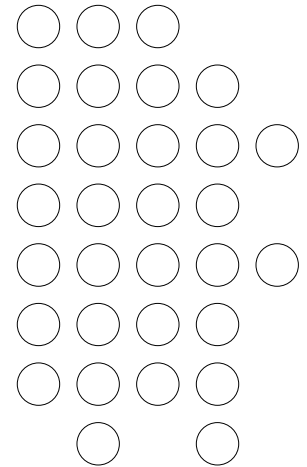


Grafos: búsqueda en profundidad

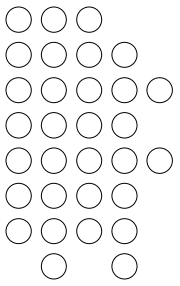


UNIVERSIDAD
DE LOS ANDES

Diseño y Análisis de Algoritmos
Cátedra de Programación
Carrera de Ingeniería de Sistemas
Prof. Isabel Besembel Carrera

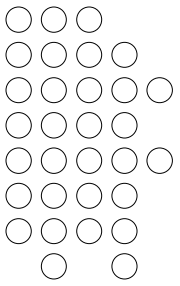


Recorridos de árboles



- Árboles binarios:
 - ❖ Preorden: raíz-izquierdo-derecho (RID)
 - ❖ Enorden: izquierdo-raíz-derecho (IRD)
 - ❖ Posorden: izquierdo-derecho-raíz (IDR)
- Pre y posorden se generalizan para los árboles
- Son recursivas por naturaleza
- Exploración de izquierda a derecha o viceversa
- Lema: cada uno de los recorridos, el tiempo $T(n)$ que se necesita para explorar un árbol binario que contiene n nodos se encuentra en $\Theta(n)$

Recorridos de árboles



➤ Demostración:

Suponga que visitar un nodo está en $O(1)$, $T(0)$ está acotado superiormente por alguna constante c , donde $c \geq T(0)$

Suponga que hay que explorar un árbol de $n > 0$ nodos, uno de ellos es la raíz, g de ellos están en el subárbol izquierdo y $n-g-1$ en el subárbol derecho, entonces

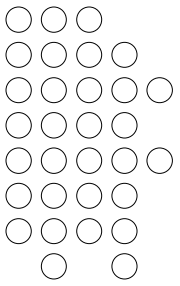
$$T(n) \leq \max_{0 \leq g \leq n-1} (T(g) + T(n-g-1) + c), \quad n > 0$$

Sin importar el orden de exploración. Se demuestra por inducción que $T(n) \leq an+b$, donde a y b son constantes.

Si $b \geq c$, la hipótesis es verdadera para $n=0$ ya que $c \geq T(0)$

Paso de inducción: $n > 0$ y la hipótesis cierta para todo $m \in [0, n-1]$

Recorridos de árboles



$$T(n) \leq \max_{0 \leq g \leq m-1} (T(g) + T(n-g-1) + c)$$

$$T(n) \leq \max_{0 \leq g \leq n-1} (ag + b + a(n-g-1) + b + c)$$

$$T(n) \leq an + 3b - a$$

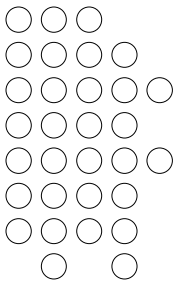
Si $a \geq 2b$ se tiene $T(n) \leq an + b$,

la hipótesis es cierta también para $m = n$, lo que demuestra que

$T(n) \leq an + b \forall n \geq 0$, por lo que $T(n)$ está en $O(n)$

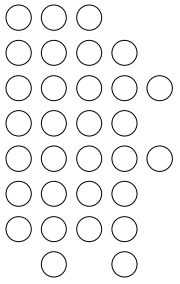
Además, $T(n)$ está en $\Omega(n)$ ya que se visitan todos los nodos y por ello $T(n)$ está en $\Theta(n)$.

