Apunte de Sistemas Operativos FIUBA

lcondoriz

May 2023

Índice

1. Introducción

Resumen hecho por lcondoriz.

2. Lista Enlazada

Una lista enlazada es una estructura de datos que consiste en una secuencia de nodos, en los cuales cada nodo tiene un puntero al siguiente nodo de la lista. La lista enlazada es una estructura de datos dinámica, ya que el tamaño de la lista puede crecer y decrecer durante la ejecución del programa.

Los requisitos que debe cumplir una lista enlazada son:

- Cada nodo debe contener al menos un dato y un puntero al siguiente nodo (o al anterior y al siguiente, en el caso de las listas doblemente enlazadas).
- El primer nodo debe tener un puntero nulo en su parte anterior (o en ambas partes, si se trata de una lista circular).
- El último nodo debe tener un puntero nulo en su parte posterior (o apuntar al primer nodo, si se trata de una lista circular).
- Los nodos deben estar conectados entre sí mediante los punteros, sin dejar nodos sueltos o desconectados.

Tipos de listas enlazadas:

1. **Lista enlazada simple:** Cada nodo tiene un puntero al siguiente nodo de la lista.
2. **Lista enlazada doble:** Cada nodo tiene un puntero al nodo anterior y al siguiente.
3. **Lista enlazada ligada circular:** El último nodo apunta al primero, y el primero al último.

2.1. Operaciones

- **Insertar:** Inserta un elemento en la lista.
- **Eliminar:** Elimina un elemento de la lista.
- **Buscar:** Busca un elemento en la lista.
- **Mostrar:** Muestra los elementos de la lista.

2.2. Caracteristicas

- **Acceso aleatorio:** No se puede acceder a un elemento de la lista de forma directa, sino que hay que recorrer la lista desde el principio hasta el elemento deseado.
- **Inserción y eliminación:** La inserción y eliminación de elementos en una lista enlazada es muy rápida, ya que no hay que desplazar elementos como en el caso de los arrays.
- **Memoria:** La memoria necesaria para una lista enlazada es mayor que la necesaria para un array, ya que cada nodo de la lista debe almacenar el puntero al siguiente nodo.

3. Pilas

Una pila es una estructura de datos que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo LIFO (del inglés Last In, First Out, «último en entrar, primero en salir»).

3.1. Operaciones

- **Push:** Inserta un elemento en la pila.
- **Pop:** Elimina un elemento de la pila.
- **Top:** Devuelve el elemento que está en la cima de la pila.
- **Empty:** Devuelve un valor booleano indicando si la pila está vacía o no.
- **Size:** Devuelve el tamaño de la pila.

4. Cola

Una cola es una estructura de datos que permite almacenar y recuperar datos, siendo el modo de acceso a sus elementos de tipo FIFO (del inglés First In, First Out, «primero en entrar, primero en salir»).

4.1. Operaciones

- **Push:** Inserta un elemento en la cola.
- **Pop:** Elimina un elemento de la cola.
- **Front:** Devuelve el elemento que está en la parte frontal de la cola.
- **Back:** Devuelve el elemento que está en la parte posterior de la cola.
- **Empty:** Devuelve un valor booleano indicando si la cola está vacía o no.
- **Size:** Devuelve el tamaño de la cola.

5. TDA conjunto

5.1. Árbol binario AB

Un árbol binario es una estructura de datos que relaciona información de manera jerárquica no lineal, con un nodo raíz y dos subárboles disjuntos, llamados izquierdo y derecho. Cada nodo puede tener, a lo sumo, dos hijos, y el grado de un árbol binario es el mayor grado de cualquiera de sus nodos.

Algunas de las características de un árbol binario son:

- La **raíz** es el primer nodo del árbol y no tiene padre.
- Los nodos hoja son los nodos que no tienen hijos.
- Los nodos internos son los nodos que tienen al menos un hijo.
- El subárbol izquierdo de un nodo es el árbol formado por su hijo izquierdo y todos sus descendientes.
- El subárbol derecho de un nodo es el árbol formado por su hijo derecho y todos sus descendientes.
- La altura de un nodo es la longitud del camino más largo desde el nodo hasta una hoja.
- La profundidad de un nodo es la longitud del camino desde la raíz hasta el nodo.
- El nivel de un nodo es su profundidad más uno.
- La altura de un árbol es la altura de su raíz.
- El orden de un árbol binario es el número total de nodos que tiene.

