Grafos: recorrido

AED II

2020

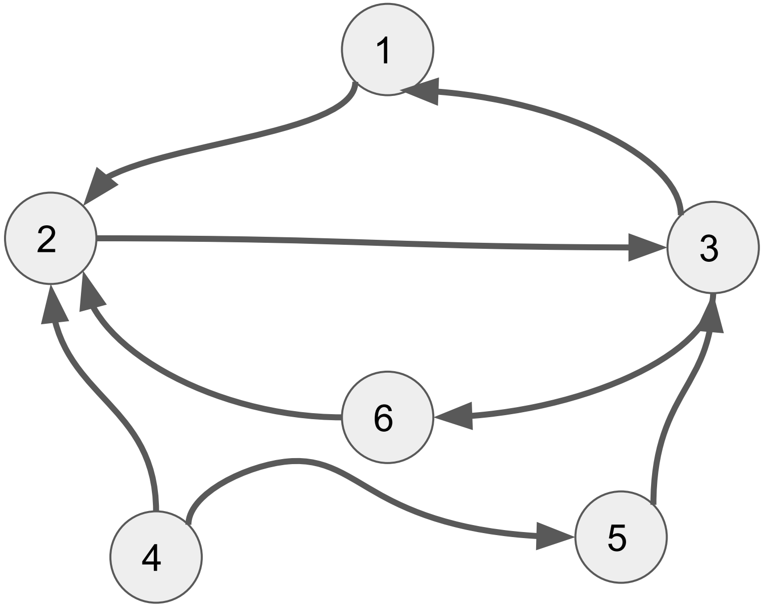
UBP UNIVERSIDAD BLAS PASCAL

clausura transitiva de un grafo

En un grafo dirigido, a menudo nos interesa conocer el conjunto de vértices que pueden ser alcanzados desde un vértice dado recorriendo los arcos del grafo en la dirección de las aristas.

La clausura transitiva es el grafo que resulta de añadir un arco dirigido desde u hacia v si existe algún camino para ir desde u hacia v.

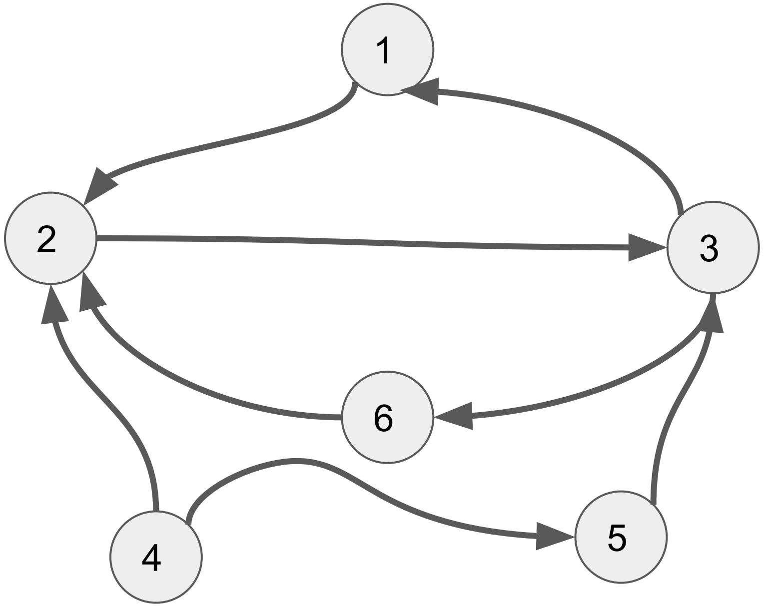
clausura transitiva de un grafo dirigido



Matriz de Adyacencia

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	1	0	0	0	0	1
4	0	1	0	0	1	0
5	0	0	1	0	0	0
6	0	1	0	0	0	0

clausura transitiva de un grafo dirigido



Matriz de Clausura Transitiva

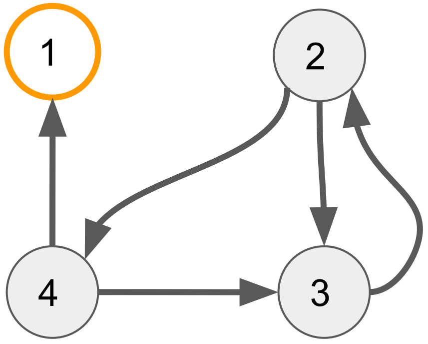
	1	2	3	4	5	6
1	1	1	1	0	0	1
2	1	1	1	0	0	1
3	1	1	1	0	0	1
4	1	1	1	0	1	1
5	1	1	1	0	0	1
6	1	1	1	0	0	1

algoritmo de Warshall

El algoritmo de Warshall realiza la clausura transitiva en un grafo representado por matriz de adyacencia

y se basa en el hecho de que "si existe un medio para ir del vértice u al v, y un medio de ir del vértice v al w, entonces existe un medio para ir de del vértice u al w".

algoritmo de Warshall

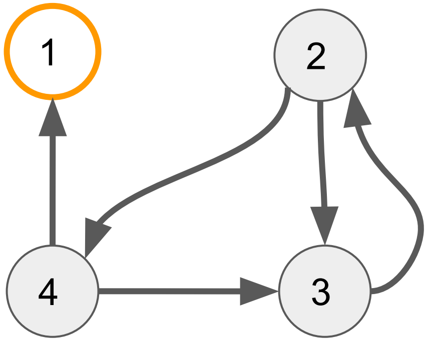


k = 1

T(0)

	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	1	0	1	0

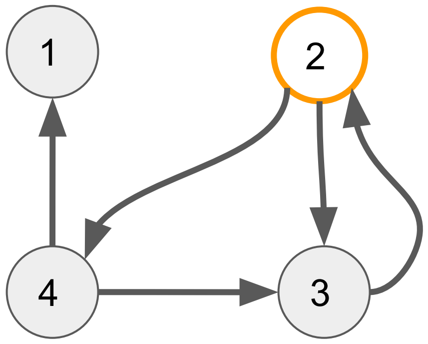
algoritmo de Warshall



T(1)

	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	1	0	1	0

algoritmo de Warshall

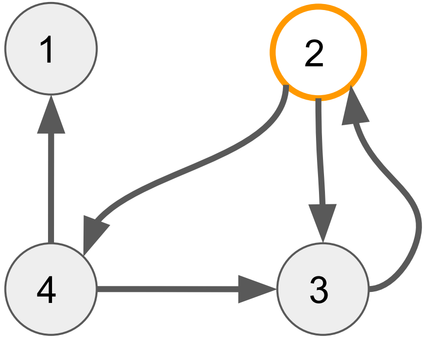


k = 2

T(1)

	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	1	0	1	0

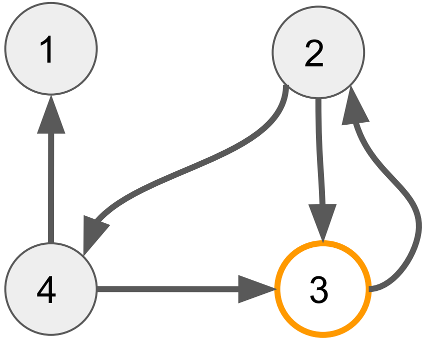
algoritmo de Warshall



T(2)

	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	0	1	1	1
4	1	0	1	0

algoritmo de Warshall

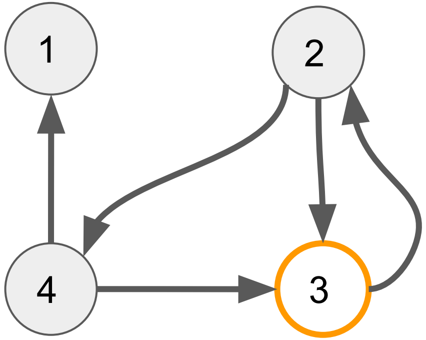


k = 3

T(2)