Búsqueda en profundidad

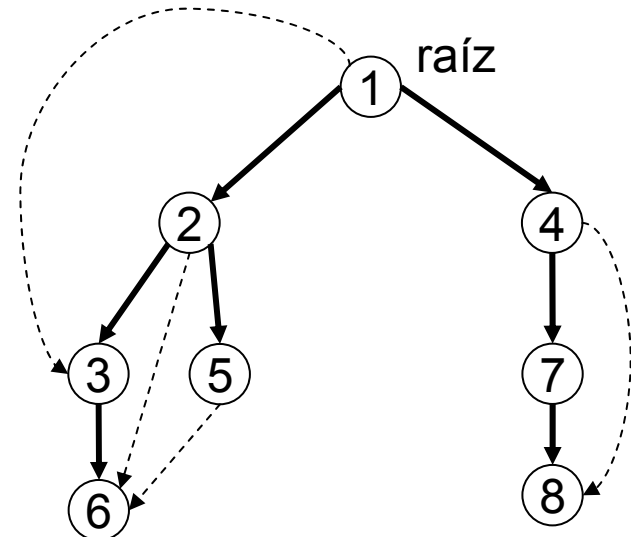
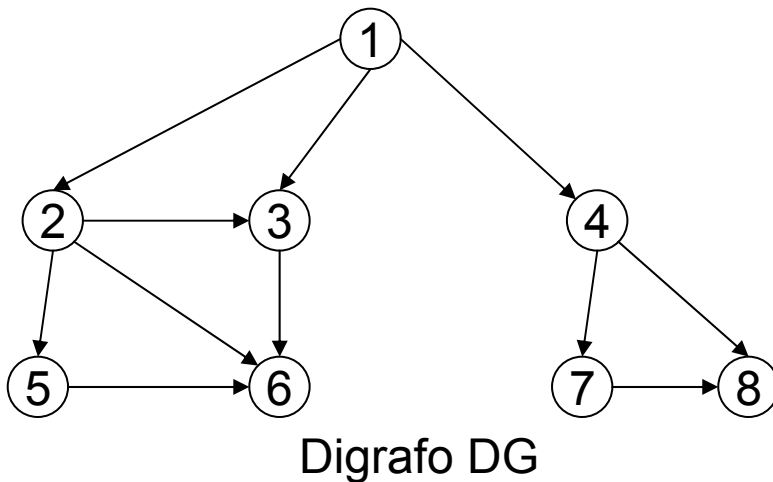


- Sea $G = \{N, A, f\}$ un grafo formado por todos los nodos que se desean visitar.
- Suponga una forma de marcar los nodos visitados
 - ❖ Se selecciona un nodo para comenzar $v \in N$ (nodo actual) y se marca como visitado
 - ❖ Si hay algún nodo adyacente a v que no ha sido visitado aún, se toma dicho nodo como nodo actual y se invoca recursivamente el proceso de recorrido en profundidad
 - ❖ Cuando están marcados como visitados todos los nodos adyacentes a v , se termina el recorrido para v .

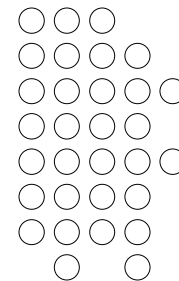
Búsqueda en profundidad



- ❖ Si queda algún nodo en G que no ha sido visitado, se repite el proceso tomando un nodo cualquiera como v
- Al recorrido se le asocia un árbol de recubrimiento



Búsqueda en profundidad del DigrafoMat

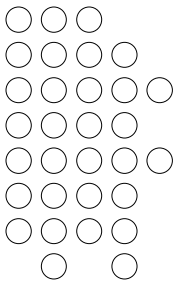


Marzo, 2005		
recorridoEnProf()		
{pre: $g(i, j) \geq 0 \quad \forall i, j$ } {pos: $marca(i) = Verdadero \quad \forall i$ }		
1	[$marca(i) = falso$] $i = 1, pos$	-pos: Definido en DigrafoMat.
2	[Si ($\neg marca(i)$) entonces busPro($i, marca$) fsi] $i = 1, pos$	-marca: Arreglo[100]De Lógico. Marca que indica si el nodo fue visitado (Verdadero) o no (Falso). -i: Entero: Variable con la posición de los nodos en la matriz