Tipos de árboles binarios: Un árbol binario es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario cada nodo puede tener cero, uno o dos hijos (subárboles). Se conoce el nodo de la izquierda como hijo izquierdo y el nodo de la derecha como hijo derecho.

- **Árbol binario de búsqueda:** Es un árbol binario que cumple con la propiedad de que para cada nodo, el valor de todos los nodos del subárbol izquierdo es menor o igual al valor del nodo y el valor de todos los nodos del subárbol derecho es mayor o igual al valor del nodo.

La diferencia entre un árbol binario y uno de búsqueda es que el primero no tiene ningún orden específico para organizar los elementos, mientras que el segundo sí lo tiene. Un árbol binario puede tener cualquier forma y distribución de los nodos, siempre que cada uno tenga como máximo dos hijos. Un árbol binario de búsqueda tiene una forma y distribución determinadas por la relación de orden entre los elementos.

5.2. Árbol binario de búsqueda ABB

Un árbol binario de búsqueda también llamado BST (acrónimo del inglés Binary Search Tree) es un tipo particular de árbol binario que presenta una estructura de datos en forma de árbol.

5.2.1. Propiedad de ABB:

Un ABB es un AB donde (esto es lo que lo diferencia de un AB común):

- El hijo izquierdo, y todos sus hijos, son menores que la raíz.
- El hijo derecho, y todos sus hijos, son mayores que la raíz.

5.2.2. Operaciones

- **Insertar:** Inserta un elemento en el árbol.
- **Eliminar:** Elimina un elemento del árbol.
- **Buscar:** Busca un elemento en el árbol.
- **Recorridos:** Recorre el árbol en distintos órdenes.

5.2.3. Insertar

Los ABB crecen "de arriba hacia abajo", es decir que se genera un nodo cuando se encuentra nulo el puntero apropiado.

5.2.4. Eliminar o Borrado

Casos Posibles [3] :

1. **Nodo hoja:** Se elimina el nodo y se pone a nulo el puntero del padre.
2. **Nodo con un hijo:** Se elimina el nodo y se pone a nulo el puntero del padre. El hijo pasa a ser hijo del padre del nodo eliminado.
3. **Nodo con dos hijos:**

Caso 3: Eliminar nodo con dos hijos.

1. Localizar el nodo predecesor o sucesor del nodo a eliminar.
   - predecesor es "el mayor de los menores"
   - sucesor es "el menor de los mayores"
   - Para la implemtacion es igual de eficiente usar uno u otro.
2. El valor del predecesor (o sucesor) se copia en el nodo a eliminar.
3. Eliminar el nodo del predecesor (o sucesor).

5.2.5. Busqueda

Debemos comenzar por el nodo raíz e ir descendiendo a izquierda o derecha, ya sea que el valor que estemos buscando sea menor o mayor que el dato del nodo que estemos comparando

5.2.6. Recorridos

Los recorridos se clasifican en dos categorías. Video YouTube [4]:

- Profundidad: En los recorridos en profundidad se procesa cada nodo, su subárbol izquierdo y el derecho. Se usa una Pila.
  - Preorden. $\Theta(n)$
  - Postorder $\Theta(n)$
  - Inorder $\Theta(n)$
- Anchura o por niveles $\Theta(n)$: En los recorridos en anchura se procesa cada nodo por niveles. Se usa una Cola.

5.3. Arboles balanceados

Tipos de árboles balanceados:

- ABB balanceado por su altura.
- ABB balanceado por el peso.

5.3.1. ABB balanceado por su altura

Un árbol binario está balanceado por su altura con diferencia permitida d sii para todo nodo x del árbol se verifica:

$$|h_{izq}(x) - h_{der}(x)| \leq d \tag{1}$$

5.3.2. ABB balanceado por el peso

Un árbol binario está balanceado por su peso con diferencia permitida d sii para todo nodo x del árbol se verifica:

$$|peso_{izq}(x) - peso_{der}(x)| \leq d \tag{2}$$

5.4. Árboles AVL

Los árboles AVL se balancean por altura. Se trata de un tipo de árbol binario de búsqueda que cumple la propiedad de que para cada nodo, la diferencia entre la altura de su subárbol izquierdo y el de su subárbol derecho es como máximo 1.

