

# Grafos – Inicial

Programación Competitiva

# Definición

## Grafo (Wikipedia)

Un grafo es un conjunto de objetos llamados  **Vértices** o **Nodos** (**Vertex** ) , unidos por enlaces llamados  **Aristas** o **Arcos** (**Edges** )

# Definición

## Grafo (Wikipedia)

Un grafo es un conjunto de objetos llamados  **Vértices** o **Nodos** (**Vertex** ) , unidos por enlaces llamados  **Aristas** o **Arcos** (**Edges** )



**Figura 1:**



Grafo Dirigido



Directed Graph/Digraph



**Figura 2:**



Grafo no Dirigido

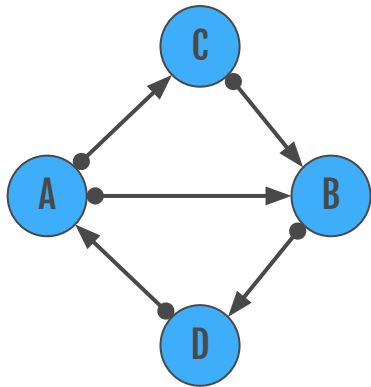


Undirected Graph/Undigraph

# Definición

## Grafo (Wikipedia)

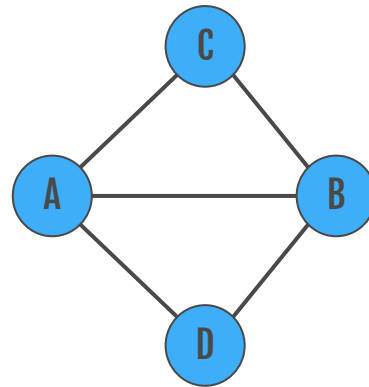
Un grafo es un conjunto de objetos llamados  **Vértices** o **Nodos** (Vertex ) , unidos por enlaces llamados  **Aristas** o **Arcos** (Edges )



**Figura 1:**

 Grafo Dirigido

 Directed Graph/Digraph



**Figura 2:**

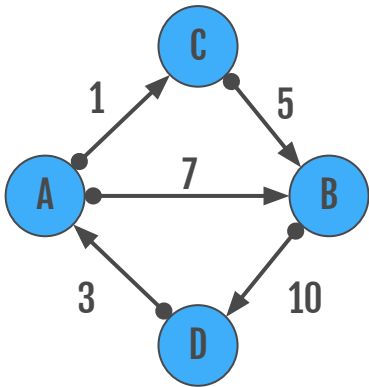
 Grafo no Dirigido

 Undirected Graph/Undigraph

# Definición

## Grafo (Wikipedia)

Un grafo es un conjunto de objetos llamados  **Vértices** o **Nodos** (Vertex ) , unidos por enlaces llamados  **Aristas** o **Arcos** (Edges )



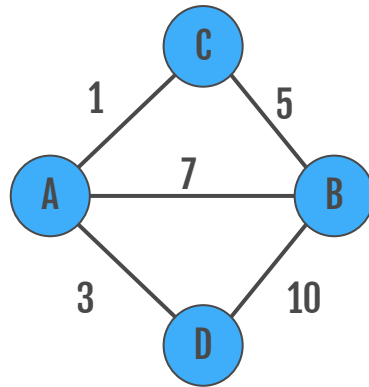
**Figura 1:**



Grafo Dirigido ponderado



Directed Weighted Graph



**Figura 2:**



Grafo no Dirigido ponderado



Undirected Weighted Graph

# Ejemplos Prácticos

## Redes sociales

- En Facebook, dos usuarios, pueden o no ser amigos. Esta relación es mutua, es decir que puede pensarse como un **grafo no dirigido**, en donde cada usuario es un **nodo** y cada relación de amistad es una **arista**.
- En Instagram, un usuario puede seguir a otro, sin necesidad que ese otro usuario lo siga a uno. Es decir que puede pensarse como un **grafo dirigido**, en donde cada usuario es un nodo y cada relación seguidor-seguido es una arista.

En general, dada cualquier situación en la que tenemos pares de cosas relacionadas, probablemente sirve verla como un grafo.

# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

**Uso de memoria:**  **$O(n^2)$**

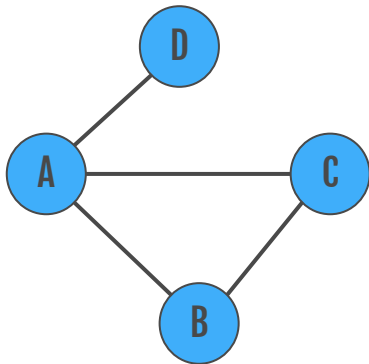
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

Grafo no Dirigido



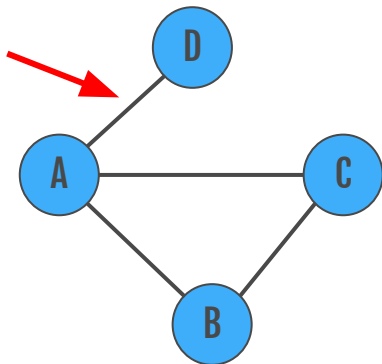
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

Grafo no Dirigido

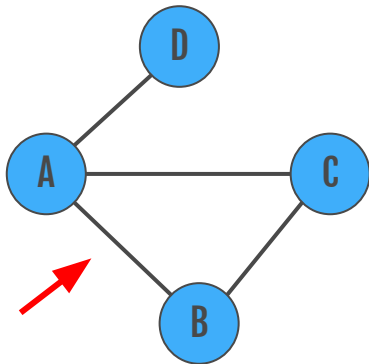
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo no Dirigido

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

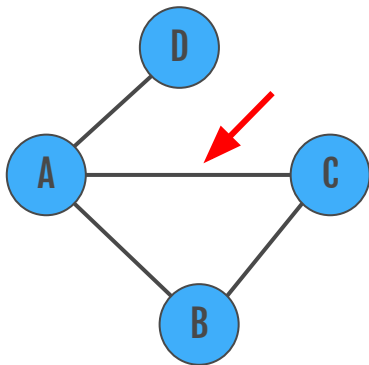
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo no Dirigido

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

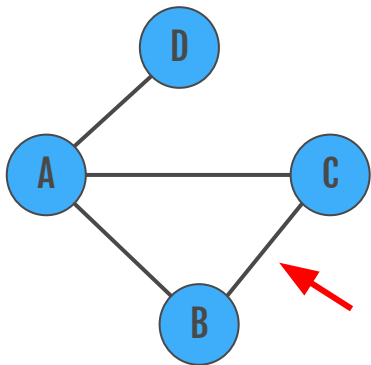
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  **$i$**  y el nodo  **$j$**  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo no Dirigido

	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

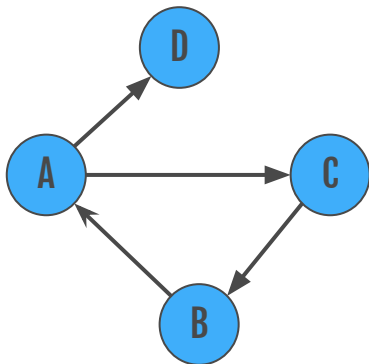
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  **$i$**  y el nodo  **$j$**  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



	A	B	C	D
A	0	0	1	1
B	1	0	0	0
C	0	1	0	0
D	0	0	0	0

Grafo Dirigido

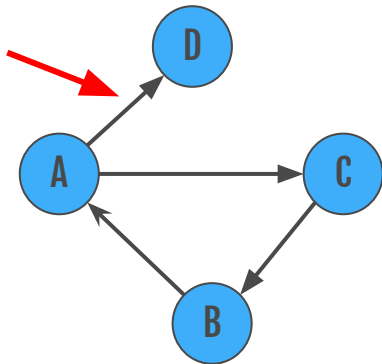
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



	A	B	C	D
A	0	0	1	1
B	1	0	0	0
C	0	1	0	0
D	0	0	0	0

Grafo Dirigido

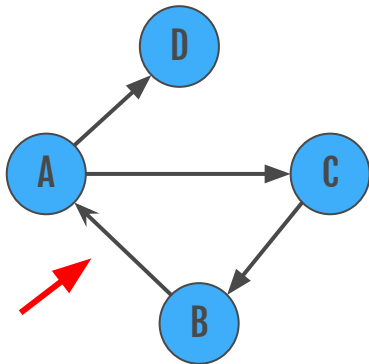
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo Dirigido

	A	B	C	D
A	0	0	1	1
B	1	0	0	0
C	0	1	0	0
D	0	0	0	0

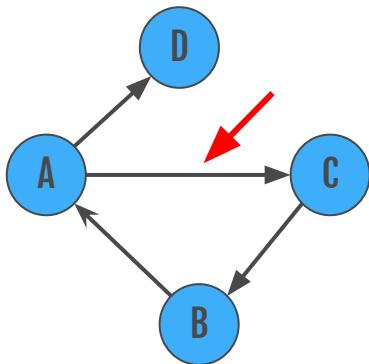
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo Dirigido

	A	B	C	D
A	0	0	1	1
B	1	0	0	0
C	0	1	0	0
D	0	0	0	0



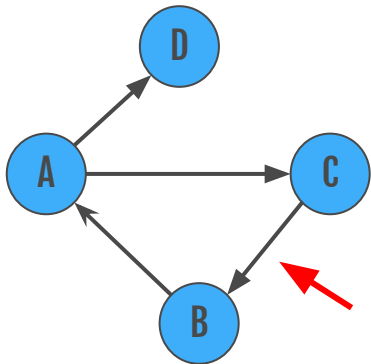
# Representación de Grafos

## Matriz de Adyacencia

La **matriz de adyacencia** es una matriz booleana de  **$N \times N$**  donde  **$N$**  es la cantidad de nodos del grafo.

Si en la posición  **$(i, j)$**  de la matriz tiene un **1 (o true)** es porque existe una arista entre el nodo  $i$  y el nodo  $j$  o **0 (o false)** en caso contrario donde no existe ninguna relación entre esos 2 nodos.

Uso de memoria:  **$O(n^2)$**



Grafo Dirigido

	A	B	C	D
A	0	0	1	1
B	1	0	0	0
C	0	1	0	0
D	0	0	0	0

# Implementación

## Matriz de Adyacencia

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1000;

int graph[N][N] = {0};

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        graph[u][v] = 1;
        graph[v][u] = 1;
    }
    return 0;
}
```

Grafo no Dirigido

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1000;

int graph[N][N] = {0};

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        graph[u][v] = 1;
    }
    return 0;
}
```

Grafo Dirigido

# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia

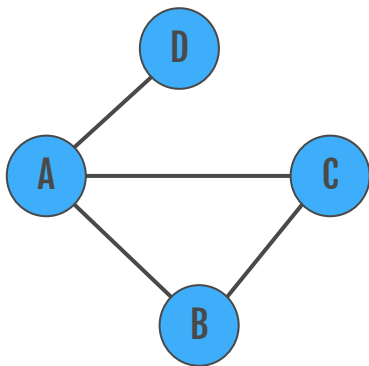
# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia



Grafo no Dirigido

A: {B, C, D}
B: {A, C}
C: {A, B}
D: {A}

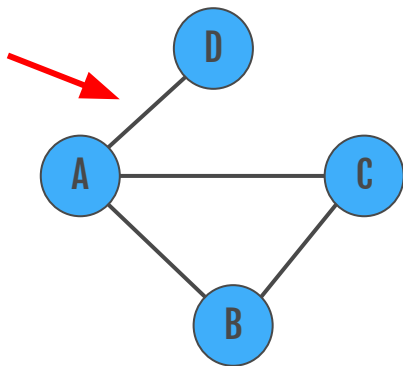
# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia



Grafo no Dirigido

A: {B, C, <span style="border: 1px solid red;">D</span> }
B: {A, C}
C: {A, B}
D: { <span style="border: 1px solid red;">A</span> }

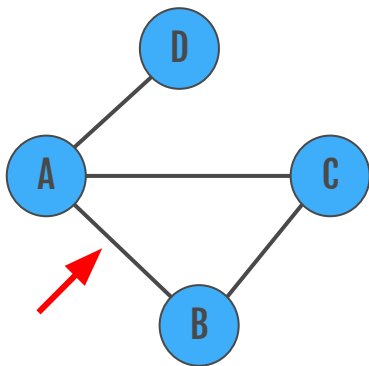
# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia



Grafo no Dirigido

A: {B, C, D}
B: {A, C}
C: {A, B}
D: {A}

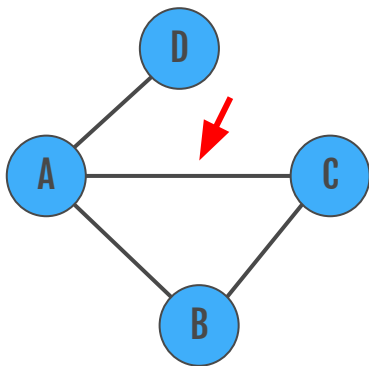
# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia



Grafo no Dirigido

A: {B, <span style="border: 1px solid red;">C</span> , D}
B: {A, C}
C: { <span style="border: 1px solid red;">A</span> , B}
D: {A}

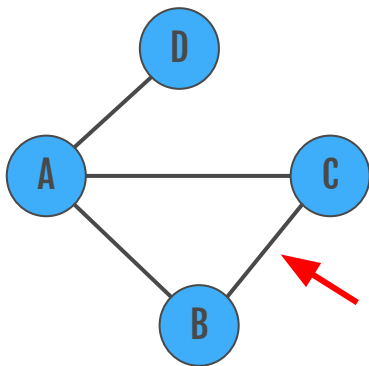
# Representación de Grafos

## Lista de Adyacencia

La **lista de adyacencia** es un vector de vectores de enteros, que en el **i-esimo** vector tiene el numero **j** si hay una arista entre los nodos **i** y **j**.

**Uso de memoria:**  $O(m)$ , donde **m** es la cantidad de aristas.

Mucho más utilizada en Programación Competitiva que la Matriz de Adyacencia



Grafo no Dirigido

A: {B, C, D}
B: {A, C}
C: {A, B}
D: {A}



# Implementación (Opción 1)

## Lista de Adyacencia

```
#include <bits/stdc++.h>
using namespace std;

const int N = 2e5 + 2;
int n, m;
vector<int> adj[N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    return 0;
}
```

Grafo no Dirigido

```
#include <bits/stdc++.h>
using namespace std;
```

```
const int N = 2e5 + 2;
int n, m;
vector<int> adj[N];

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }
    return 0;
}
```

Grafo Dirigido

# Implementación (Opción 2)

## Lista de Adyacencia

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
vector<vector<int>> adj;

int main() {
    cin >> n >> m;
    adj.resize(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    return 0;
}
```

Grafo no Dirigido

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
vector<vector<int>> adj;

int main() {
    cin >> n >> m;
    adj.resize(n);
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
    }
    return 0;
}
```

Grafo Dirigido



¿Preguntas?



The image features a light blue, cloud-like shape in the center containing the text '¿Preguntas?'. Surrounding this central element is a decorative network diagram consisting of several blue circular nodes of varying sizes, some with concentric circles, connected by thin black lines. The nodes are positioned at the top right and bottom left corners, with ellipses indicating the network continues beyond the visible nodes.

# Algunas Definiciones

## Definición: Vecino y Grado

- Dada una arista que conecta dos vértices  $u$  y  $v$ , decimos que  $u$  es vecino de  $v$  (y que  $v$  es vecino de  $u$ ). Además, a la cantidad de vecinos de  $u$  se le llama el **grado** de  $u$ .

## Definición: Distancia

- Definimos la distancia de  $u$  a  $v$  como el menor  $n$  tal que hay un camino de largo  $n$  de  $u$  a  $v$

# ¿Como recorrer un Grafo?

## DFS ( Depth-First-Search o Busqueda en Profundidad)

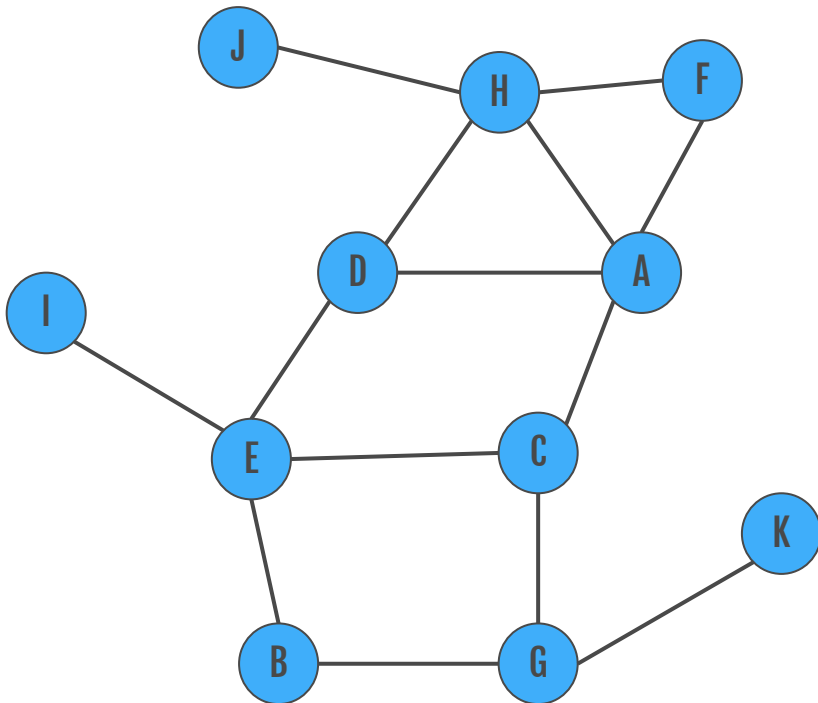
Funciona de la siguiente manera:

- Arrancamos de cierto nodo y lo marcamos como visitado.
- Luego, evaluamos sus vecinos de a uno, y cada vez que encontramos uno que no esté marcado como visitado, seguimos procesando a partir de él.
- Cuando no quedan vecinos por visitar salgo para atrás (vuelvo en la recursión).
- Este procedimiento es recursivo, por lo que se puede implementar así.

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

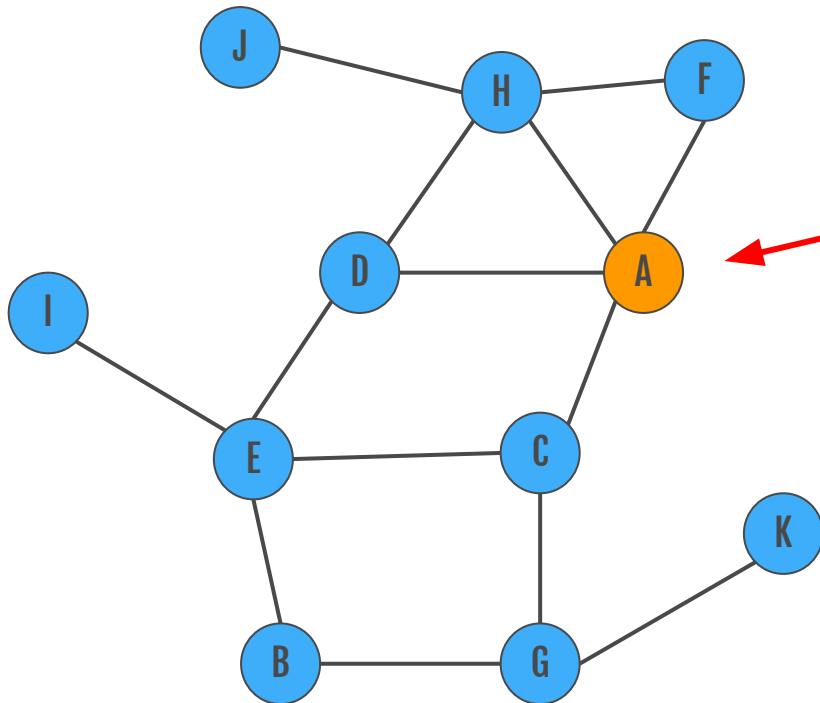
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:

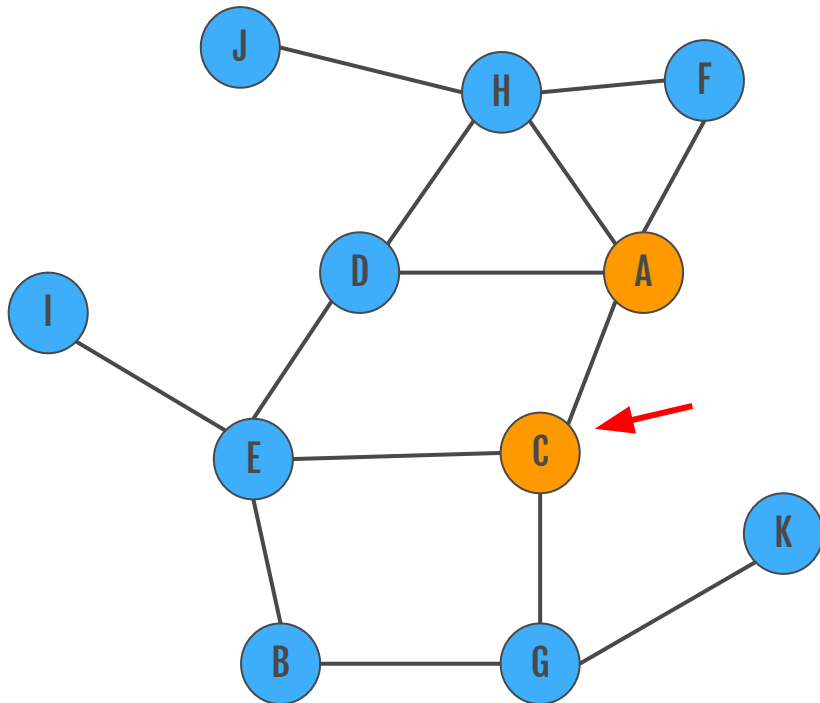


No visitado  
Visitado y en stack  
Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