Marzo, 2005		
busPro(Entero+: na , Arreglo[100]De Lógico: mar)		
{pre: $1 \leq na \leq pos$ } {pos: $marca(i) = Verdadero \quad \forall i$ }		
1	$mar(na) = Verdadero$	-pos, nodoAdyacente(): Definidos en DigrafoMat.
2	$nadya = nodoAdyacente(na)$	-mar: Arreglo[100]De Lógico. Marca que indica si el nodo fue visitado (Verdadero) o no (Falso).
3	[Si ($\neg mar(nadya(i))$) entonces busPro($nadya(i), mar$) fsi] $i = 1, pos$	-i: Entero+: Variable con la posición de los nodos en la matriz -nadya: Arreglo[100]De Entero+. Vector auxiliar que contiene los nodos adyacentes

$$T(n) = \Theta(\text{máx}(N, A))$$

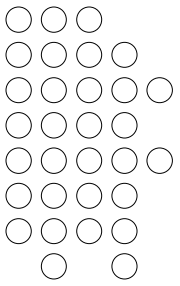
Búsqueda en profundidad del DigrafoLis



Marzo, 2005		
recorridoEnProf()		
{pre: g.n ≥ 0 }		{pos: marca (i) = Verdadero ∀ i}
1	[marca(i) = falso] i = 1, g.numEle()	- g: Definido en DigrafoLis
2	[Si (¬ marca(i)) entonces busPro(i, marca) fsi] i = 1, g.numEle()	- marca : Arreglo[100]De Lógico. Marca que indica si el nodo fue visitado (Verdadero) o no (Falso). - i : Entero: Variable auxiliar para los nodos
Marzo, 2005		
busPro(Entero+: na, Arreglo[100]De Lógico: mar)		
{pre: g.n ≥ 0 ∧ 1 ≤ na ≤ g.n}		{pos: marca (i) = Verdadero ∀ i}
1	mar(na) = Verdadero	- g, nodoAdyacente() : Definidos en DigrafoLis
2	nadya = nodoAdyacente(na)	- mar : Arreglo[100]De Lógico. Marca que indica si el nodo fue visitado (Verdadero) o no (Falso).
3	nadya.cursorAlInicio()	- i : Entero+: Variable auxiliar
4	[Si(¬mar(nadya.conLista().Info())) entonces busPro(nadya.conLista().Info(), mar) fsi nadya.cursorAlProximo()] i = 1, nadya.numEle()	- nadya : Lista[Entero+]. Lista auxiliar que contiene los nodos adyacentes - cursorAlInicio(), numEle(), conLista(), cursorAlProximo() . Definidos en Lista