Se agrega un atributo al nodo que es el factor de balanceo (FB). Este factor tiene tres valores permitidos: 0, 1 o -1, en cualquier otro caso se necesita rebalancear.

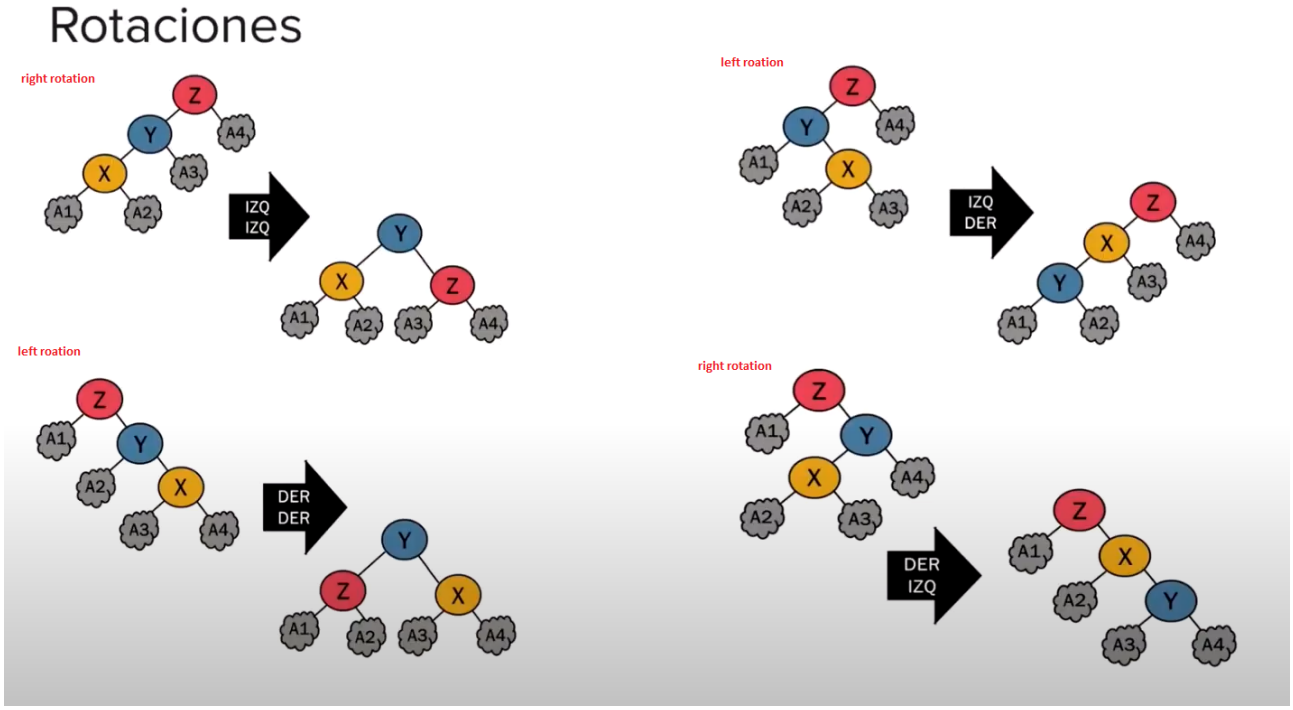

Figura 1: Rotaciones Árbol AVL

5.5. Árboles multivía o de m vías

Estas estructuras mejoran la eficiencia del almacenamiento habilitando la posibilidad de tener varios datos o claves en el mismo nodo.

Las claves siempre estarán ordenadas de forma creciente y de manera contigua, es decir que dos claves no puede haber nunca espacio no utilizado. No hay "posiciones vacías" entre dos claves.

Un árbol de m vías está formado por nodos que pueden contener hasta m-1 claves $(k_0, k_1, k_2, ..., k_{m-2})$.

Las condiciones que verifica esta estructura son:

1. Cada nodo tiene m vías (eventualmente pueden estar nulas) y m-1 lugares para almacenar claves.
2. Las claves (o datos) se almacenan de forma creciente y de manera contigua.
3. Las claves de los subárboles que están a la izquierda de la iésima clave son menores que la iesima clave.
4. Las claves de los subárboles que están a la derecha de la iésima clave son mayores que la iesima clave.
5. No esta balanceado.