Visitado y en stack

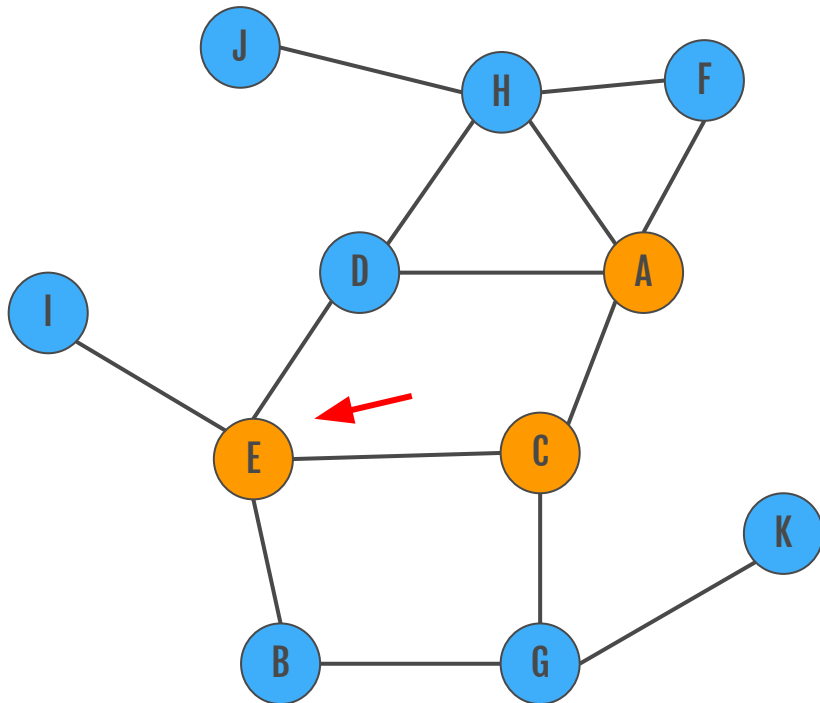
Visitado y fuera del stack



# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

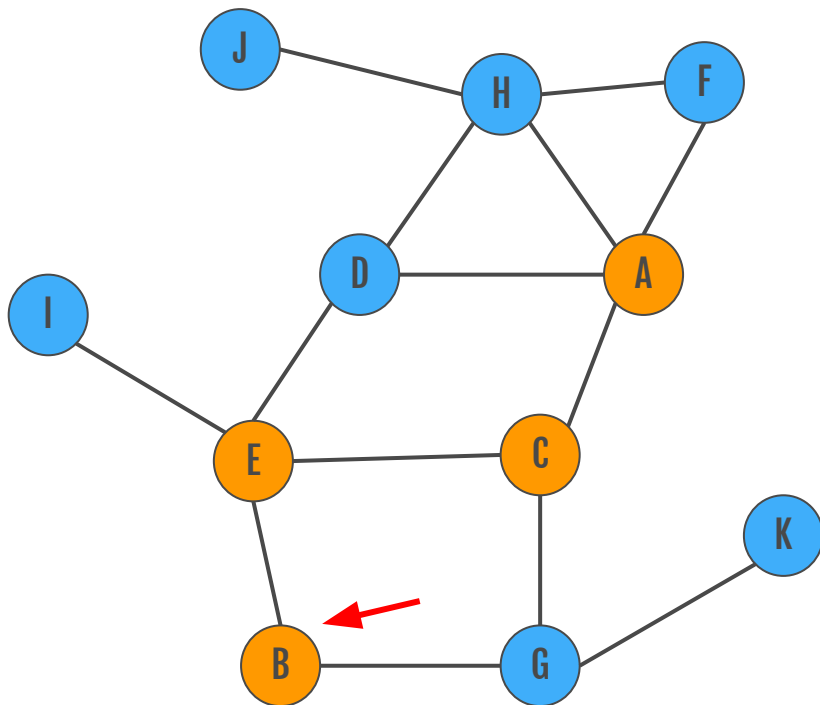
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

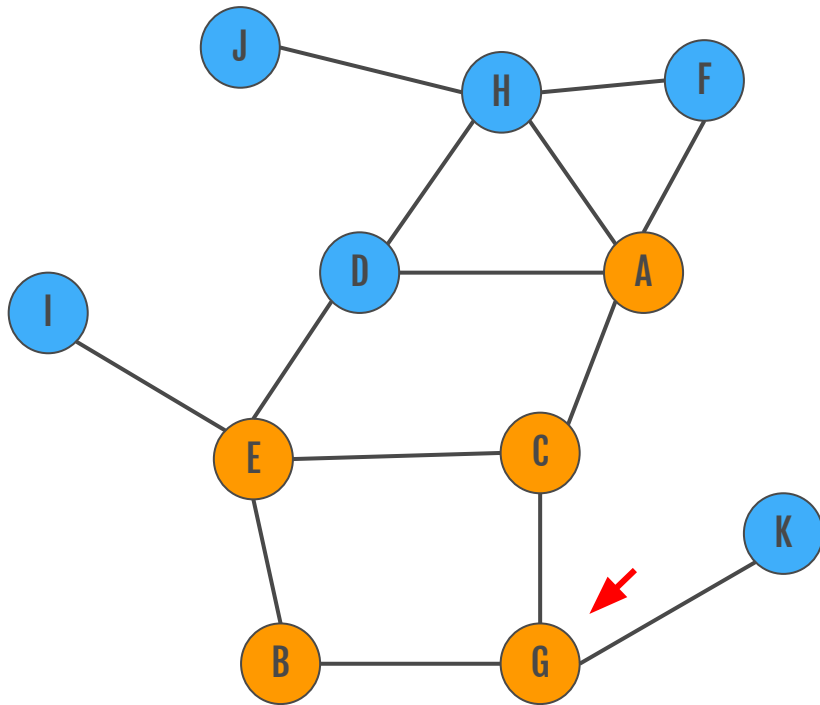
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

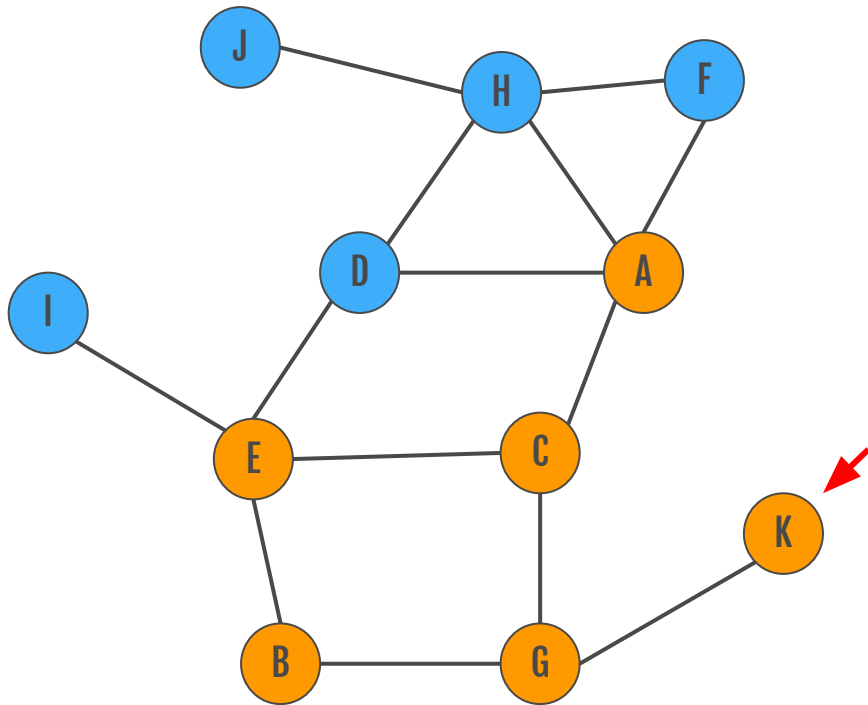
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

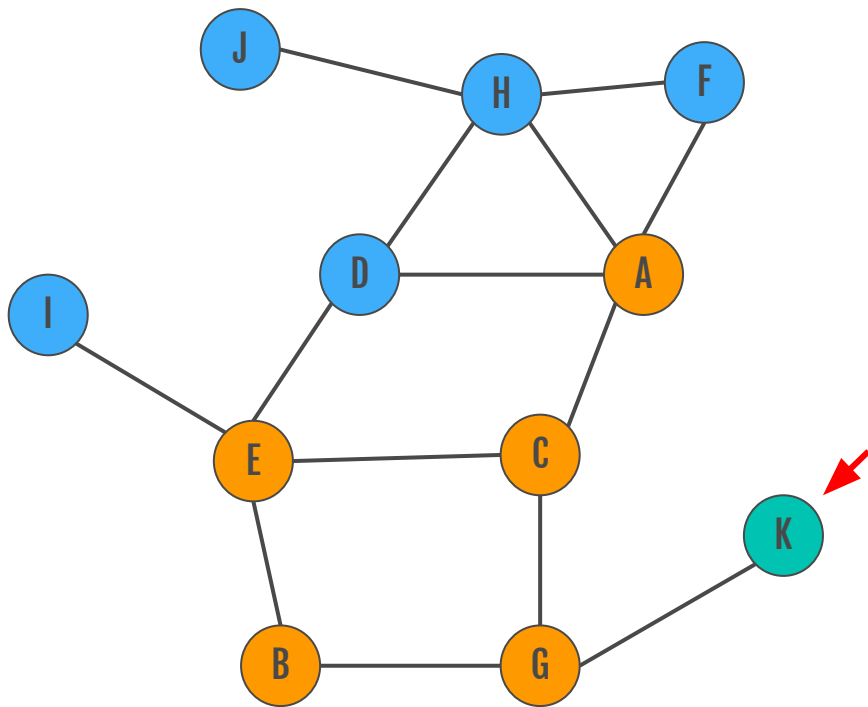
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:

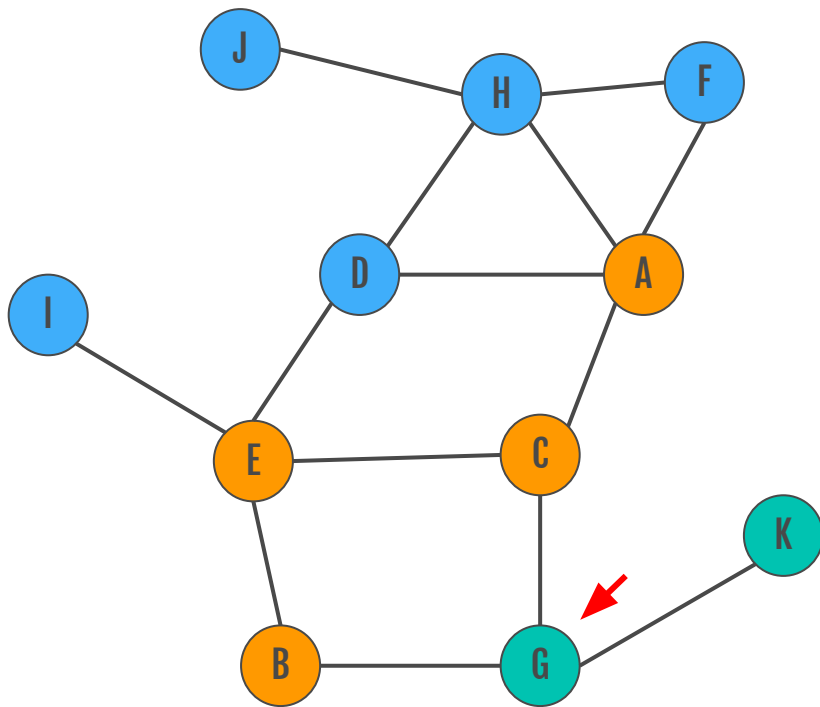


No visitado  
Visitado y en stack  
Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

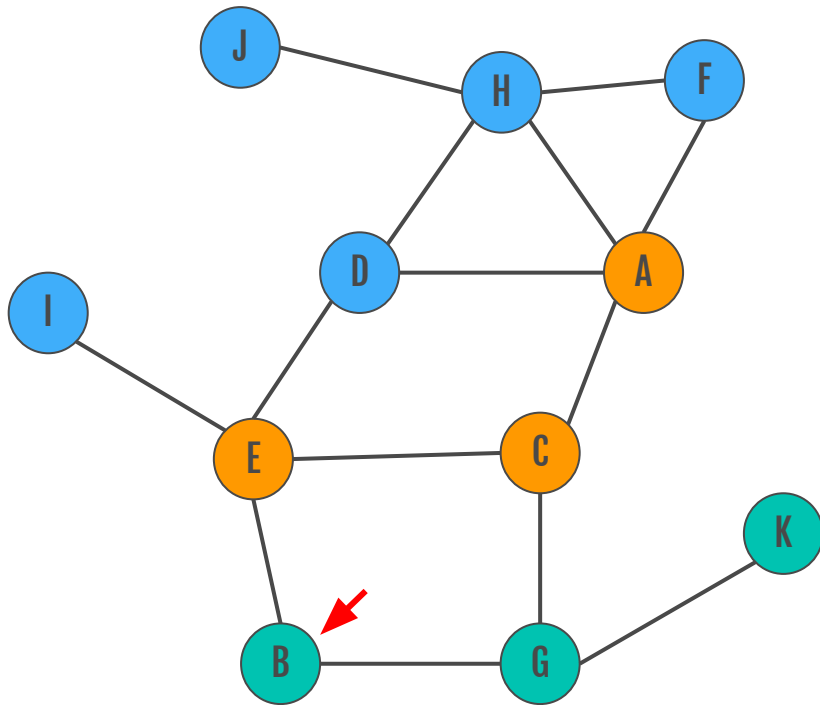
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

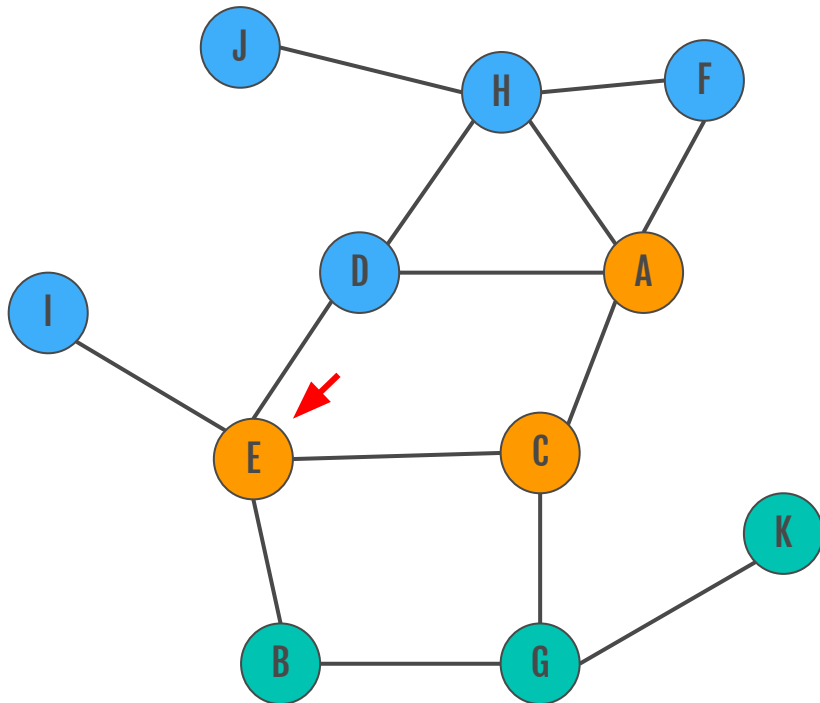
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

Visitado y en stack

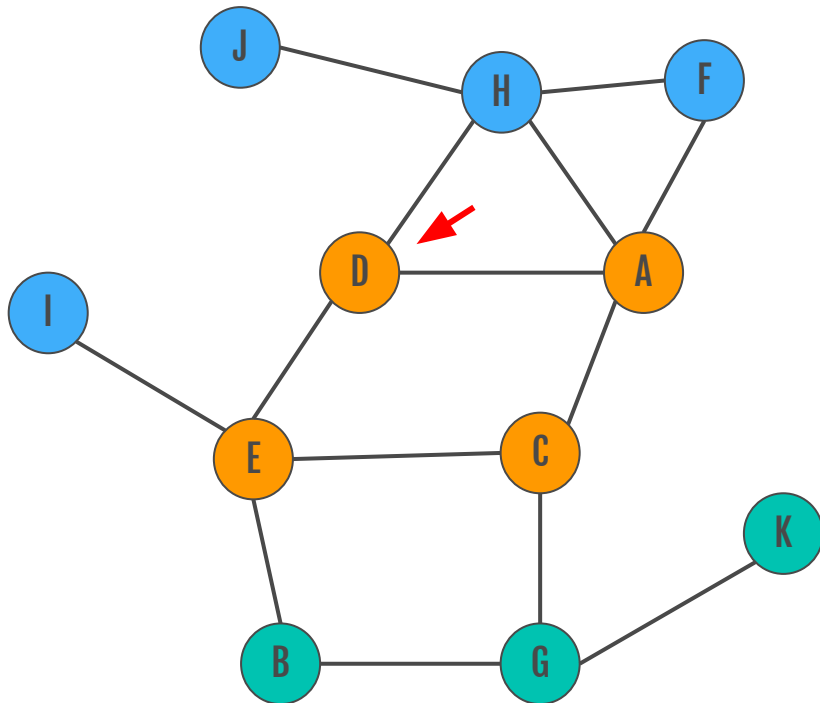
Visitado y fuera del stack



# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

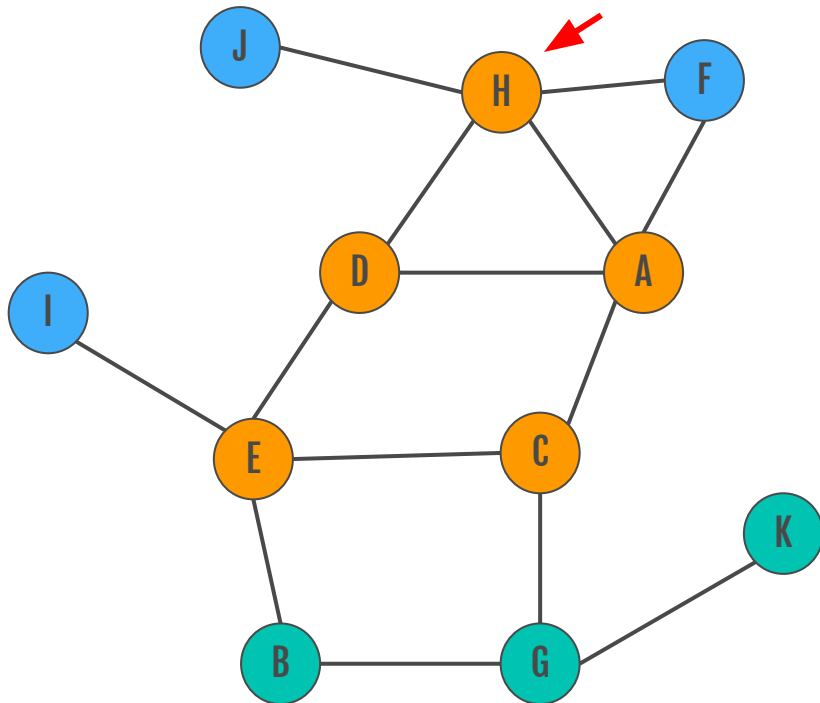
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

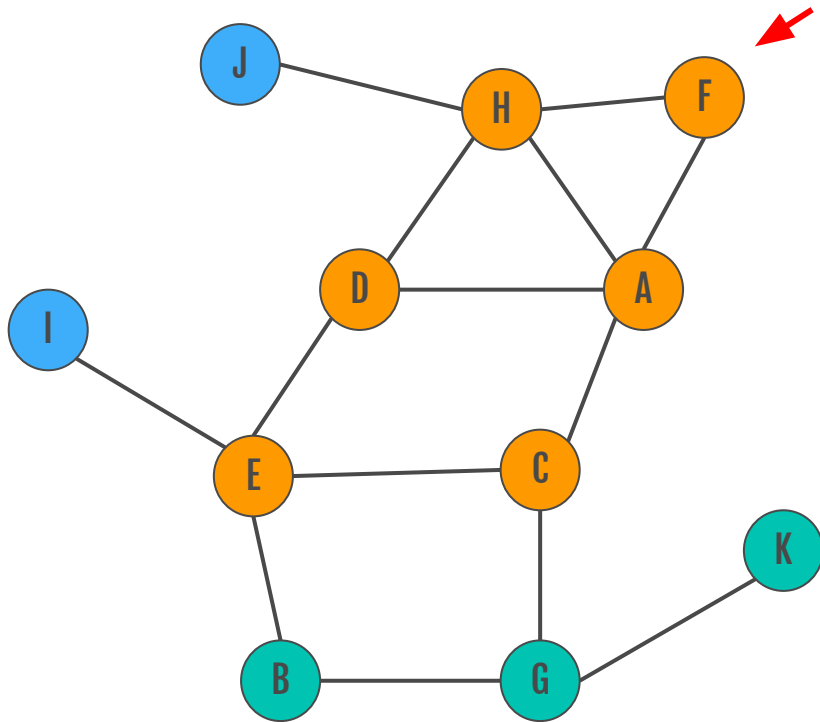
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

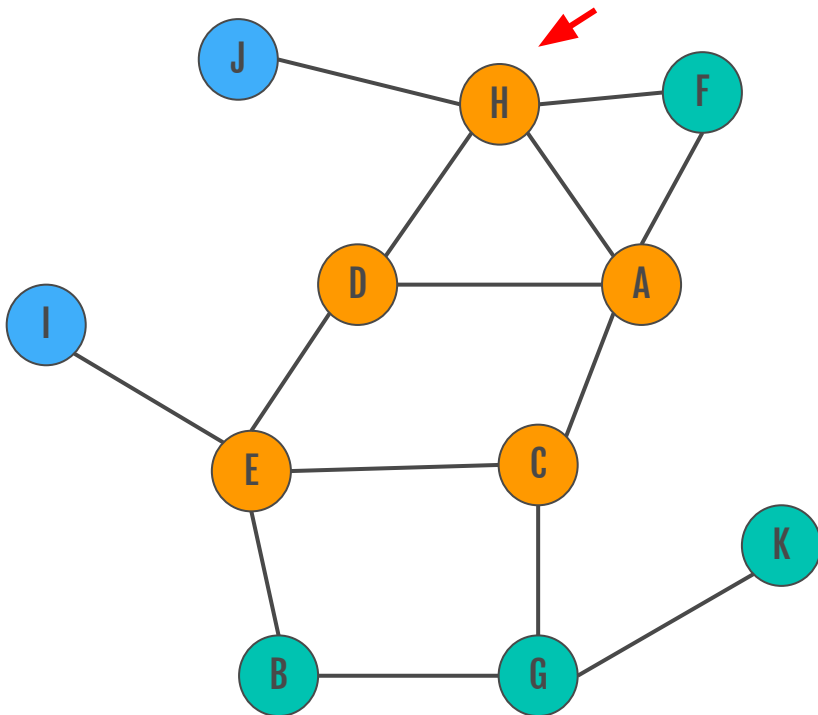
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

