



# Taller de Programación

---

## **Grafos**

Jhonny Felípez Andrade  
[jrfelizamigo@yahoo.es](mailto:jrfelizamigo@yahoo.es)



# Contenido

---

- Nociones de Grafos.
- Propiedades.
- Estructura de datos para grafos.



# Nociones de Grafos

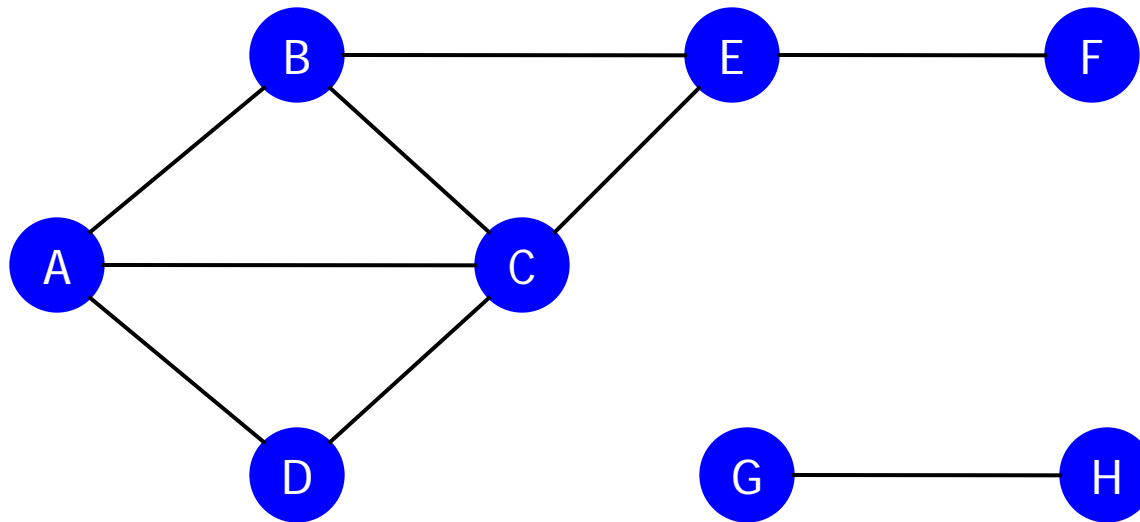
---

- Un grafo  $G = (V, E)$  se define como un conjunto de vértices  $V$  y un conjunto de arcos  $E$ , cada uno de los cuales consiste en un par ordenado o desordenado de vértices en  $V$ .

