
- La segunda para guardar los nodos que vamos recorriendo, esta vacía.



Clase 12: Algoritmo DFS

Ejemplo del recorrido:



- Agregamos al nodo 1.

1	2	3	4	5	6	7	8
F	F	F	F	F	F	F	F

1

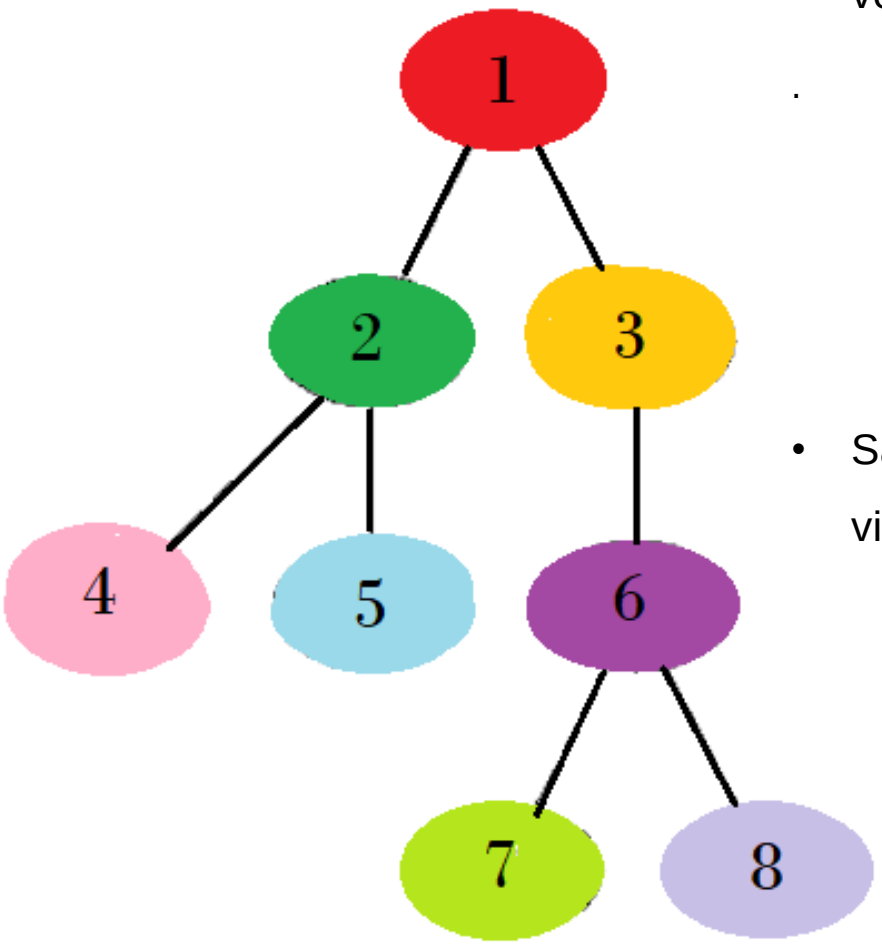
- Sacamos al nodo 1 y agregamos sus vecinos que no fueron visitados, en cualquier orden.

1	2	3	4	5	6	7	8
T	F	F	F	F	F	F	F

2, 3

Clase 12: Algoritmo DFS

Ejemplo del recorrido:



- Sacamos primero al nodo 3, y miramos sus vecinos.

1	2	3	4	5	6	7	8
T	*F*	*T*	*F*	*F*	*F*	*F*	*F*

2, 6

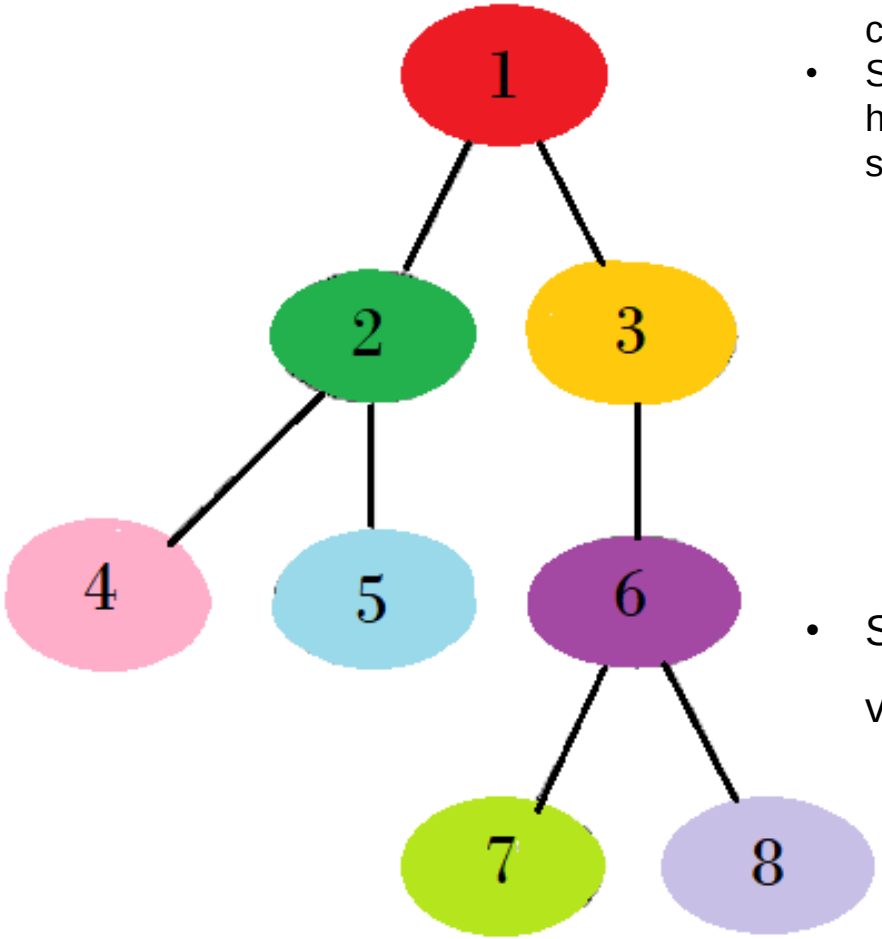
- Sacamos al nodo 6 y miramos sus vecinos no visitados.

1	2	3	4	5	6	7	8
T	*F*	*T*	*F*	*F*	*T*	*F*	*F*

2, 7, 8

Clase 12: Algoritmo DFS

Ejemplo del recorrido:



- Notar que el 2 quedó, pero primero vamos a agotar los caminos hacia abajo del nodo 3.
- Sacamos al nodo 8 y como no tiene vecinos sin visitar, no hacemos nada (más que marcarlo como visitado al sacarlo, como a todos). Lo mismo luego con el nodo 7.

1	2	3	4	5	6	7	8
T	*F*	*T*	*F*	*F*	*T*	*T*	*T*

2

- Sacamos al nodo 2 y miramos sus vecinos no visitados.

1	2	3	4	5	6	7	8
T	*T*	*T*	*F*	*F*	*T*	*T*	*T*

4, 5

- Sacamos al nodo 5 que no tiene nodos por visitar, y lo mismo con el 4, y terminamos.

