Ayudantía 9: Algoritmos en grafos

Cristian Alonso Carrasco

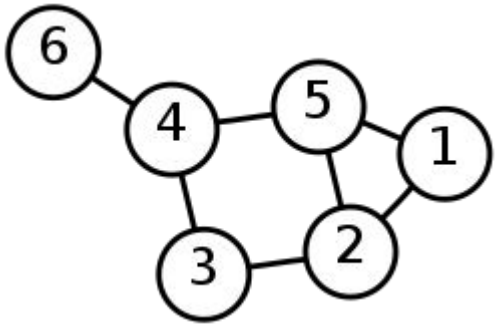
Contenidos

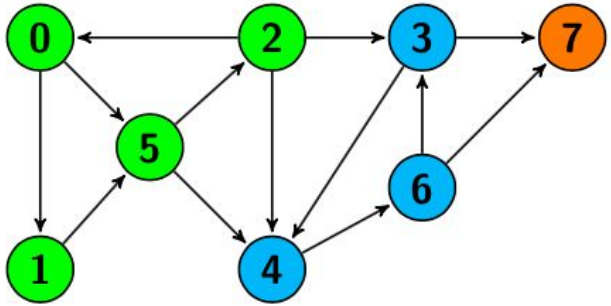
- Repaso de algoritmos sobre grafos.
  DFS, SCC, BFS, TOPOSORT
- Ejercicios de dfs

¿Qué es un Grafo?

grafo:

- **Estructura de datos** compuesta por **nodos** los cuales están unidos por **aristas.** Estas aristas pueden estar dirigidas o no. Un grafo puede ser dirigido o no dirigido.





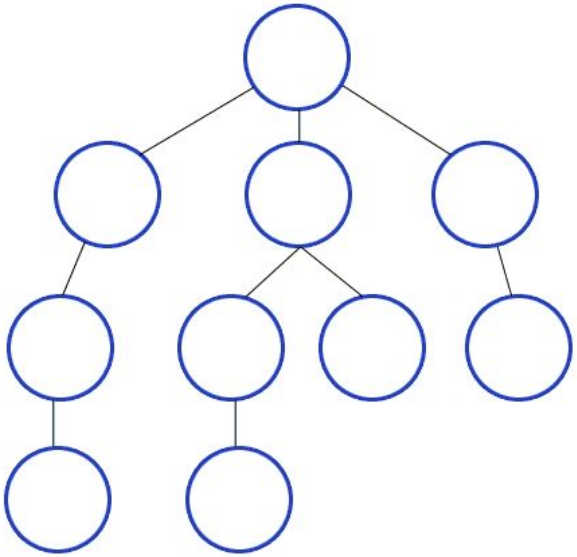
DFS

- **Depth First Search o Búsqueda en profundidad.**

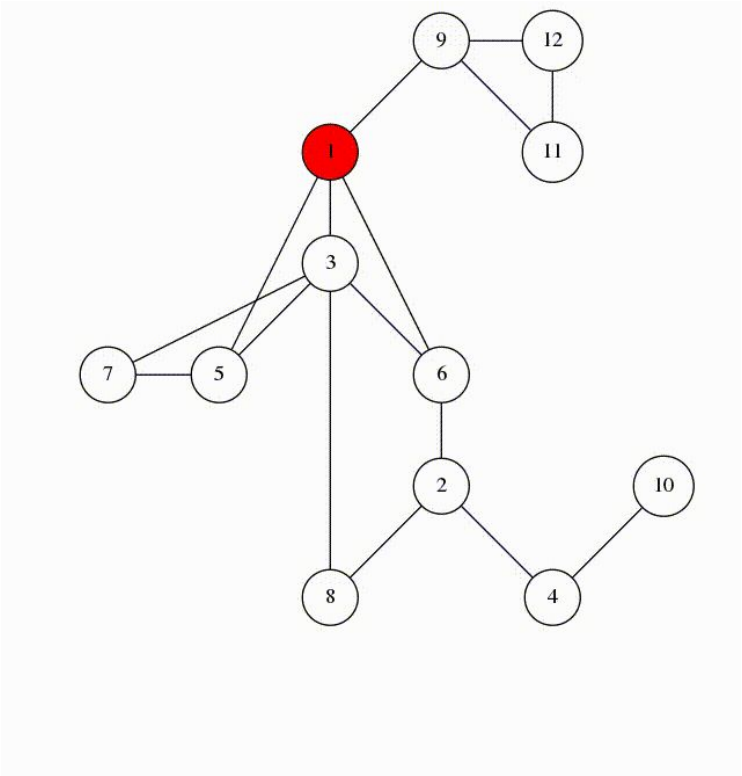
- **grafo no dirigido o no dirigido**

- Este algoritmo nos ayuda a recorrer un grafo de forma ordenada. Su funcionamiento es similar al backtracking. recorre un camino hasta el fondo y luego sigue explorando otros caminos.