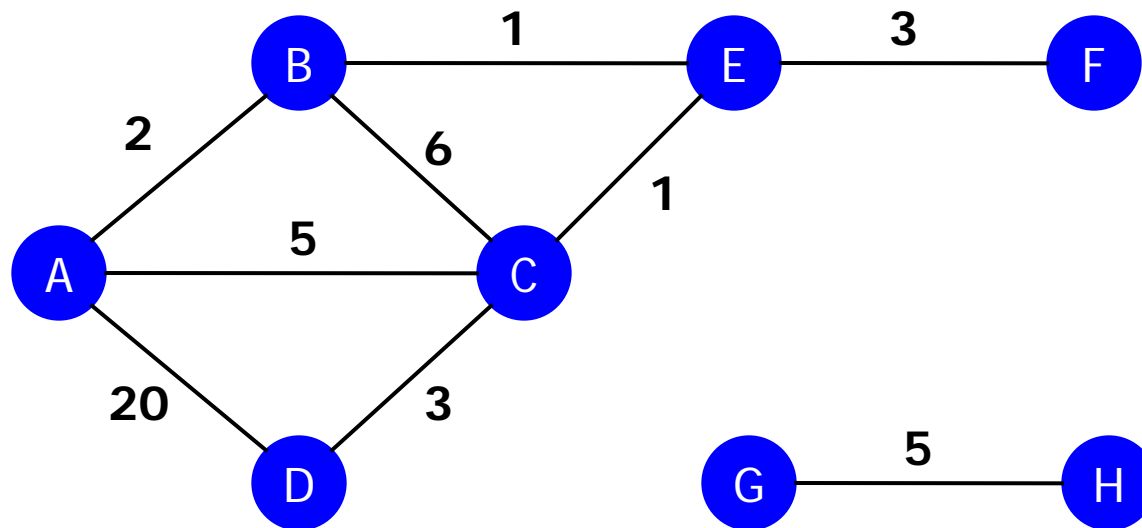


# Simple Grafo

---



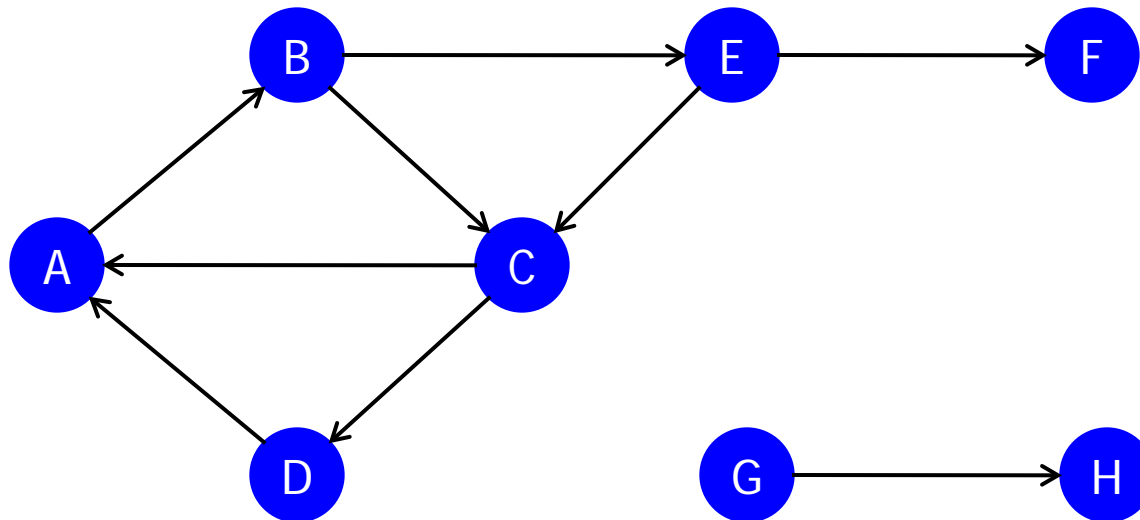
# Grafo ponderado o etiquetado





# Grafo dirigido

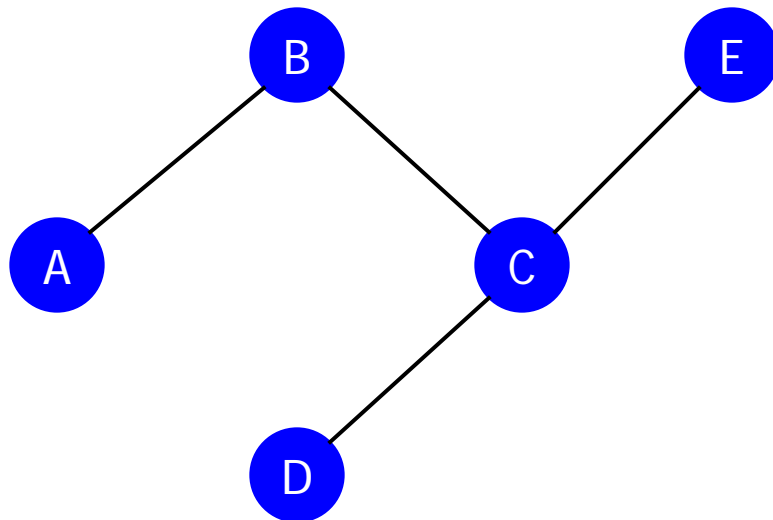
---





# Árbol

---

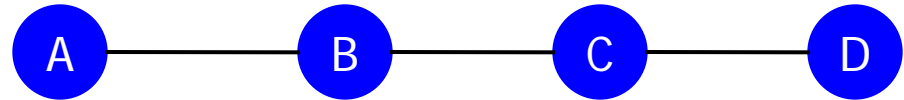




# Otra terminología

---

- Camino



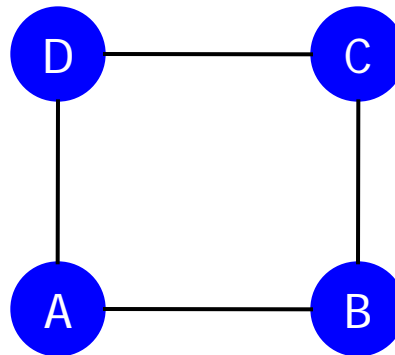




# Otra terminología

---

- Camino
- Ciclo

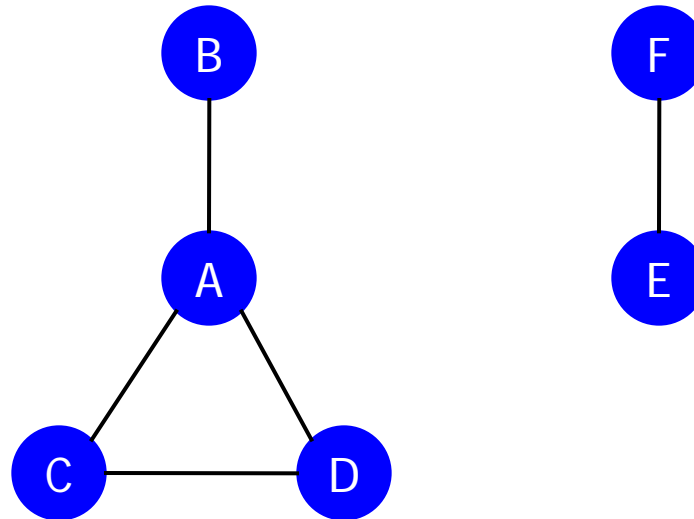




# Otra terminología

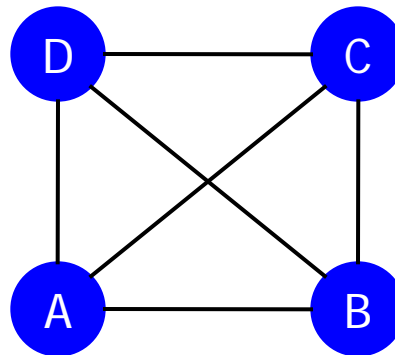
---

- Camino
- Ciclo
- Conectado



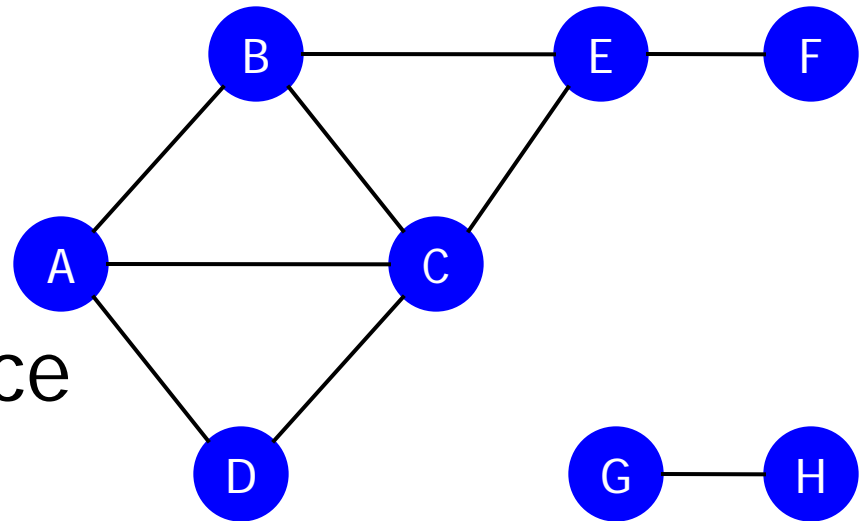
# Otra terminología

- Camino
- Ciclo
- Conectado
- Completo



# Otra terminología

- Camino
- Ciclo
- Conectado
- Grado de un vértice

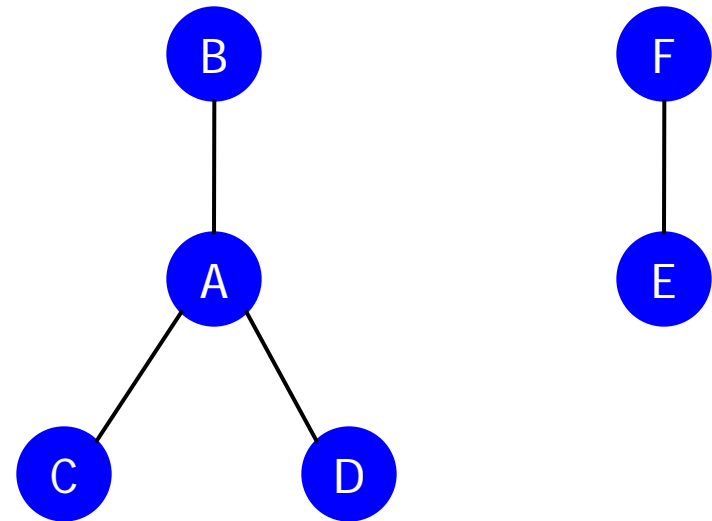




# Otra terminología

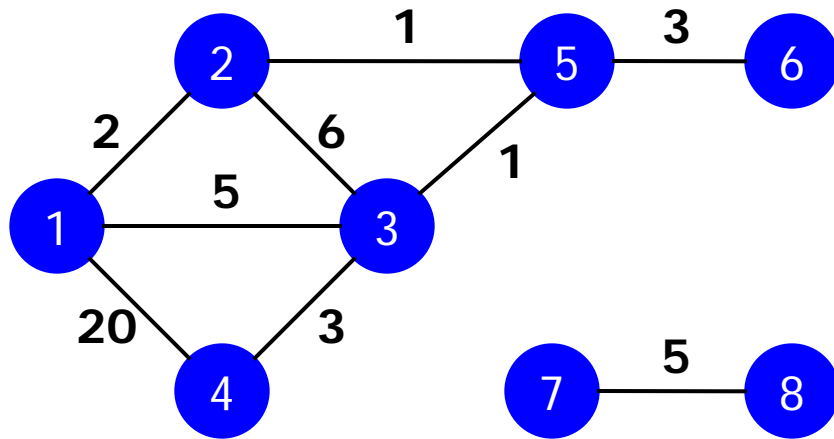
---

- Camino
- Ciclo
- Conectado
- Grado de un vértice
- Bosque



# Estructura de datos para grafos

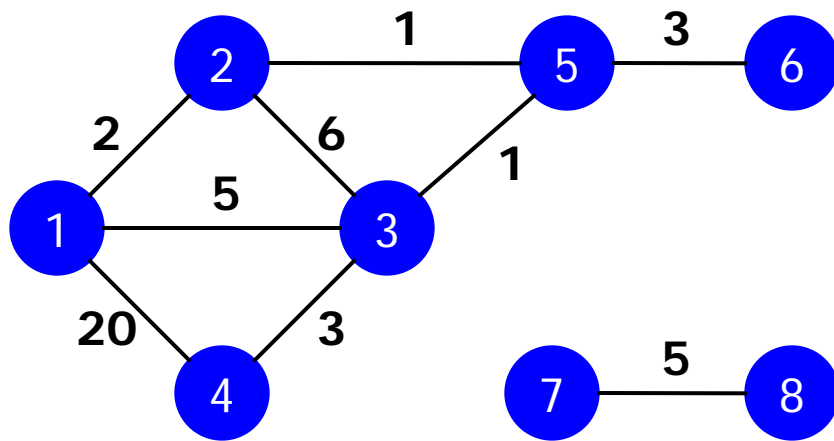
- **Matriz de adyacencias.** Permite consultas rápidas ¿esta  $(i,j)$  en  $G$ ?  
Las actualizaciones son inmediatas cuando se insertan o eliminan arcos.  
Se desaprovecha gran cantidad de espacio.



	1	2	3	4	5	6	7	8
1		2	5	20				
2	2		6		1			
3	5	6		3	1			
4	20		3					
5		1	1			3		
6					3			
7								5
8							5	

# Estructura de datos para grafos

- **Lista de Adyacencias como matriz.** Ocupa mucho espacio. La estructura de datos es sencilla de programar para grafos estáticos (que no cambian). Cada fila para cada vértice puede ser una lista.



**grado**

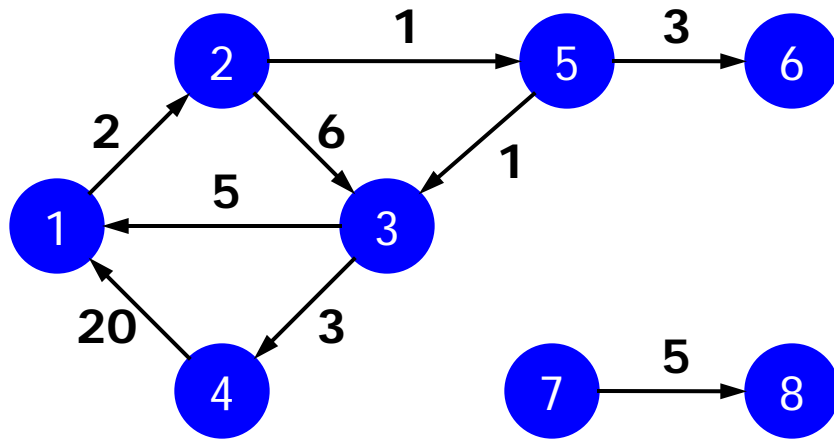
1	2	3	4	5	6	7	8
3	3	4	2	3	1	1	1

**arcos**

	1	2	3	4	5	6	7	8
0	2	1	1	1	2	5	8	7
1	3	3	2	3	3	0	0	0
2	4	5	4	0	6	0	0	0
3	0	0	5	0	0	0	0	0

# Estructura de datos para grafos

- **Lista de Adyacencias como matriz.** Ocupa mucho espacio. La estructura de datos es sencilla de programar para grafos estáticos (que no cambian). Cada fila para cada vértice puede ser una lista.