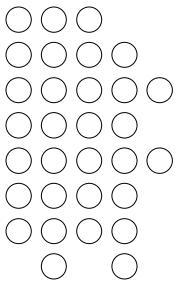
$$T(n) = \Theta(\text{máx}(N,A))$$

Búsqueda en profundidad

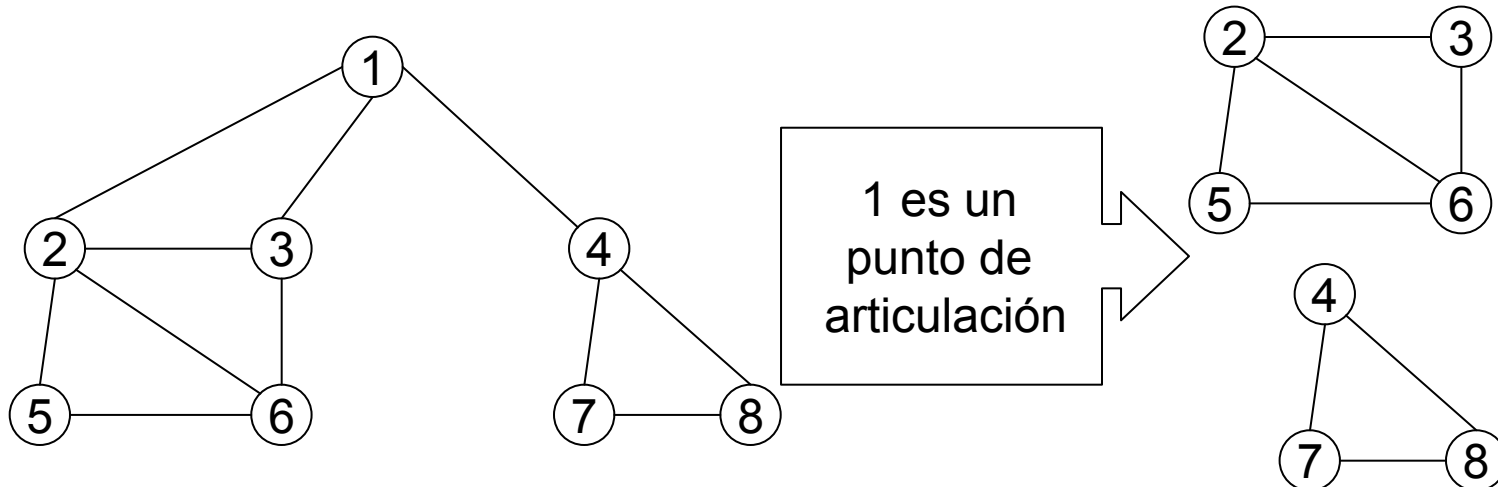


- Si el grafo es conexo, se tendrá un único árbol de recorrido en profundidad
- Si el grafo no es conexo, se tendrá un bosque de árboles, uno por cada componente conexo
- Numera los nodos del árbol en preorden
 - ❖ Se coloca en el algoritmo recorridoEnProf
$$np = np + 1 \quad \text{como paso 1}$$
$$\text{prenum}(i) = np \quad \text{como paso 2}$$
 - ❖ en el algoritmo busPro
$$np = 0 \quad \text{como paso 2}$$

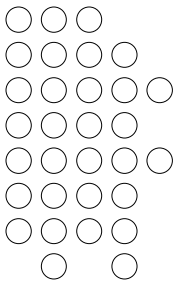
Puntos de articulación



- Un nodo v es un punto de articulación si el subgrafo obtenido al borrar todas las aristas que incidan en v ya no es conexo