	1	2	3	4
1	0	0	0	0
2	0	0	1	1
3	0	1	1	1
4	1	0	1	0

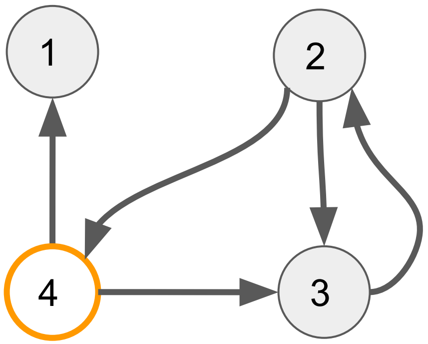
algoritmo de Warshall



T(3)

	1	2	3	4
1	0	0	0	0
2	0	1	1	1
3	0	1	1	1
4	1	1	1	1

algoritmo de Warshall

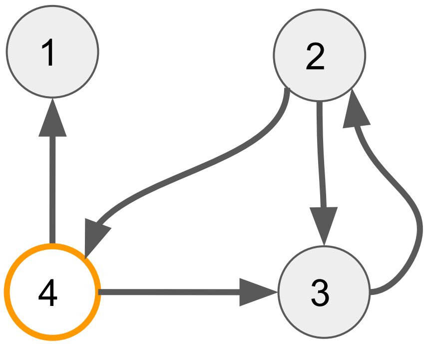


k = 4

T(3)

	1	2	3	4
1	0	0	0	0
2	0	1	1	1
3	0	1	1	1
4	1	1	1	1

algoritmo de Warshall



T(4)

	1	2	3	4
1	0	0	0	0
2	1	1	1	1
3	1	1	1	1
4	1	1	1	1

algoritmo de Warshall

```
for (int k = 0; k < v; k++)
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            ct[i][j] = ct[i][j] || (ct[i][k] && ct[k][j]);
```

algoritmo de Warshall

Si podemos ir de **i** a **k** y podemos ir de **k** a **j**, entonces existe un camino de **i** a **j**

```
for (int k = 0; k < v; k++)
    for (int i = 0; i < v; i++)
        for (int j = 0; j < v; j++)
            ct[i][j] = ct[i][j] || (ct[i][k] && ct[k][j]);
```

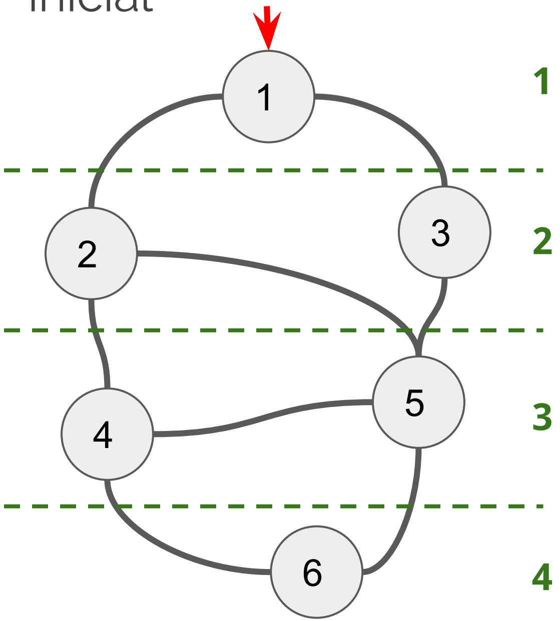
Ejercicio 1.09.

Implementá una función para obtener y mostrar la matriz de clausura transitiva por el método de Warshall.

Analizá la complejidad computacional de este método.

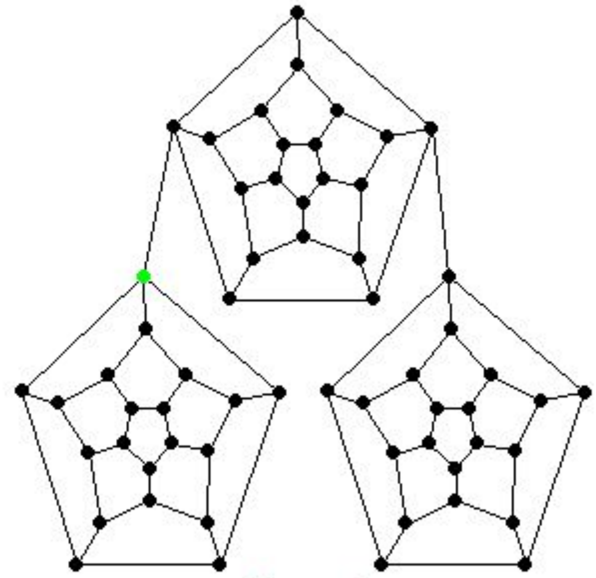
recorrido de un grafo: amplitud

El recorrido en amplitud o anchura (BFS - Breadth First Search) consiste en recorrer el grafo <u>en niveles</u> a partir del nodo que consideremos como nodo inicial



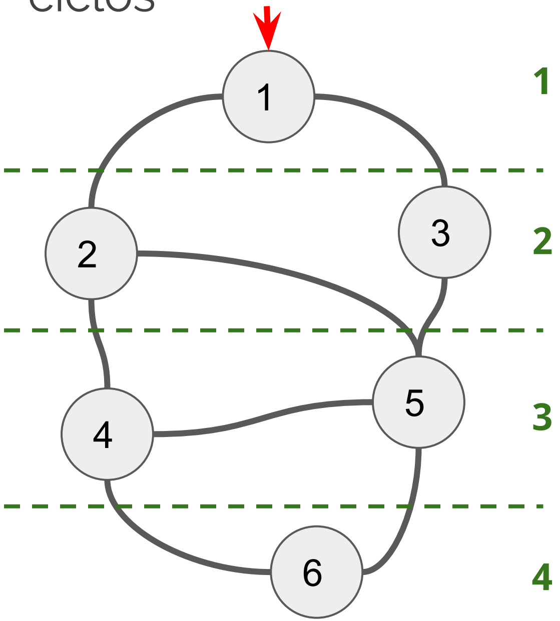
1, 2, 3, 4, 5, 6

recorrido de un grafo: amplitud



recorrido de un grafo: amplitud

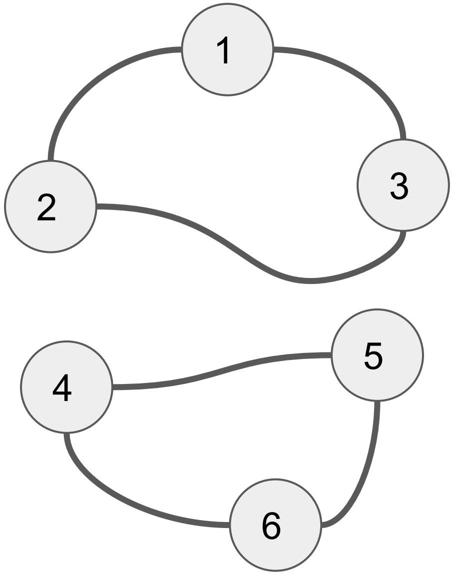
Debemos, sin embargo resolver una situación que se presenta en los grafos cíclicos: un vértice se visita más de una más de una vez cuando hay ciclos



La solución consiste en llevar registro de los nodos ya visitados: esto se puede implementar con un vector booleano

recorrido de un grafo: amplitud

Otro problema puede ser que tengamos un grafo desconectado o no conexo



En este caso deberíamos utilizar el mismo algoritmo, considerando cada nodo como nodo inicial.

recorrido de un grafo: amplitud

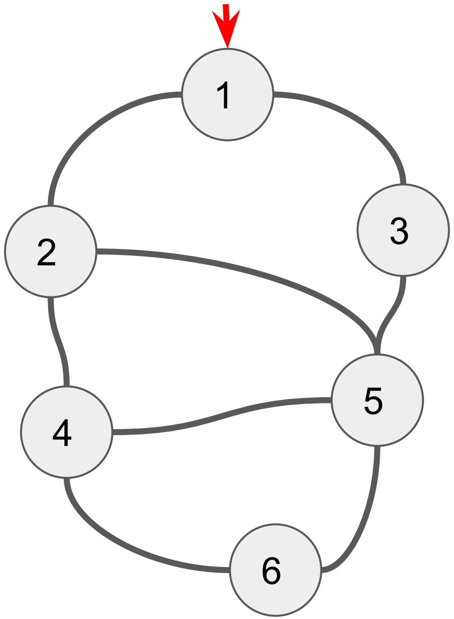
Algoritmo

Con G = (V, A) tomando el vértice origen u.