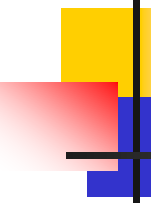
**grado**

1	2	3	4	5	6	7	8
1	2	2	1	2	0	1	0

**arcos**

	1	2	3	4	5	6	7	8
0	2	3	1	1	3	0	8	0
1	0	5	4	0	6	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0





```
private final int MAX_VERT = 20;
private Vertice vertices[]; // lista de vertices
private int matrizdeadyacencias[][]; // matriz de adyacencias
private int numvertices; // número de vertices
// -----
public Grafo() { // constructor
    vertices = new Vertice[MAX_VERT];
    // matriz de adyacencias
    matrizdeadyacencias = new int[MAX_VERT][MAX_VERT];
    numvertices = 0;
    // coloca la matriz de adyacencias a cero
    for (int j = 0; j < MAX_VERT; j++)
        for (int k = 0; k < MAX_VERT; k++)
            matrizdeadyacencias[j][k] = 0;
} // fin constructor
// -----
public void adiVertice(char e) {
    vertices[numvertices++] = new Vertice(e);
}
// -----
public void adiArco(int x, int y) {
    matrizdeadyacencias[x-1][y-1] = 1;
}
// -----
public void despliegaVertice(int v) {
    System.out.print(vertices[v].dato + " ");
}
```



# Recorrido de grafos

---

- La operación básica en la mayoría de los algoritmos de grafos consiste en recorrer el grafo completamente.
- Hay dos algoritmos para los recorridos:
  - Búsqueda en anchura (BFS).
  - Búsqueda en profundidad (DFS).



# Búsqueda en anchura

---

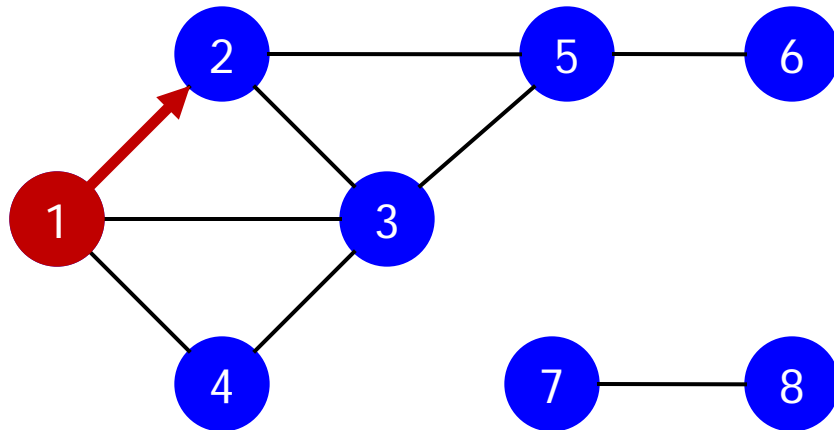
- La búsqueda en anchura se da cuando no tiene importancia el orden en el que visitemos los vértices y los arcos del grafo o estamos buscando el camino mas corto.

# Búsqueda en anchura BFS

- BFS tiene las siguientes reglas:

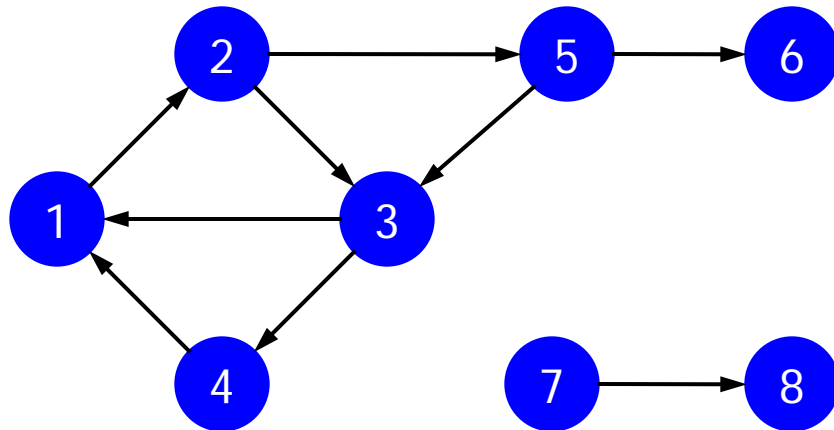
1. Seleccione un nodo no visitado X, visite este. Tiene que ser la raíz de un árbol BFS, que se está formando. Su nivel se denomina nivel actual.
2. Desde cada nodo Z del nivel actual, visite todos los vecinos no visitados de Z. Los nuevos nodos visitados forman un nuevo nivel que se convierten en el siguiente nivel actual.
3. Repita el paso 2 hasta que no se puedan visitar más nodos.
4. Si existen nodos no visitados, repita desde el paso 1.

# Búsqueda en anchura



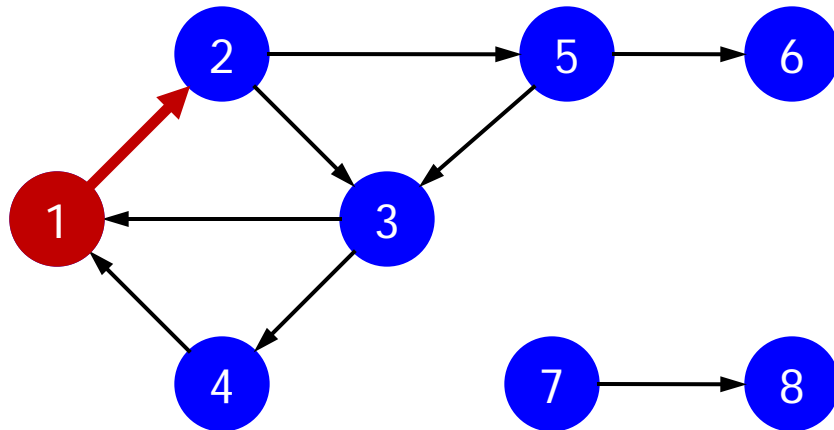
Cola: 1

# Búsqueda en anchura



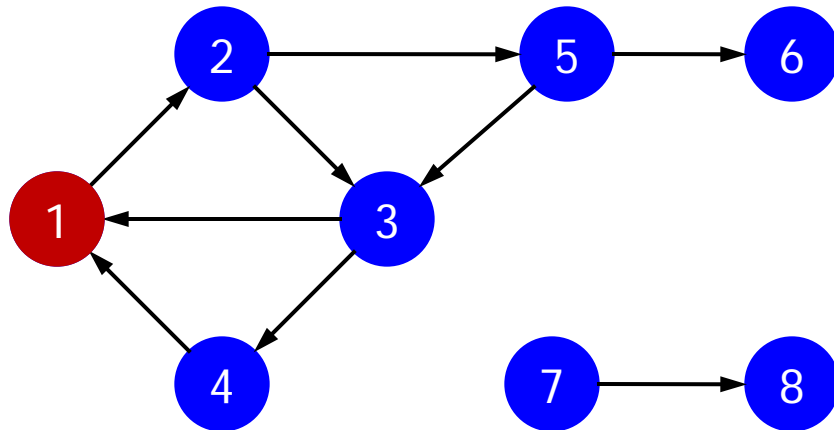
Cola:

# Búsqueda en anchura



Cola: 1

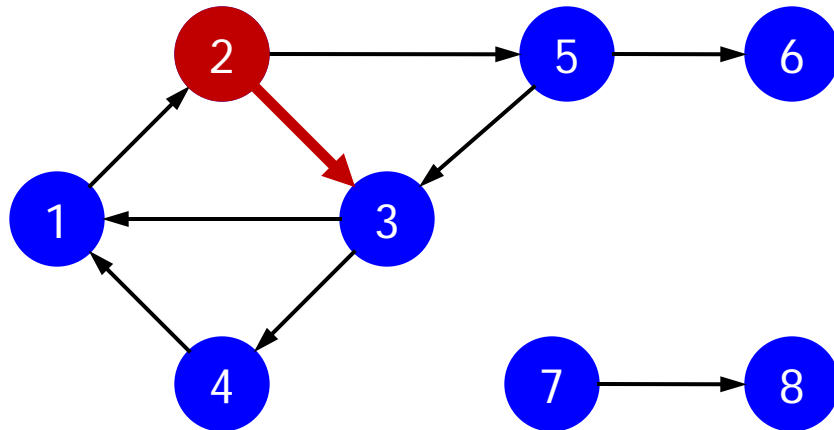
# Búsqueda en anchura



Cola: 1 2

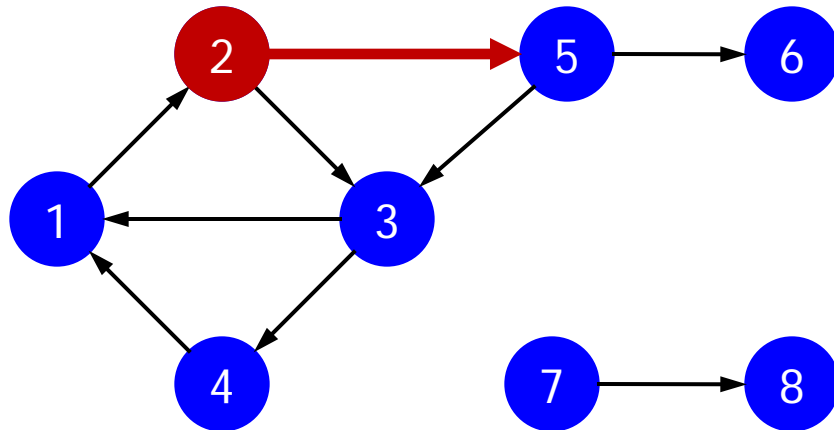


# Búsqueda en anchura



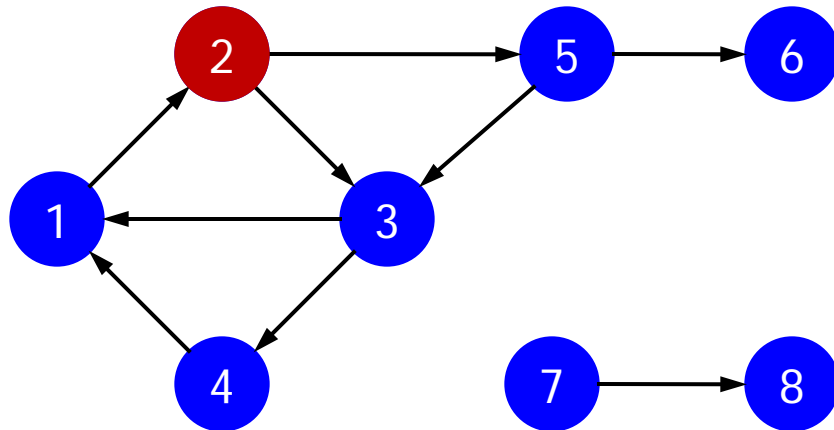
Cola: 2

# Búsqueda en anchura



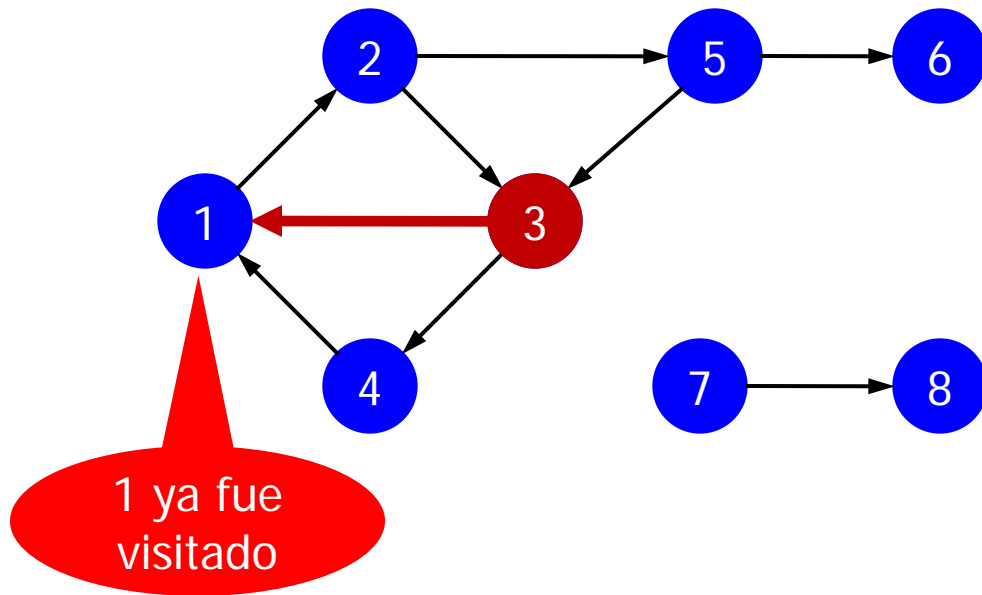
Cola: 2 3

# Búsqueda en anchura



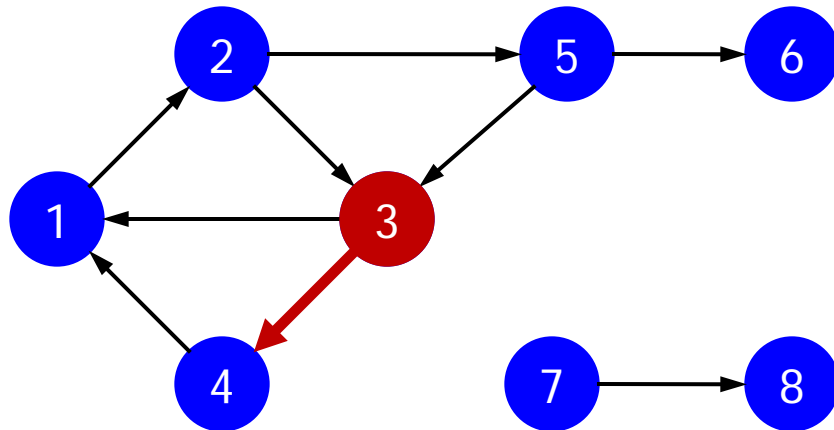
Cola: 2 3 5

# Búsqueda en anchura



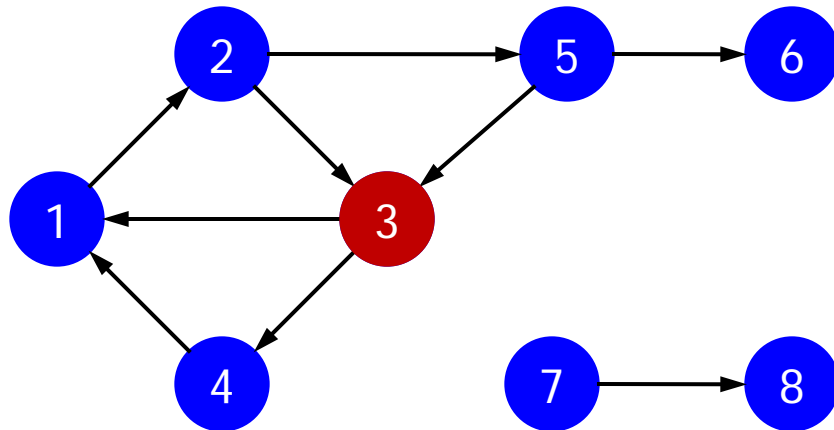
Cola: 3 5

# Búsqueda en anchura



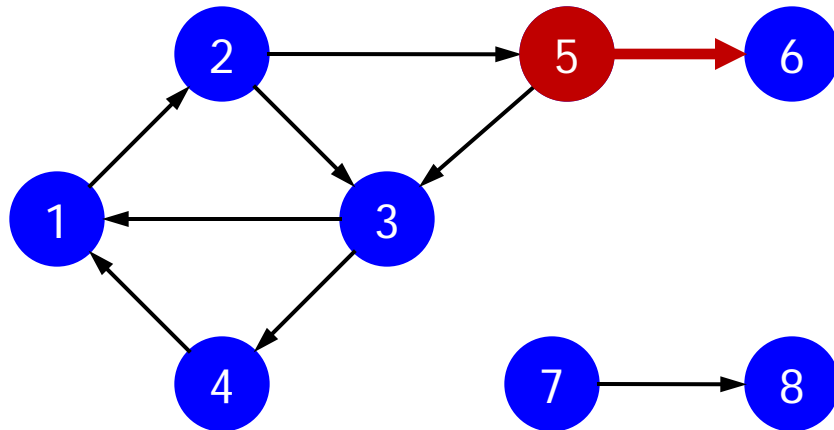
Cola: 3 5

# Búsqueda en anchura



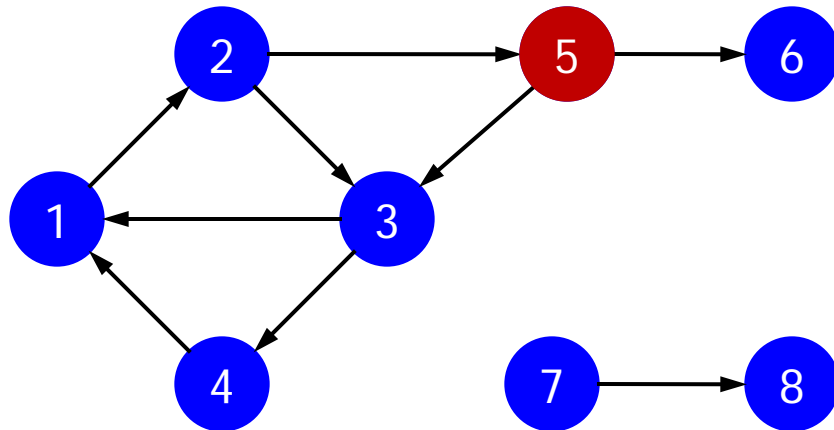
Cola: 3 5 4

# Búsqueda en anchura



Cola: 5 4

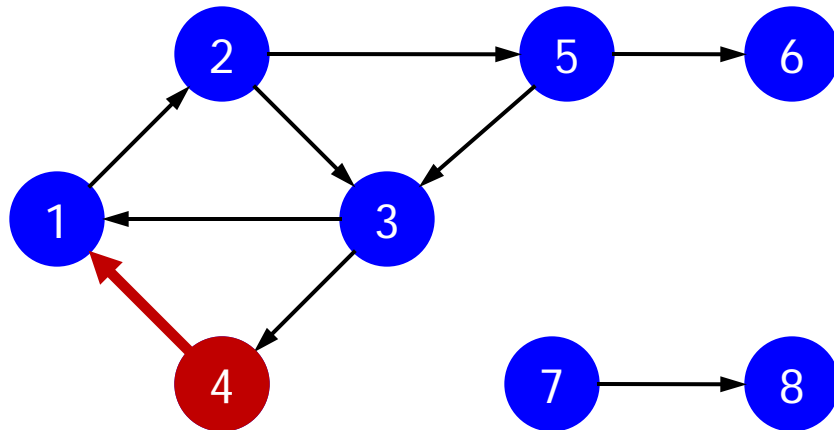
# Búsqueda en anchura



Cola: 5 4 6

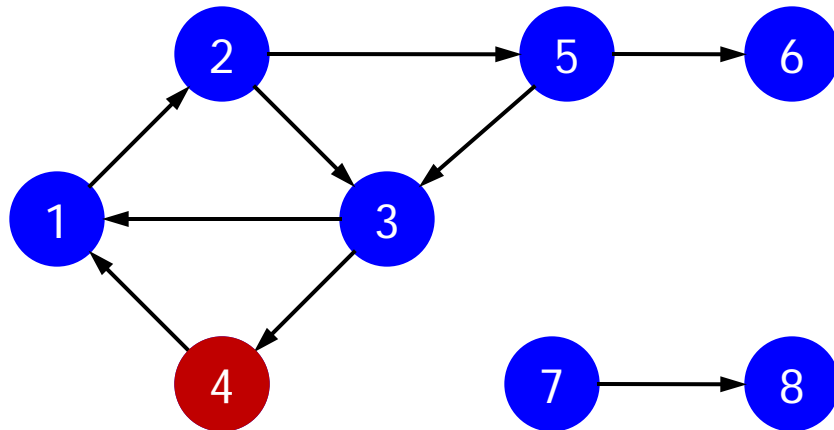


# Búsqueda en anchura



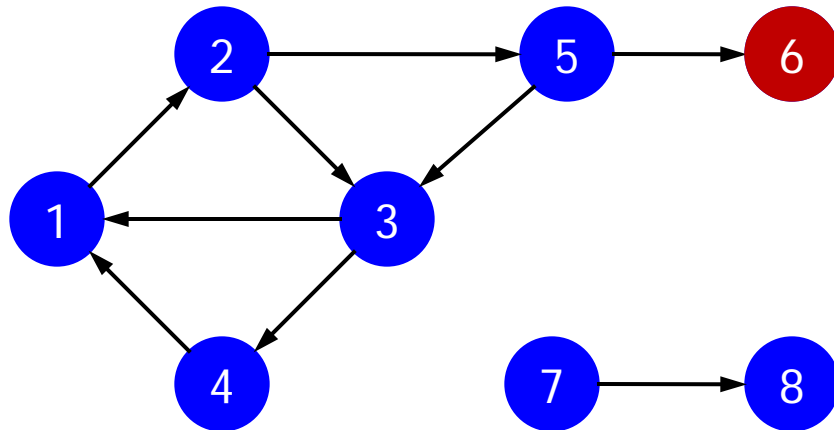
Cola: 4 6

# Búsqueda en anchura



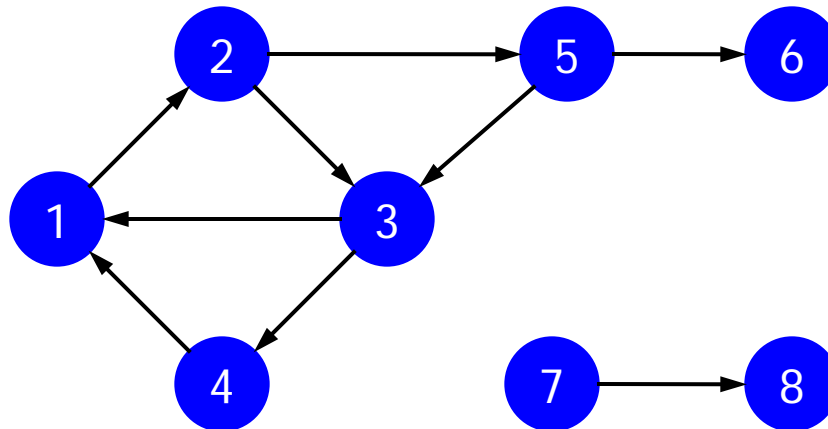
Cola: 4 6

# Búsqueda en anchura

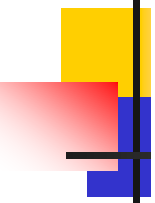


Cola: 6

# Búsqueda en anchura

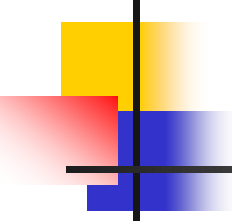


Cola:



```
private final int MAX_VERT = 20;
private Vertice vertices[]; // lista de vertices
private int matrizdeadyacencias[][]; // matriz de adyacencias
private int numvertices; // número de vertices
// -----
public Grafo() { // constructor
    vertices = new Vertice[MAX_VERT];
    // matriz de adyacencias
    matrizdeadyacencias = new int[MAX_VERT][MAX_VERT];
    numvertices = 0;
    // coloca la matriz de adyacencias a cero