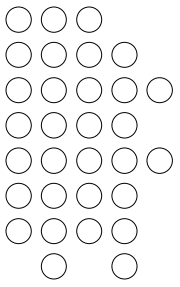


Grafo biconexo y bicoherente

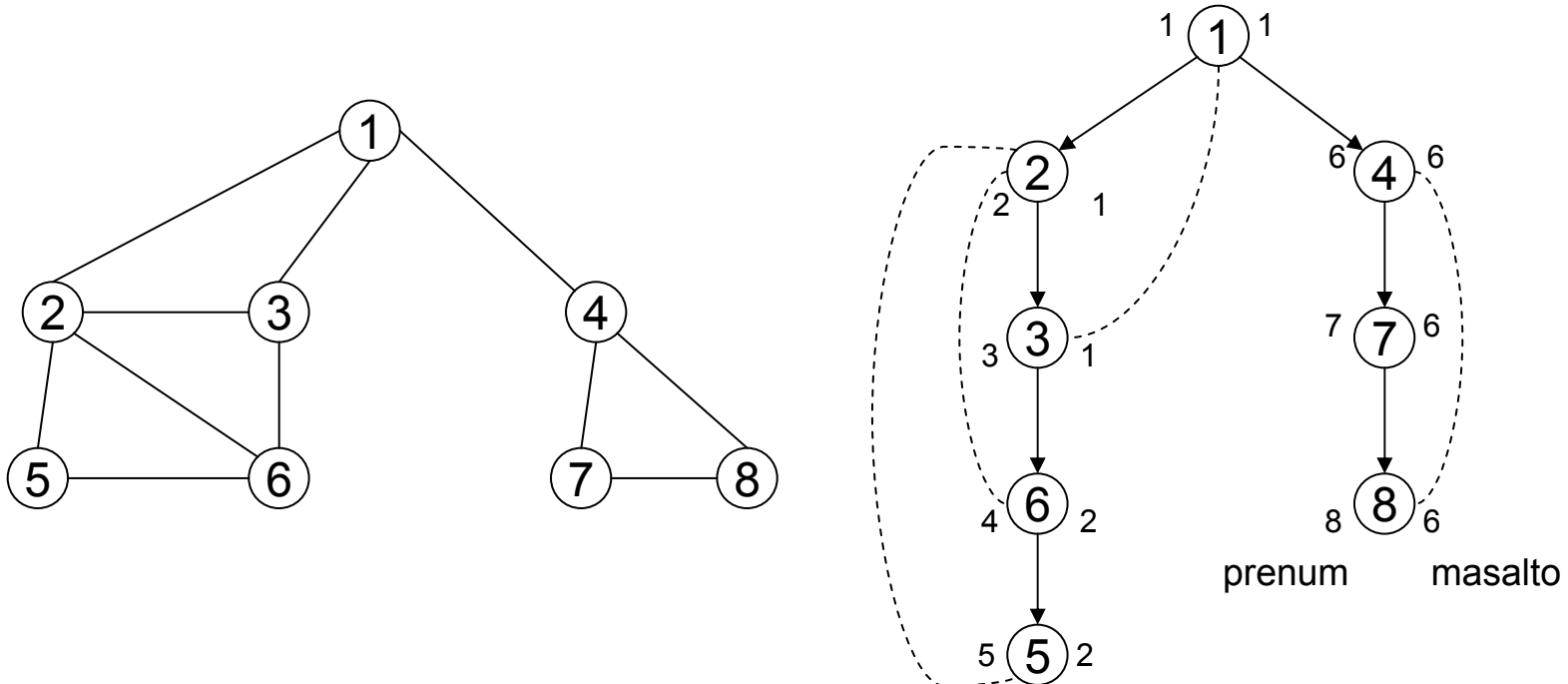


- Un grafo G es biconexo si es conexo y no tiene puntos de articulación
 - ❖ Ejemplo: si G representa una red de telecomunicaciones, entonces G asegura que la red puede seguir funcionando aunque falle uno de los equipos de los nodos.
- Un grafo G es bicoherente si todo punto de articulación está unido mediante al menos 2 aristas con cada componente del subgrafo restante
 - ❖ Si además de biconexo, G es bicoherente, se tiene la seguridad que la red seguirá funcionando aunque falle una línea de transmisión

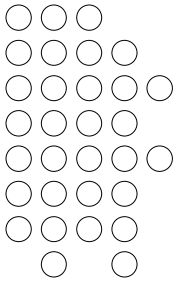
Cálculo de los puntos de articulación



- Se numeran los nodos de G con **prenum**
- Se calcula el valor de **masalto** para cada nodo de G

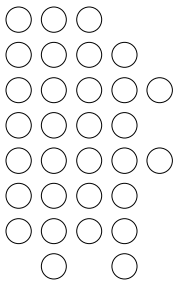


Cálculo de los puntos de articulación



- ❖ $\text{prenum}(v)$: se calcula para cada nodo haciendo el recorrido en profundidad (preorden)
- ❖ $\text{masalto}(v) = \min(\text{prenum}(v), \text{prenum}(w), \text{masalto}(x))$
donde
 - (v) es el nodo actual
 - (w) Es el nodo antecesor de v alcanzable por una arista que no pertenece a las aristas del árbol del recorrido en profundidad (T)
 - (x) Todo hijo de v desde los cuales se puede recorrer hasta un antecesor de v