1. Encolar el vértice origen u.
2. Marcar el vértice u como visitado.
3. Procesar la cola hasta que no haya más vértices.
4. Extraer u de la cola y procesarlo
5. Para todo vértice v adyacente a u,(u,v) $\in$ A,
6. si v no ha sido visitado
7. encolar y visitar v

recorrido de un grafo: amplitud

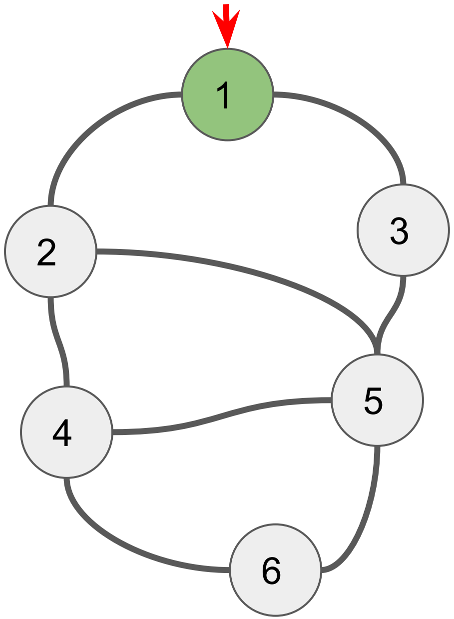
El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



	1	2	3	4	5	6		
Visitado:	0	0	0	0	0	0		
Cola:								
Salida:	[redacted]							

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



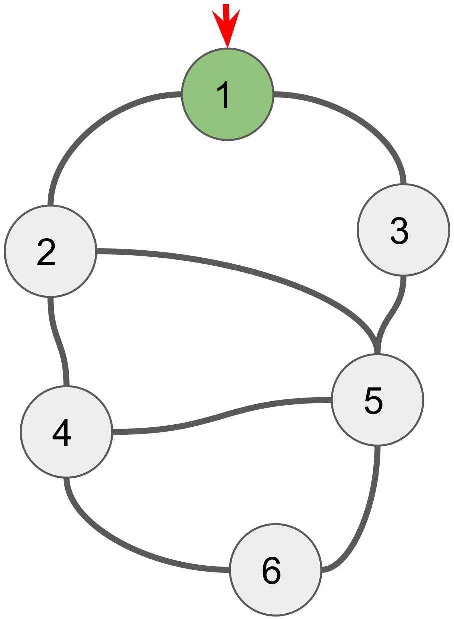
	1	2	3	4	5	6
Visitado:	1	0	0	0	0	0

Cola:	1							

Salida:

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



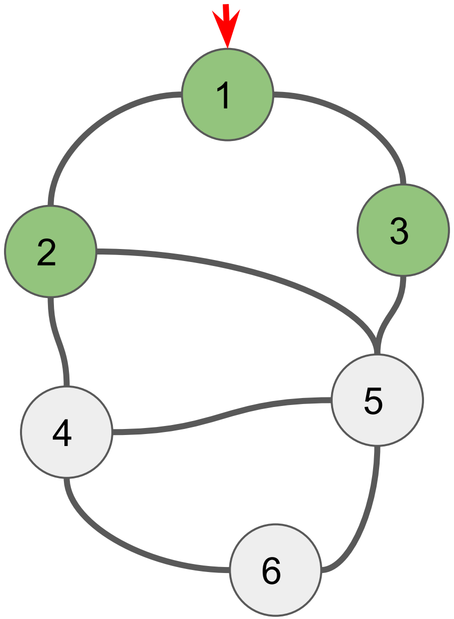
	1	2	3	4	5	6
Visitado:	1	0	0	0	0	0

Cola:

Salida: 1

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



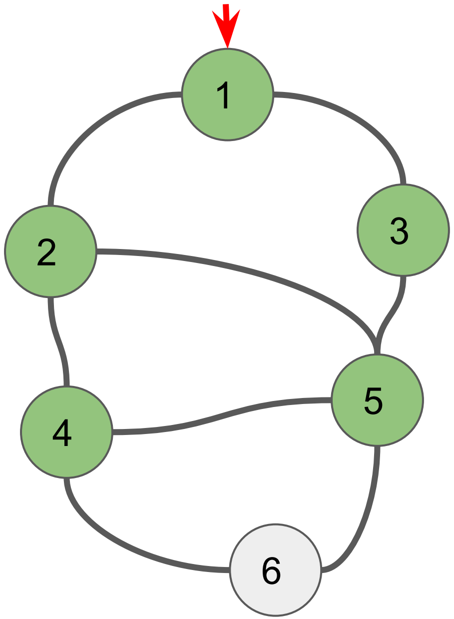
	1	2	3	4	5	6
Visitado:	1	1	1	0	0	0

Cola:	2	3						

Salida:	1

recorrido de un grafo: amplitud

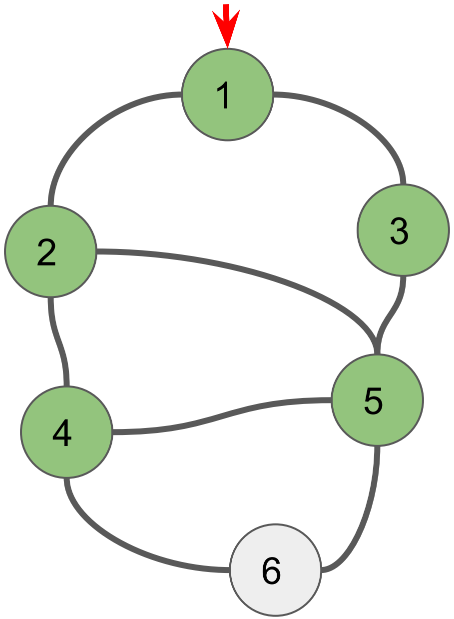
El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



	1	2	3	4	5	6		
Visitado:	1	1	1	1	1	0		
Cola:	3	4	5					
Salida:	1	2						

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



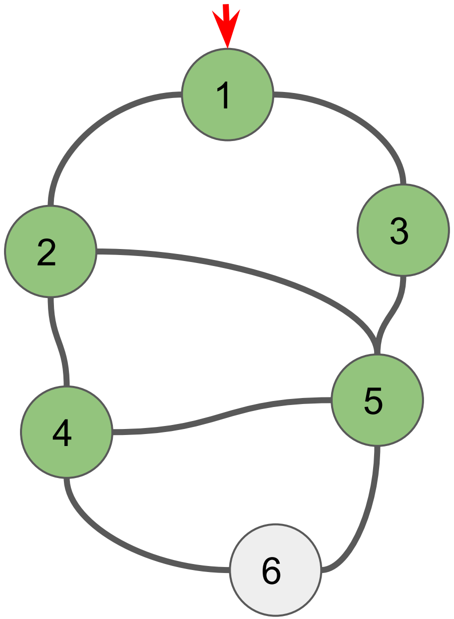
	1	2	3	4	5	6
Visitado:	1	1	1	1	1	0

Cola:	4	5						

Salida:	1	2	3

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



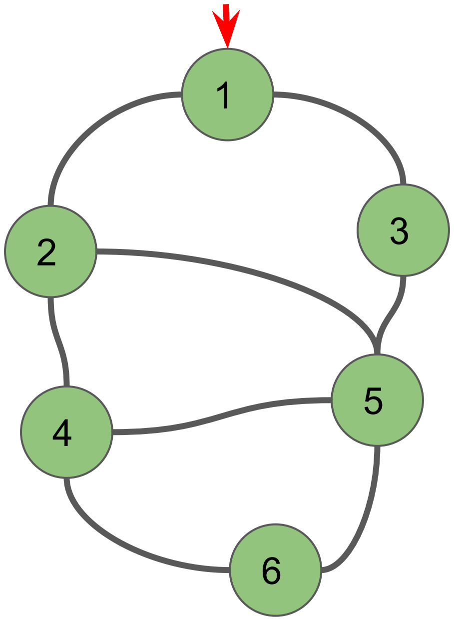
	1	2	3	4	5	6
Visitado:	1	1	1	1	1	0

Cola:	5							

Salida:	1	2	3	4

recorrido de un grafo: amplitud

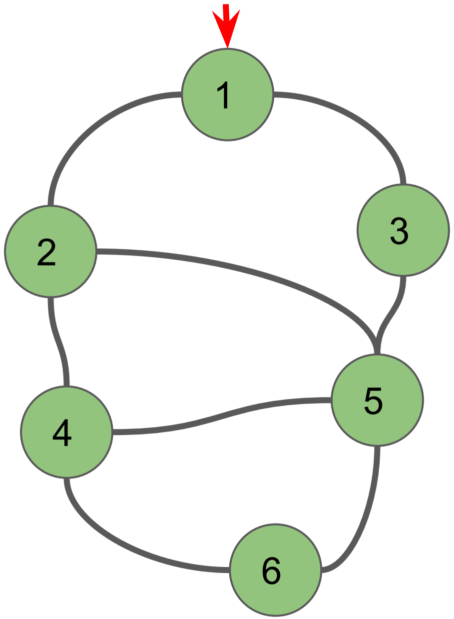
El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