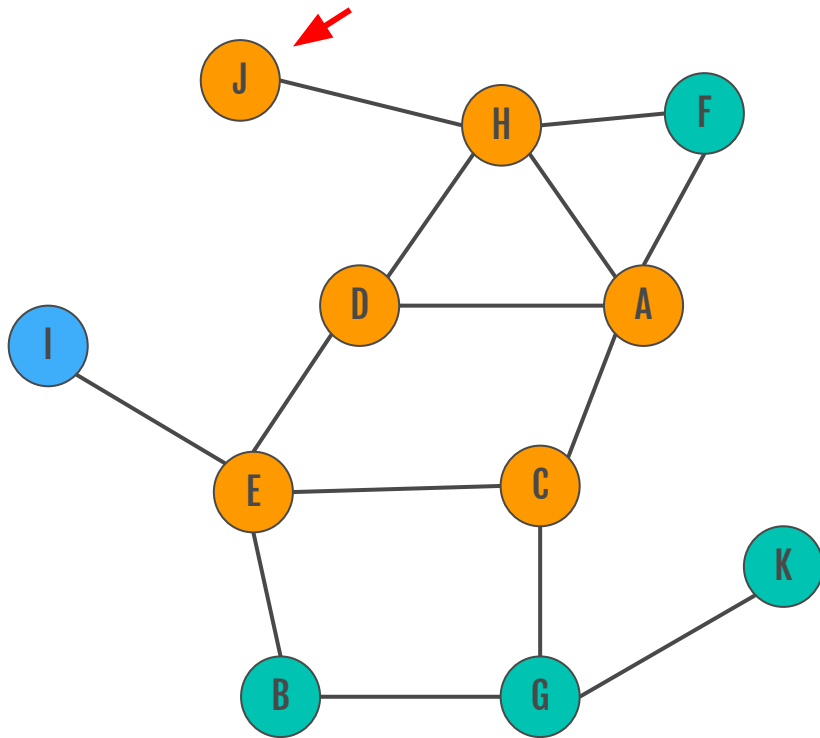
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

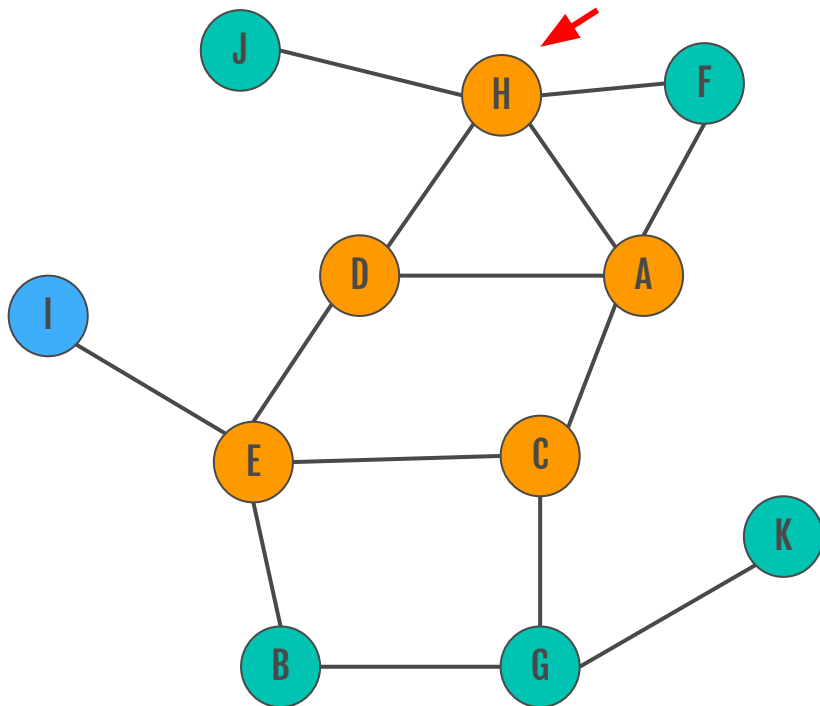
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

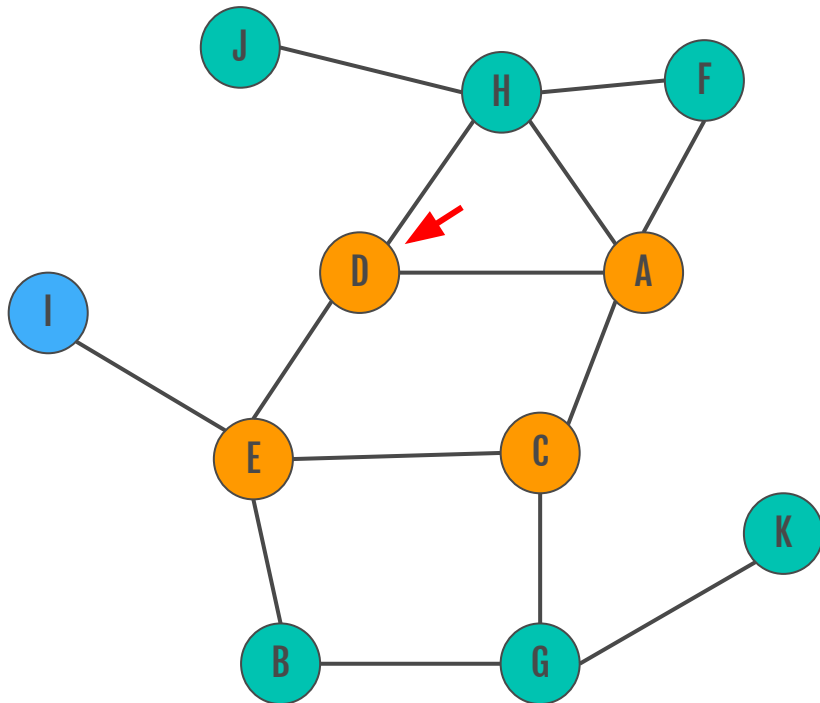
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



No visitado

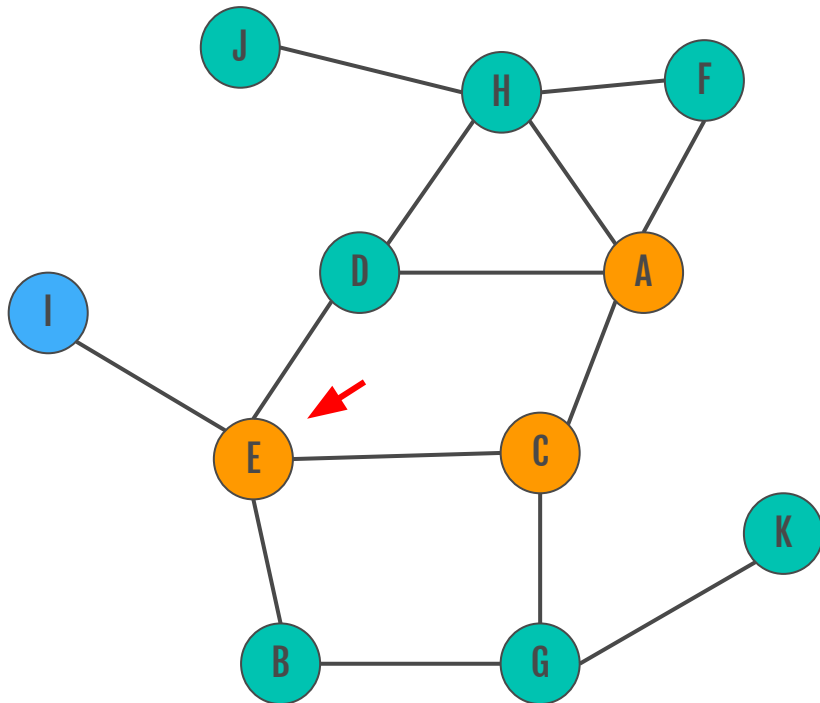
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:



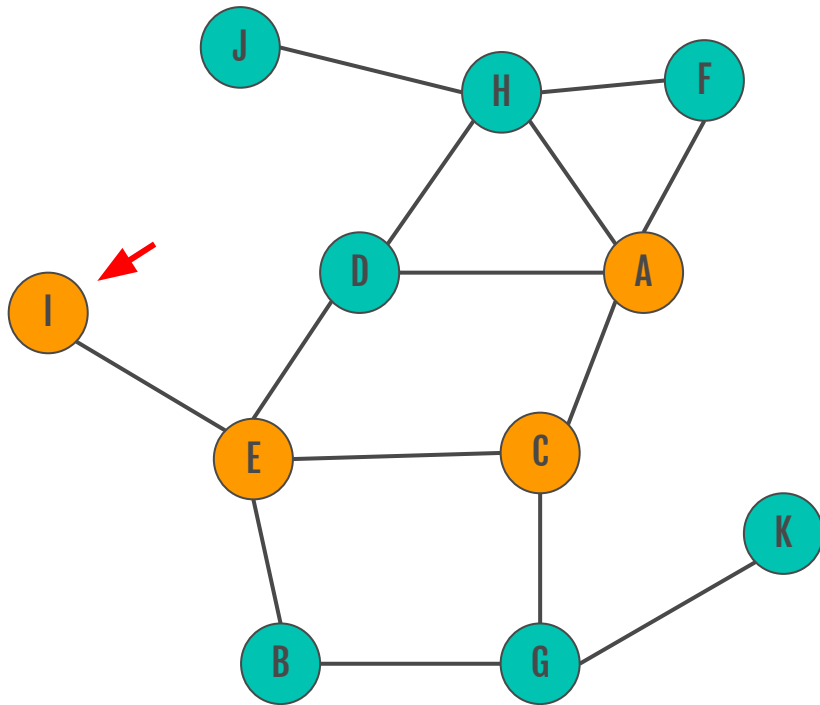
No visitado  
Visitado y en stack  
Visitado y fuera del stack



# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:

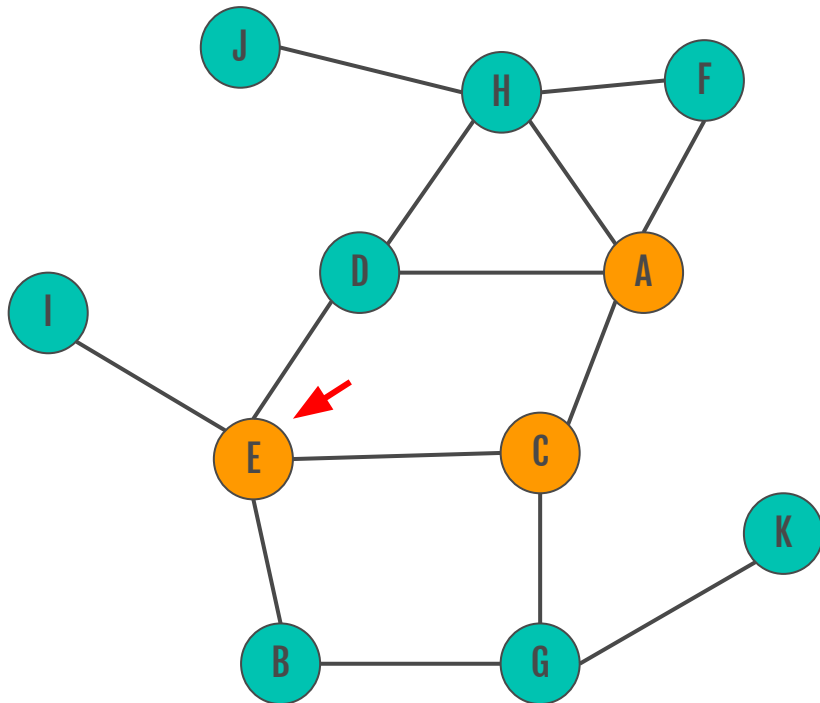


No visitado  
Visitado y en stack  
Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Búsqueda en Profundidad)

Simulación Gráfica:

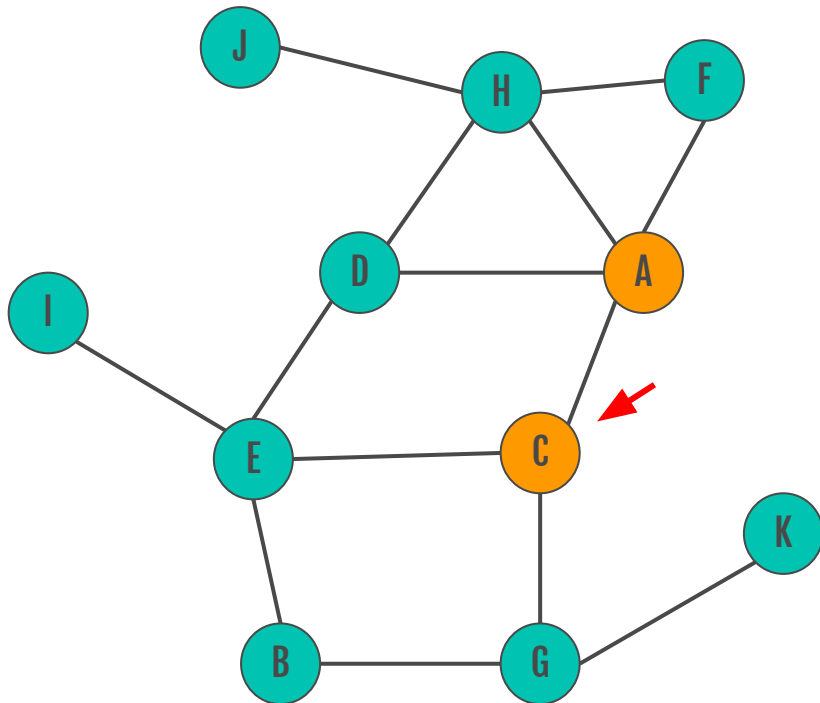


No visitado  
Visitado y en stack  
Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

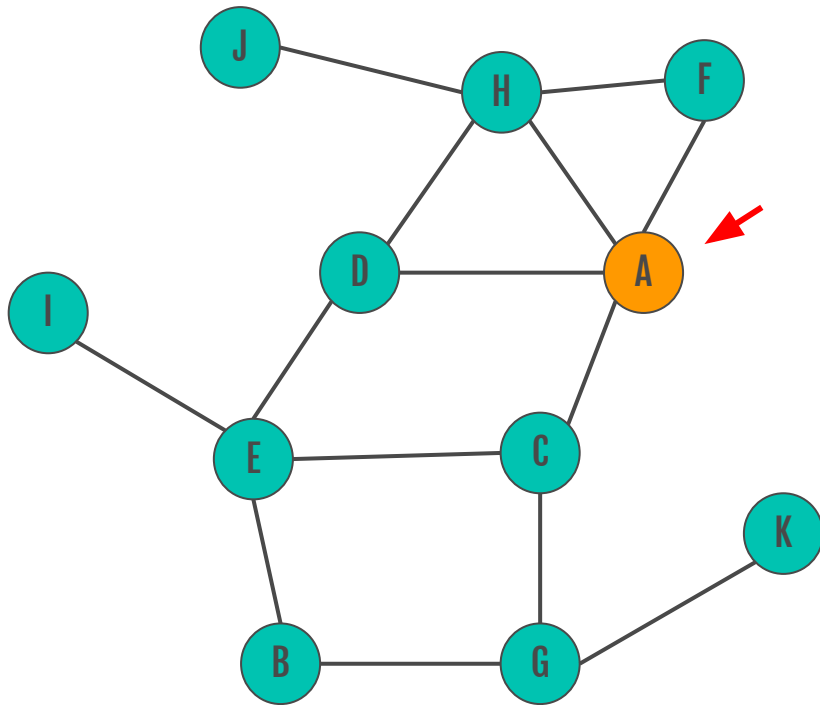
Visitado y en stack

Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:

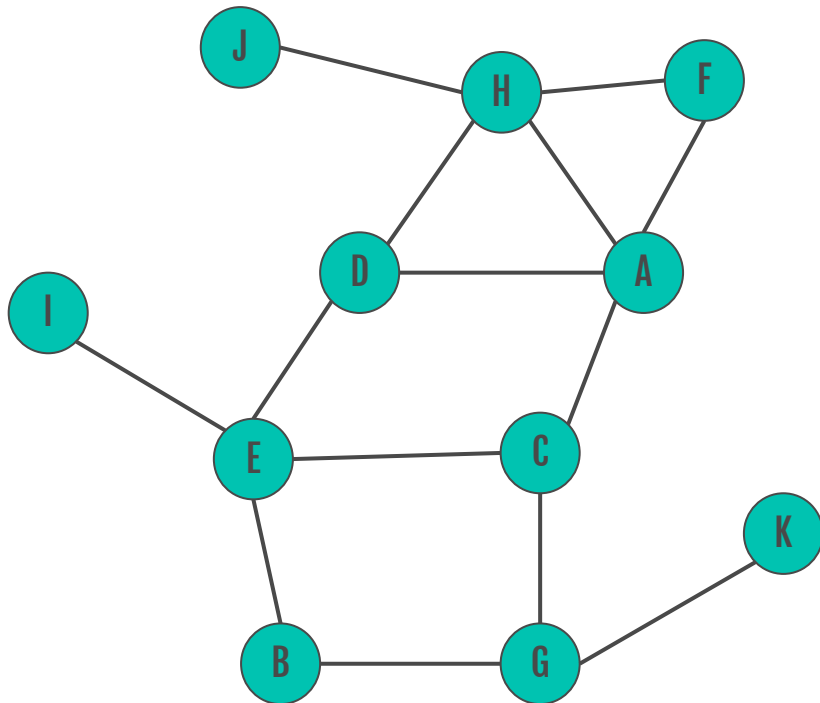


No visitado  
Visitado y en stack  
Visitado y fuera del stack

# ¿Como recorrer un Grafo?

DFS (🇺🇸 Depth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



No visitado

Visitado y en stack

Visitado y fuera del stack

# Implementación

## Implementación Recursiva DFS

```
void dfs(int u) {  
    visited[u] = 1;  
    for (int v : adj[u]) {  
        if (visited[v])  
            continue;  
        dfs(v);  
    }  
}  
  
// Usage:  
// dfs(0);
```

```
for (int i = 0; i < n; ++i) {  
    if (!visited[i]) {  
        dfs(i);  
        count++;  
    }  
}
```

Misma implementación para Grafo no Dirigido y Dirigido



**¿Preguntas?**



# ¿Como recorrer un Grafo?

## BFS ( Breadth-First-Search o Busqueda en Anchura)

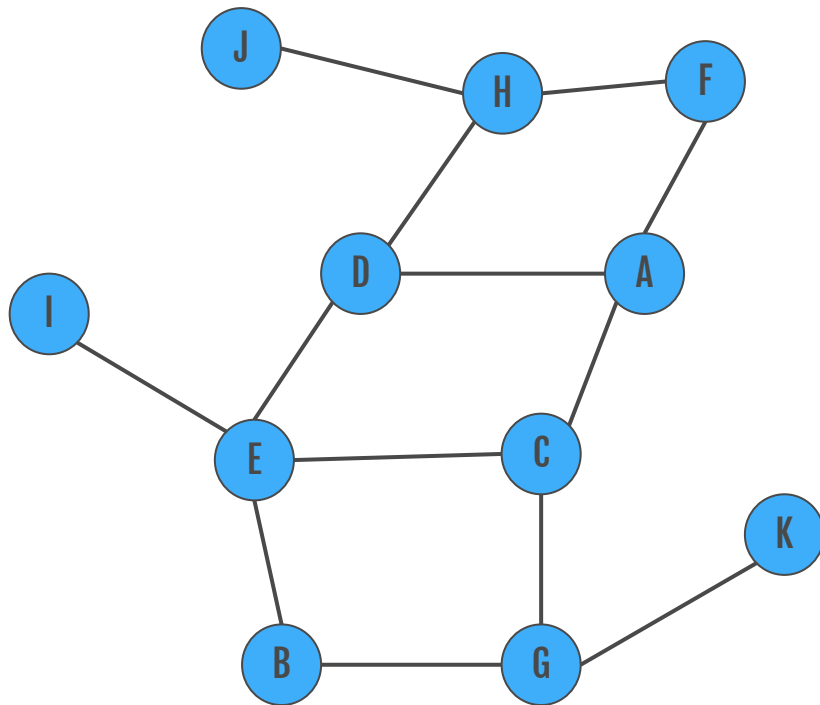
- 1) **Inicio:** Comienza desde un nodo inicial.
- 2) **Cola/Queue:** Utiliza una estructura de datos **Cola/Queue** para mantener un seguimiento de los nodos que necesitas visitar. Agrega el nodo inicial a esta fila.
- 3) **Visita el nodo:** Saca el primer nodo de la **cola** y visita sus vecinos.
- 4) **Vecinos:** Para cada vecino del nodo actual que no ha sido visitado, agrégalo a la fila y márcalo como visitado.
- 5) **Repetición:** Repite los pasos (3) y (4) hasta que hayas visitado todos los nodos alcanzables desde el nodo inicial.