Cálculo de los puntos de articulación



Marzo/05

puntosDeArticulacion()

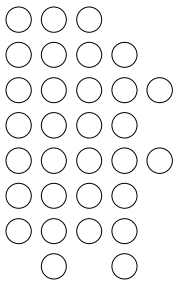
{pre: $|N| > 0$ }

{pos: $|N| > 0 \wedge G' = G \wedge \text{padre} \supset \text{bosque en profundidad}$ }

<p>1 recorridoEnProf() 2 Recorrer T en posorden y para cada nodo na se calcula masalto(na) 3 [Si (na = raíz \wedge na no tiene mas de 1 hijo) entonces na es un punto de articulación sino Si (na \neq raíz \wedge na tiene un hijo x / masalto(x) \geq prenum(na)) entonces na es un punto de articulación fsi] na \in N</p>	<p>-na, x: Entero. Nodo actual y nodo hijo del actual. -recorridoEnProf(). Realiza la búsqueda en profundidad calculando prenum de cada nodo</p>
---	---

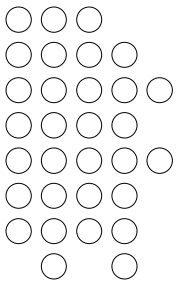
- Ejercicio: combinar los pasos 1 y 2 para calcular prenum y masalto dentro del recorrido en profundidad

Búsqueda en profundidad



- El subgrafo predecesor de una búsqueda en profundidad forma un bosque en profundidad compuesto de varios árboles en profundidad donde $A_\pi = \{(\text{padre}(v), v) : v \in N \text{ y } \text{padre}(v) \neq 0\}$
- Para cada na , se tienen dos variables auxiliares para indicar cuando se descubre na $d(na)$ y cuando se termina de trabajar con na $f(na)$
 - ❖ Para cada na , $d(na) < f(na)$.
 - ❖ El nodo na es blanco antes de $d(na)$, gris entre $d(na)$ y $f(na)$ y negro después de $f(na)$.

Búsqueda en profundidad



26/11/98

busPro()

{pre: $n > 0$ }

{pos: $n > 0 \wedge G' = G \wedge \text{padre} \supset \text{bosque en profundidad}$ }

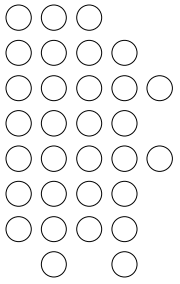
```

1 [ color(u), padre(u) = 'blanco', 0 ] u ∈ N
2 t = 0
3 [ Si ( color(u) = 'blanco' ) entonces
   visitaBusPro(u, t, color, padre, d, f)
   fsi ] u ∈ N

```

-**u, t**: Entero. Variable auxiliar y contador.
 -**color, padre, d, f**: Arreglo[n] De Entero.
 Contienen el color del nodo, el nodo predecesor inmediato, la etiqueta donde se empezó a procesar el nodo y donde se terminó de procesar cada nodo.
 -**visitaBusPro()**. Rutina recursiva para hacer la búsqueda en profundidad.

Búsqueda en profundidad



26/11/98

visitaBusPro(Entero: u, t, Arreglo[n]De [Entero]: color, padre, d, f)

{pre: $n > 0 \wedge u \in N$ }

{pos: $n > 0 \wedge G' = G \wedge \text{padre} \supset \text{bosque en profundidad}$ }

```

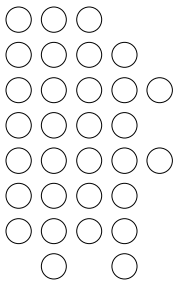
1  color(u), t, d(u) = 'gris', t + 1, t
2  [ Si ( color(v) = 'blanco' ) entonces
3     padre(v) = u
4     visitaBusPro(v, t, color, padre, d, f)
   fsi ]  $\forall v \in u.\text{listaAdyacencia}$ 
   color(u), t, f(u) = 'negro', t + 1, t
   regrese

```

-u, v, t: Entero. Variables auxiliares y contador.
-color, padre, d, f: Arreglo[n] De [Entero].
 Contienen el color del nodo, el nodo predecesor inmediato, la etiqueta donde se empezó a procesar el nodo y donde se terminó de procesar cada nodo.
-visitaBusPro(). Rutina recursiva para hacer la búsqueda en profundidad.