Clase 12: Algoritmo DFS

Implementación:

DFS puede ser implementado de forma recursiva o iterativa utilizando una pila. La versión recursiva suele ser más corta y fácil de entender, mientras que la versión iterativa puede ser más eficiente en términos de espacio. DFS se utiliza en una variedad de aplicaciones, como detectar ciclos en un grafo, encontrar componentes conexos, comprobar si un grafo es bipartito, y resolver problemas de búsqueda y optimización combinatoria.

Clase 12: Algoritmo DFS

Implementación:

Algoritmo en pseudocodigo:

```
1   método DFS( origen):
2       creamos una pila S
3       agregamos origen a la pila S
4       marcamos origen como visitado
5       mientras S no este vacío:
6           sacamos un elemento de la pila S llamado v
7           para cada vertice w adyacente a v en el Grafo:
8               si w no ah sido visitado:
9                   marcamos como visitado w
10                  insertamos w dentro de la pila S
```

Se ve que cada vértice se lo mira a lo sumo una vez, y en una visita al nodo se lo agrega/saca de la pila, y se hacen operaciones de tiempo constante, por lo que tenemos **O(V)** tiempo para los vértices.
Además, cada arista se ve también a lo sumo una vez, tomando **O(E)** .
En total, la complejidad es **O(V+E),**
que en términos de grafos es óptimo.

Clase 12: Algoritmo DFS

unab

Implementación:

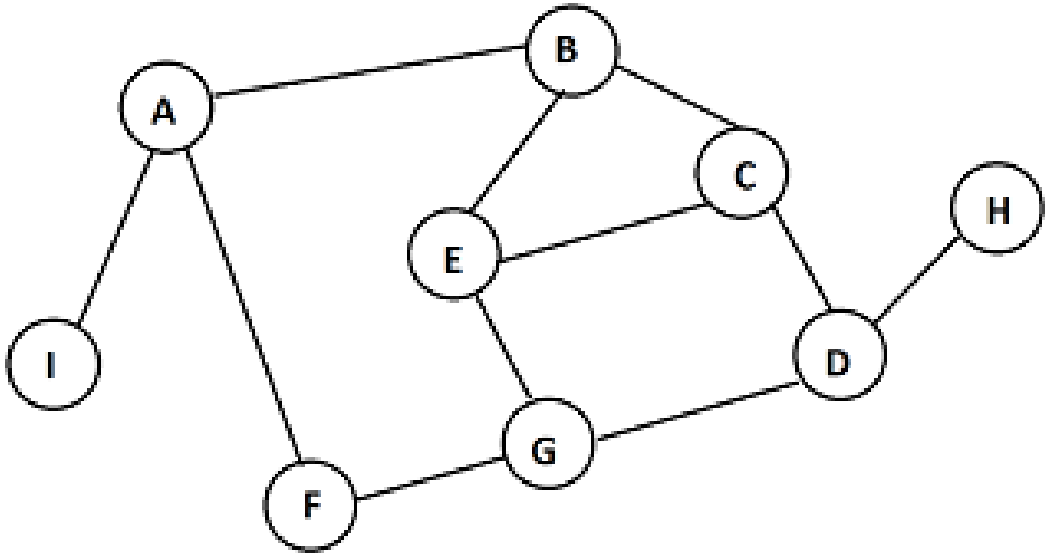
Algoritmo en pseudocódigo:

Usando recursión

```
1  método DFS( origen):
2      marcamos origen como visitado
3          para cada vertice v adyacente a origen en el Grafo:
4              si v no ah sido visitado:
5                  marcamos como visitado v
6                  llamamos recursivamente DFS( v )
```

Algoritmo de recorrido BFS

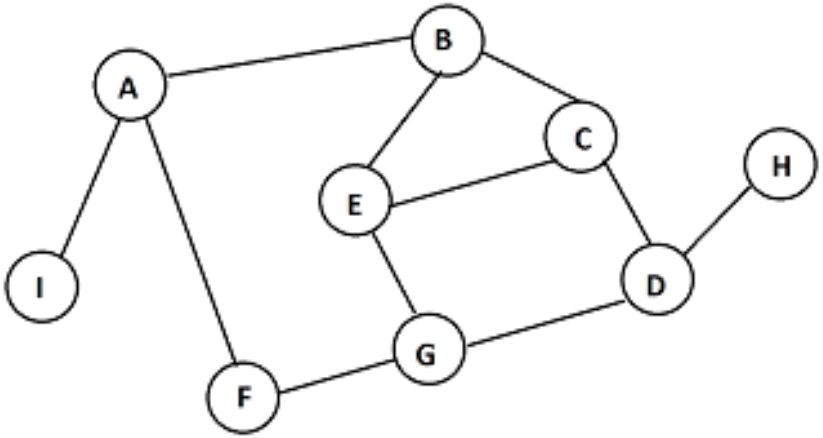
BFS(“Breadth First Search”, “Buscar primero a lo ancho”) es una técnica para **recorrer** un grafo, a partir de **un nodo origen**. Se trata de “ir expandiendo” la exploración de los nodos del grafo, en “todas las direcciones a la vez”. De esta forma, el grafo se irá recorriendo en orden de **distancia** al nodo origen, formándose una suerte de “onda expansiva” a su alrededor. Supongamos que tenemos el siguiente grafo.

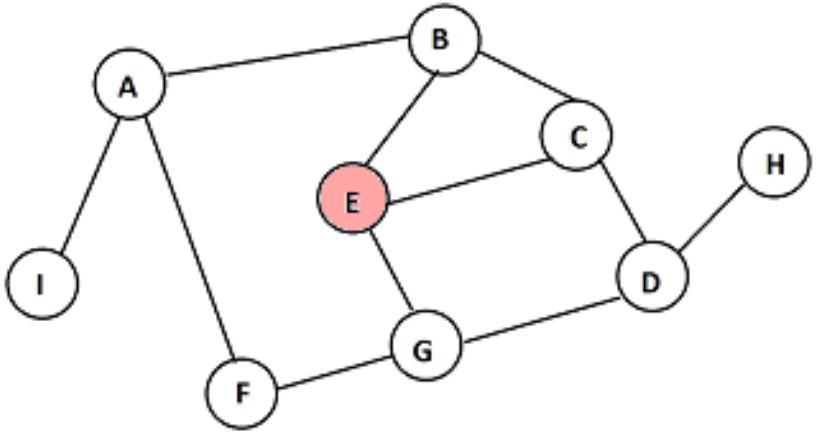


Clase 12: Algoritmo BFS

Ejemplo del recorrido:

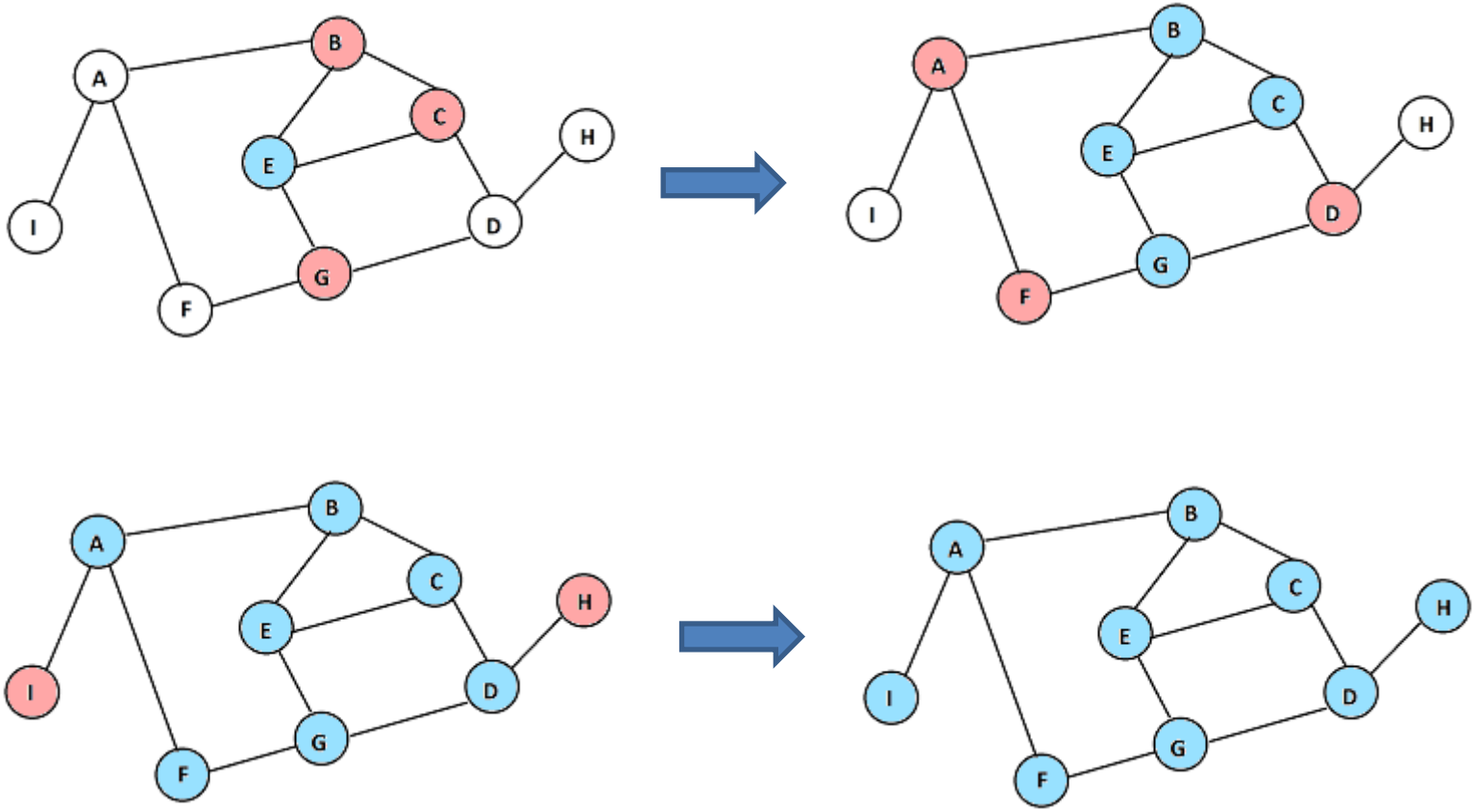
A continuación se muestra el orden en que el algoritmo de BFS irá descubriendo los nodos del grafo, si se comienza una exploración **desde el nodo E**:





Clase 12: Algoritmo BFS

Ejemplo del recorrido:



En cada paso, se muestran en **rosa** los nodos recién descubiertos,
y en **celeste** los que ya habían sido descubiertos en pasos anteriores.

Ejemplo del recorrido:

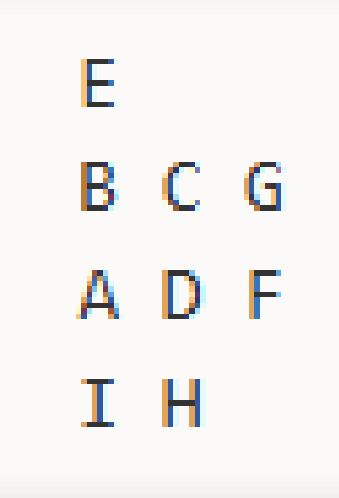
Observando estos ejemplos, es posible caracterizar el mecanismo fundamental del algoritmo de BFS de la siguiente manera:

- Se comienza inicialmente **con el nodo origen, como único nodo rosa** recién descubierto.
- La exploración se expande, **desde los nodos rosa** recién descubiertos, descubriendo nuevos nodos **que no habían sido visitados antes**. Es decir, todos los nodos **blancos** (que nunca habían sido vistos) que resultan ser vecinos de los nodos rosa, **serán los rosa del próximo paso**.
- El paso anterior se repite una y otra vez descubriendo más y más nodos en orden, hasta que en algún paso ya no se descubran nodos nuevos. Es decir, **se termina el algoritmo cuando ya no hay ningún nodo rosa**.

Clase 12: Algoritmo BFS

Ejemplo del recorrido:

El algoritmo BFS garantiza que los vértices se visiten en orden de distancia desde el vértice origen. Por lo tanto, si existe un camino más corto entre el vértice origen y cualquier otro vértice del grafo, BFS lo encontrará. Además, BFS es un algoritmo eficiente, con una complejidad de tiempo O(|V|+|E|) donde |V| es el numero de vértices y |E| el número de aristas en una lista de adyacencias.



Si ejecutamos el algoritmo anterior sobre el grafo del ejemplo, empezando con el nodo marcado como E , y hacemos que el programa muestre cuáles son los valores contenidos en rosas actuales en cada paso, se obtendría una salida como la siguiente (un paso por línea)

Clase 12: Algoritmo BFS

Ejemplo del recorrido:

El algoritmo BFS sigue estos pasos generales:

1. Inicializar: Selecciona un vértice origen y crea una Cola vacía. Marca el vértice origen como encolado y lo encola.
2. Explorar: Mientras la cola no esté vacía, realiza los siguientes pasos:

   a. Desencola el siguiente vértice en la cola (llamémosle 'v').

   b. Procesa el vértice 'v' (por ejemplo, imprímelo, guárdalo en una lista de vértices visitados, etc.).

   c. Encuentra todos los vértices adyacentes a 'v' que no han sido encolados y los marca como encolados. A continuación, encola estos vértices en la cola.
3. Finalizar: Cuando la cola esté vacía, el algoritmo BFS habrá visitado todos los vértices alcanzables desde el vértice origen.