	1	2	3	4	5	6		
Visitado:	1	1	1	1	1	1		
Cola:	5	6						
Salida:	1	2	3	4				

recorrido de un grafo: amplitud

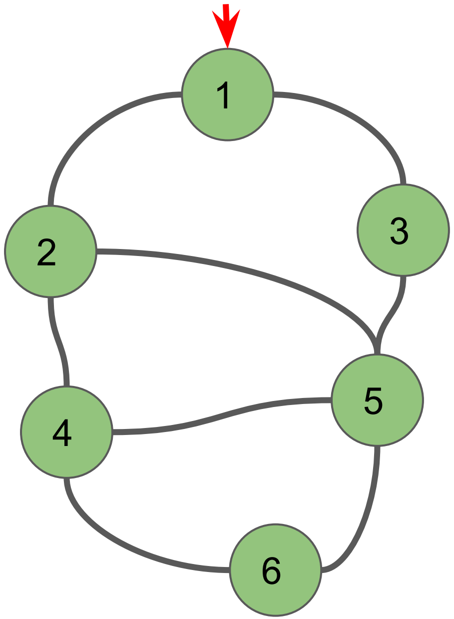
El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



	1	2	3	4	5	6		
Visitado:	1	1	1	1	1	1		
Cola:	6							
Salida:	1	2	3	4	5			

recorrido de un grafo: amplitud

El algoritmo que presentamos se asemeja al que usamos para un árbol binario e incluye una cola.



	1	2	3	4	5	6		
Visitado:	1	1	1	1	1	1		
Cola:								
Salida:	1	2	3	4	5	6		

Ejercicio 1.10.

Implementá una función que permita recorrer un grafo en amplitud partiendo de su representación en listas de adyacencia.

Usamos la implementación de cola de la biblioteca estándar de plantillas (STL):

```
#include <queue>

queue <int> cola;            // creamos una instancia de queue para valores int

int valor  = cola.front();   // asigna el primer valor de la cola

cola.push(valor);            // ingresa un valor en la cola

cola.pop();                  // extrae el primer valor de la cola

// referencia en: http://www.cplusplus.com/reference/queue/queue/
```

Ejercicio 1.11.

Implementá una función que permita recorrer un grafo en amplitud partiendo de su representación con matriz de adyacencia.

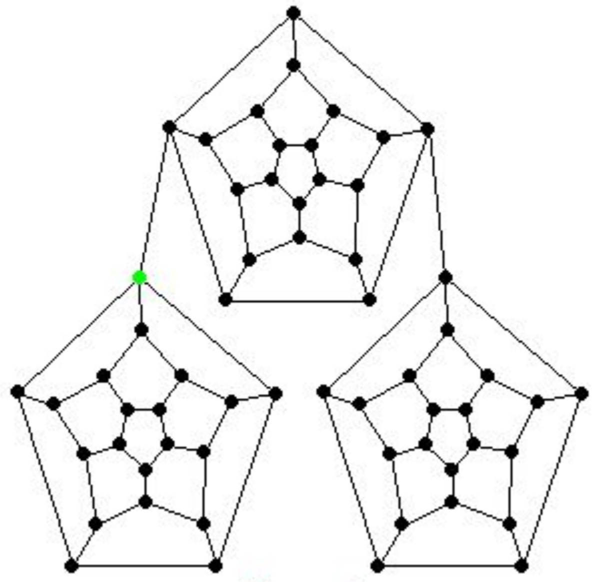
recorrido de un grafo: profundidad

El recorrido en profundidad (DFS - Depth First Search) Es una generalización del recorrido preorden de un árbol.

La estrategia consiste en partir de un vértice determinado V y a partir de allí, cuando se visita un nuevo vértice, explorar cada camino que salga de él. Hasta que no se haya finalizado de explorar uno de los caminos no se comienza con el siguiente. Un camino deja de explorarse cuando se llega a un vértice ya visitado.

Si existían vértices no alcanzables desde v el recorrido queda incompleto; entonces, se debe seleccionar algún vértice como nuevo vértice de partida, y repetir el proceso.

recorrido de un grafo: profundidad



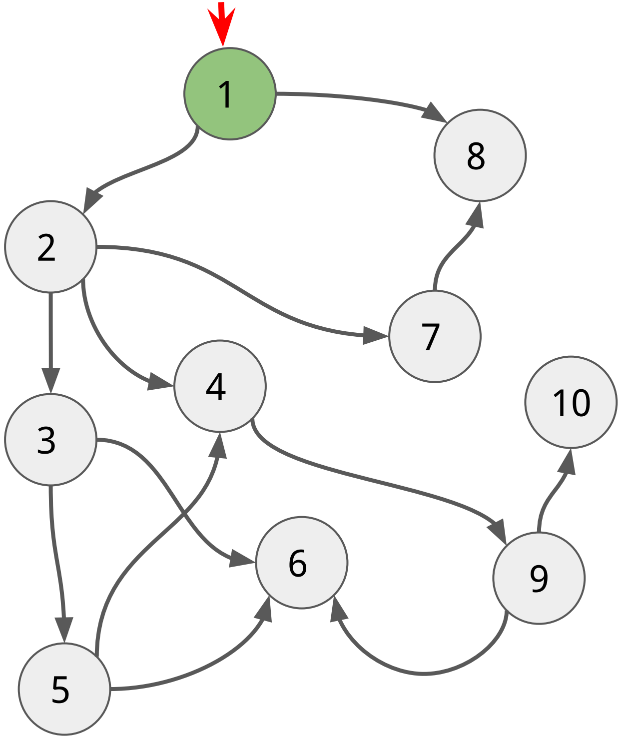
recorrido de un grafo: profundidad

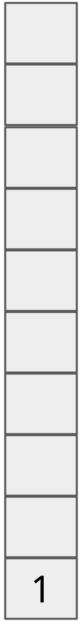
Algoritmo

dado G = (V, A)

1. Marcar todos los vértices como no visitados.
2. Elegir vértice u como punto de partida.
3. Marcar u como visitado.
4. Para todo v adyacente a u,(u,v) $\in$ A, si v no ha sido visitado, llamar recursivamente para ejecutar (3) y (4) para v.