Ejemplo gráfico



Ejemplo gráfico



pseudo-codigo DFS

```
dfs(nodo u):
        if not u->visitado:
                u->visitado = True
                for v in u->vecinos:
                        dfs(v)
```

DFS implementado de forma básica. solo recorre los nodos una vez pero no hace nada más.

variaciones DFS

```
DFS(grafo G)
    PARA CADA vértice u ∈ V[G] HACER
            estado[u] ← NO_VISITADO
            padre[u] ← NULO
    tiempo ← 0
    PARA CADA vértice u ∈ V[G] HACER
            SI estado[u] = NO_VISITADO ENTONCES
                    DFS_Visitar(u,tiempo)
```

```
DFS_Visitar(nodo u, int tiempo)
    estado[u] ← VISITADO
    tiempo ← tiempo + 1
    d[u] ← tiempo
    PARA CADA v ∈ Vecinos[u] HACER
            SI estado[v] = NO_VISITADO ENTONCES
                    padre[v] ← u
                    DFS_Visitar(v,tiempo)
    estado[u] ← TERMINADO
    tiempo ← tiempo + 1
    f[u] ← tiempo
```

En este caso el algoritmo va guardando caminos de un nodo a otro guardando padres (permite buscar caminos de vuelta) y también guarda el tiempo que nos demoramos en llegar a cierto nodo

Ejercicio

Escriba un algoritmo que determine si es posible llegar del vértice v al u en un grafo dirigido. Indique el camino encontrado.

¿Cómo se puede relacionar con backtracking?

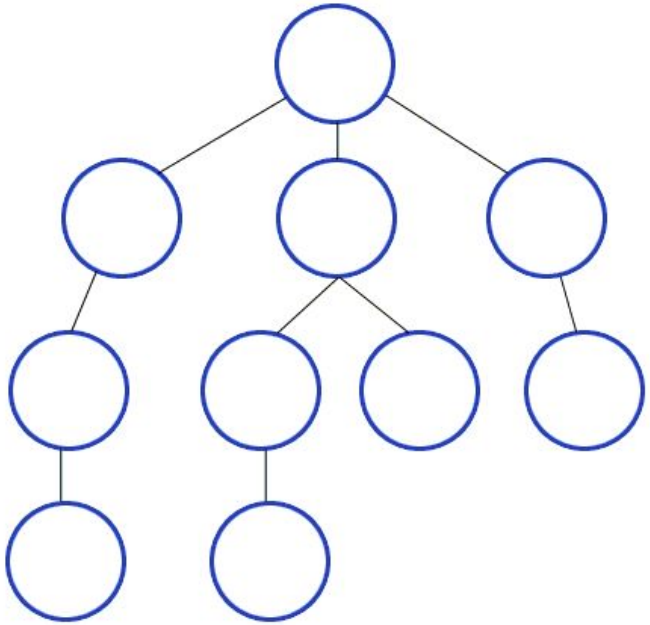
BFS

- **Breadth First Search o Búsqueda en anchura.**

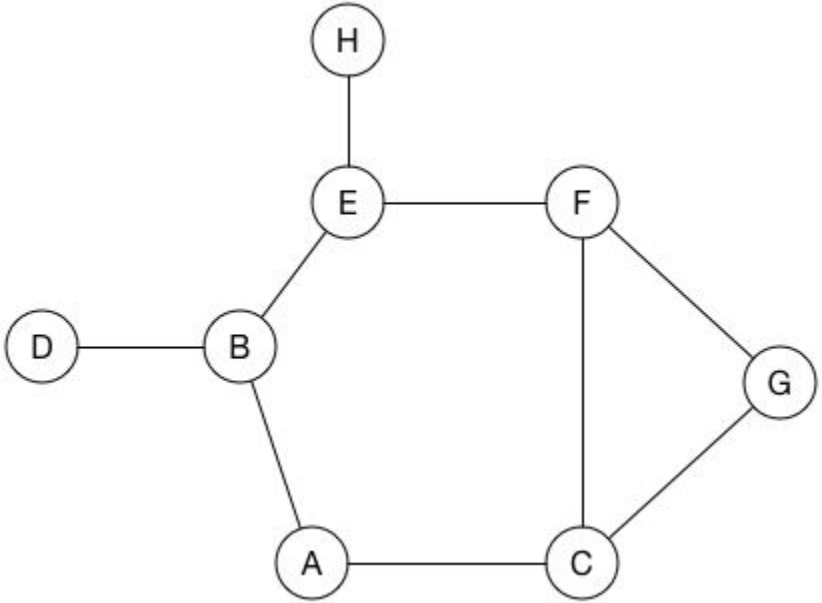
- Este también es un algoritmo de búsqueda sólo que revisa todos los vecinos antes de avanzar en profundidad.