Clase 12: Algoritmo BFS

unab

Implementación:

Algoritmo en pseudocódigo

```
1   método BFS(Grafo,origen):
2       creamos una cola Q
3       agregamos origen a la cola Q
4       marcamos origen como visitado
5       mientras Q no este vacío:
6           sacamos un elemento de la cola Q llamado v
7           para cada vertice w adyacente a v en el Grafo:
8               si w no ah sido visitado:
9                   marcamos como visitado w
10                  insertamos w dentro de la cola Q
```

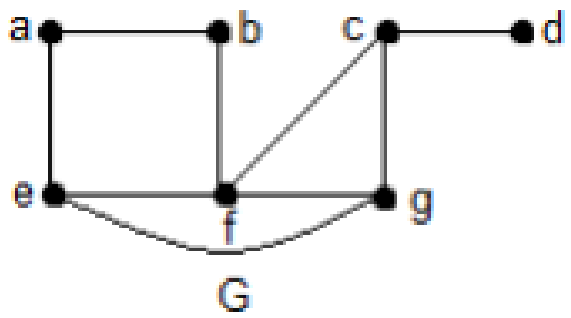
Árbol generador

- Un **árbol** es un grafo no dirigido, conexo y sin ciclos (grafo simple).
- Sea G un **grafo** simple. Un árbol generador de G es un subgrafo de G que es un árbol y contiene todos los vértices de G,
- Un **grafo** simple que admite un **árbol generador** es necesariamente conexo, ya que existe un camino en el árbol generador entre dos vértices cualesquiera. El reciproco también es válido, es decir, todo grafo simple conexo tiene un árbol generador.

Árbol generador

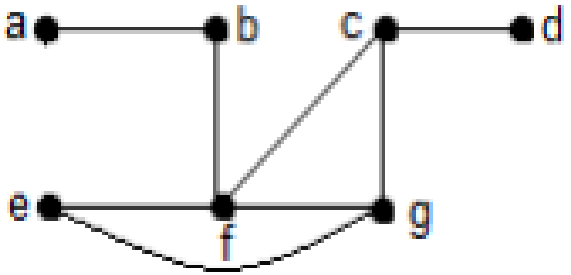
Obtener un árbol generador del siguiente grafo simple G:

El grafo G es conexo, pero no es un árbol porque contiene ciclos.

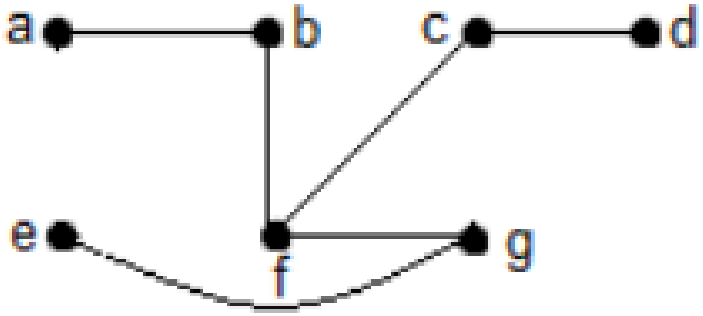


G

Eliminamos el ciclo (a,e)
Todavía hay ciclos.

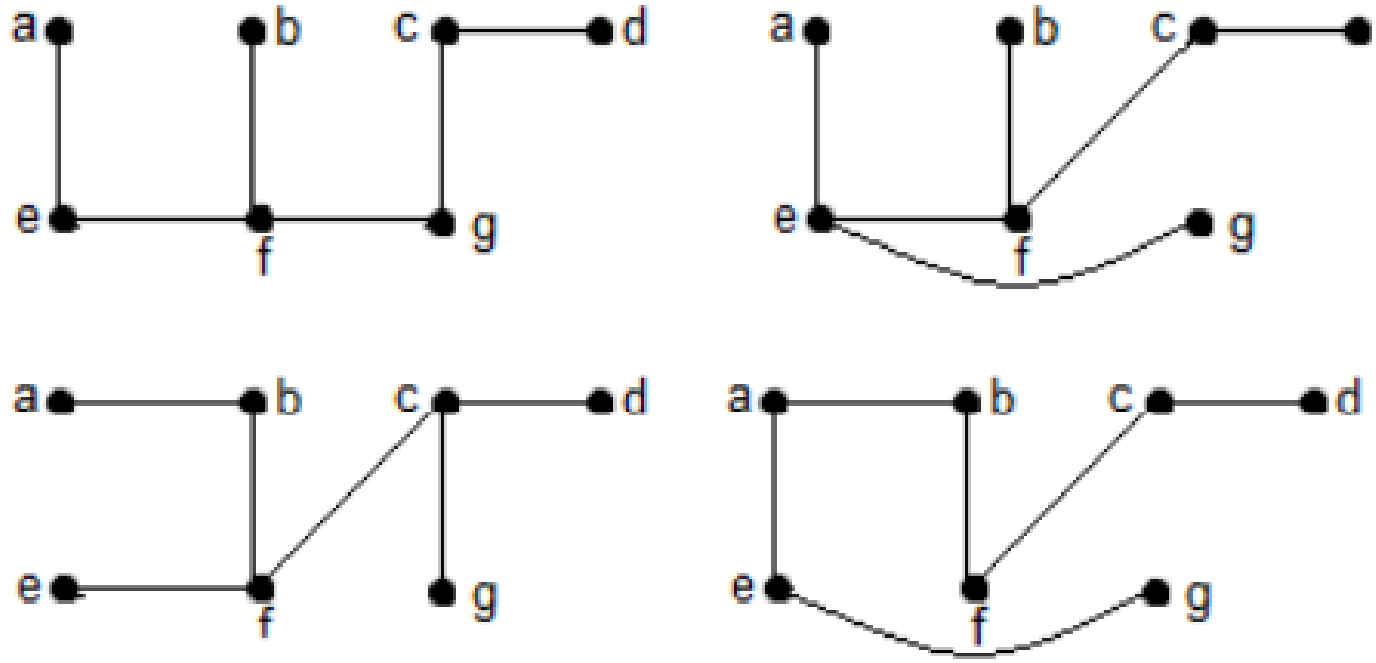


Eliminamos la arista $\{e, f\}$ y la arista $\{c, g\}$ para obtener un grafo simple y sin ciclos.



Árbol generador

Este no es el único árbol generador de G. Cada uno de los siguientes árboles también es un árbol generador de G.

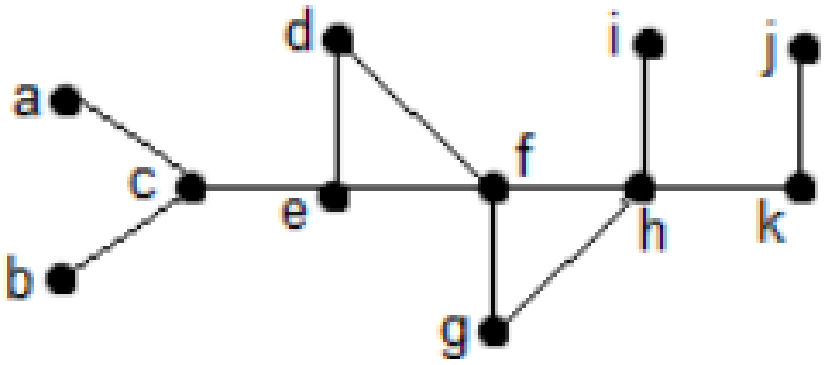


teorema de Cayley *"un mismo grafo puede tener hasta* n^{n-2} *árboles generadores diferentes"* (n es el número de vértice del grafo).