# ¿Como recorrer un Grafo?

**BFS** ( **Breadth-First-Search** o  **Busqueda en Profundidad**)

Simulación Gráfica:

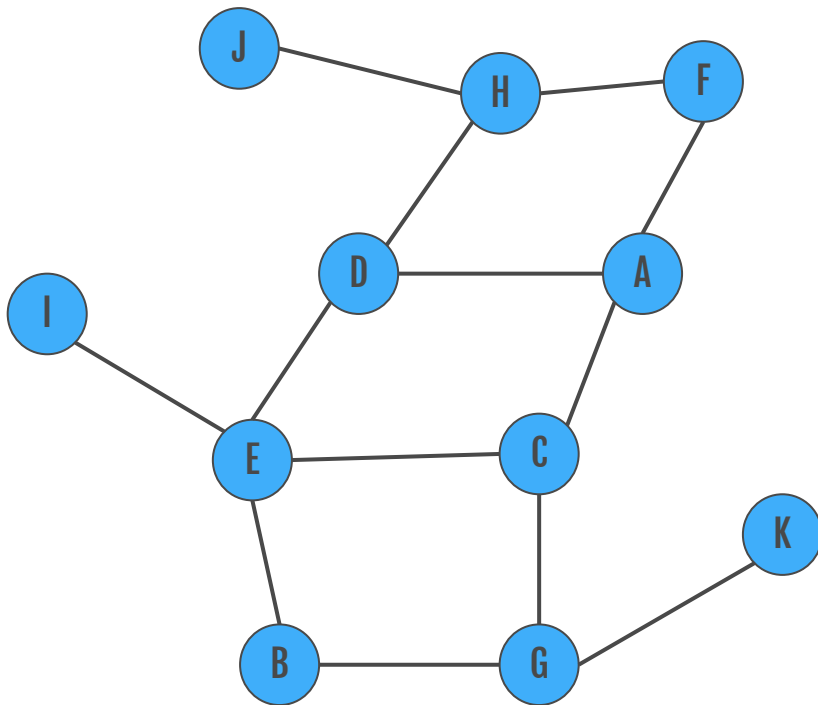


Queue: {}

# ¿Como recorrer un Grafo?

**BFS** ( **Breadth-First-Search** o  **Busqueda en Profundidad**)

Simulación Gráfica:

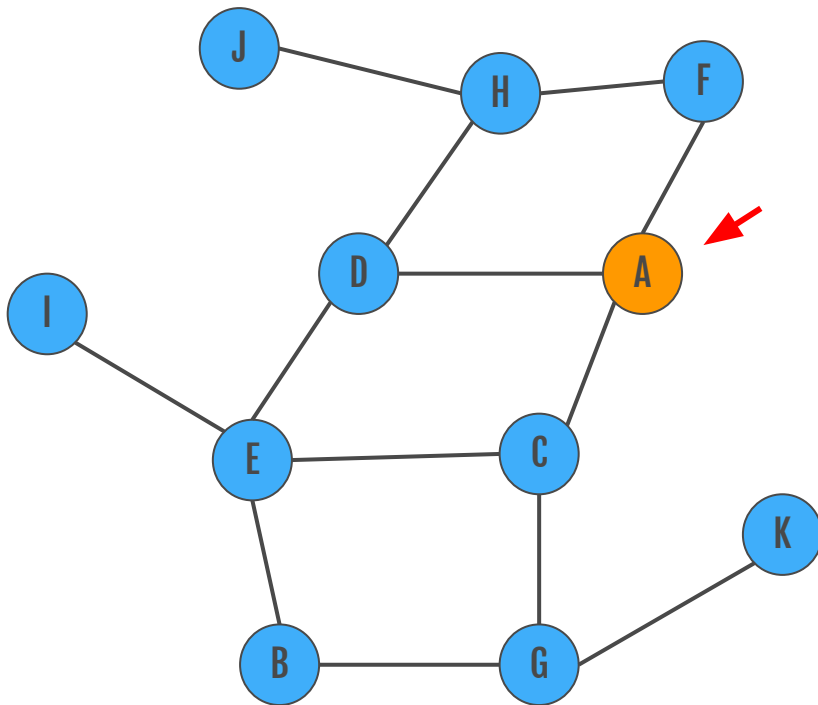


Queue: {A}

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



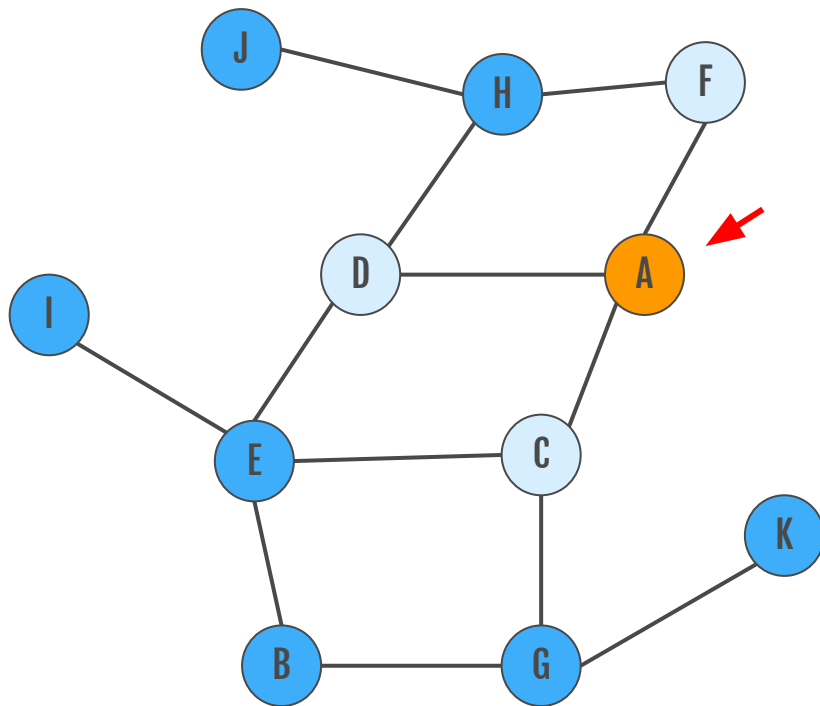
Queue: {}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



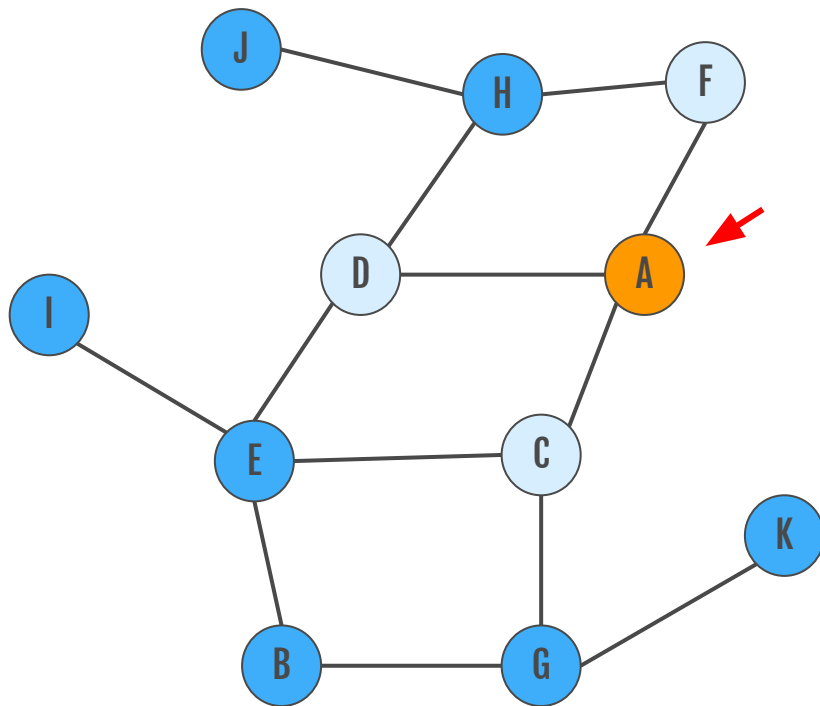
Queue: {}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



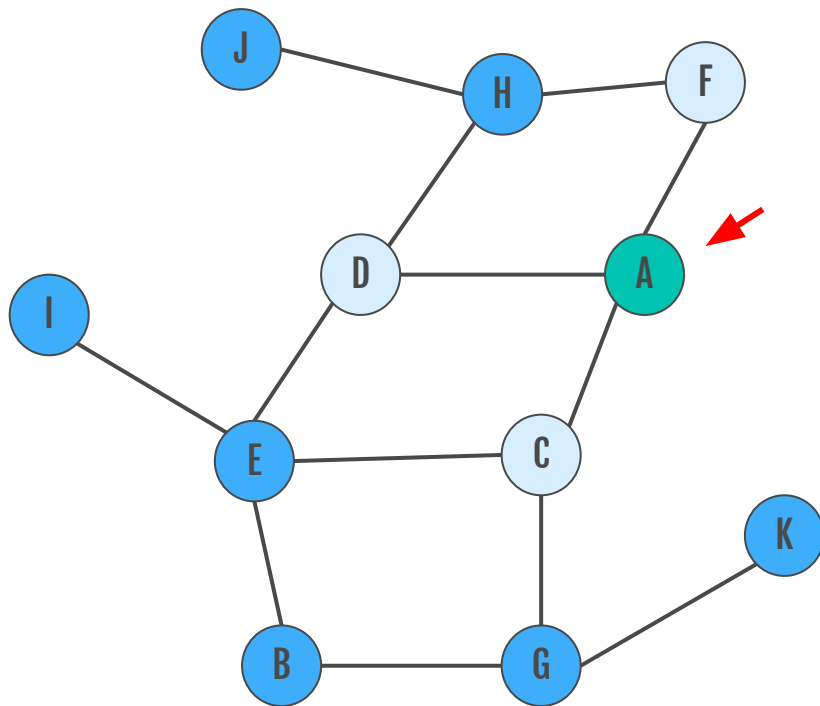
Queue: {C, D, F}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



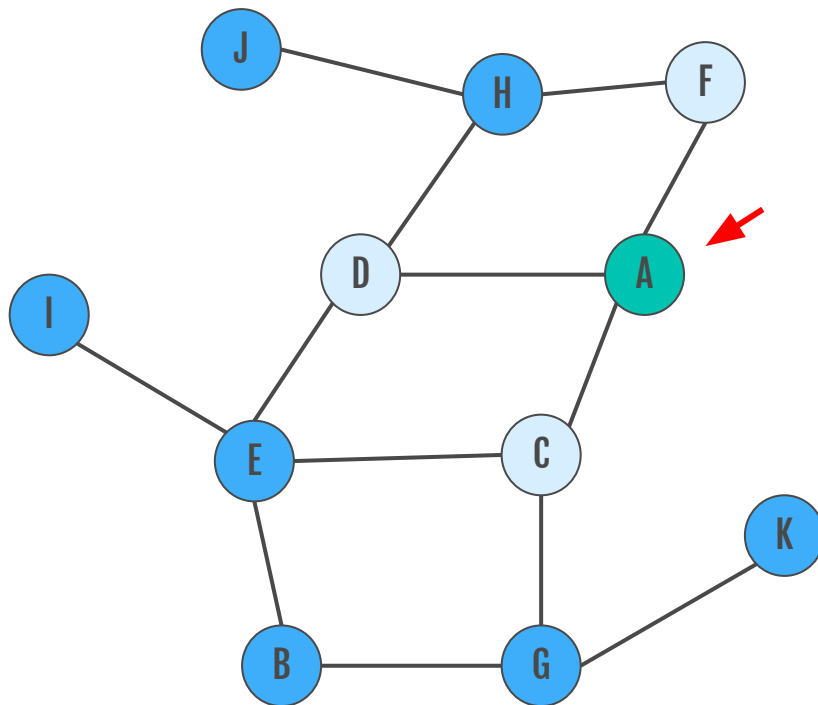
Queue: {C, D, F}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



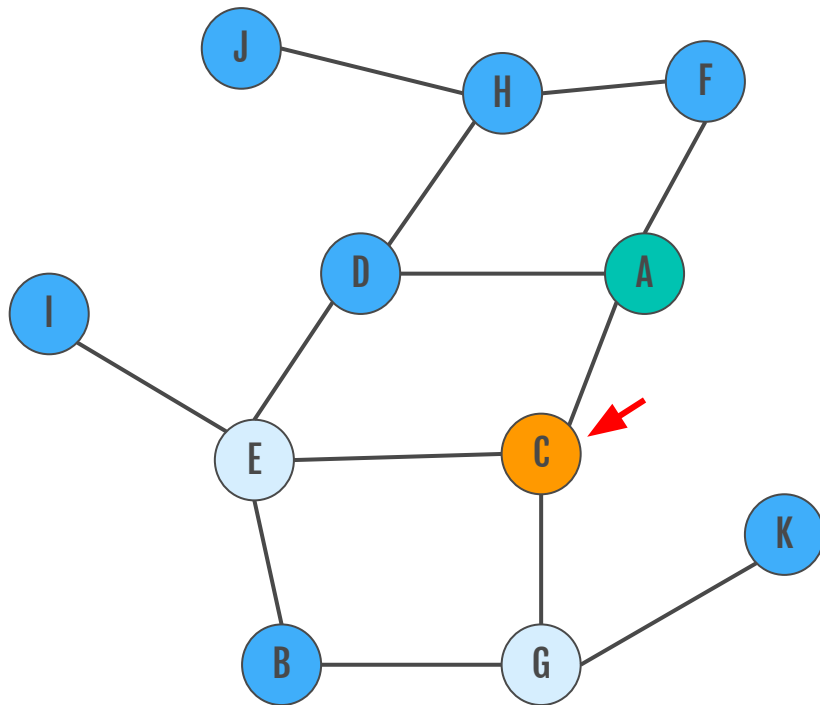
Queue: {C, D, F}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



Queue: {D, F}

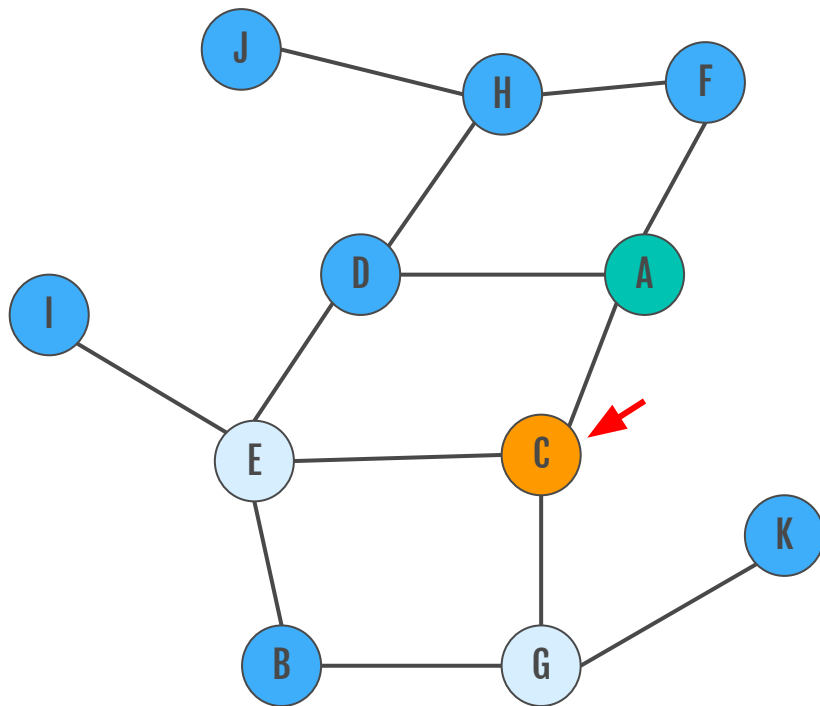
No visitado  
Actual Nodo  
Visitados  
Vecinos



# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



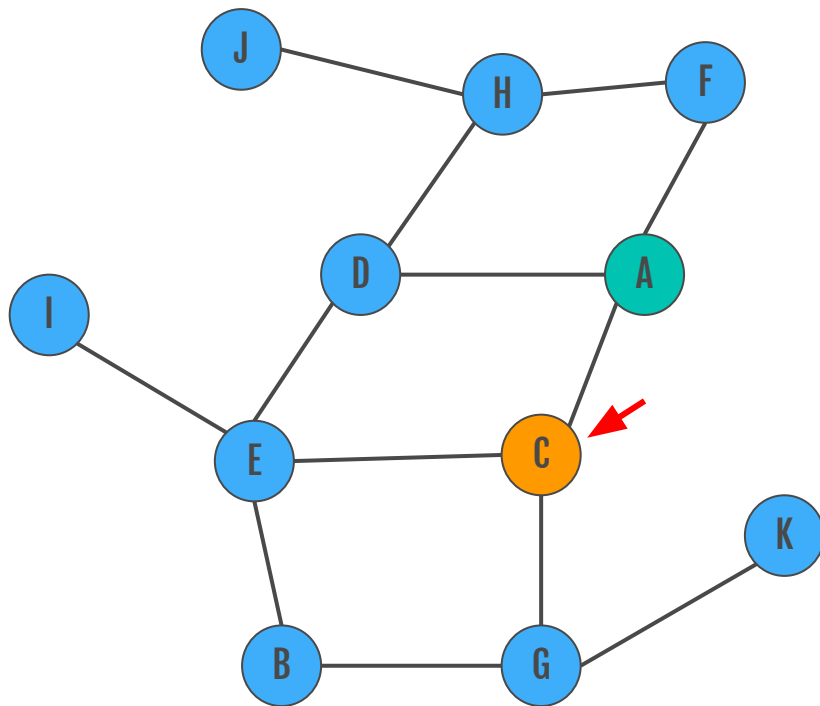
Queue: {D, F, E, G}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



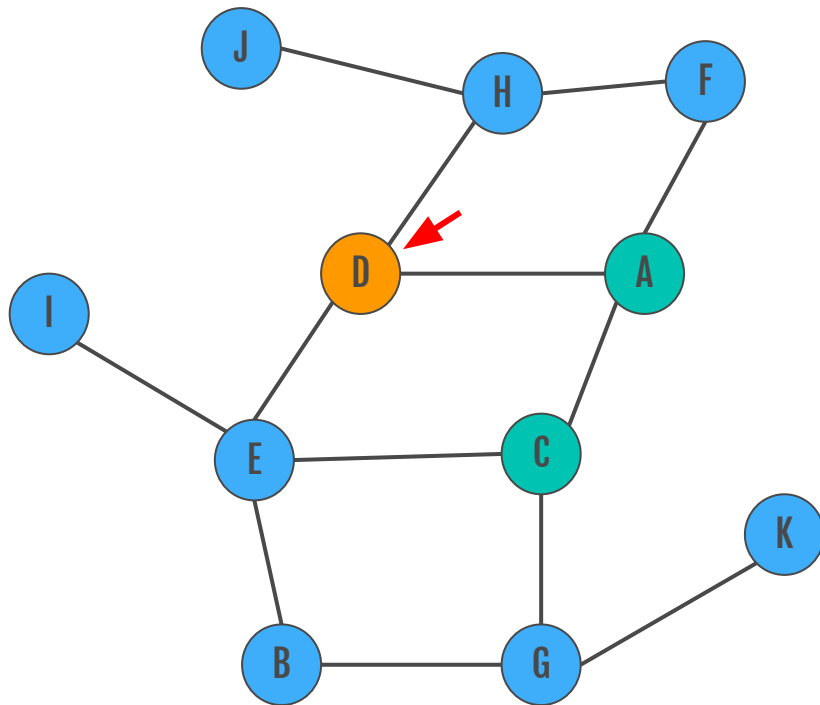
Queue: {D, F, E, G}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



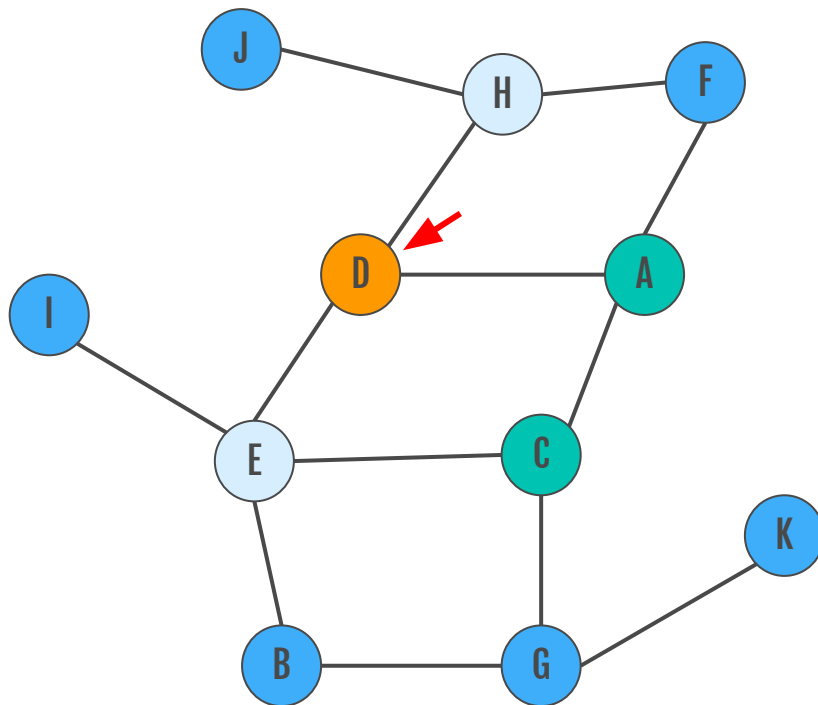
Queue: {F, E, G}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



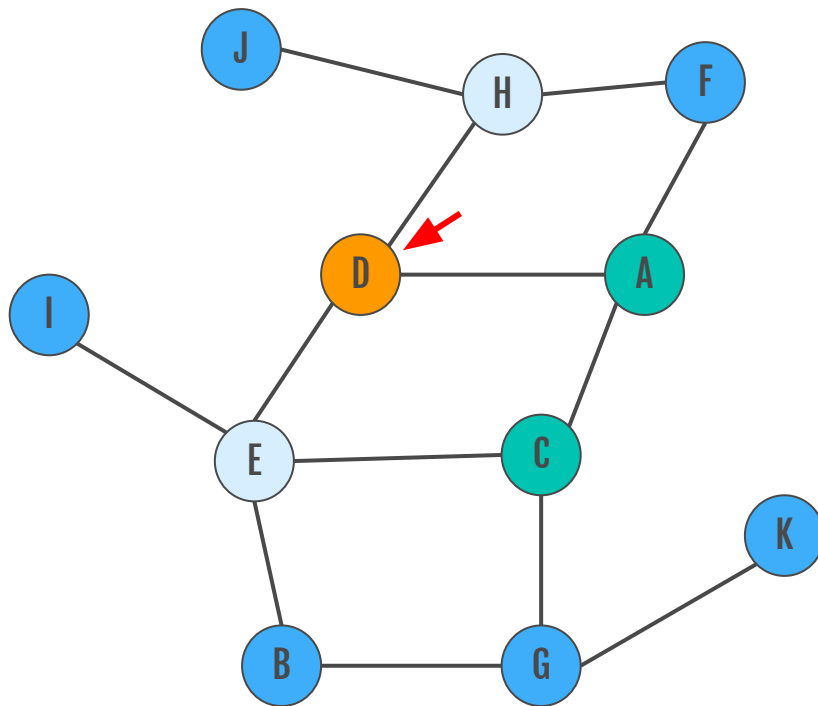
Queue: {F, E, G}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