- puede adaptarse para grafos dirigidos y no dirigidos

BFS gráficamente



BFS gráficamente

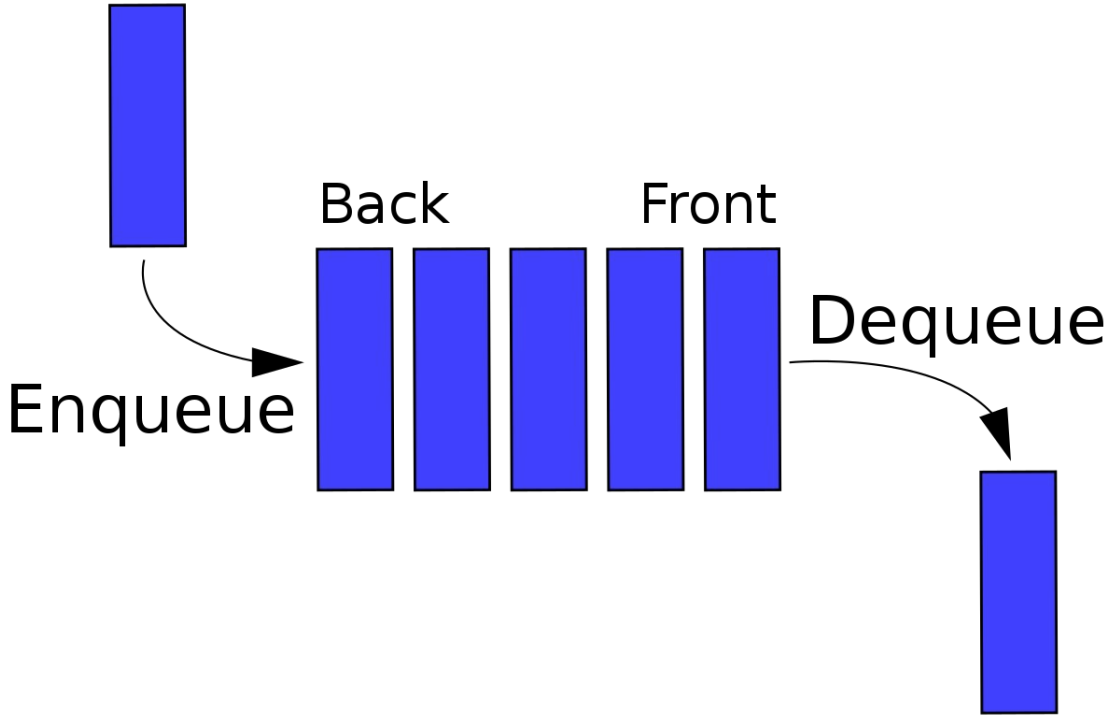


BFS pseudocódigo

```
bfs(nodo u):
    Queue = [u]
    while Queue non-empty:
        v = Queue.deque() // extrae el último elemento
        if not v->visitado:
            v->visited = True
            for w in v->vecinos:
                Queue.enque(w) // pone el elemento al principio
```

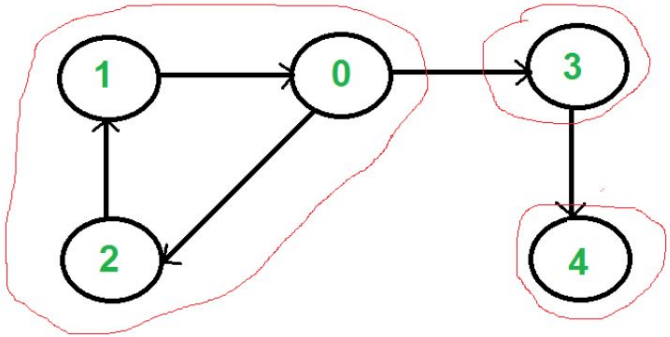
BFS implementado de forma básica. solo recorre los nodos una vez pero no hace nada más.

BFS implementado para un queue (FIFO)



SCC

- **Strongly connected components**
- Componentes fuertemente conexas
- Se usan en un grafo dirigido
- Una componente fuertemente conexa de un grafo es un conjunto de vértices tales que de cada vértice de dicho conjunto es posible llegar a otro vértice de dicho conjunto (pasando por vértices del conjunto).



https://www.geeksforgeeks.org/strongly-connected-components/

¿Cómo encontramos estos componentes?