    for (int j = 0; j < MAX_VERT; j++)
        for (int k = 0; k < MAX_VERT; k++)
            matrizdeadyacencias[j][k] = 0;
} // fin constructor
// -----
public void adiVertice(char e) {
    vertices[numvertices++] = new Vertice(e);
}
// -----
public void adiArco(int x, int y) {
    matrizdeadyacencias[x-1][y-1] = 1;
}
// -----
public void despliegaVertice(int v) {
    System.out.print(vertices[v].dato + " ");
}
```

```
public void bfs() { // búsqueda en anchura
    LinkedList<Integer> cola = new LinkedList<Integer>();
    vertices[0].visitado = true;
    despliegaVertice(0);
    cola.add(0);
    int v2;
    while (!cola.isEmpty()) {
        int v1 = cola.remove();
        while ((v2 = obtVerticeAdyNoVisitado(v1)) != -1) {
            vertices[v2].visitado = true;
            despliegaVertice(v2);
            cola.add(v2);
        }
    }
    for (int j = 0; j < numvertices; j++)
        vertices[j].visitado = false;
} // fin bfs()
```



---

```
<terminated> Aplicacion (1) [Java Application] C:\Program Files  
Vertices visitados (BFS) : 1 2 3 5 4 6
```



# Búsqueda en profundidad

---

- La búsqueda en profundidad es la misma que en el rastreo exhaustivo. Ambas técnicas consisten en la búsqueda entre todas las posibilidades, avanzando si es posible y retrocediendo tan pronto como se agotan las posibilidades de avanzar a vértices no explorados.
- En la búsqueda en profundidad como una búsqueda en anchura se utiliza una pila en vez de una cola.