# BFS ( Breadth-First-Search o Búsqueda en Profundidad)

### Simulación Gráfica:



Queue: {F, E, G, H}

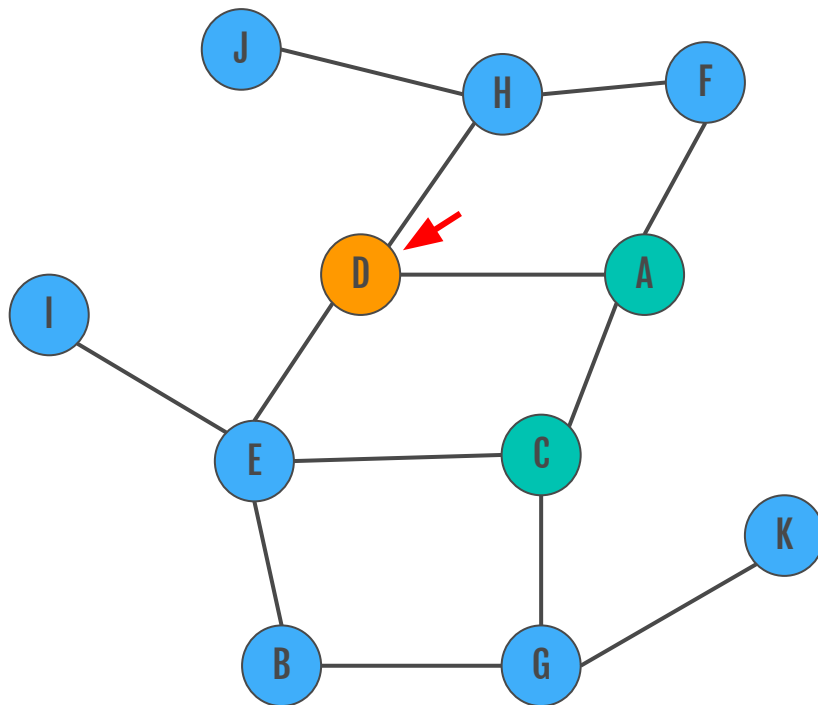
Diagram illustrating the components of a node in a graph search algorithm:

- No visitado
- Actual Nodo
- Visitados
- Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



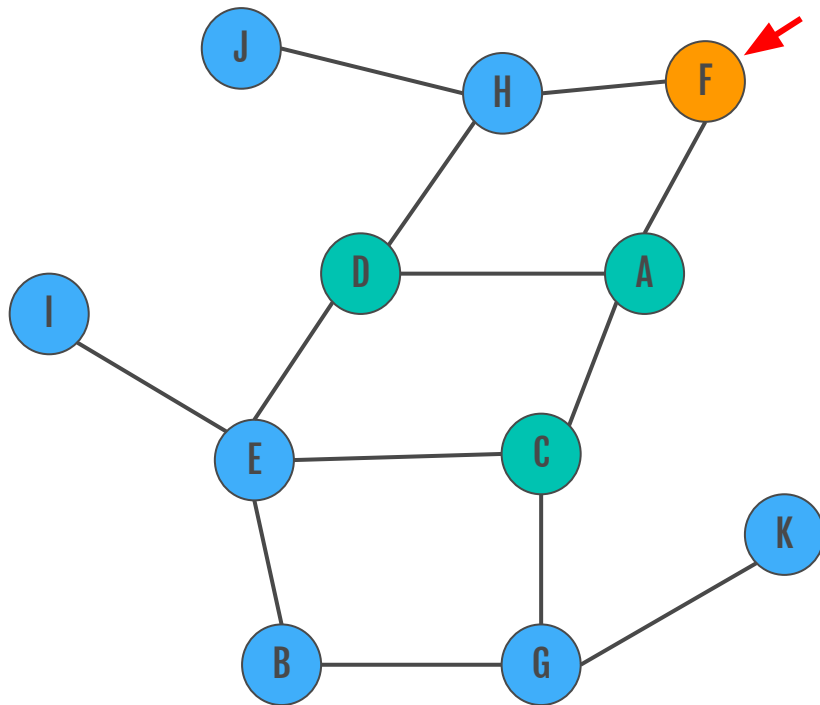
Queue: {F, E, G, H}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



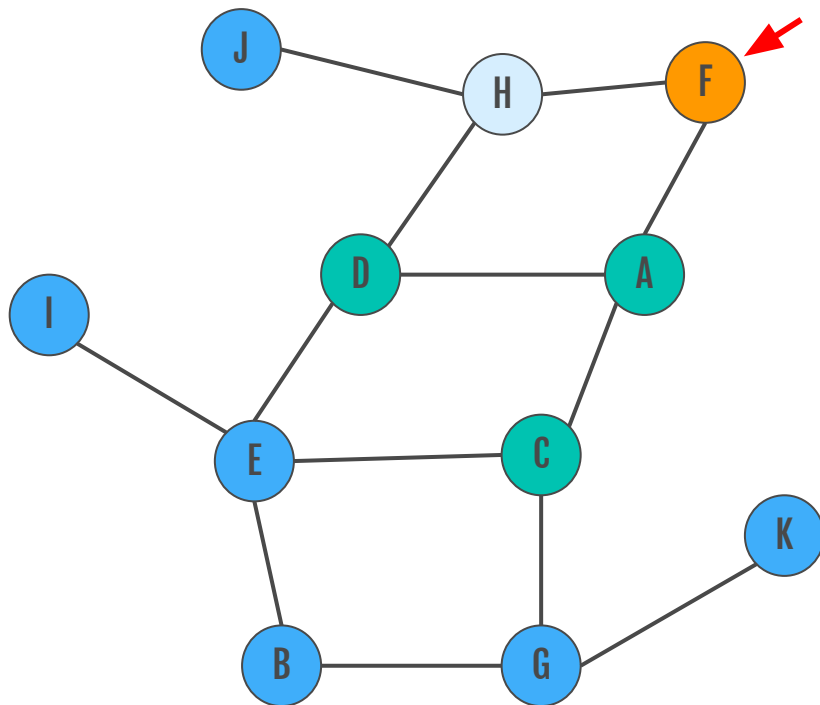
Queue: {E, G, H}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



Queue: {E, G, H}

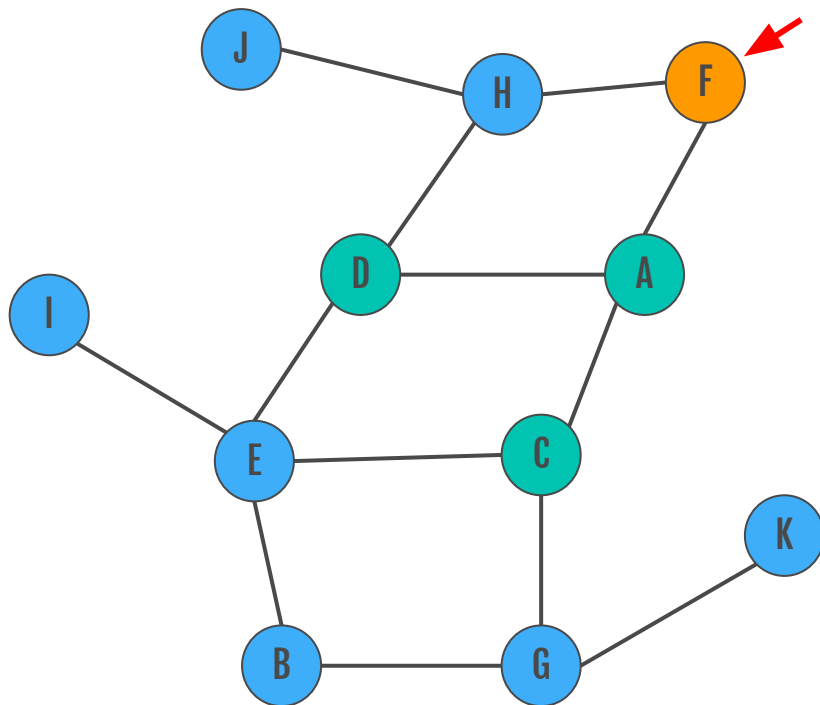
No visitado  
Actual Nodo  
Visitados  
Vecinos



# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



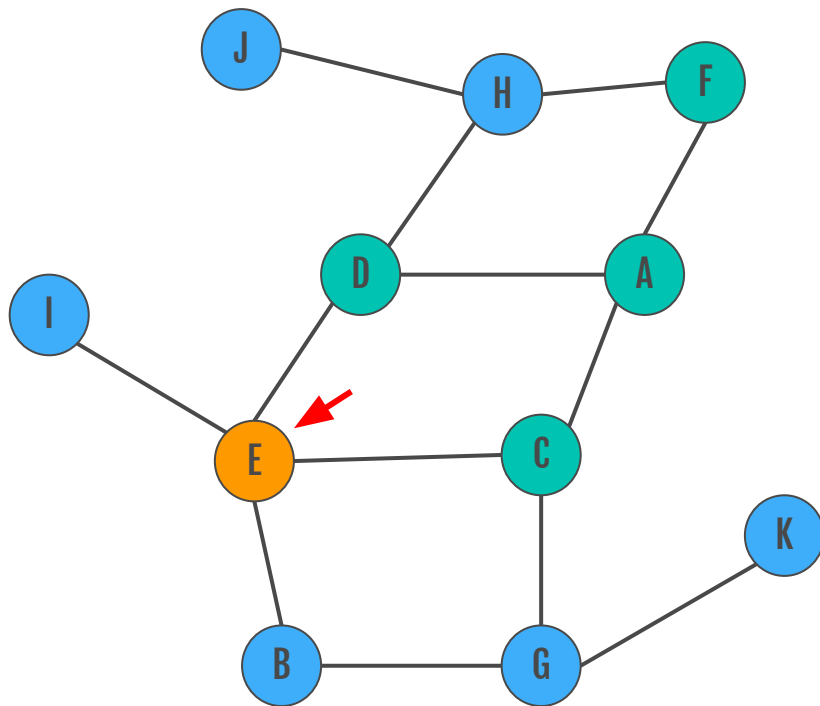
Queue: {E, G, H}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



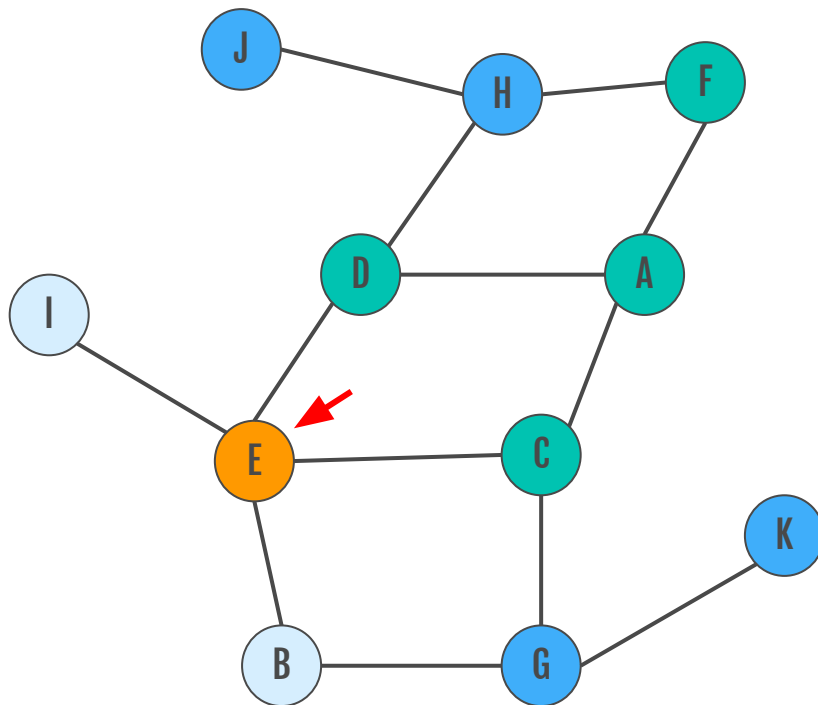
Queue: {G, H}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



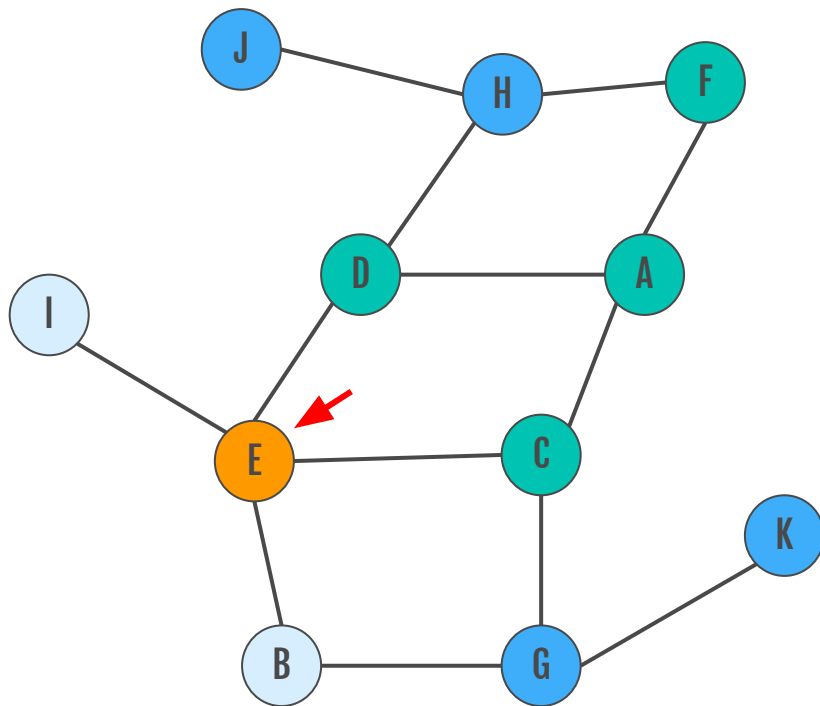
Queue: {G, H}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



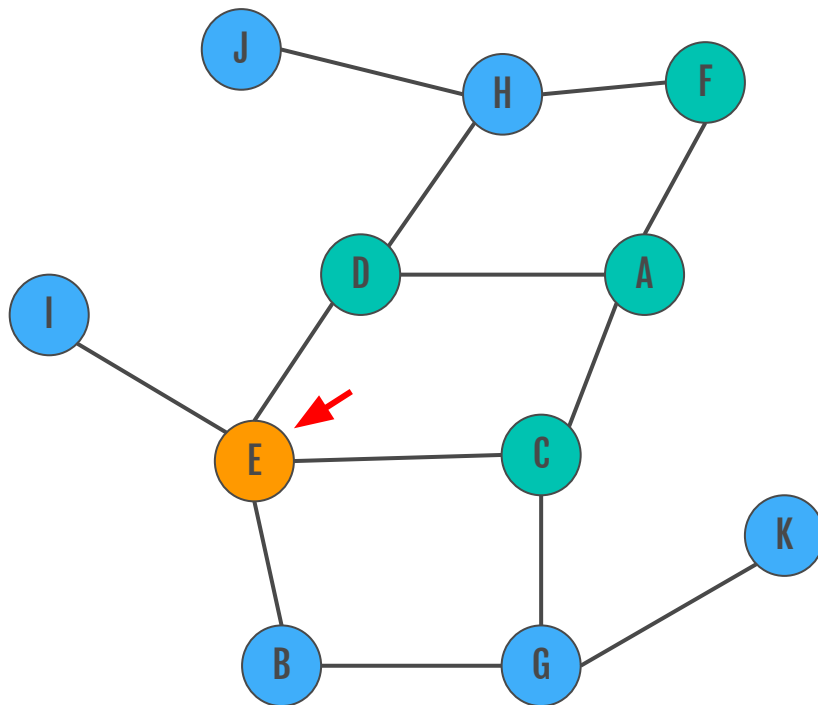
Queue: {G, H, B, I}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



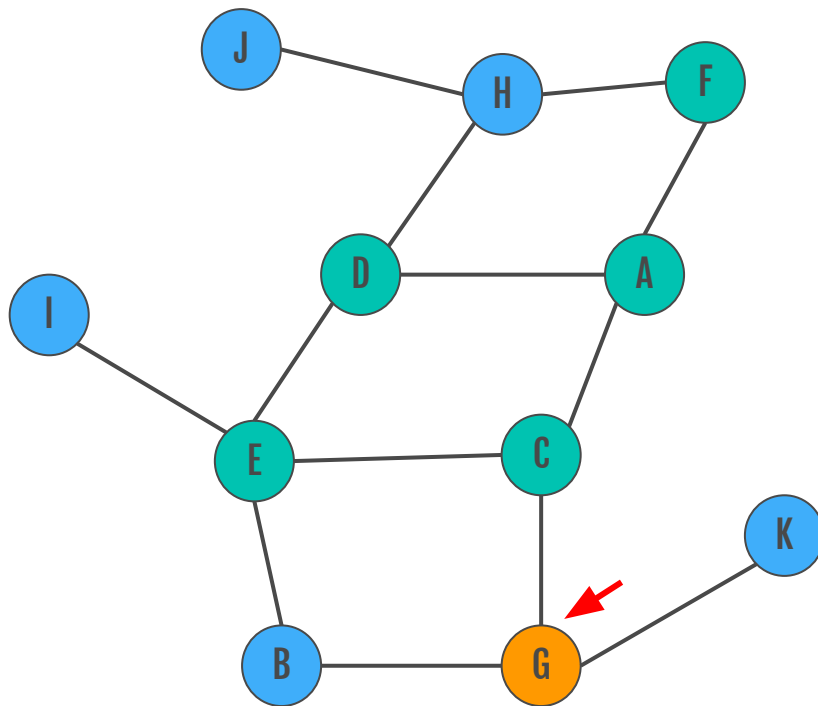
Queue: {G, H, B, I}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



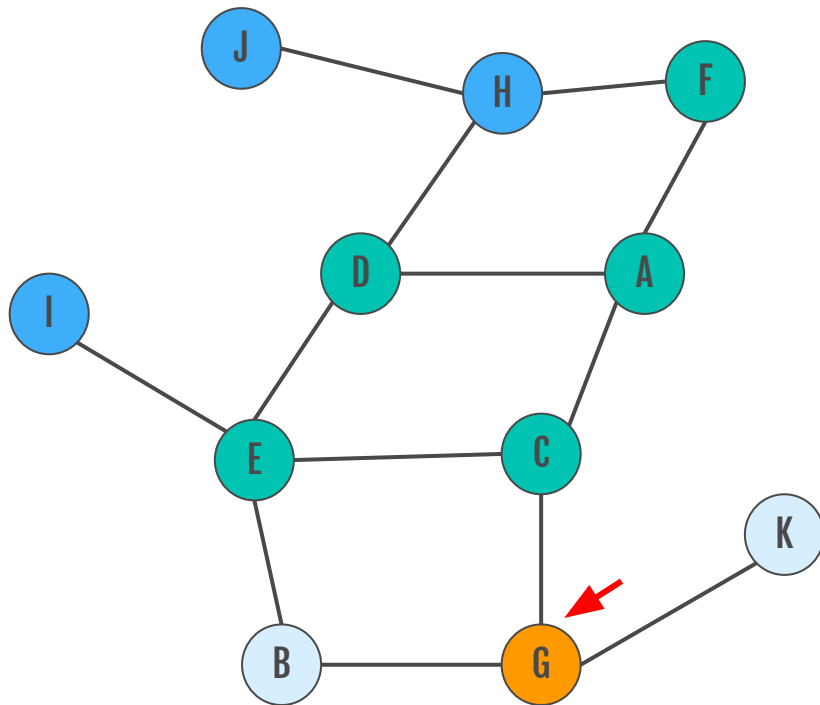
Queue: {H, B, I}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



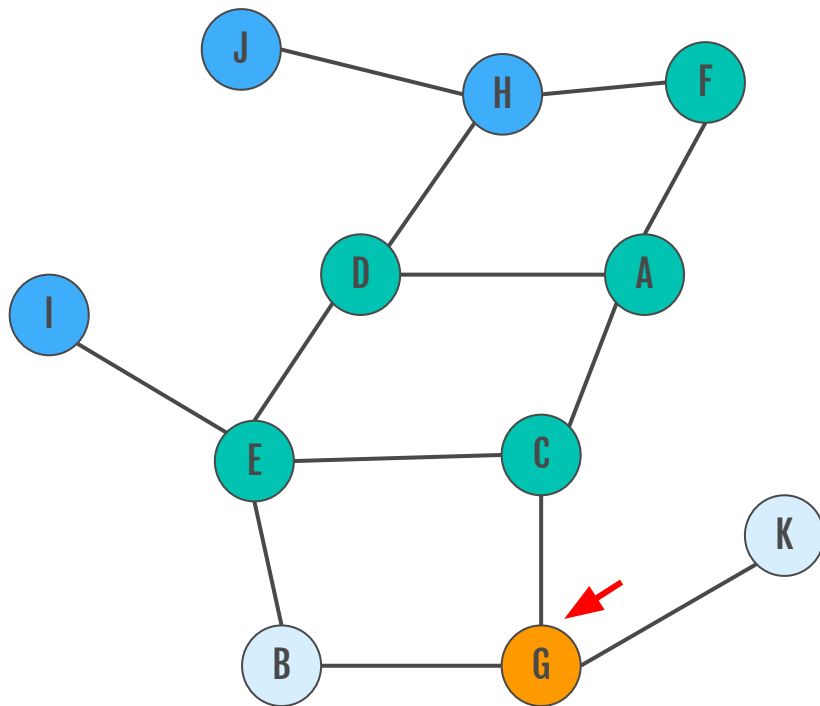
Queue: {H, B, I}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