recorrido de un grafo: profundidad



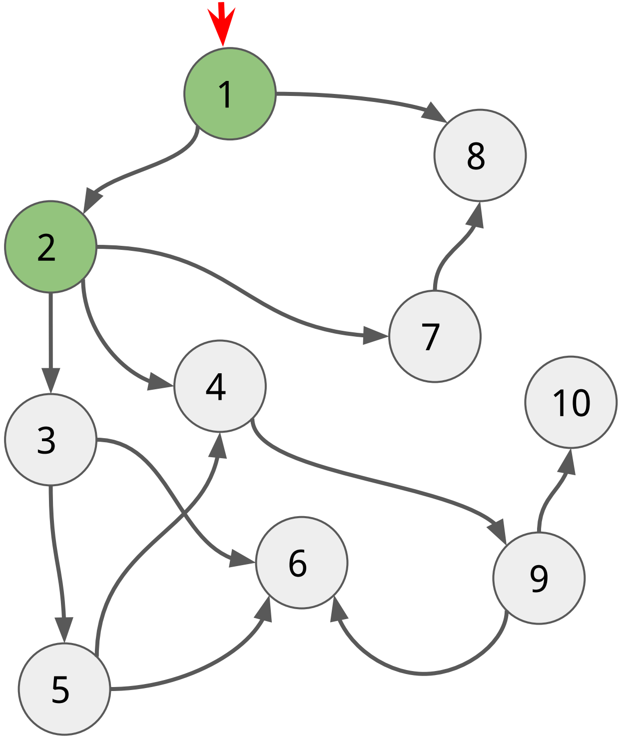


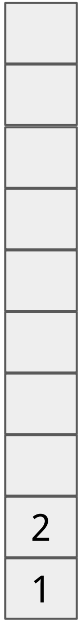
Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	0	0	0	0	0	0	0	0	0

Salida: 1

recorrido de un grafo: profundidad



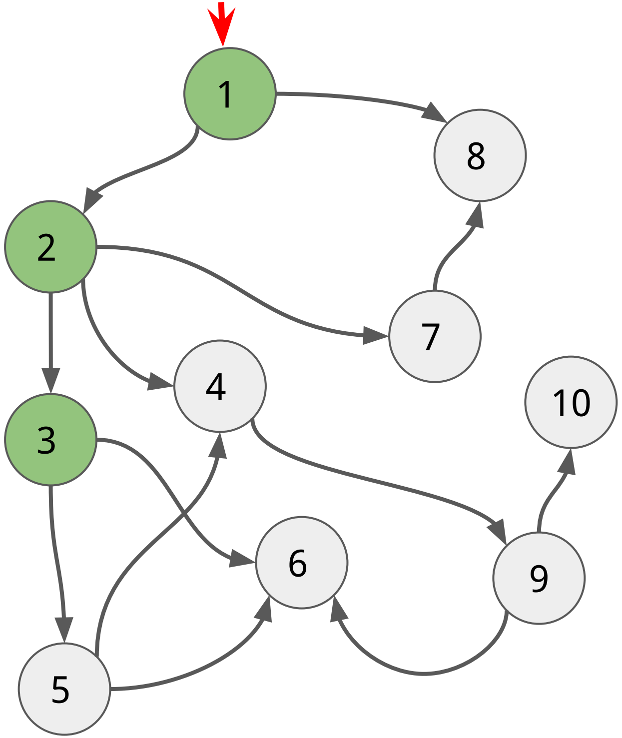


Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	0	0	0	0	0	0	0	0

Salida: 1 2

recorrido de un grafo: profundidad



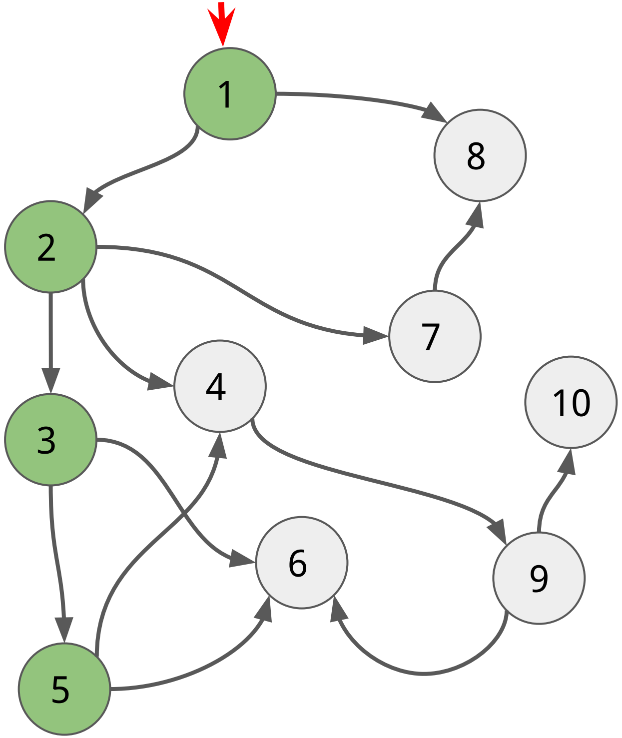
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	0	0	0	0	0	0	0

Salida: 1 2 3

recorrido de un grafo: profundidad



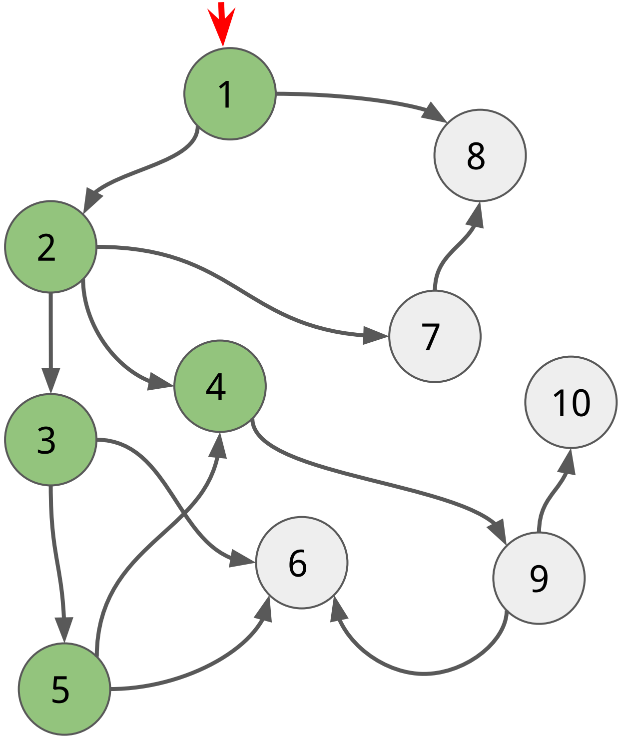
Stack (memoria)
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	0	1	0	0	0	0	0

Salida: 1 2 3 5

recorrido de un grafo: profundidad

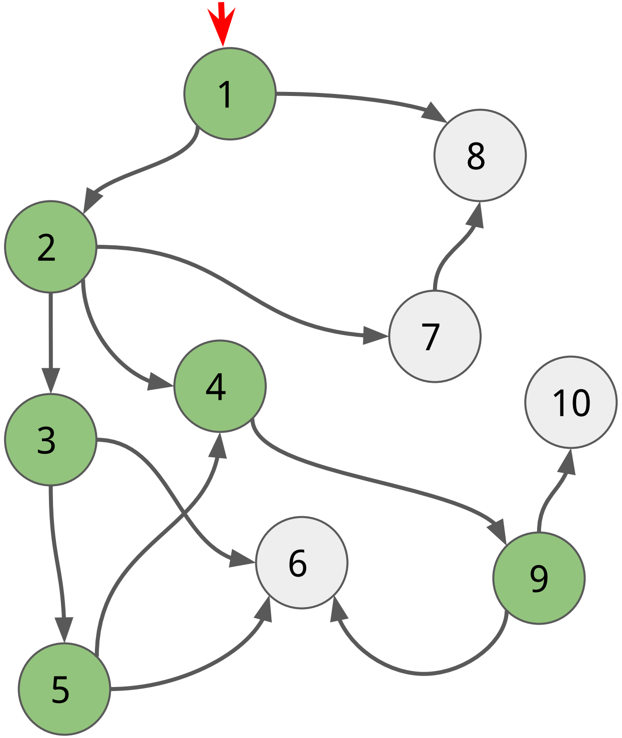


4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	0	0	0	0	0
Salida:	1	2	3	5	4					

recorrido de un grafo: profundidad



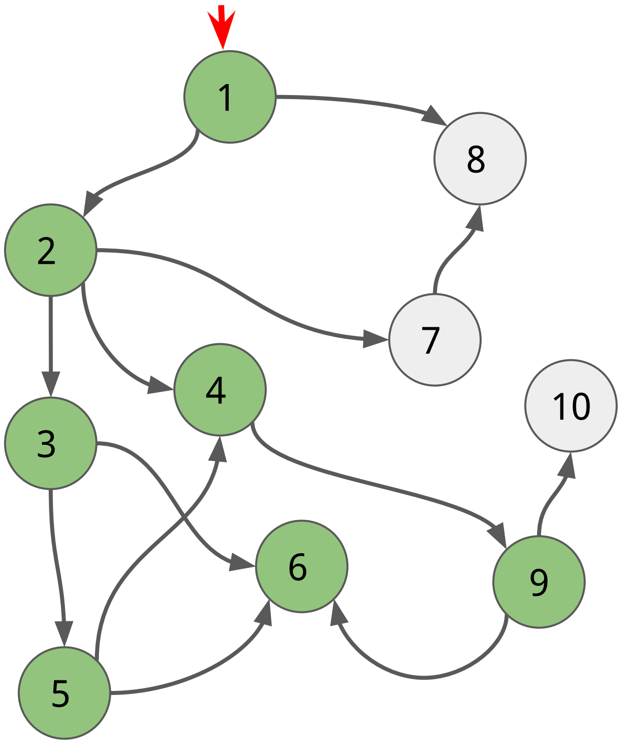
9
4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	0	0	0	1	0