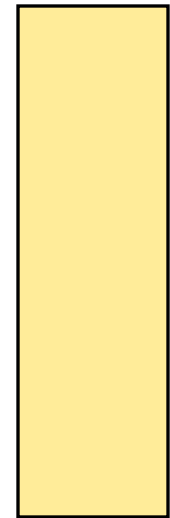
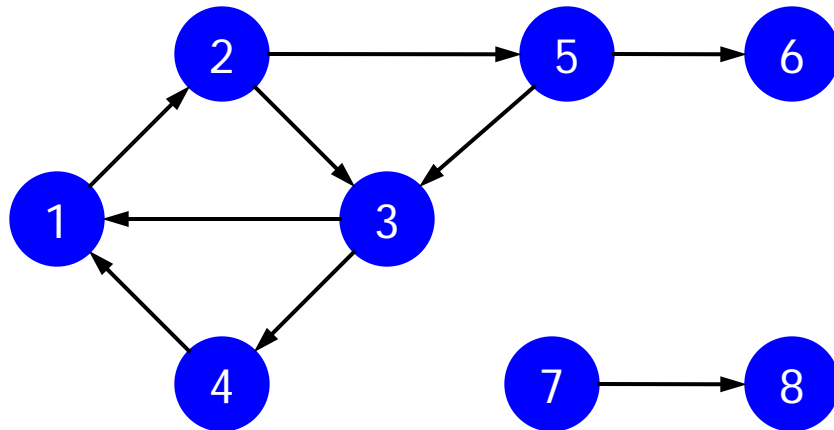


# Búsqueda en profundidad DFS

- DFS tiene las siguientes reglas:

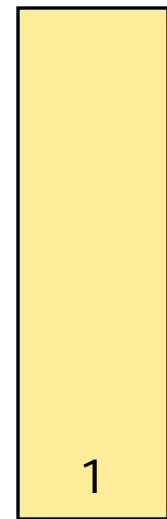
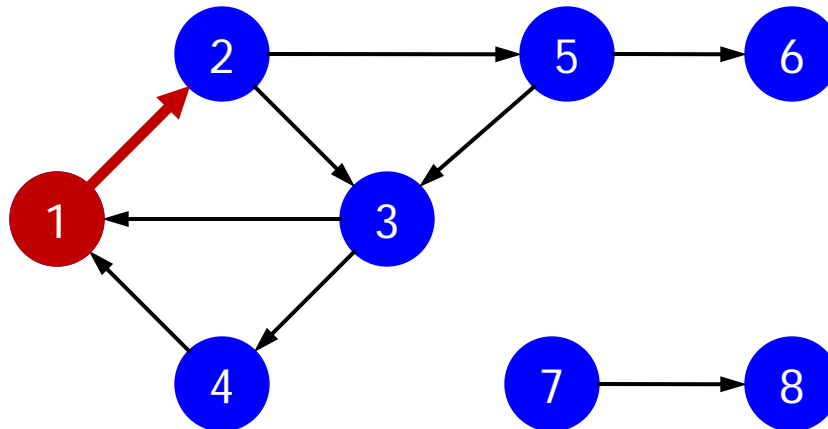
1. Seleccione un nodo no visitado X, visite este. Éste se denomina nodo actual.
2. Busque un nodo no visitado vecino desde el nodo actual, visite este. Éste será el nuevo nodo actual.
3. Si el nodo actual tiene a todos los vecinos como visitados, **regrese** a su nodo padre, el nodo padre será el nuevo nodo actual
4. Repita los pasos 3 y 4 hasta que no se puedan visitar más nodos.
5. Si existen nodos no visitados, repita desde el paso 1.

# Búsqueda en profundidad



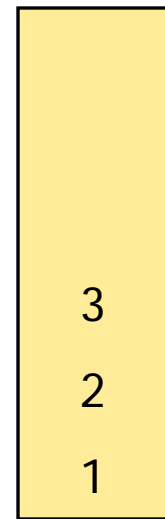
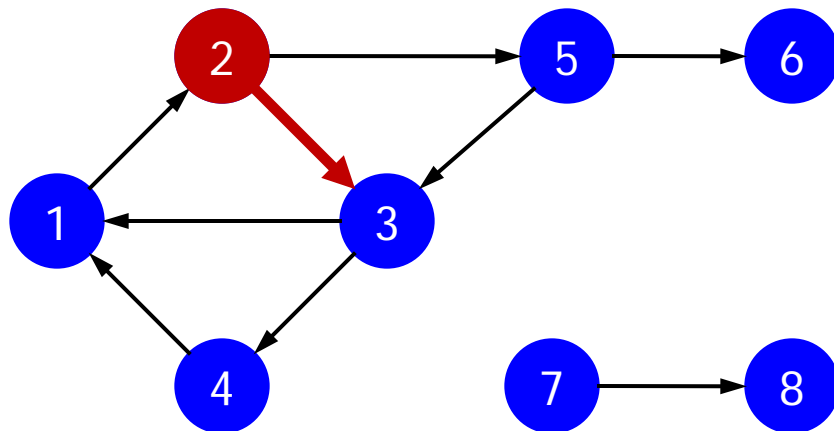
Pila

# Búsqueda en profundidad



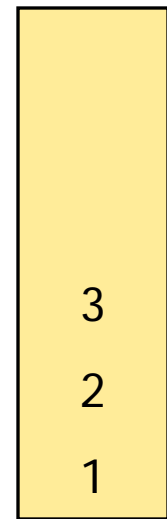
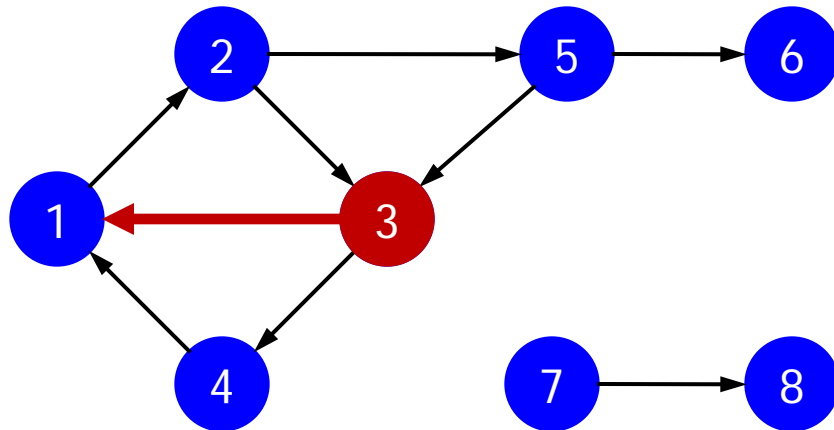
Pila

# Búsqueda en profundidad



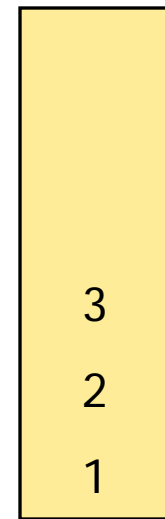
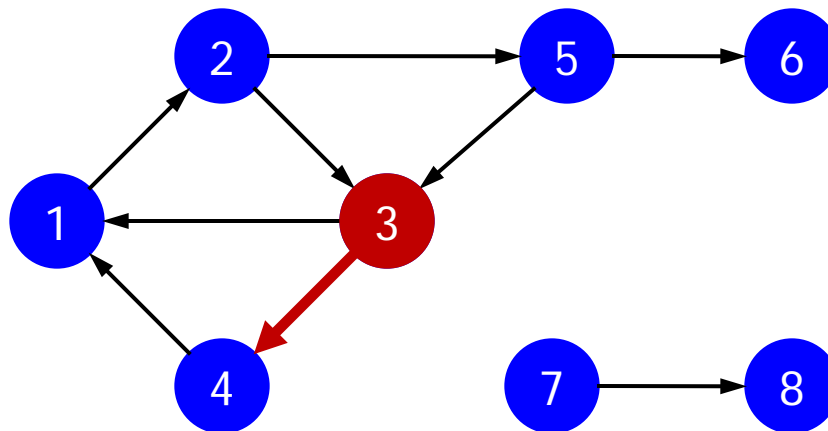
Pila