Queue: {H, B, I, K}

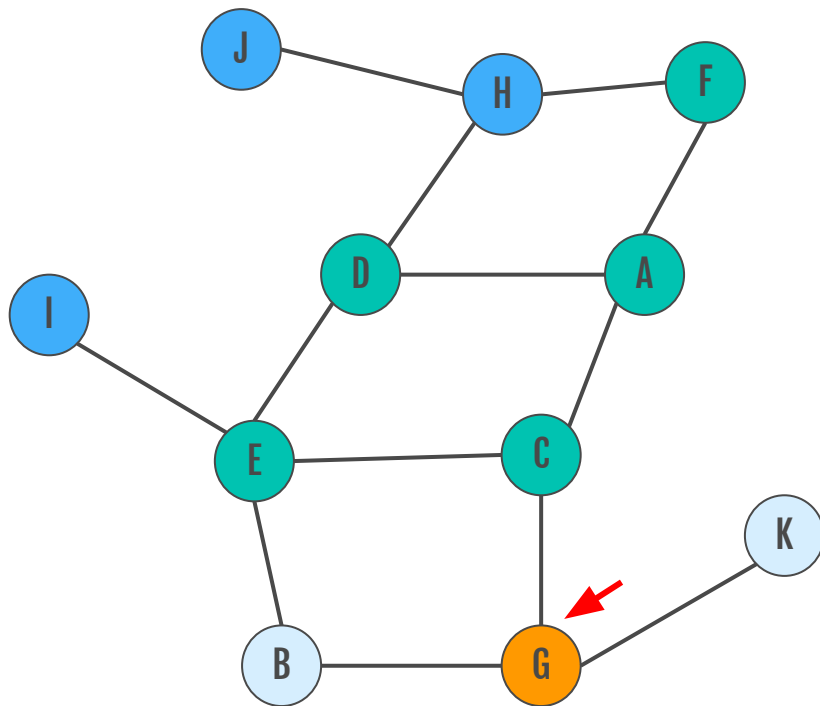
No visitado  
Actual Nodo  
Visitados  
Vecinos



# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



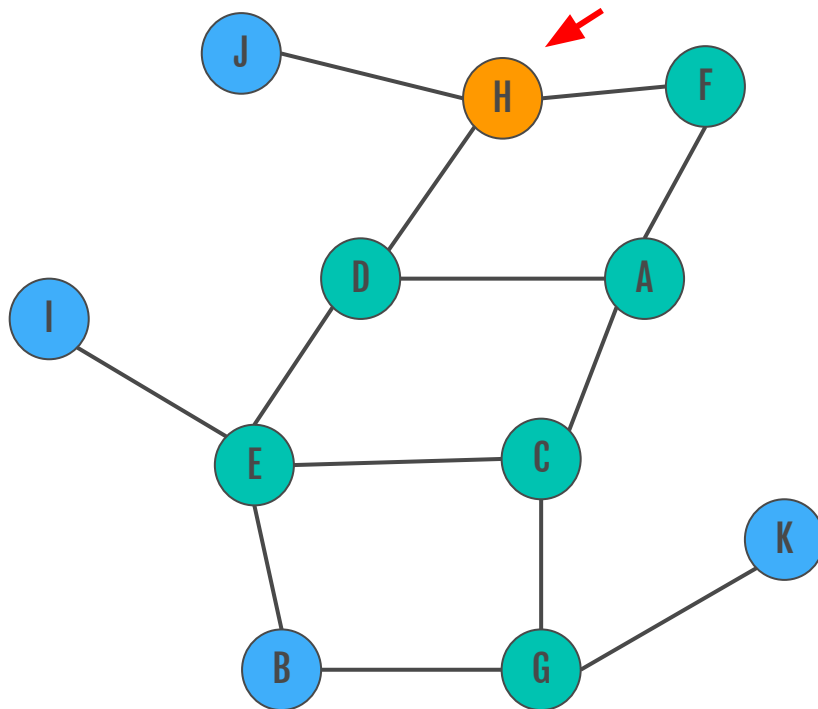
Queue: {H, B, I, K}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



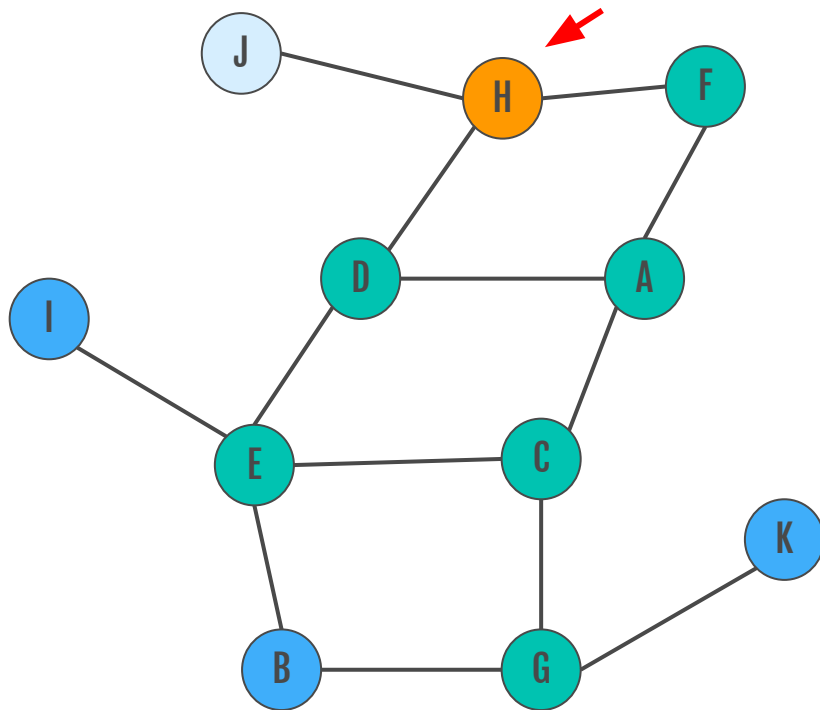
Queue: {B, I, K}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



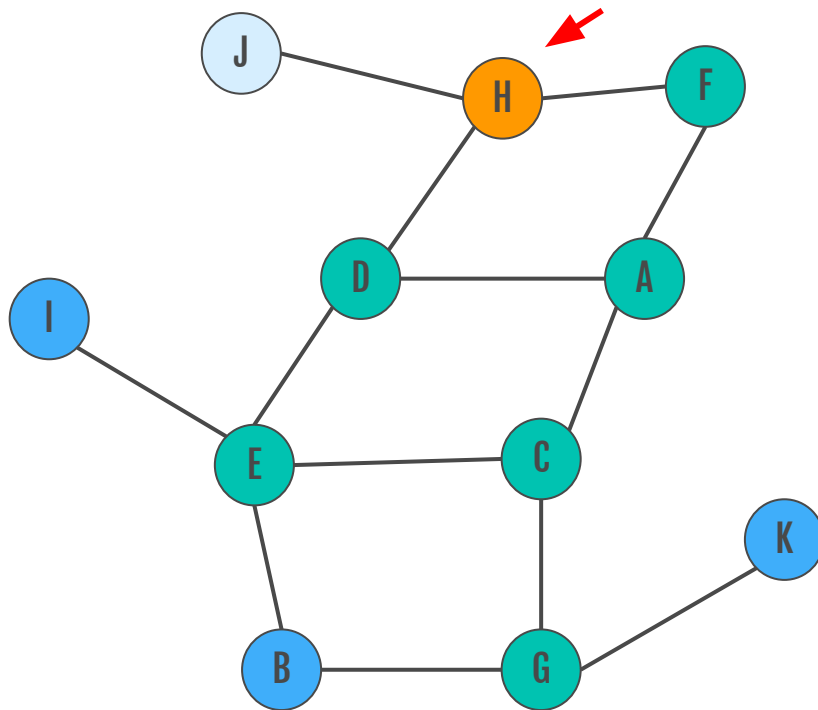
Queue: {B, I, K}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



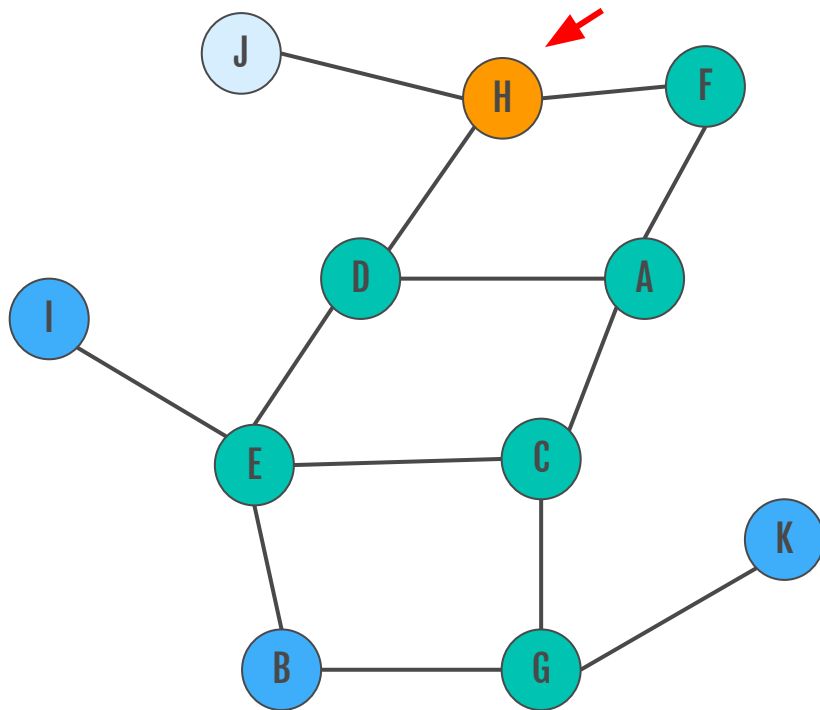
Queue: {B, I, K, J}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



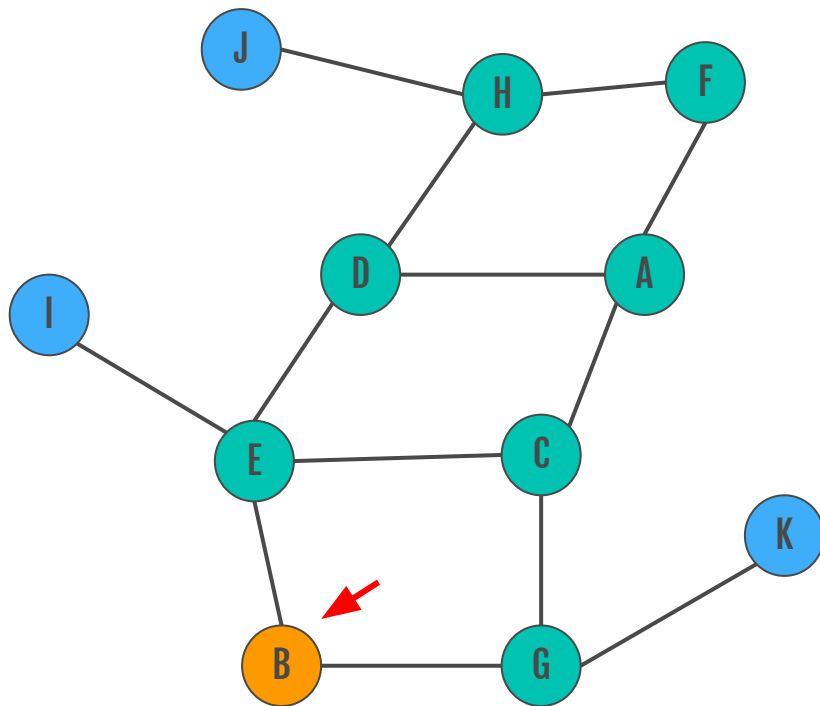
Queue: {B, I, K, J}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



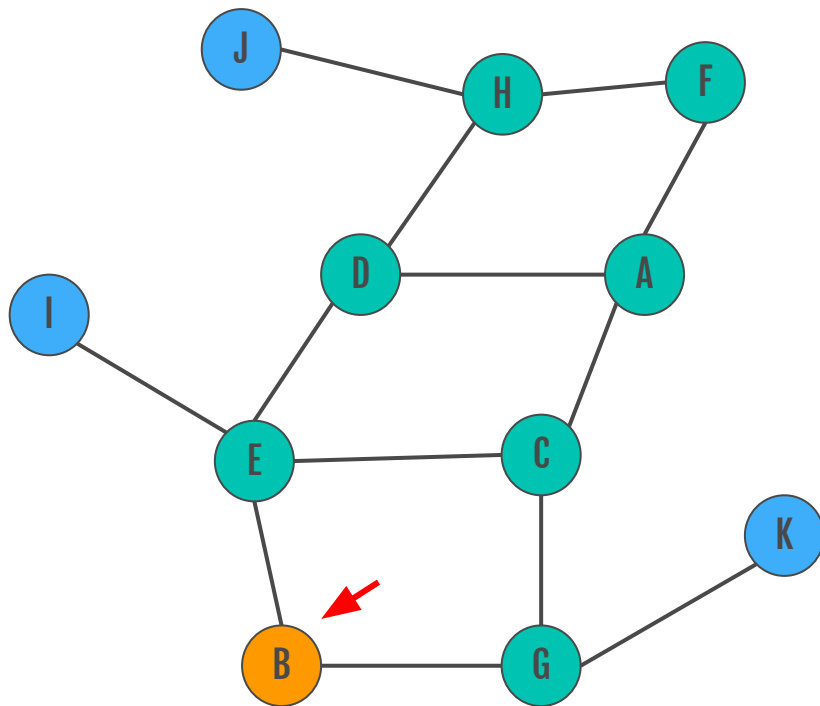
Queue: {I, K, J}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



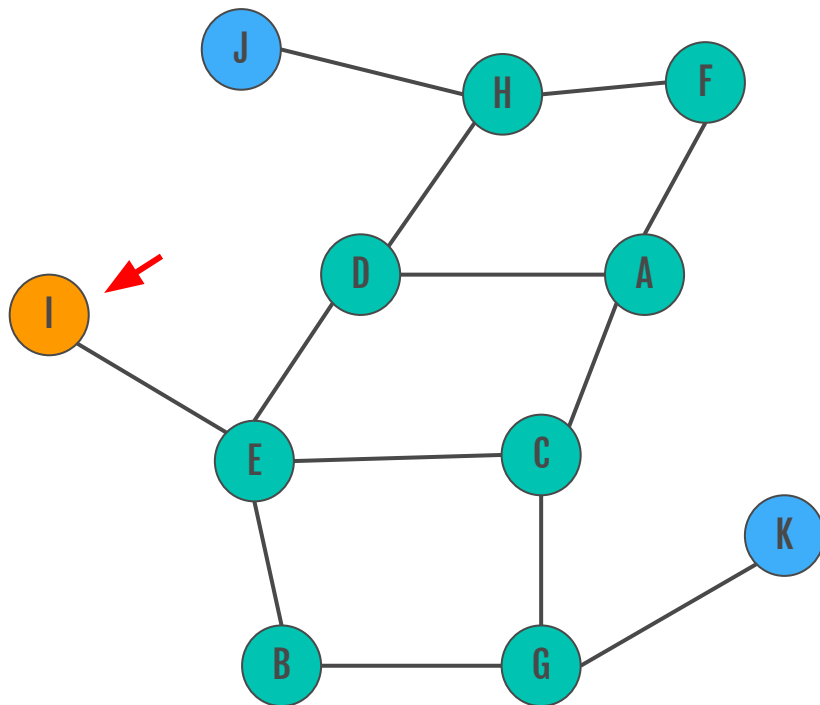
Queue: {I, K, J}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



Queue: {K, J}

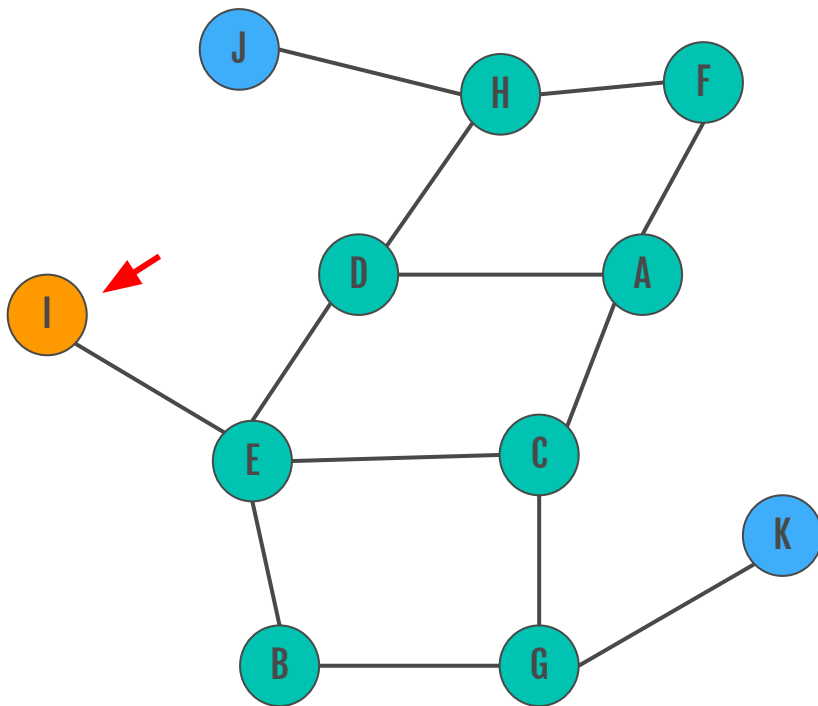
No visitado  
Actual Nodo  
Visitados  
Vecinos



# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



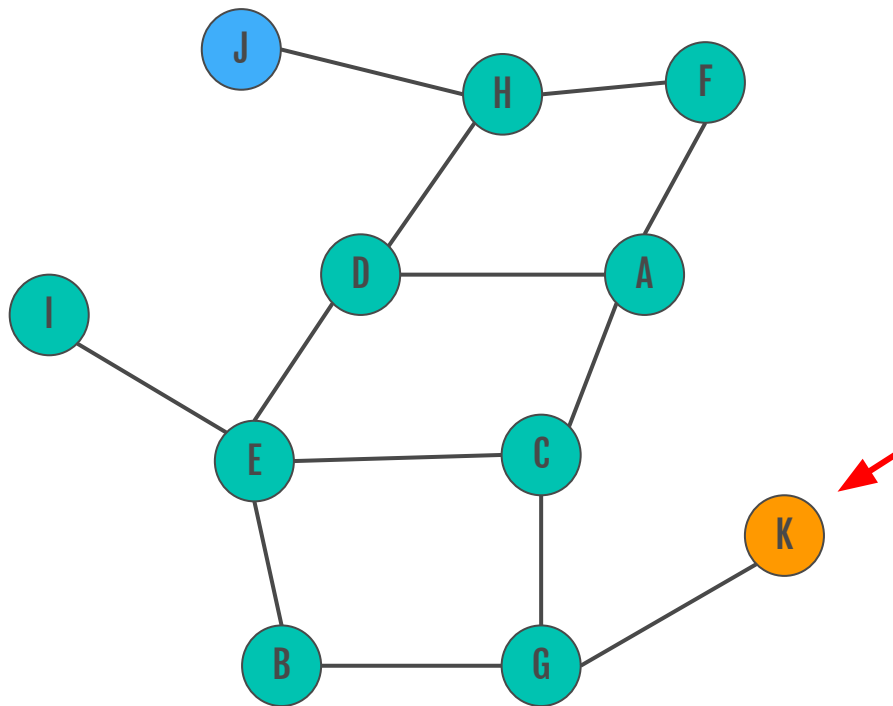
Queue: {K, J}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

# BFS ( Breadth-First-Search o Búsqueda en Profundidad)

### Simulación Gráfica:



Queue: {J}

No visitado

## Actual Nodo

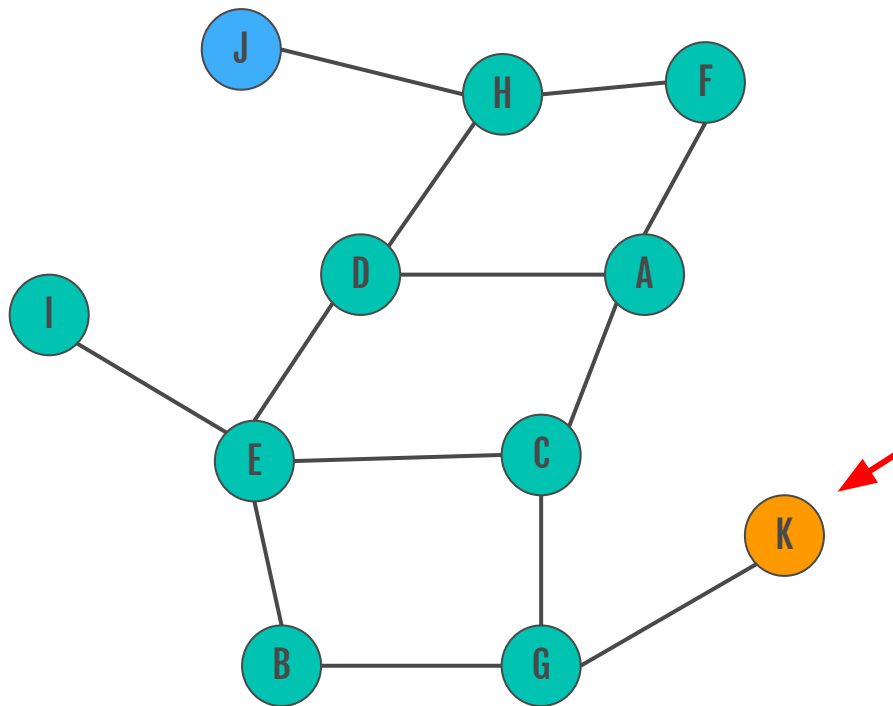
## Visitados

## Vecinos

# ¿Como recorrer un Grafo?

# BFS ( Breadth-First-Search o Búsqueda en Profundidad)

### Simulación Gráfica:



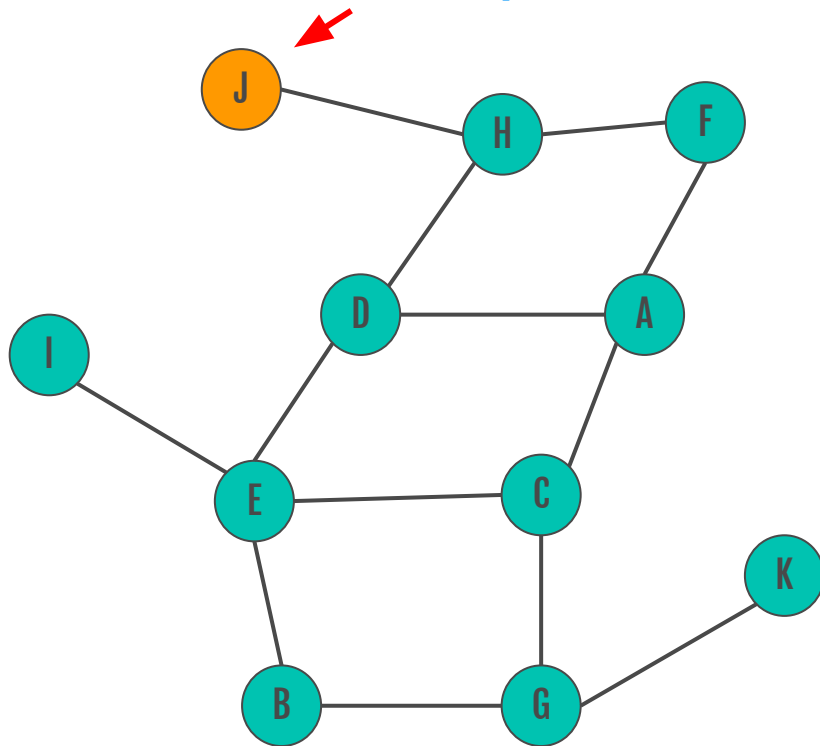
Queue: {J}

A diagram of a node in a search tree. The node is represented as a vertical stack of four colored rectangles. From top to bottom, the colors and labels are: blue with the text "No visitado", orange with the text "Actual Nodo", teal with the text "Visitados", and light blue with the text "Vecinos".