Salida: 1 2 3 5 4 9

recorrido de un grafo: profundidad



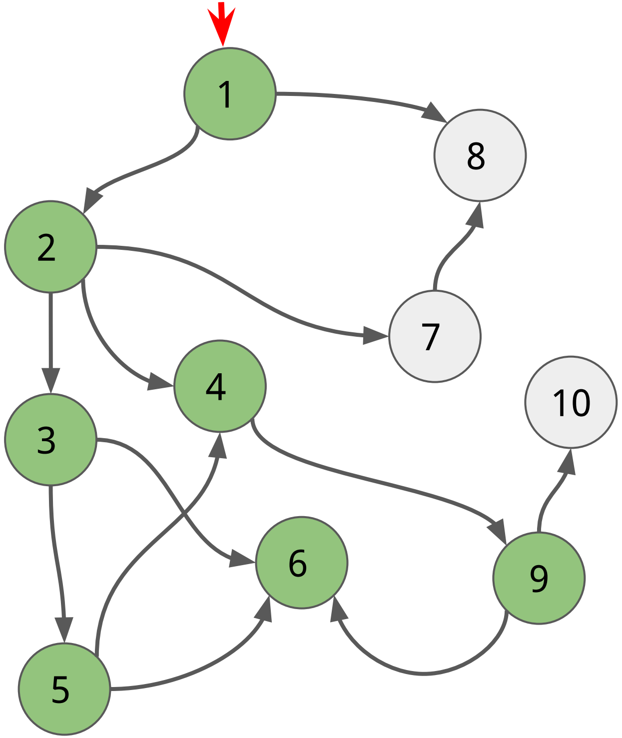
6
9
4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	0

Salida:	1	2	3	5	4	9	6			

recorrido de un grafo: profundidad



9
4
5
3
2
1

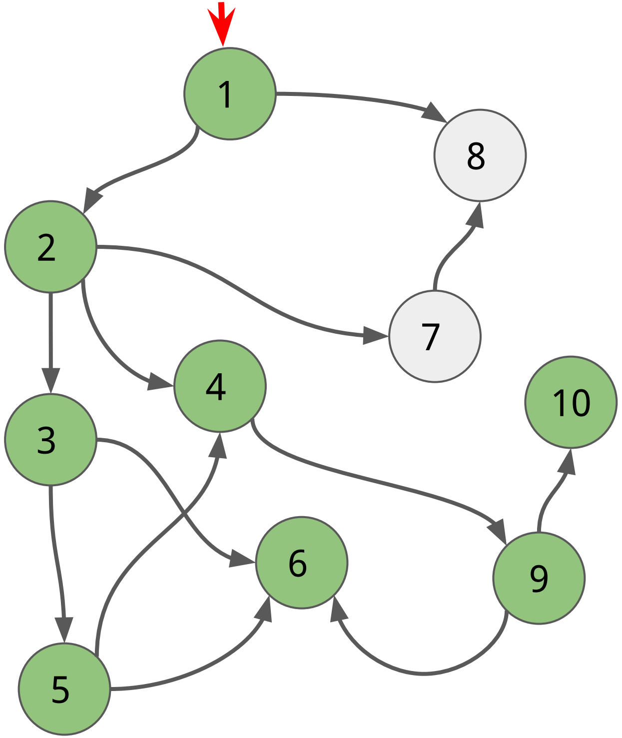
Stack (memoria)

Como el 6 no tiene vértices adyacentes volvemos a la llamada recursiva del vértice 9

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	0

Salida: 1 2 3 5 4 9 6

recorrido de un grafo: profundidad



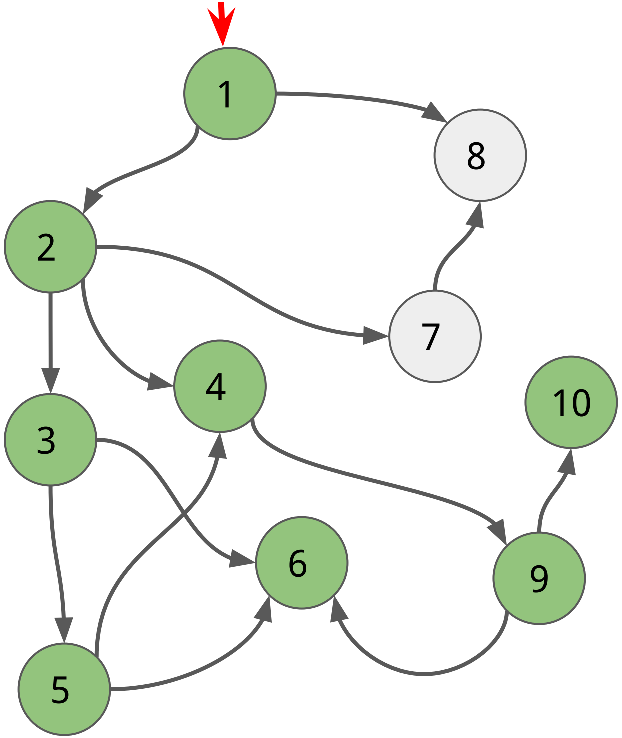
10
9
4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1

Salida:	1	2	3	5	4	9	6	10	

recorrido de un grafo: profundidad



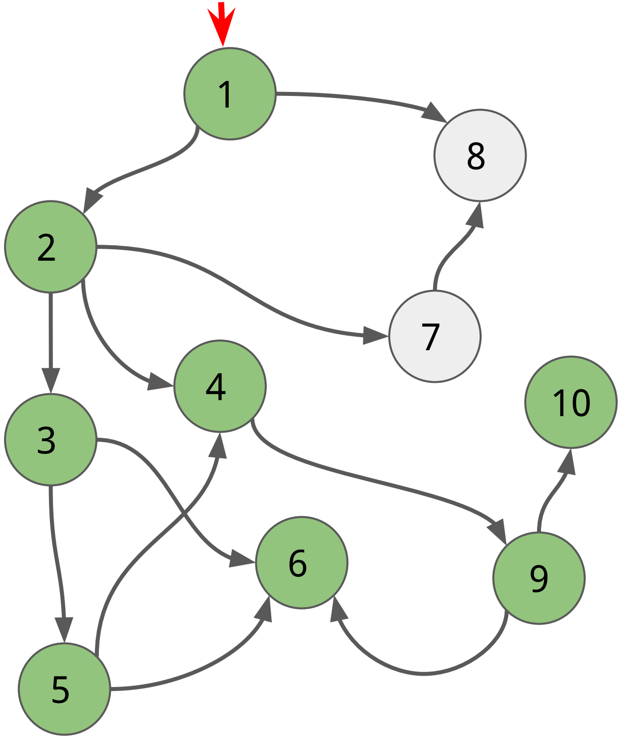
9
4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1