- **Algoritmo de kosaraju**

1. para cada vértice u del grafo marcarlo como no-visitado y definir L lista vacía
2. para cada vértice u del grafo aplicar subrutina visitar(u)

Visitar(u):
    si u está no-visitado:
        marcar u como visitado
        para cada hijo v de u aplicar subrutina visitar(u)
        poner al final de la Lista L el vértice u.

3. **Invertir la dirección de todas las aristas del grafo**
4. Recorrer la lista L de adelante hacia atrás y para cada nodo u ejecutar asignar(u,u)

¿Cómo encontramos estos componentes?

- **Algoritmo de kosaraju**

asignar(u, root):
    si u no ha sido asignado a un componente asignar u al componente cuya root es root
    para cada hijo v de u en este nuevo grafo aplicamos asignar(v, root)
    si u ya se había asignado no hacemos nada

https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm

Ejemplo de Kosaraju

https://www.programiz.com/dsa/strongly-connected-components

Algoritmo dentro de Kosaraju

¿Qué algoritmo de los ya vistos se ocupa dentro de kosaraju?

Kosaraju utiliza DFS 2 veces. una vez dentro de visitar() y otra dentro de asignar()

TOPOSORT

- **Topological sort**
- Es una forma de ordenar un grafo dirigido. Este orden cumple que si un vértice **v** tiene un arco que apunta a **u** entonces, entonces **v** aparece antes que **u**.
- Para aplicar este ordenamiento no podemos tener componentes fuertemente conexas

[...., v,..., u,...]

TOPOSORT

¿Qué pasa en este caso?

¿[...., v,..., u,...] o [...., u,..., v,...]?

NO se puede hacer para componentes fuertemente conectada

TOPOSORT

```
Toposort():
        Stack = [ ]
        // N es número de nodos
        Visitado = [False] * N
        // recorremos los nodos en un orden específico
        for i in range(0,N):
                if not Visitado[i]:
                        v = nodos[i]
                        recursive-topo(v, Stack, Visitado)

        imprimir Stack en orden inverso
```

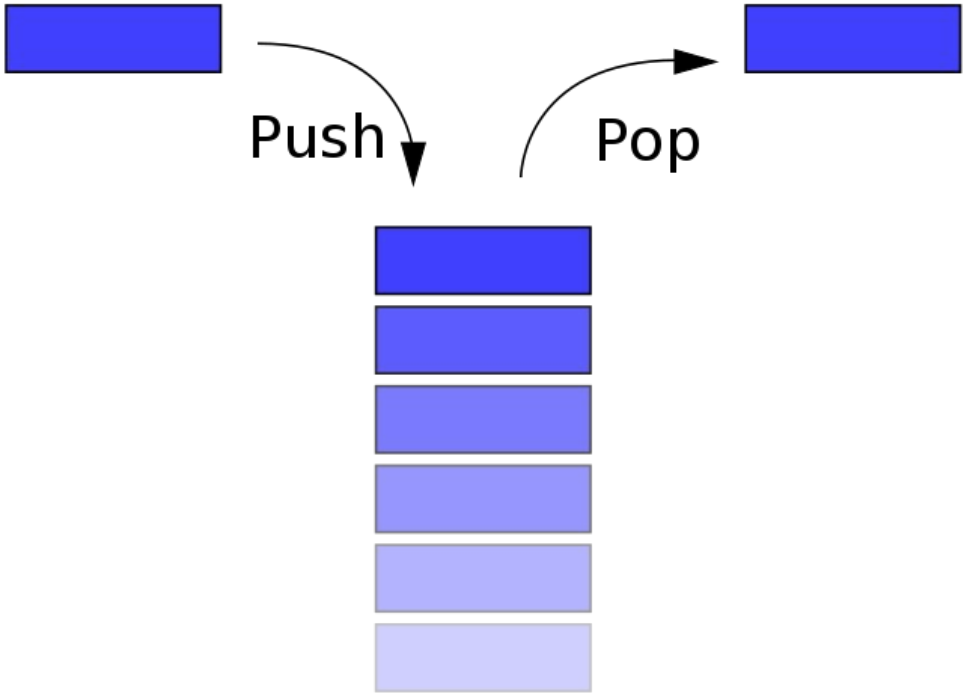
TOPOSORT

```
recursive-topo(v, Stack, Visitado):
        index = v->index
        Visitado[index] = True

        for u in v->vecinos:
                u_index = u->index

                if not Visitado[u_index]:
                        Visitado[u_index] = True
                        recursive-topo(u, Stack, Visitado)
        stack.push(v)
```

TOPOSORT



Implemente DFS iterativo