Árbol generador DFS

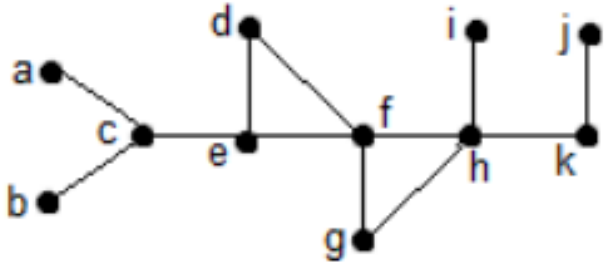
La forma más común de construir árboles generadores tiene que ver con procedimientos de búsqueda, **búsqueda en profundidad(DFS)** y **búsqueda en anchura(BFS)**.

Utilicemos la búsqueda en profundidad para obtener un árbol generador del siguiente grafo G

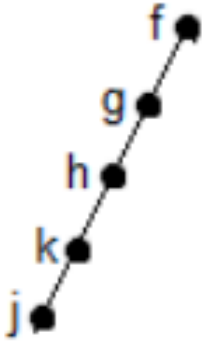


Clase 12: Árbol generador

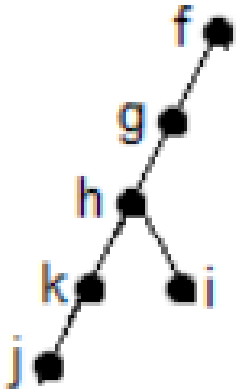
Árbol generador DFS



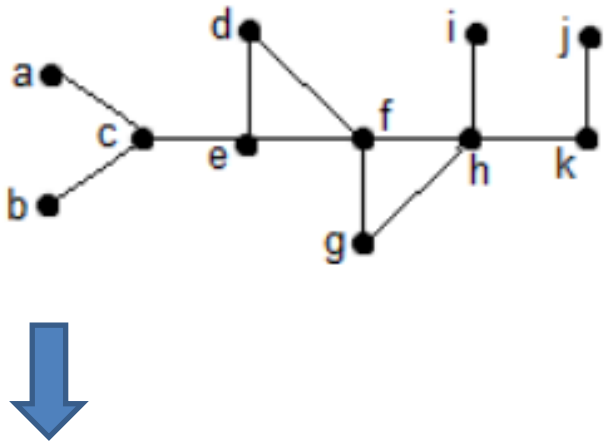
- Elegimos un vértice inicial f.
- Tomamos uno de sus vértices adyacentes, g
- Tomamos los vértices adyacentes de g y elegimos uno, h.
- Tomamos los vértices adyacentes de h y elegimos uno, k.
- Tomamos el vértice adyacentes de k y elegimos j.



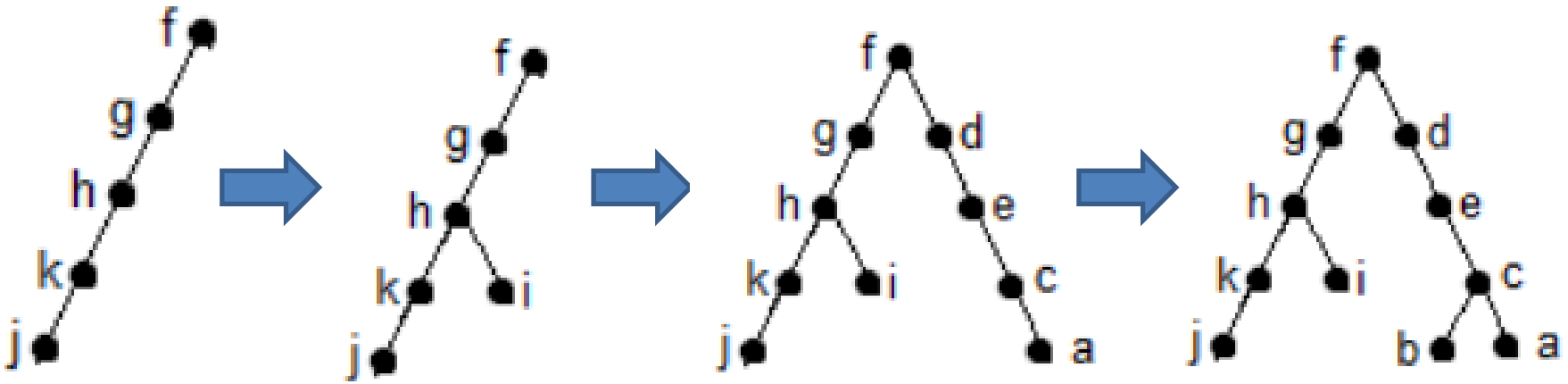
- Retrocedemos, no hay mas vértices para tomar en j.
- Retrocedemos, no hay mas vértices para tomar en k que no hayan sido utilizados.
- Tomamos el vértice adyacente i de h y lo agregamos al árbol.



Árbol generador DFS



- Retrocedemos hasta h, no quedan vértices sin utilizar.
- Retrocedemos hasta g, no quedan vértices sin utilizar.
- Retrocedemos hasta f
- Tomamos uno de sus vértices adyacentes, d.
- Tomamos el vértice adyacentes a d, e.
- Tomamos el vértice adyacentes a e, c.
- Tomamos los vértices adyacentes de c, a
- Retrocedemos hasta c, y tomamos el vértice b.



Árbol generador DFS

ALGORITMO: Busqueda en profundidad

```
def  Busq_Prof (G: grafo conexo de vértices v_1, v_2, . . . v_n )
     T:= árbol que consta sólo del vértice v_1
     visita ( v_1)
def  visita (v: vértice de G)
     for cada vértice w adyacente a v y que no esté en T
          añadir el vértice w y la arista { v, w } a T
          visita (w)
```

G es conexo, todo vértice de G se utiliza en alguna fase del algoritmo y se añade al árbol. De esto se concluye que T es un árbol generador de G

Árbol generador DFS

G (V, E) se representa mediante listas de adyacencia.

*El método **dfs(v)** se aplica únicamente sobre vértices no visitados*
➡ *sólo una vez sobre cada vértice.*

***dfs(v)** depende del número de vértices adyacentes que tenga (longitud de la lista de adyacencia).*

➡ *el tiempo de todas las llamadas a **dfs(v)**: O(|E|)*

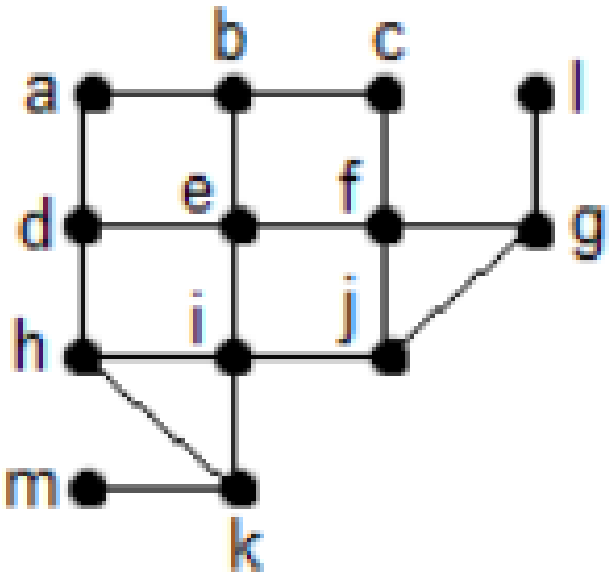
añadir el tiempo asociado al bucle de dfs(grafo): O (|V|)).