Salida:	1	2	3	5	4	9	6	10

recorrido de un grafo: profundidad



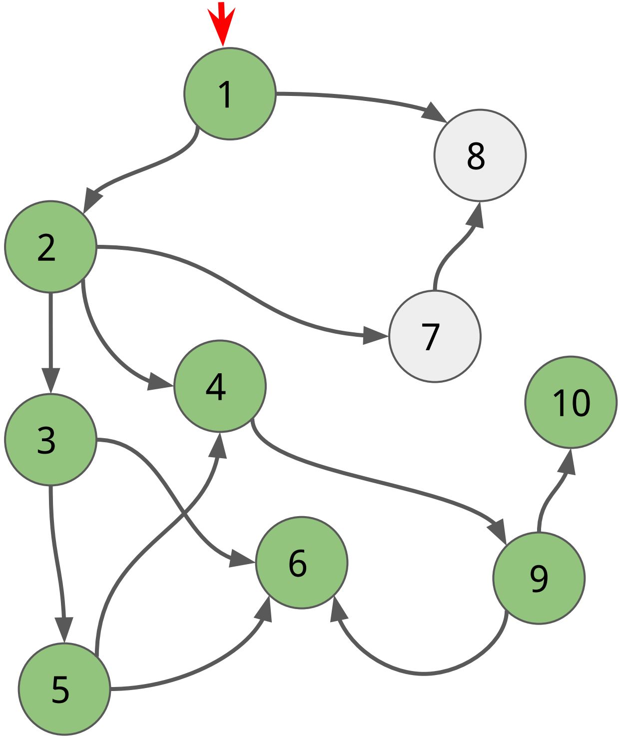
4
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1

Salida: 1 2 3 5 4 9 6 10

recorrido de un grafo: profundidad



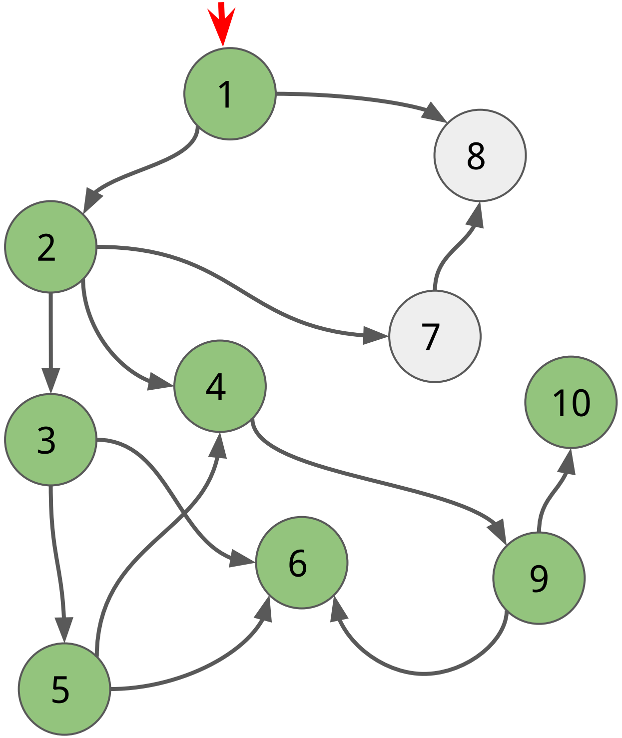
5
3
2
1

Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1

Salida:	1	2	3	5	4	9	6	10

recorrido de un grafo: profundidad



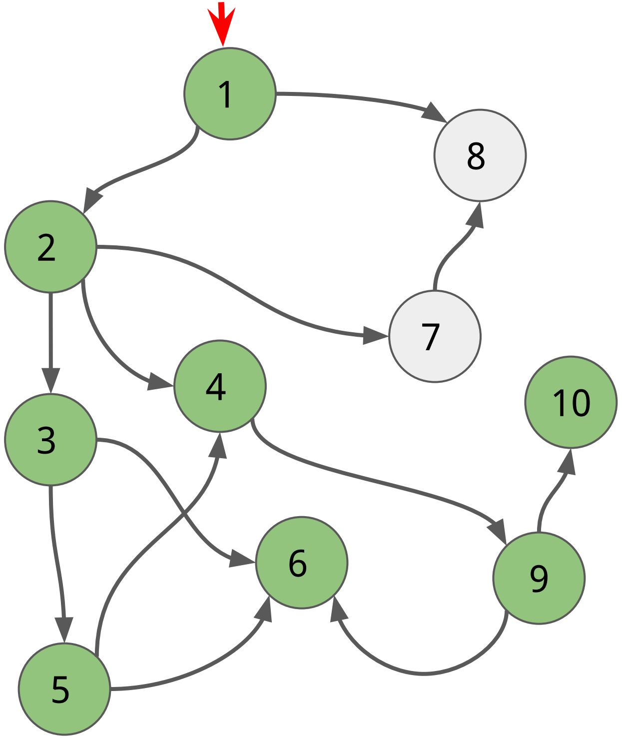
Stack (memoria)
3
2
1

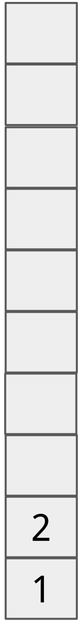
Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1

Salida:	1	2	3	5	4	9	6	10

recorrido de un grafo: profundidad

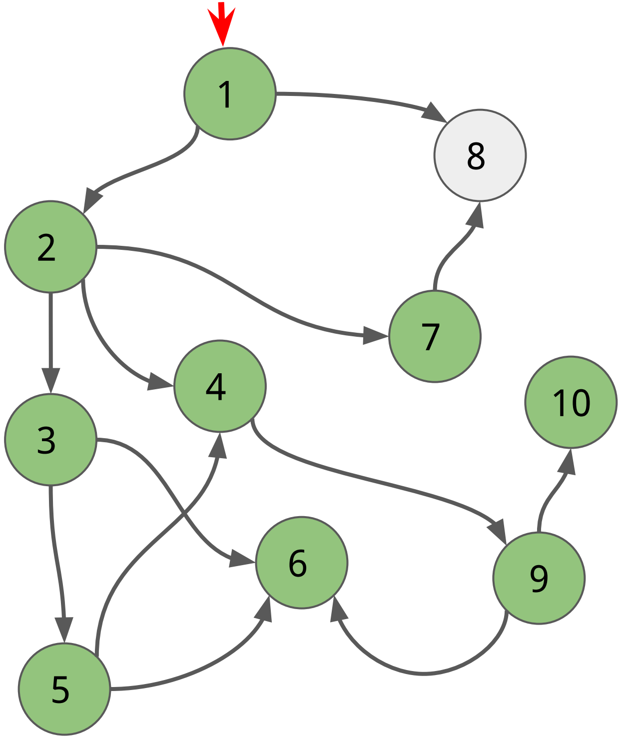




Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	0	0	1	1
Salida:	1	2	3	5	4	9	6	10		

recorrido de un grafo: profundidad

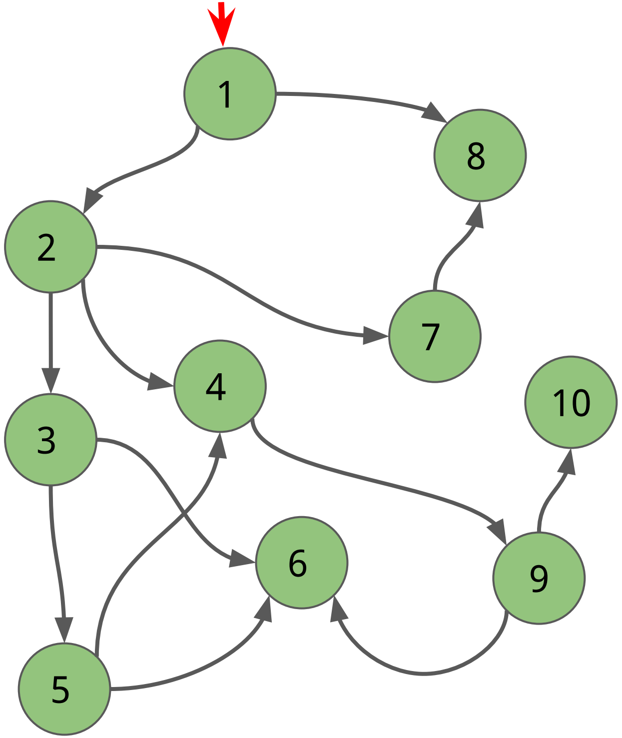


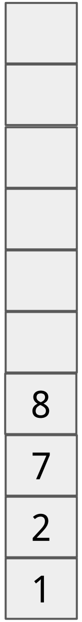
	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	0	1	1
Salida:	1	2	3	5	4	9	6	10	7	