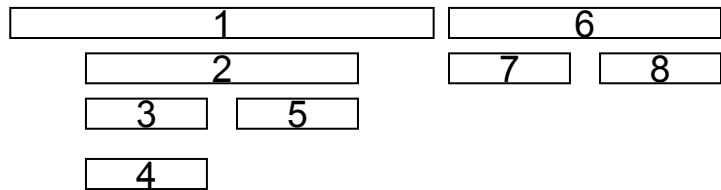
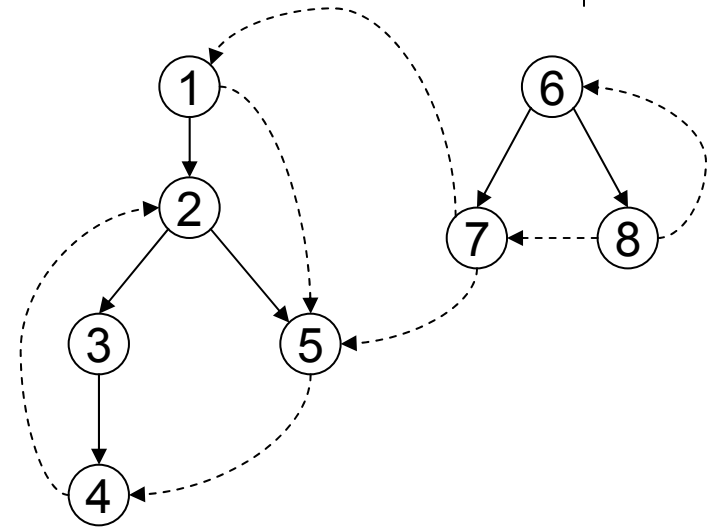
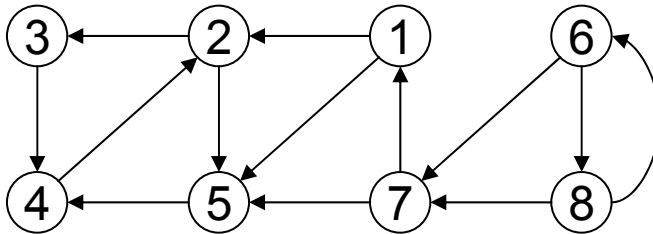
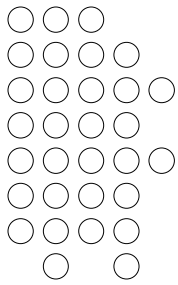
$$T(n) = \Theta(N + A)$$

Teorema del paréntesis



- **Teorema del paréntesis:** Para cualquier búsqueda en profundidad en un grafo dirigido o no $G = (\mathbb{N}, A)$ y cualquier par de nodos u y v , una de las tres condiciones es cierta:
 - Los intervalos $[d(u), f(u)]$ y $[d(v), f(v)]$ son disjuntos
 - $[d(u), f(u)]$ está contenido enteramente dentro de $[d(v), f(v)]$ y u es un descendiente de v en el árbol en profundidad, o
 - $[d(v), f(v)]$ está contenido enteramente dentro de $[d(u), f(u)]$ y v es un descendiente de u en el árbol en profundidad.
- Corolario 7: El nodo v es un descendiente propio de u en el bosque en profundidad para $G \Leftrightarrow d(u) < d(v) < f(v) < f(u)$.

Ejemplo



(1 (2(3(4) (5))))(6(7) (8))