➡ *Tiempo del recorrido en profundidad es O(|V|+|E|) .*

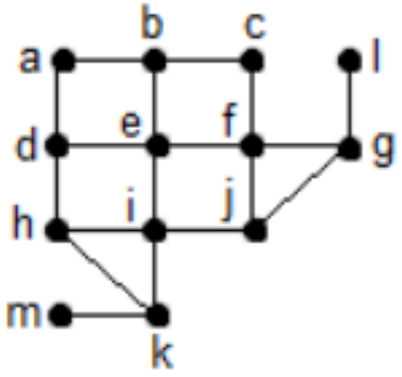
Árbol generador BFS

También se puede obtener un árbol generador de un grafo simple mediante la **búsqueda en anchura** o **por niveles (BFS)**.

Utilicemos la búsqueda por niveles para obtener un árbol generador del siguiente grafo G

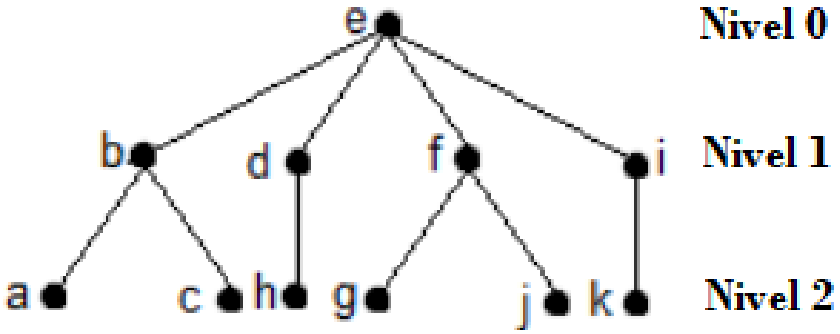


Árbol generador BFS

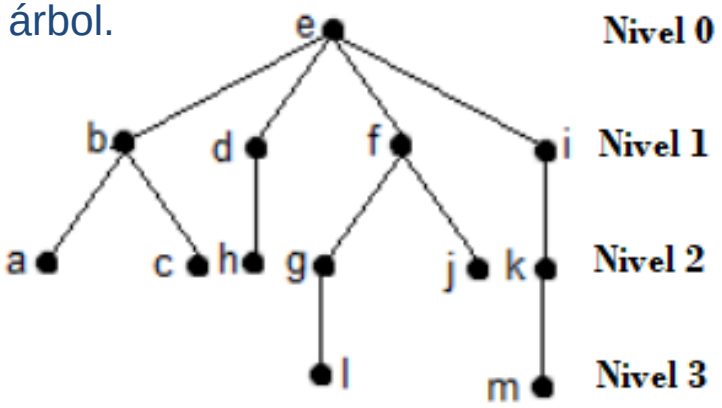


1. Elegimos un vértice como raíz.
2. Agregamos todas las aristas incidentes

3. Se agregan las aristas que conectan los vértices del nivel 1 con vértices que aún no están en el árbol. P



4. Se agregar las aristas que unen los vértices del nivel 2 con vértices adyacentes que no están en aún en el árbol.



Árbol generador BFS

def Busq_Anch (G: grafo conexo de vértices $v_1, v_2, \ldots v_n$)

T:= árbol que consta sólo del vértice v_1

L:= lista vacía

poner v_1 en la lista L de los vértices no procesados

while L no esté vacía

borrar el primer vértice v de L

for cada vértice w adyacente a v

if w no está en L y no está en T **then**

añadir el vértice w al final de la lista L

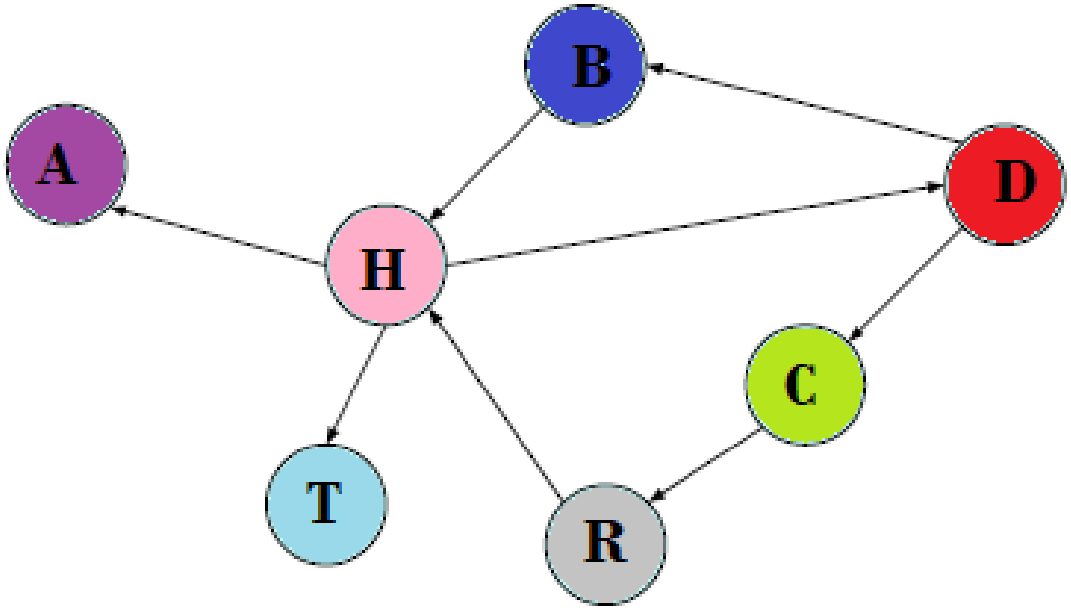
añadir el vértice w y la arista $\{v, w\}$ a T

Consideramos que los vértices del grafo G están ordenados y etiquetados por ejemplo de la siguiente manera: v_1, v_2, . ., v_n