# Búsqueda en profundidad



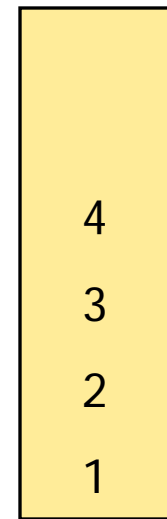
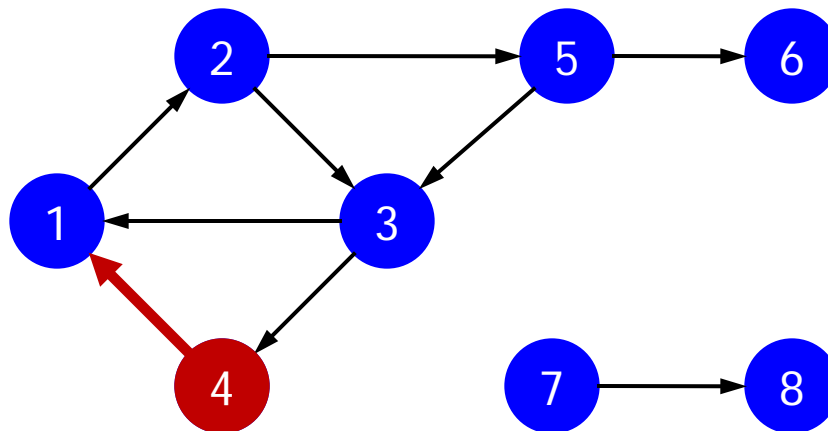
Pila

# Búsqueda en profundidad



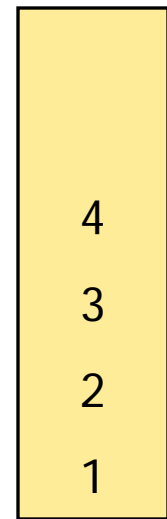
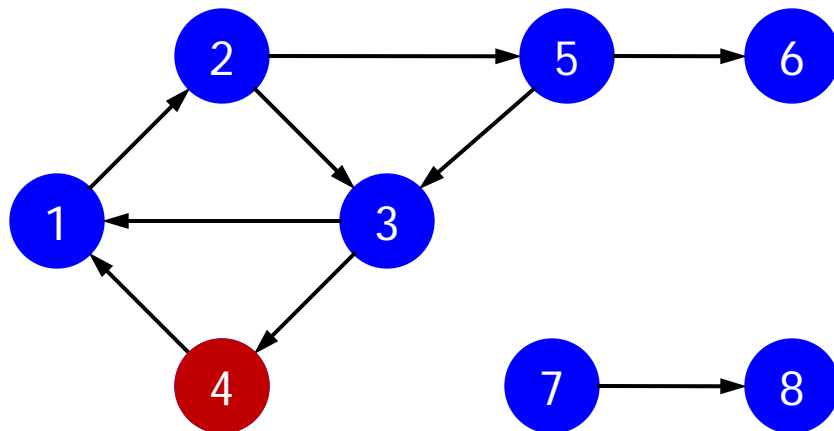
Pila

# Búsqueda en profundidad



Pila

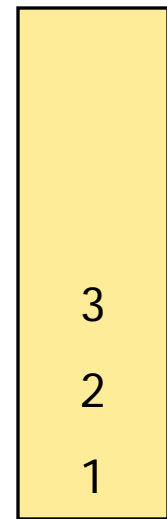
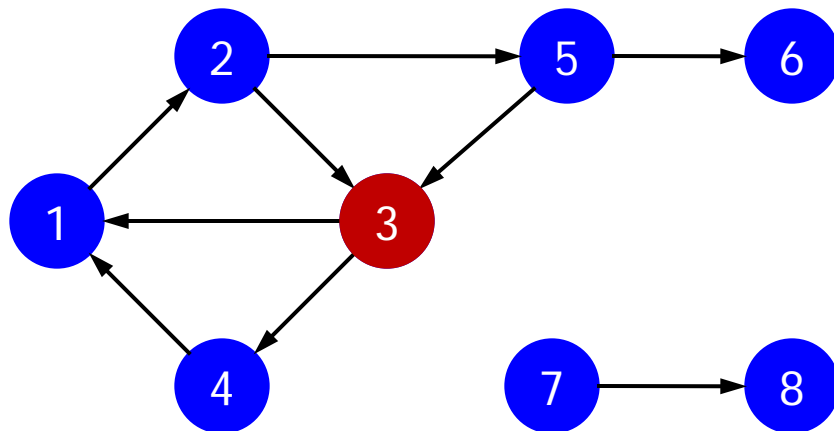
# Búsqueda en profundidad



Pila

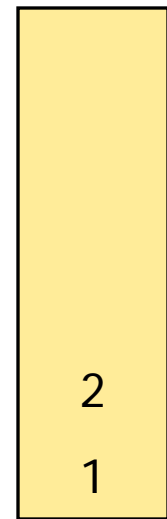
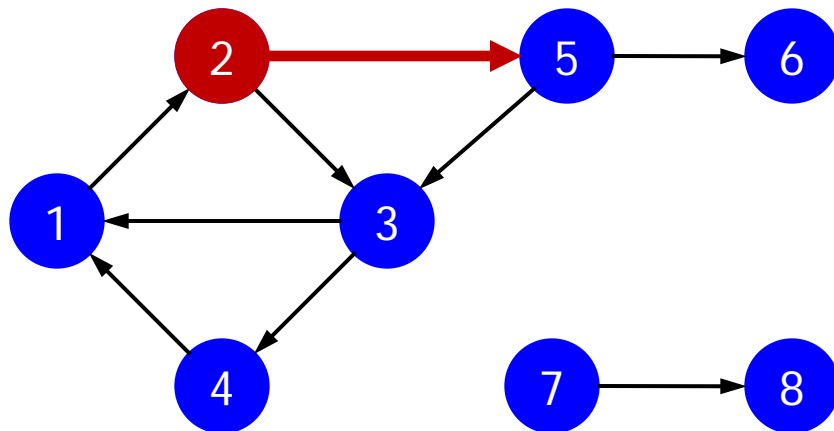


# Búsqueda en profundidad



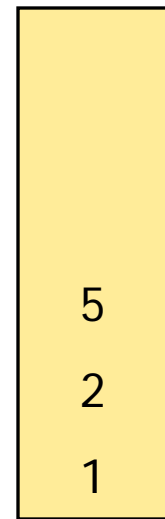
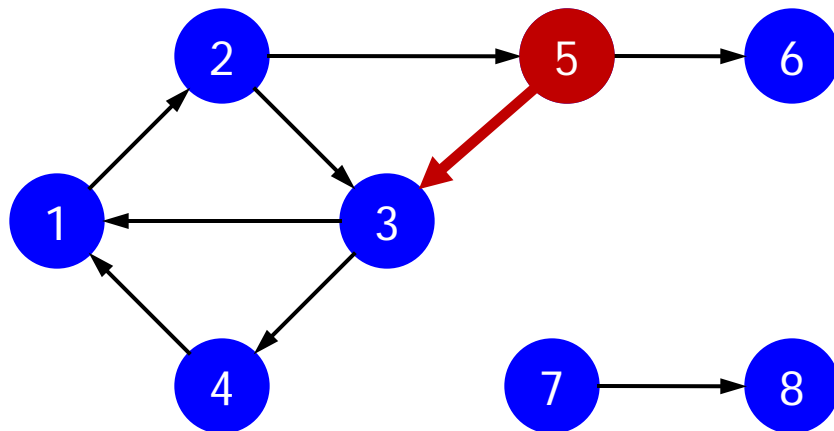
Pila

# Búsqueda en profundidad



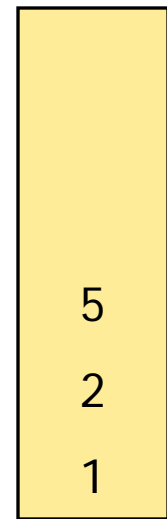
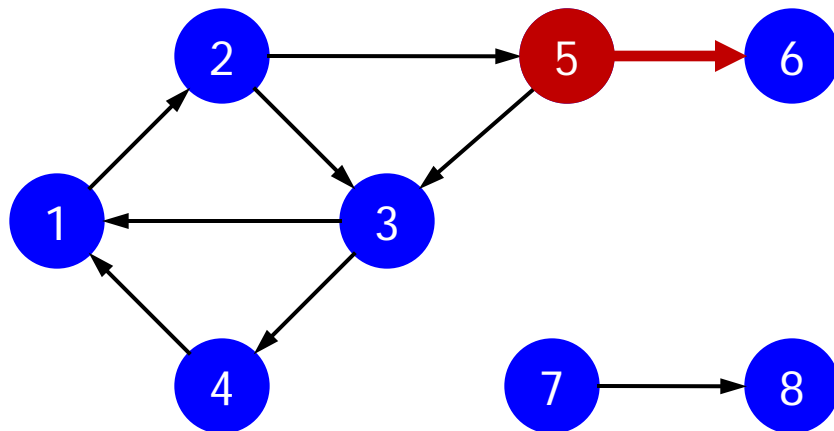
Pila