7
2
1

Stack (memoria)

recorrido de un grafo: profundidad

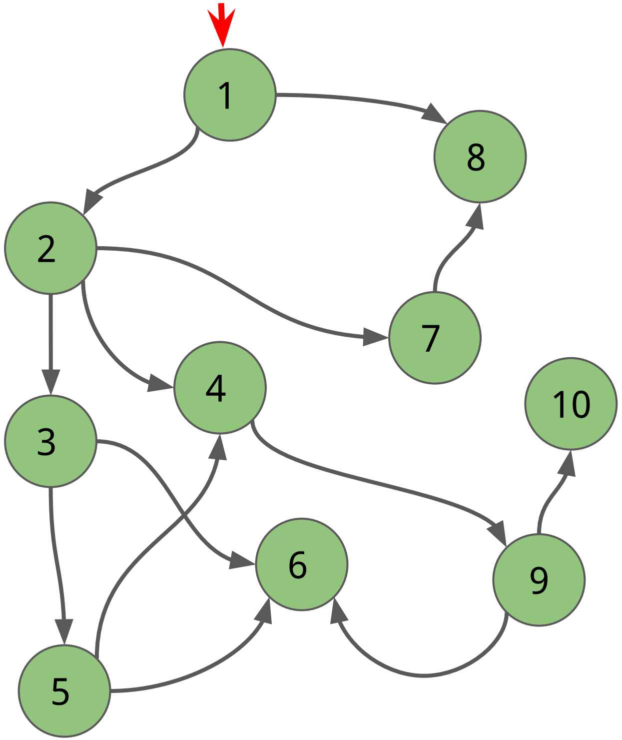




Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	1	1	1
Salida:	1	2	3	5	4	9	6	10	7	8

recorrido de un grafo: profundidad



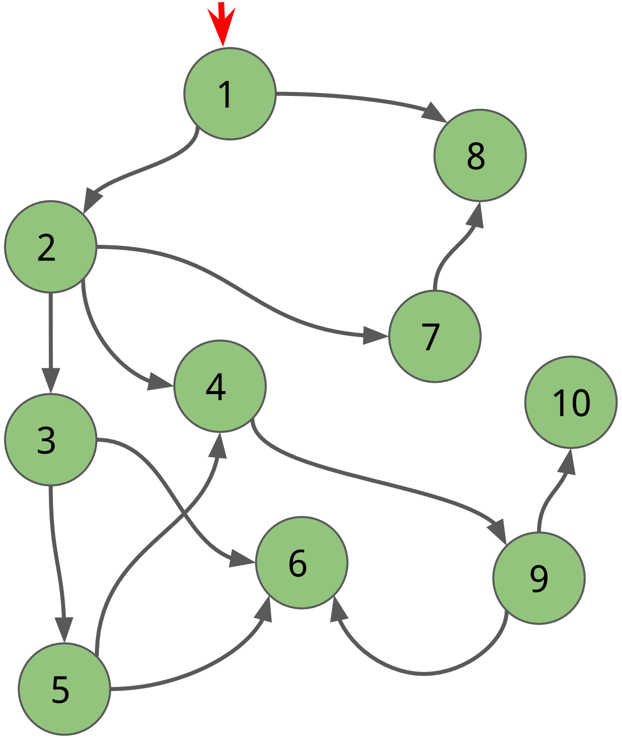
Stack (memoria)
7
2
1

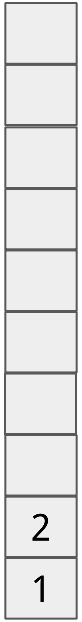
Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	1	1	1

Salida:	1	2	3	5	4	9	6	10	7	8

recorrido de un grafo: profundidad

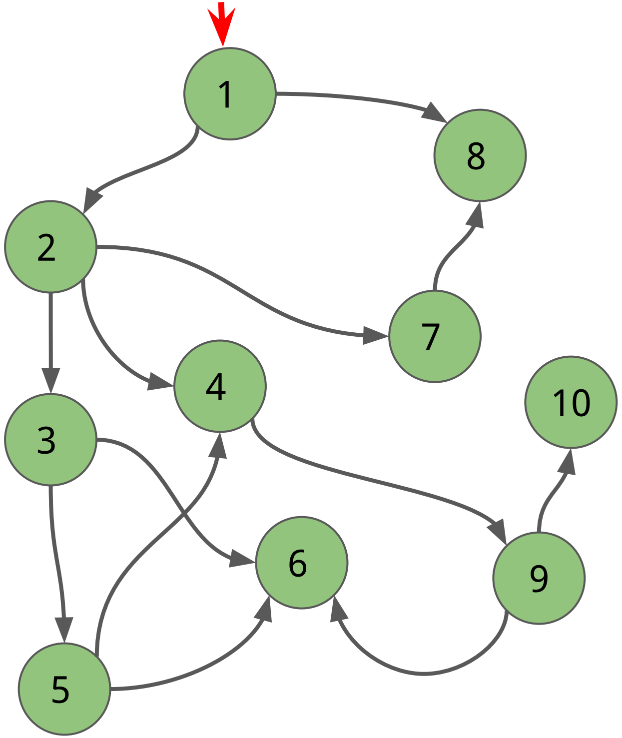


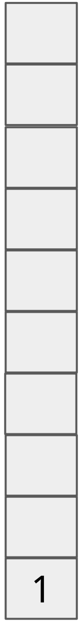


Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	1	1	1
Salida:	1	2	3	5	4	9	6	10	7	8

recorrido de un grafo: profundidad



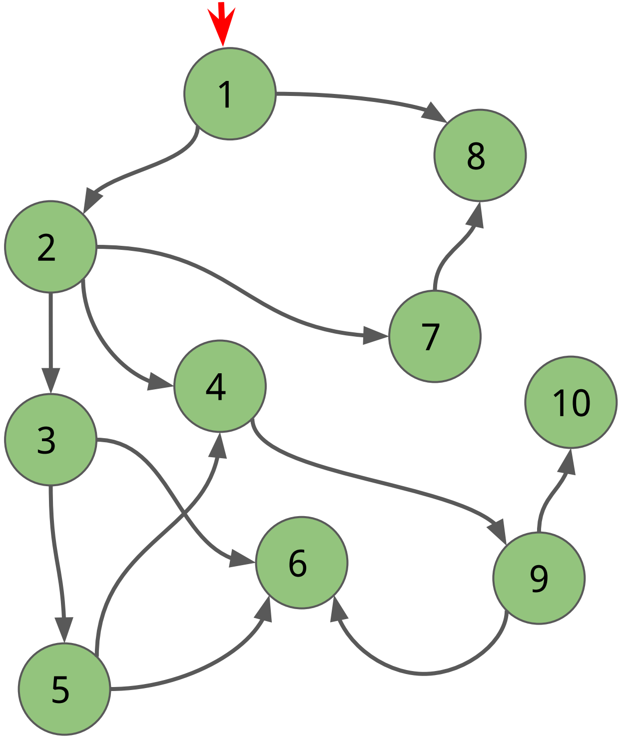


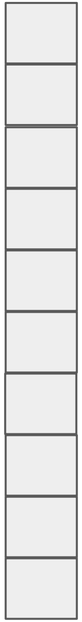
Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	1	1	1

Salida:	1	2	3	5	4	9	6	10	7	8

recorrido de un grafo: profundidad





Stack (memoria)

	1	2	3	4	5	6	7	8	9	10
Visitado:	1	1	1	1	1	1	1	1	1	1
Salida:	1	2	3	5	4	9	6	10	7	8

Ejercicio 1.12.

Implementá una función que permita recorrer un grafo en profundidad partiendo de su representación en listas de adyacencia