Costo $T(|V|,|E|)$ es de $O(|V|+|E|)$

Clase 12: Árbol generador

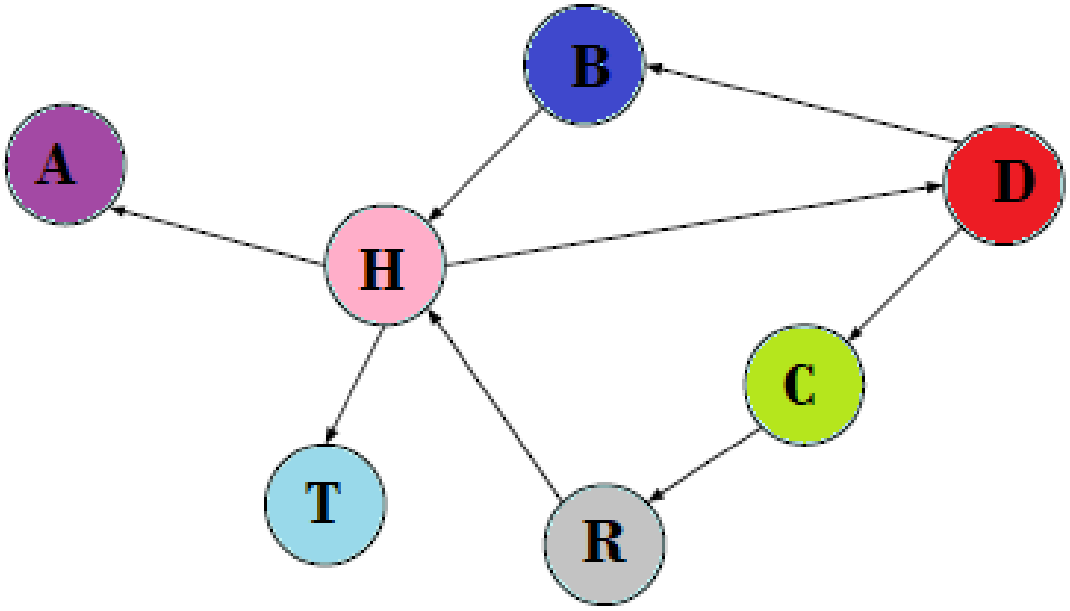
Ejemplo DFS:

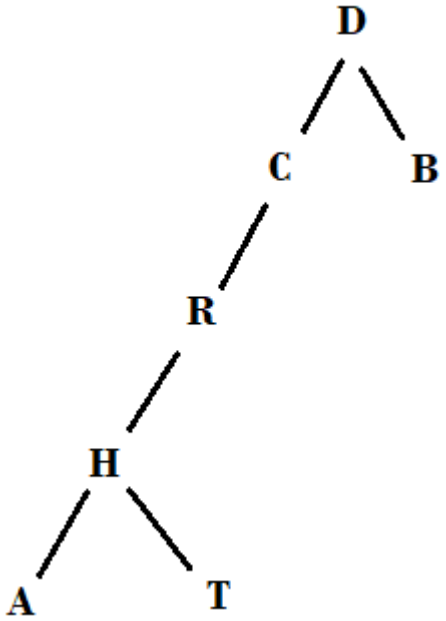


Tomamos como vertice de partida D

Clase 12: Árbol generador

Ejemplo DFS:

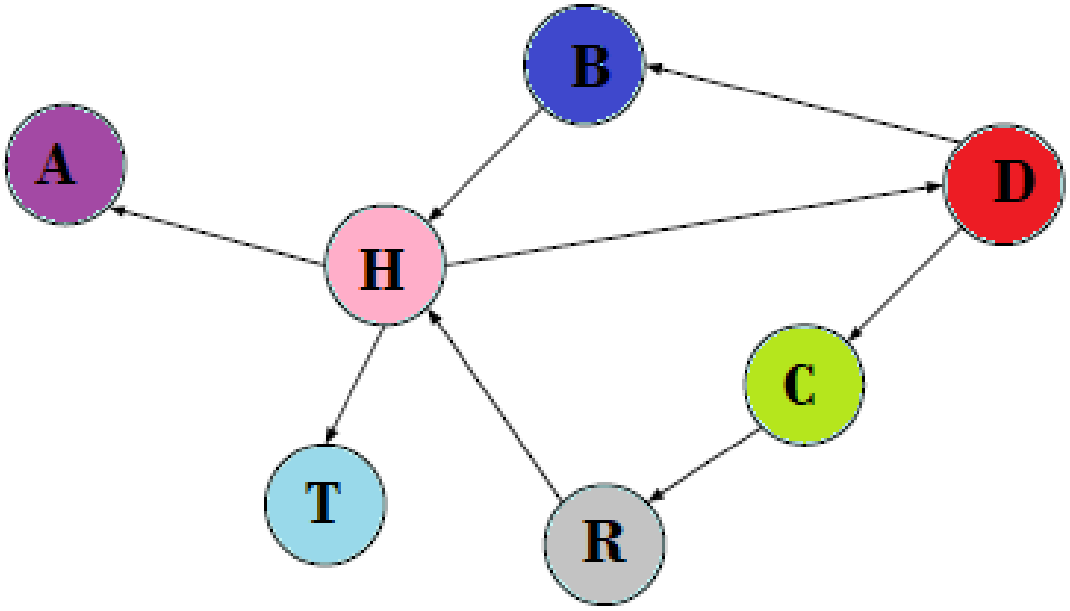


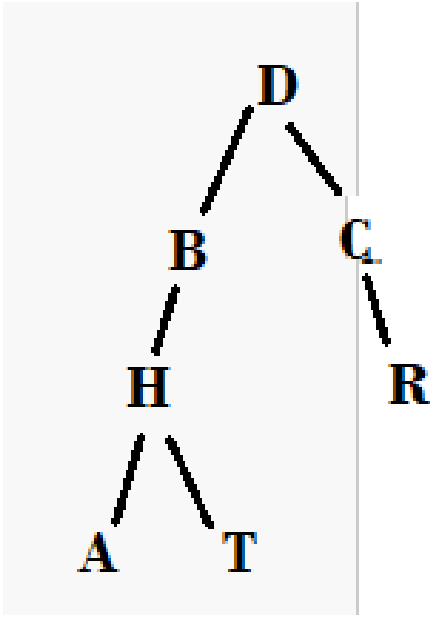


Tomamos como vertice de partida D

Clase 12: Árbol generador

Ejemplo DFS:

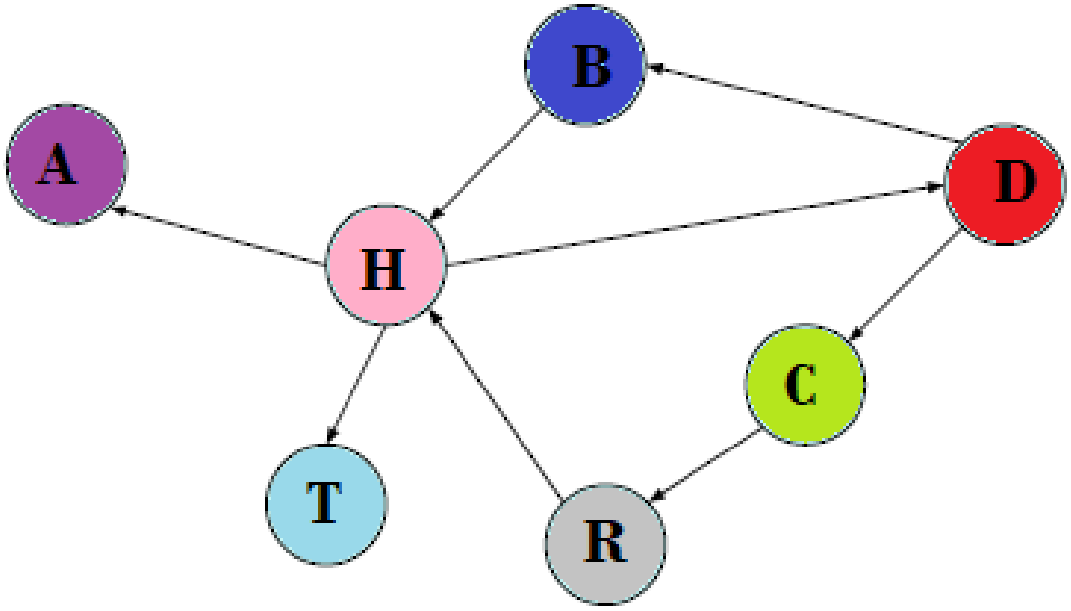




Tomamos como vertice de partida D

Clase 12: Árbol generador

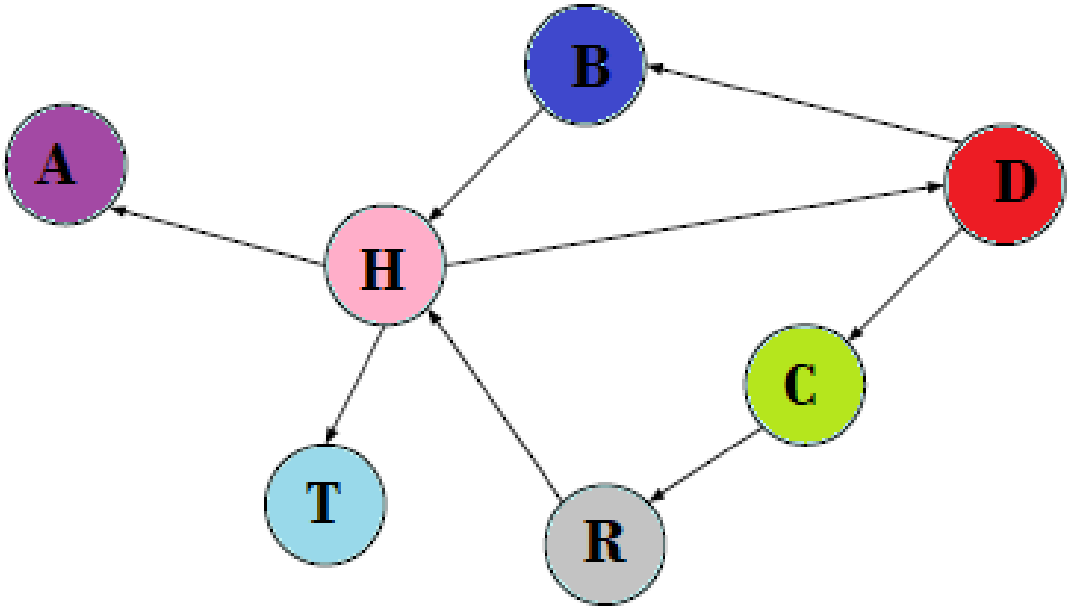
Ejemplo BFS:

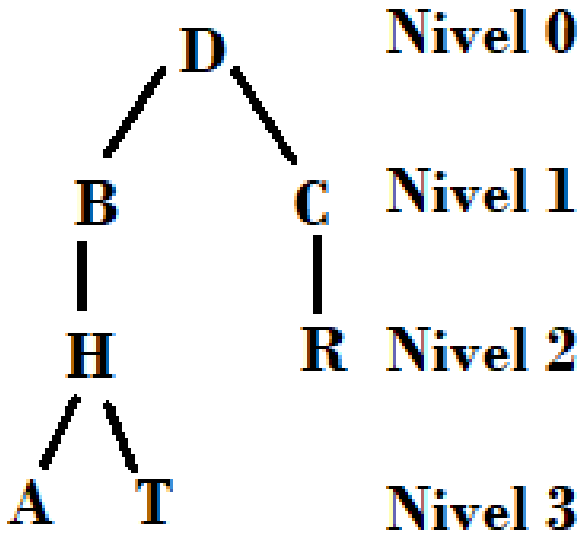


Tomamos como vertice de partida D

Clase 12: Árbol generador

Ejemplo BFS:





Tomamos como vertice de partida D

Aplicación del DFS:

- Encontrar las componentes conexas de un grafo no dirigido.
- Dado un grafo (dirigido o no dirigido) comprobar si tiene algún ciclo o no.
- Encontrar las componentes fuertemente conexas de un grafo dirigido.

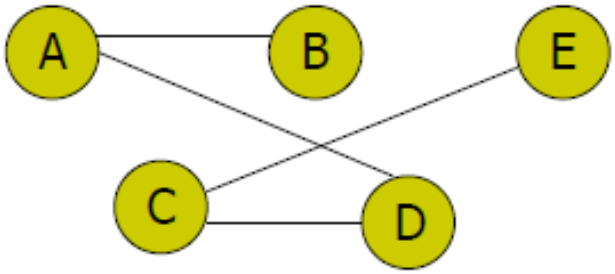
Aplicación del DFS:

Encontrar las componentes conexas de un grafo no dirigido

- Si el grafo es conexo
  - Un recorrido desde cualquier vértice
  - Visitará a TODOS los vértices del grafo

- Si no lo es
  - Partiendo de un vértice, tendremos una componente conexa ⇨ conjunto de vértices recorrido
  - Para descubrir otras
    - Repetir recorrido desde un vértice no visitado
    - Hasta que todos los vértices hayan sido visitados

Aplicación del DFS:



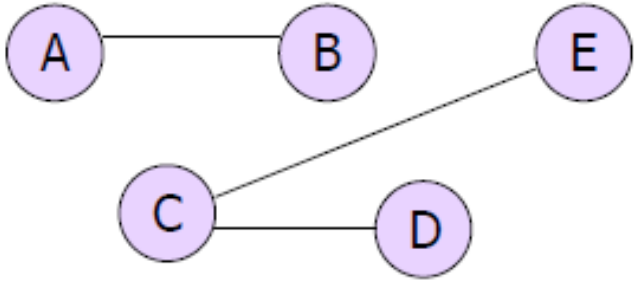
Recorrido desde E

E C D A B

Conjunto recorrido =

conjunto de Vértices

ES CONEXO



Recorrido desde E

E C D

Componente Conexa 1

Recorrido desde B

B A

Componente Conexa 2

Aplicación del DFS:

Prueba de aciclicidad

- **Grafo no dirigido.** Hacer un recorrido dfs. Existe algún ciclo si y sólo si aparece algún arco que no es del árbol de expansión.
- **Grafo dirigido.** Hacer un dfs. Existe un ciclo si y sólo si aparece algún arco de retroceso.

Orden de complejidad de la prueba de aciclicidad: igual que los recorridos.

- Con matrices de adyacencia: $\mathbf{O(|V|^2)}$.
- Con listas de adyacencia: $\mathbf{O(|V| + |E|)}$.

Consultas

Temas a desarrollar la próxima clase

- Ordenamiento topológico.
- Ejemplos de aplicación. Distintas implementaciones.
- Análisis de la eficiencia de cada uno.
- Problema del camino mínimo.
- Algoritmos de Dijkstra y Floyd. Árbol generador mínimo.
- Algoritmos de Prim y Kruskal. Análisis del tiempo de ejecución de los algoritmos vistos.