# Búsqueda en profundidad



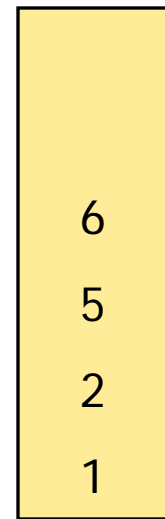
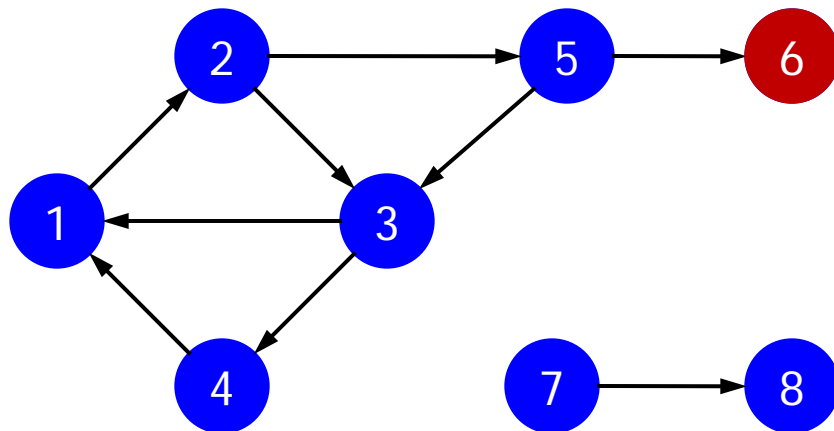
Pila

# Búsqueda en profundidad



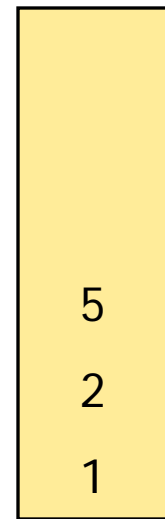
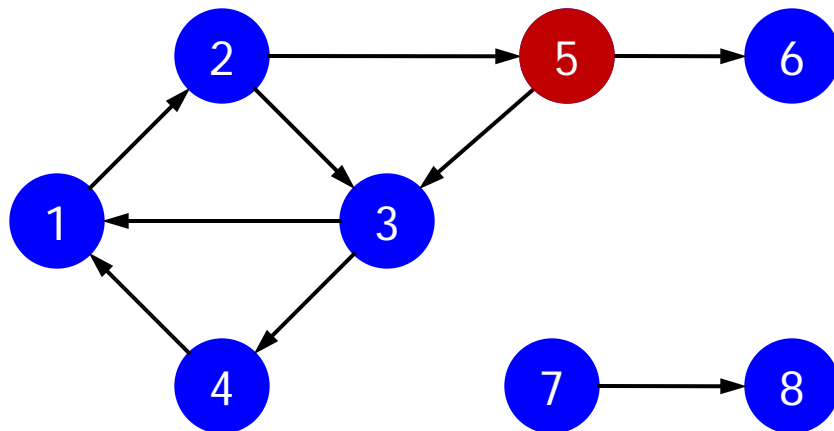
Pila

# Búsqueda en profundidad



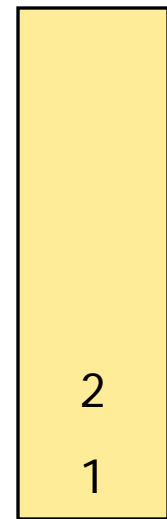
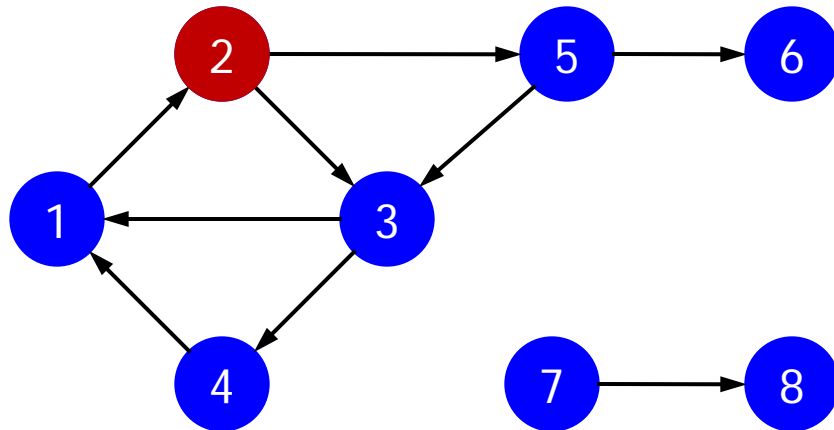
Pila

# Búsqueda en profundidad



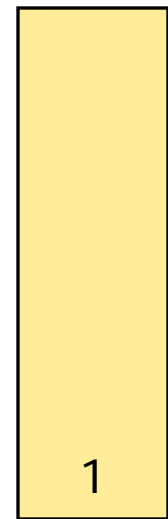
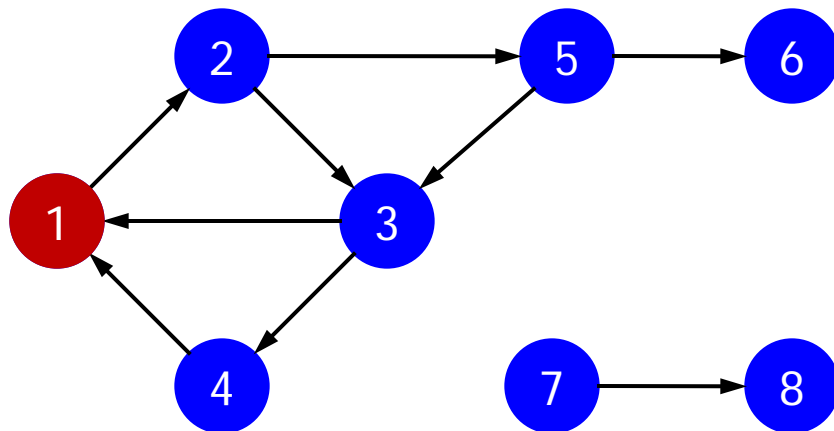
Pila

# Búsqueda en profundidad



Pila

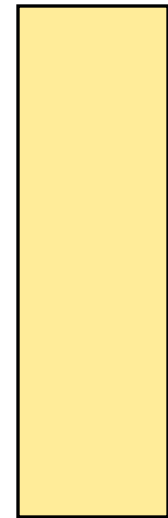
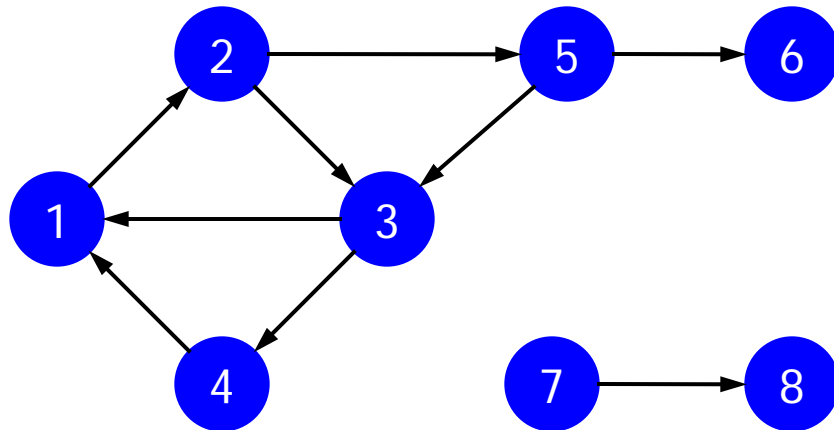
# Búsqueda en profundidad



Pila

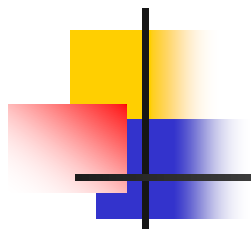


# Búsqueda en profundidad



Pila

```
public void dfs() { // búsqueda en profundidad
    Stack <Integer> pila = new Stack<Integer>();
    vertices[0].visitado = true;
    despliegaVertice(0);
    pila.push(0);
    while (!pila.isEmpty()) {
        int v = obtVerticeAdyNoVisitado(pila.peek());
        if(v == -1)
            pila.pop();
        else {
            vertices[v].visitado = true;
            despliegaVertice(v);
            pila.push(v);
        }
    }
    for (int j = 0; j < numvertices; j++)
        vertices[j].visitado = false;
} // fin dfs()
```



```
<terminated> Aplicacion (1) [Java Application] C:\Program  
Vertices visitados (DFS) : 1 2 3 4 5 6
```



# Bibliografía

---

- Data Structures & Algorithms in Java, 2da Ed, Robert Lafore, 2003.



# Taller de Programación

---

Gracias