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



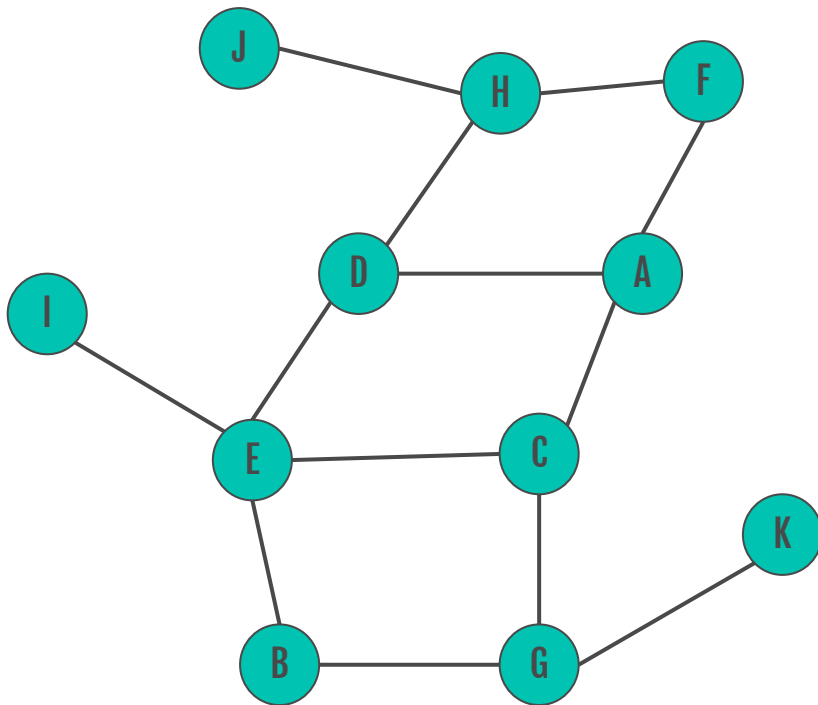
Queue: {}

No visitado  
Actual Nodo  
Visitados  
Vecinos

# ¿Como recorrer un Grafo?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Profundidad)

Simulación Gráfica:



Queue: {}

No visitado  
Actual Nodo  
Visitados  
Vecinos

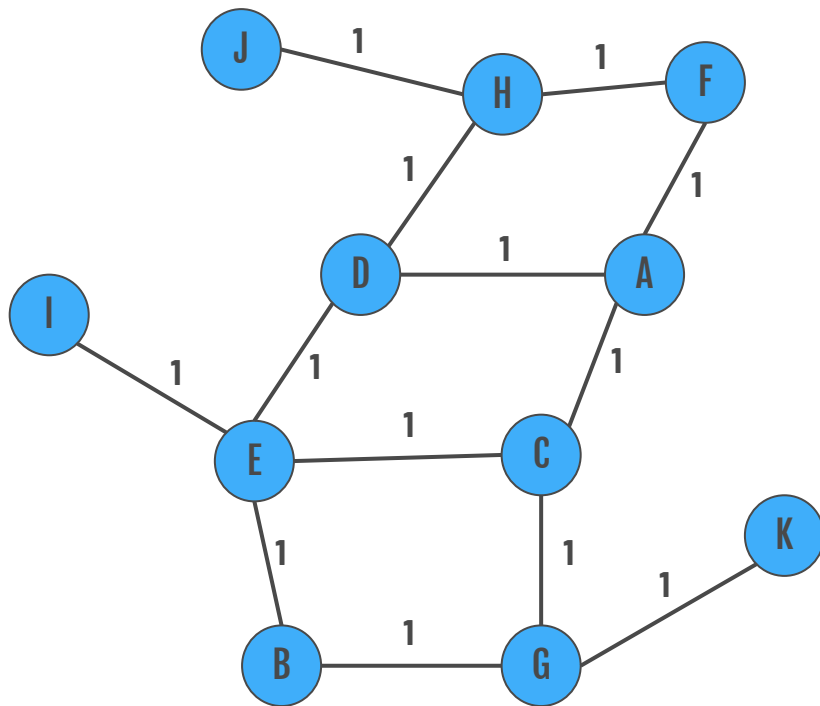
# ¿Distancias Minimas con BFS?

## BFS ( Breadth-First-Search o Busqueda en Anchura)

- Es un algoritmo para recorrer grafos, que a su vez sirve para calcular la distancia mínima desde un nodo  $u$  a cada uno de los otros.
- Inicialmente se procesa  $u$ , que tiene distancia a sí mismo 0, por lo que seteamos  $d_u = 0$ . Además es el único a distancia 0.
- Los vecinos de  $u$  tienen sí o sí distancia 1, por lo que seteamos  $d_v = 1$  para todo  $v$  vecino de  $u$ . Además, estos son los únicos a distancia 1.
- Ahora tomamos los nodos a distancia 1, y para cada uno nos fijamos en sus vecinos cuya distancia aún no calculamos. Estos son los nodos a distancia 2.
- Así sucesivamente, para saber cuáles son los nodos a distancia  $k + 1$ , tomamos los vecinos de los nodos a distancia  $k$  que no hayan sido visitados aún.

# ¿Distancias Minimas con BFS?

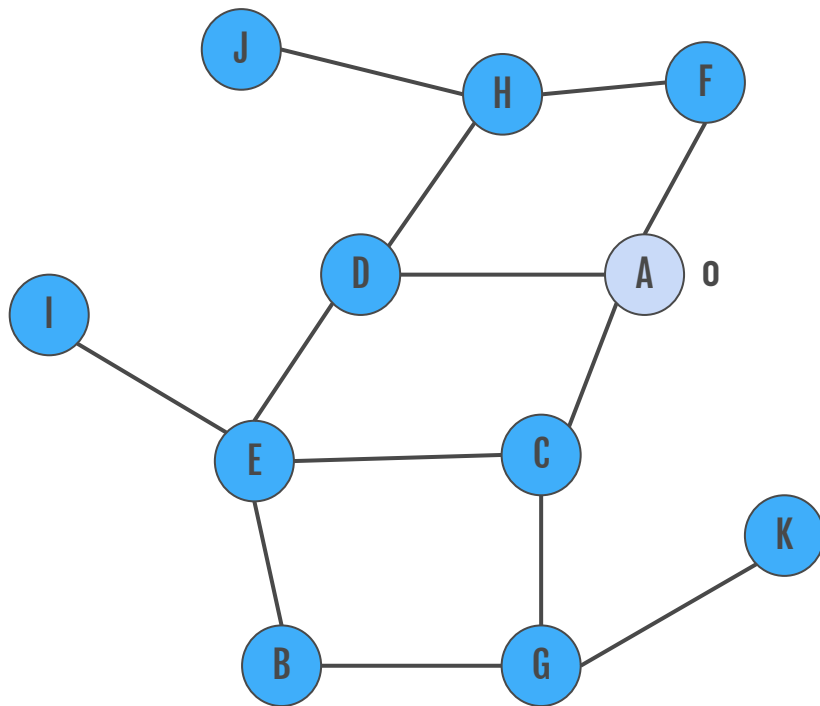
BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Anchura)



Queue: {}

# ¿Distancias Minimas con BFS?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Busqueda en Anchura)

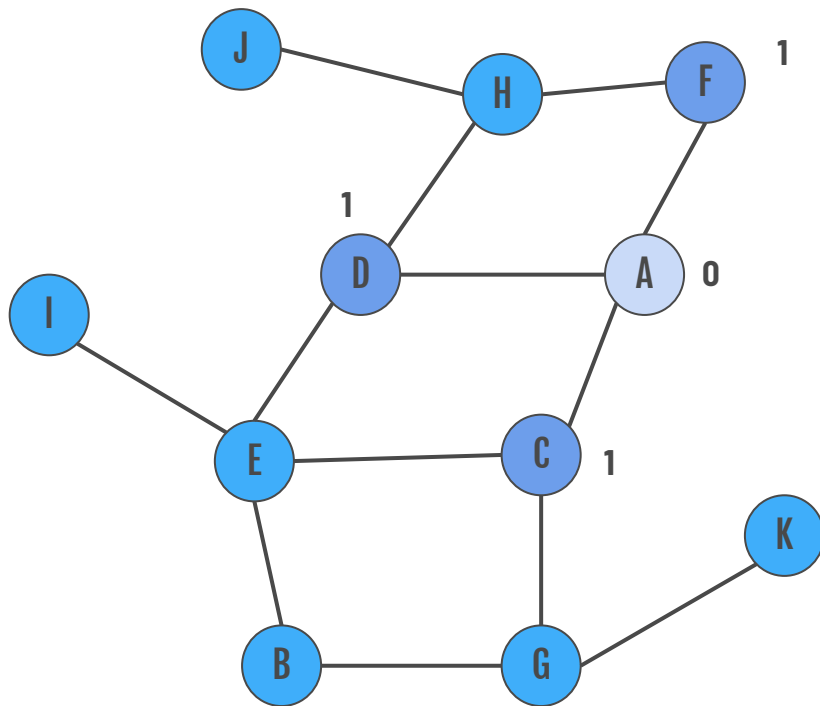


Queue: {}



# ¿Distancias Minimas con BFS?

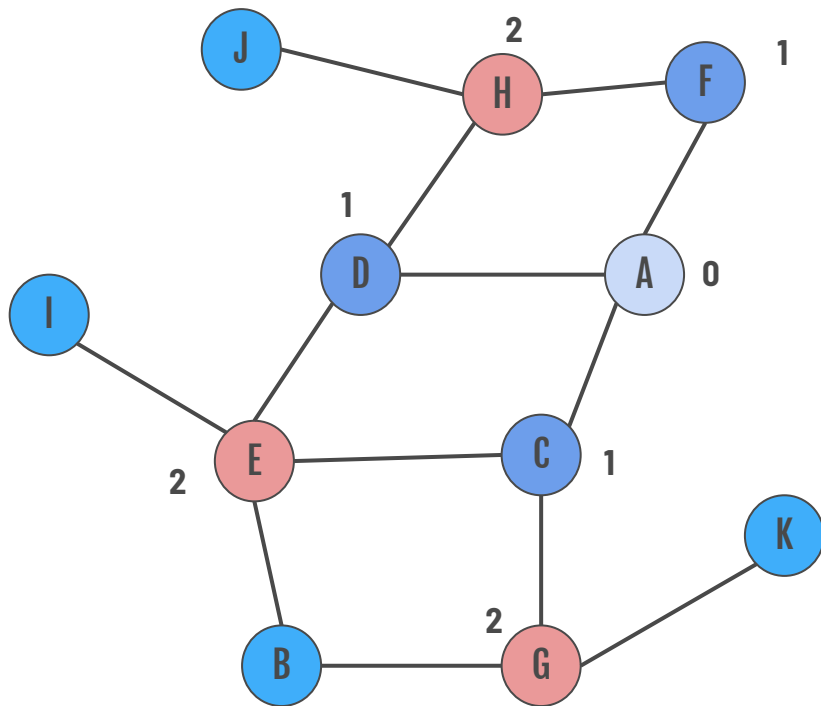
BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Búsqueda en Anchura)



Queue: {}

# ¿Distancias Minimas con BFS?

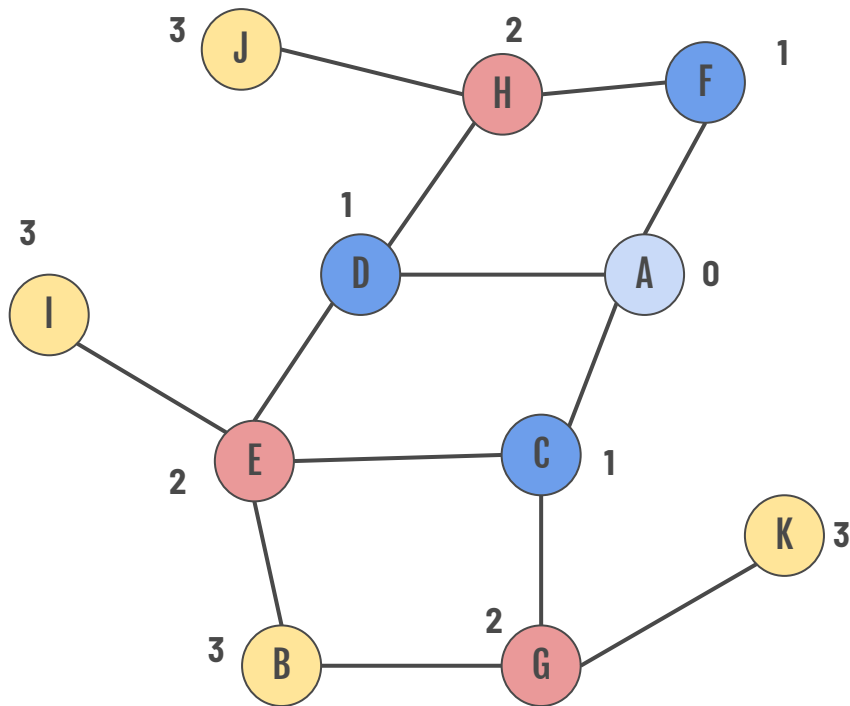
BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Búsqueda en Anchura)



Queue: {}

# ¿Distancias Minimas con BFS?

BFS (🇺🇸 Breadth-First-Search o 🇪🇸 Búsqueda en Anchura)



Queue: {}



**¿Preguntas?**



# ¿Como recorrer un Grafo?

 Flood Fill o  Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

## Counting Rooms

You are given a map of a building, and your task is to count the number of its rooms. The size of the map is  $n \times m$  squares, and each square is either floor or wall. You can walk left, right, up, and down through the floor squares.

### Input

The first input line has two integers  $n$  and  $m$ : the height and width of the map.

Then there are  $n$  lines of  $m$  characters describing the map. Each character is either `.` (floor) or `#` (wall).

### Output

Print one integer: the number of rooms.

### Constraints

- $1 \leq n, m \leq 1000$

# ¿Como recorrer un Grafo?

🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

# ¿Como recorrer un Grafo?

🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#



# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos



# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos



# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos



**Counting Rooms**

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos



# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos



# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

####.#.#

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

**CSES Counting Rooms**

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## 🇺🇸 Flood Fill o 🇪🇸 Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#

 **Counting Rooms**

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	#	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#



**Counting Rooms**

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# ¿Como recorrer un Grafo?

## Flood Fill o Relleno por Inundación

- Nos permite **recorrer** un **grafo** cuando no lo tenemos de forma explícita por ejemplo una cuadrícula.

**Input:**

5 8

#####

# ..#...#

#####

# ..#...#

#####

**Output:**

3

#	#	.	#	#	#	#	#
#	.	.	#	.	.	.	#
#	#	#	#	.	#	.	#
#	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#



**Counting Rooms**

No visitado

Visitado y en stack

Visitado y fuera del stack

Actual Nodo

Vecinos

# Implementación

Flood Fill

# Coding



# Implementación (Option 1)

## Flood Fill

```
#define between(a, b, c) ((a) <= b && b <= (c))

void floodfill(int x, int y) {
    if (!between(0, x, n - 1) || !between(0, y, m - 1) || visited[x][y] ||
        grid[x][y] == '#') {
        return;
    }
    visited[x][y] = 1;

    floodfill(x, y - 1);
    floodfill(x - 1, y);
    floodfill(x + 1, y);
    floodfill(x, y + 1);
}
```

# Implementación (Option 2)

## Flood Fill

```
#define between(a, b, c) ((a) <= b && b <= (c))

const int d4x[4] = {0, -1, 1, 0};
const int d4y[4] = {-1, 0, 0, 1};

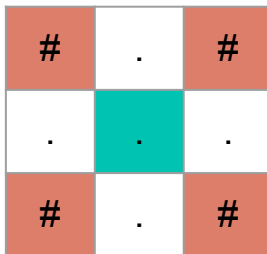
void floodfill(int x, int y) {
    if (!between(0, x, n - 1) || !between(0, y, m - 1) || visited[x][y] ||
        grid[x][y] == '#') {
        return;
    }
    visited[x][y] = 1;

    for(int i = 0; i < 4; ++i) {
        floodfill(x + d4x[i], y + d4y[i]);
    }
}
```

# Implementación (Option 2)

## Flood Fill

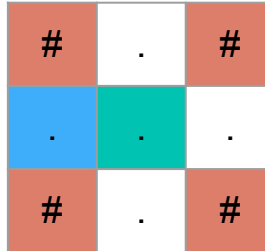
```
const int d4x[4] = {0, -1, 1, 0};  
const int d4y[4] = {-1, 0, 0, 1};
```



# Implementación (Option 2)

## Flood Fill

```
const int d4x[4] = {0, -1, 1, 0};  
const int d4y[4] = {-1, 0, 0, 1};
```

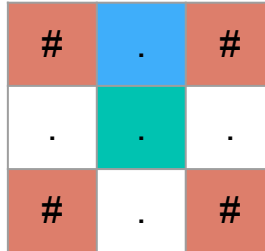


```
floodfill(x, y - 1);  
floodfill(x - 1, y);  
floodfill(x + 1, y);  
floodfill(x, y + 1);
```

# Implementación (Option 2)

## Flood Fill

```
const int d4x[4] = {0, -1, 1, 0};  
const int d4y[4] = {-1, 0, 0, 1};
```

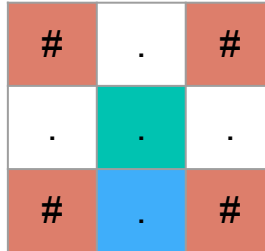


```
floodfill(x, y - 1);  
floodfill(x - 1, y);  
floodfill(x + 1, y);  
floodfill(x, y + 1);
```

# Implementación (Option 2)

## Flood Fill

```
const int d4x[4] = {0, -1, 1, 0};  
const int d4y[4] = {-1, 0, 0, 1};
```

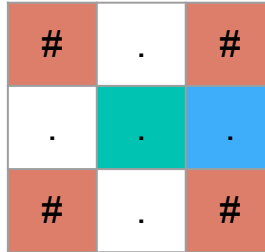


```
floodfill(x, y - 1);  
floodfill(x - 1, y);  
floodfill(x + 1, y);  
floodfill(x, y + 1);
```

# Implementación (Option 2)

## Flood Fill

```
const int d4x[4] = {0, -1, 1, 0};  
const int d4y[4] = {-1, 0, 0, 1};
```



```
floodfill(x, y - 1);  
floodfill(x - 1, y);  
floodfill(x + 1, y);  
floodfill(x, y + 1);
```

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};

void floodfill(int x, int y) {
    if (!between(0, x, n - 1) || !between(0, y, m - 1) || visited[x][y] ||
        grid[x][y] == '#') {
        return;
    }
    visited[x][y] = 1;

    for(int i = 0; i < 8; ++i) {
        floodfill(x + d8x[i], y + d8y[i]);
    }
}
```



# Implementación (Option 2)

## Flood Fill

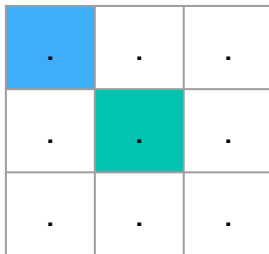
```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```



# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.



# Implementación (Option 2)

## Flood Fill

```
const int d8x[8] = {-1, 0, -1, 1, -1, 1, 0, 1};  
const int d8y[8] = {-1, -1, 0, -1, 1, 0, 1, 1};
```

.	.	.
.	.	.
.	.	.

# Definición

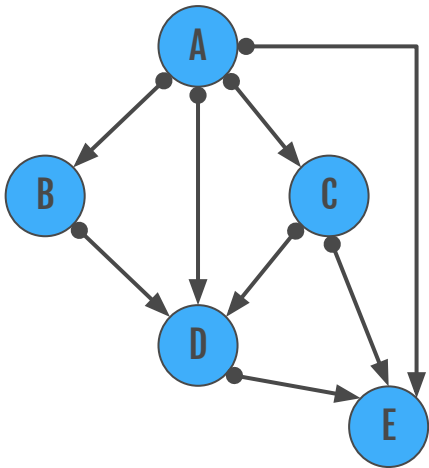
## Ordenamiento Topológico - Topological Sorting

Dado un Grafo **Dirigido**, un **orden topológico** es un ordenamiento de los nodos de modo que para cada arista  $\mathbf{x} \rightarrow \mathbf{y}$ ,  $\mathbf{x}$  viene antes que  $\mathbf{y}$  en el orden.

# Definición

## 🇪🇸 Ordenamiento Topológico - 🇺🇸 Topological Sorting

Dado un Grafo **Dirigido**, un **orden topológico** es un ordenamiento de los nodos de modo que para cada arista  $x \rightarrow y$ ,  $x$  viene antes que  $y$  en el orden.



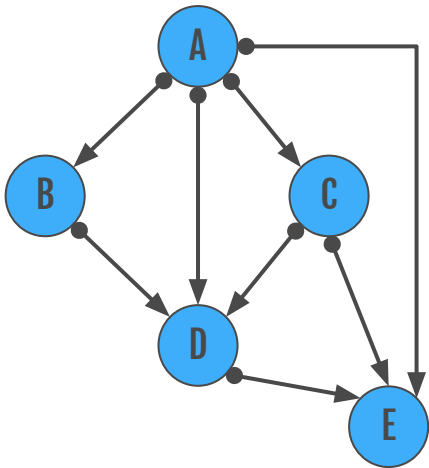
Este Grafo tiene dos órdenes topológicos:

- 1) A, B, C, D, E
- 2) A, C, B, D, E

# Definición

## 🇪🇸 Ordenamiento Topológico - 🇺🇸 Topological Sorting

Dado un Grafo **Dirigido**, un **orden topológico** es un ordenamiento de los nodos de modo que para cada arista  $x \rightarrow y$ ,  $x$  viene antes que  $y$  en el orden.





Este Grafo tiene dos órdenes topológicos:

- 1) A, B, C, D, E
- 2) A, C, B, D, E

# Definición

## Ordenamiento Topológico - Topological Sorting

- **No** todo grafo tiene orden topológico.
- Concretamente, un grafo tiene orden topológico **si y sólo si** no tiene **ciclo** (camino desde algún nodo a sí mismo). A estos grafos se los llama **DAG** ( **Directed-Acyclic-Graph** o  Grafo Dirigido sin Ciclos).

# Definición

## Kahn's Algorithm

- **Degree (grado):** Es el número de conexiones que salen de un nodo.
- **Indegree (grado de entrada):** Es el número de conexiones que llegan a un nodo desde otros nodos.



# Definición

## Kahn's Algorithm

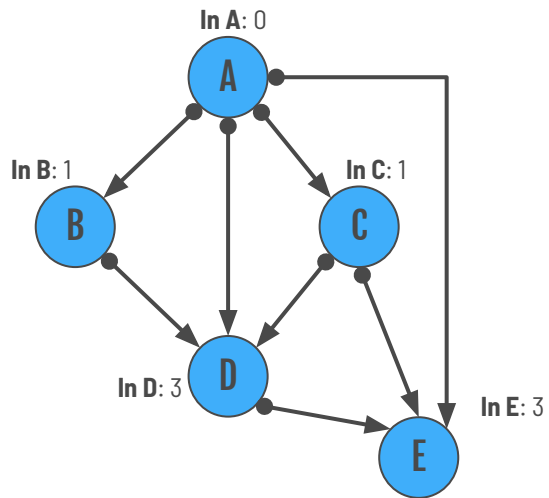
- **Degree (grado)**: Es el número de conexiones que salen de un nodo.
- **Indegree (grado de entrada)**: Es el número de conexiones que llegan a un nodo desde otros nodos.

### Algoritmo:

- Obtenemos el **Indegree/grado de entrada** de cada nodo y metemos los de grado **0** en una cola.
- Mientras que la cola no esté vacía, tomamos el nodo del frente y **eliminamos** virtualmente cada una de sus **aristas** del grafo, actualizando el grado de entrada de sus vecinos y metiendo a la cola los **nodos** que pasen a tener grado **0**.

# Implementación (BFS)

## Kahn's Algorithm

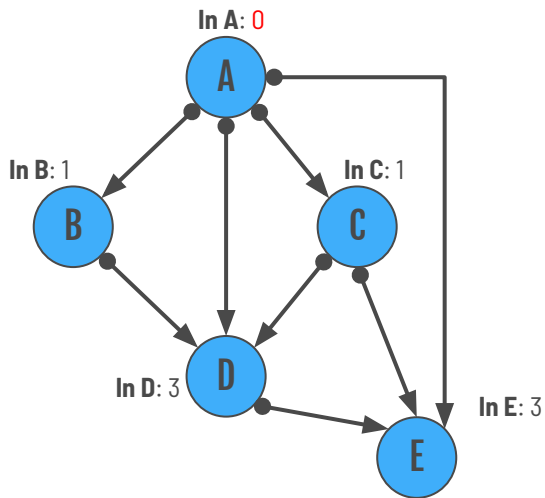




# Implementación (BFS)

## Kahn's Algorithm

Order:  $\{\emptyset\}$

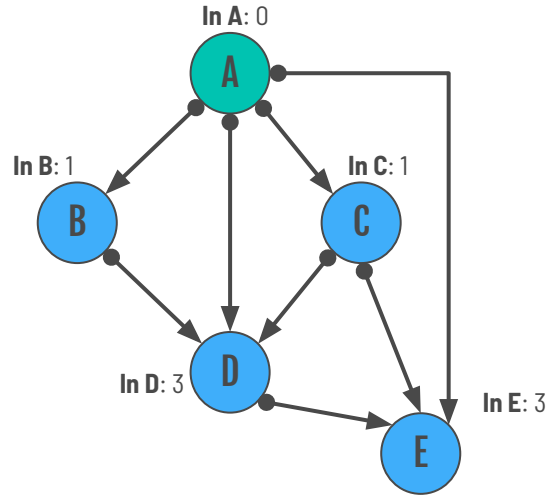


Queue:  $\{\emptyset\}$

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

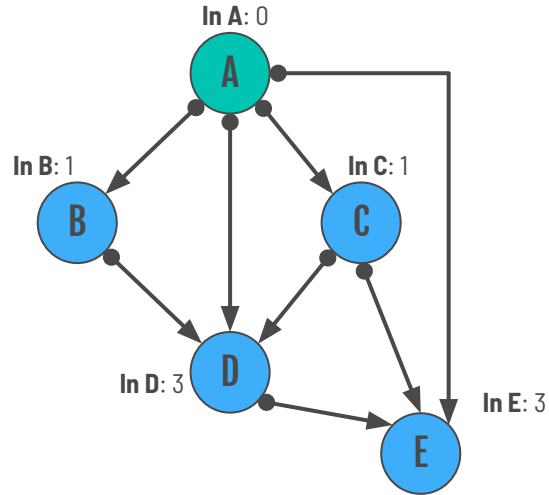


Queue: {A}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

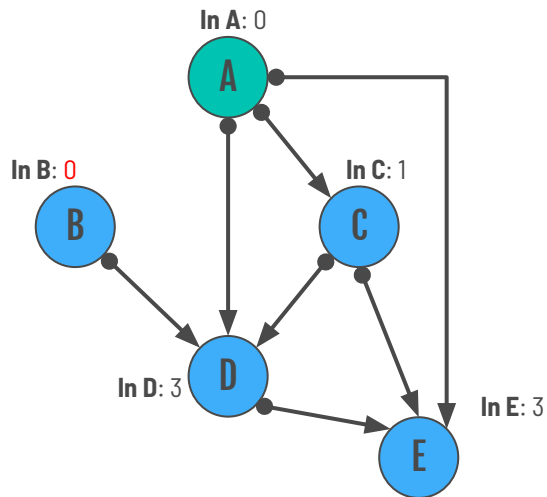


Queue: {}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

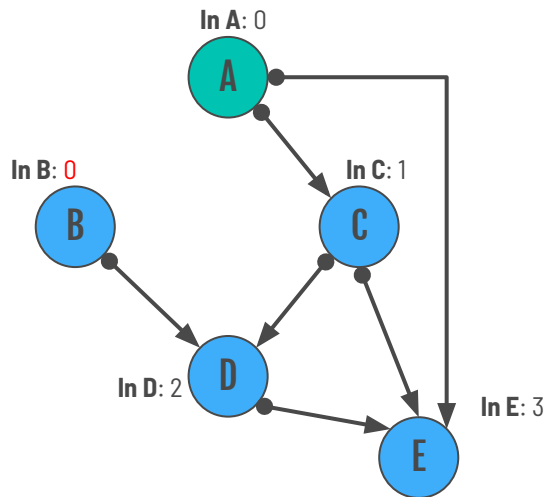


Queue: {B}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

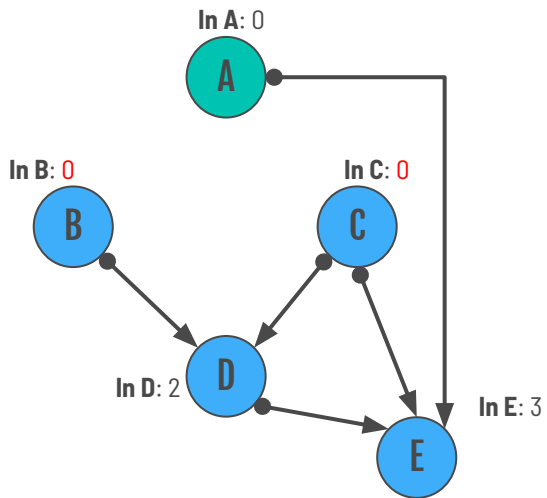


Queue: {B}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

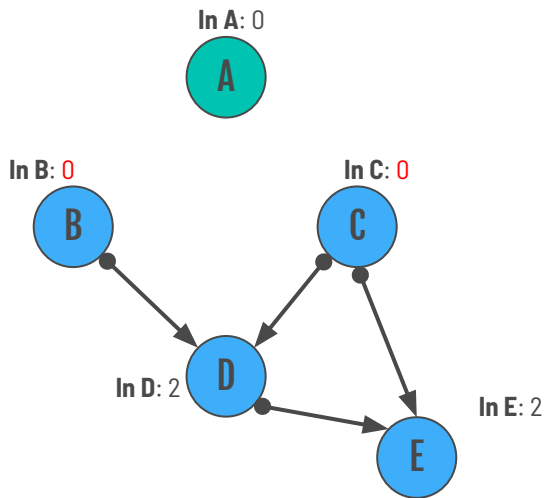


Queue: {B, C}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A}

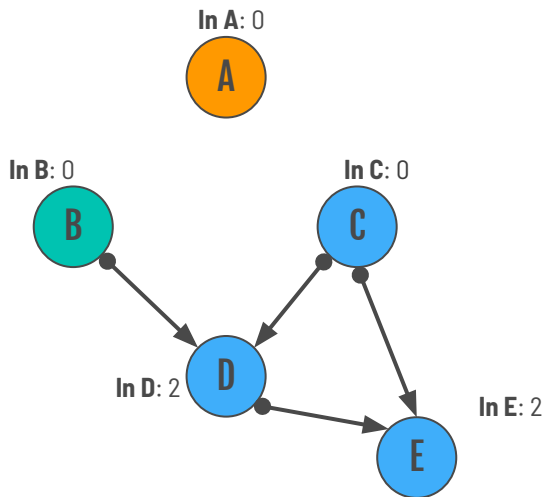


Queue: {B, C}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B}



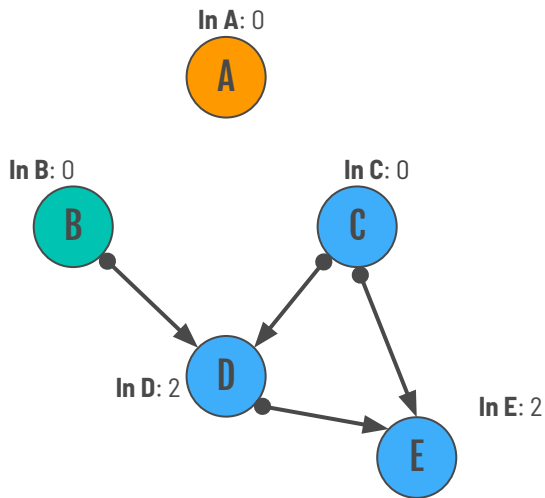
Queue: {**B**, C}



# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B}

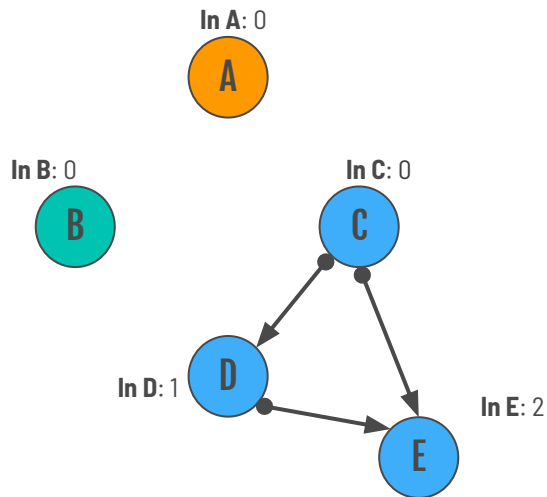


Queue: {C}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B}

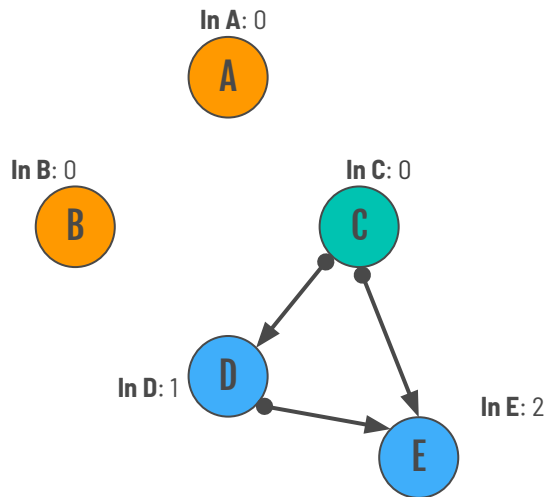


Queue: {C}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B}

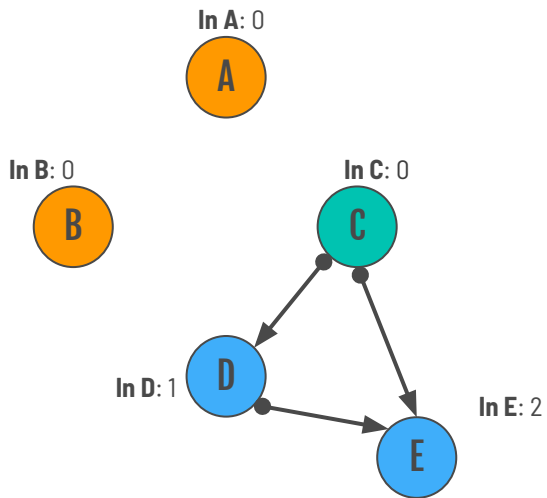


Queue: {**C**}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B}

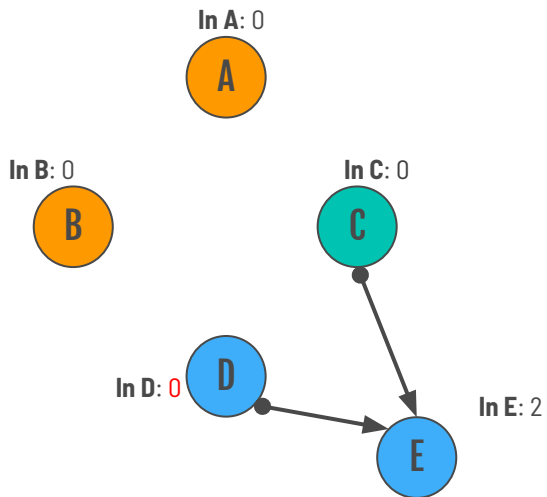


Queue: {}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C}

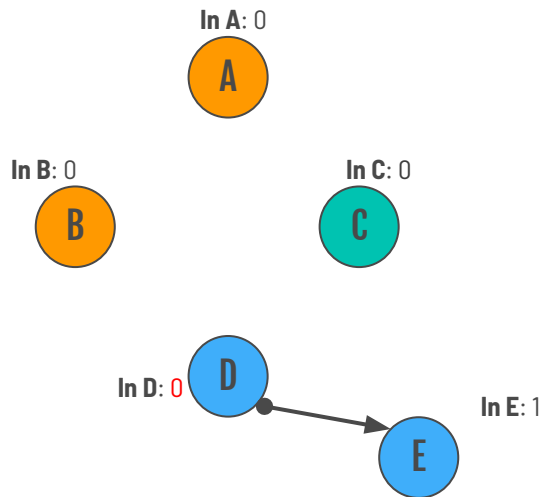


Queue: {D}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C}

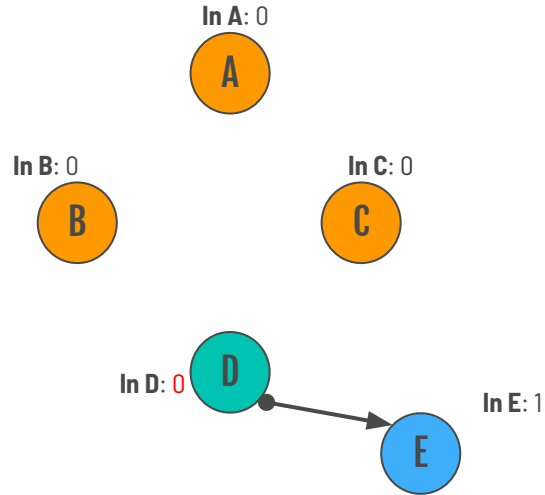


Queue: {D}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C, D}

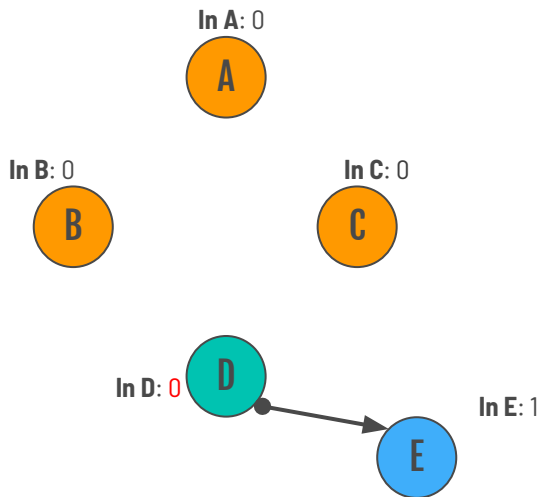


Queue: {**D**}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C, D}



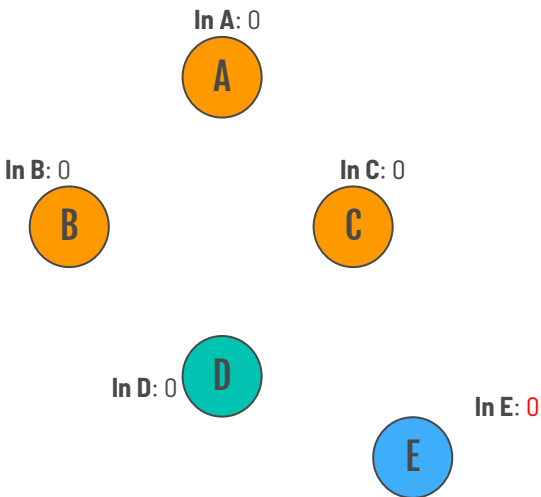
Queue: {}



# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C, D}

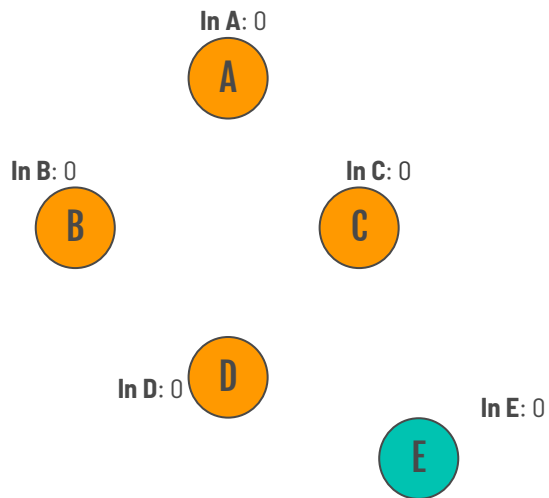


Queue: {E}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C, D, E}

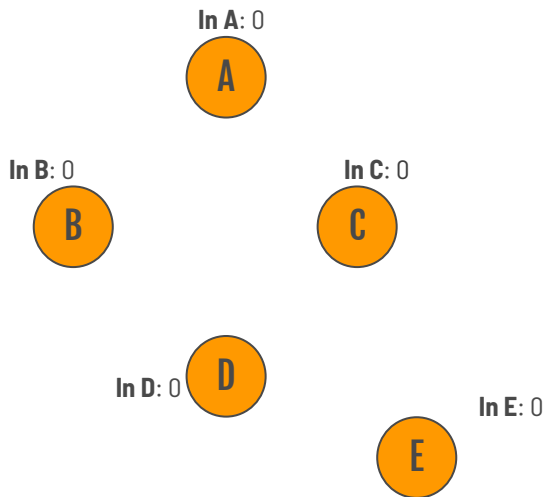


Queue: {**E**}

# Implementación (BFS)

## Kahn's Algorithm

Order: {A, B, C, D, E}



Queue: {}

# Implementación (Kahn's Algorithm)

C++

```
const int N = 2e5 + 2;

int n, m;
vector<int> adj[N];
int indegree[N] = {0};

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
        indegree[b]++;
    }
```

```
queue<int> Q;
for (int i = 0; i < n; ++i) if (indegree[i] == 0) Q.push(i);

vector<int> toposort;
while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    toposort.push_back(u);
    for (int v : adj[u]) {
        indegree[v]--;
        if (indegree[v] == 0) Q.push(v);
    }
}
if ((int)toposort.size() < n) {
    // there is a cycle
}
// use toposort
```



¿Preguntas?



# Implementación (DFS)

## Topological Sorting

```
const int N = 2e5 + 2;  
  
int n, m;  
vector<int> adj[N];
```

```
vector<int> toposort;  
  
void dfs(int u) {  
    visited[u] = 1;  
    for (int v : adj[u]) {  
        if (visited[v])  
            continue;  
        dfs(v);  
    }  
    toposort.push_back(u);  
}
```

**Disclaimer:** Si te garantizan que el grafo que se dan es un DAG puedes utilizar esta implementación.

# Implementación (DFS)

## Topological Sorting

```
const int N = 2e5 + 2;  
  
int n, m;  
vector<int> adj[N];
```

**Disclaimer:** Si el grafo que te dan no es un DAG puedes utilizar esta otra implementación.

```
vector<int> toposort;  
  
bool dfs(int u) {  
    visited[u] = 1;  
    onstack[u] = 1;  
    for (int v : adj[u]) {  
        if (visited[v] && onstack[v]) {  
            // There is a circle  
            return true;  
        } else if (!visited[v] && dfs(v)) {  
            // There is a circle  
            return true;  
        }  
    }  
    onstack[u] = 0;  
    toposort.push_back(u);  
    return false;  
}
```

# Referencias

[1] Re, L.S. (2020-07-16) 'Grafos - Inicial', in.

<https://www.pc-arg.com/media/attachment/graphs.pdf>

[2] Fiset, W. (2018-06-19) - 'Dijkstra's Shortest Path Algorithm', in.

[https://github.com/williamfiset/Algorithms/blob/master/slides/graphtheory/graph\\_theory\\_algorithms.pdf](https://github.com/williamfiset/Algorithms/blob/master/slides/graphtheory/graph_theory_algorithms.pdf)