5.6. Árbol B

Se trata de un árbol de m vías balanceado. Las primaras cuatro condiciones representan un árbol de m-vias.Video YouTube [5]

Las condiciones que verifica esta estructura son:

1. Cada nodo tiene m vías (eventualmente pueden estar nulas) y m-1 lugares para almacenar claves.
2. Las claves (o datos) se almacenan de forma creciente y de manera contigua.
3. Las claves de los subárboles que están a la izquierda de la iésima clave son menores que la iesima clave.
4. Las claves de los subárboles que están a la derecha de la iésima clave son mayores que la iesima clave.
5. Esta balanceado.
6. Todos los nodos excepto la raíz están completos con claves al menos hasta la mitad.
7. La raíz, o bien es hoja, o bien tiene al menos dos hijos.
8. Si un nodo tiene h claves almacenadas, entonces tiene $h + 1$ hijos (salvo la raíz y las hojas).
9. Todas las hojas están en el mismo nivel.

5.6.1. Borrado

Borrado en un árbol B:

- En hojas: normalmente.
- Nodos internos: buscar reemplazante (similar ABB).
- Puede haber underflow (nodo con menos claves de las permitidas).

Borrado con Underflow:

- **Prestar:** Si un hermano tiene más claves de las permitidas, se le puede pedir prestado.
- **Fusionar:** Si un hermano tiene la mínima cantidad de claves, se fusionan los dos nodos.

5.7. Estructura B+

Cuando es necesario acceder de manera secuencial a los datos almacenados en una estructura de arbol B, puede adicionarse una lista ligada con esos datos, la cual es accessible desde los nodos hoja.

Las condiciones que verifica esta estructura son:

- Las mismas condiciones que un árbol B.
- Todos los datos se almacenan únicamente en los nodos hoja.
- Los nodos hoja se encuentran unidos entre sí como una lista enlazada para permitir principalmente recuperación en rango mediante búsqueda secuencial.

6. Cola de Prioridad

El orden de salida no es el de entrada, sino que está dado por una prioridad. Video YouTube [6] y Video YouTube Ejercicios [7].

Heaps: Propiedad de heap (de máximos) Un árbol binario AB tiene propiedad de heap si la reíz es más grande (o igual) que sus dos hijos, y estos también son heaps.

Heaps: Representación de árbol izquierdista Un árbol izquierdista es aquel que tiene todos los niveles completos, salvo a lo sumo el último, que debe completarse se izquierda a derecha.

Heaps: Representación en arreglos Árbol izquierdista con propiedad de heap → En un arreglo como si fuera el recorrido por niveles del AB.

La implemtacion de una cola de prioridad se hace atraves de un un arreglo pero se usa la representan de árbol como interpresación.

6.1. Heap: Representación en arreglo - Posiciones

Si estoy en la posición i:

- Posiciones de sus hijos:
  - hijo izquierdo: $2i + 1$
  - hijo derecho: $2i + 2$
- Posición de su padre: $\frac{i-1}{2}$

6.2. Operaciones sobre Heaps: Upheap y Downheap

Encolar:

- Al encolar, guardamos el elemento al final.
- Aplicamos **Upheap:**
  - Si el elemento es más grande que su padre, entonces se intercambian, y se aplica Upheap al padre.
  - Sino, termina.

Desencolar:

- Llevamos el último al primer lugar (podemos intercambiar)
- Aplicamos **Downheap:**
  - Si alguno de los hijos es más grande que el elemento, entonces intercambiar, con el mas más grande de ellos.
  - Sino, termina.

Complejidad:

- Upheap: $O(log(n))$
- Downheap: $O(log(n))$
- Ver el máximo: $O(1)$
- Encolar: $O(log(n))$
- Desencolar: $O(log(n))$

6.3. Heapify: Cómo construir un heap desde un arreglo

Si tengo un arreglo de n elementos, cómo creo un heap con dichos elementos.

Complejidad: $O(n)$

6.4. Heapsort

El método requiere dos etapas. Video YouTube [8].

1. Construir un heap con los elementos del arreglo.
   - heap de maximos si queremos ordenar de menor a mayor (creciente).
   - heap de minimos si queremos ordenar de mayor a menor (decreciente).
2. Desencolar los elementos del heap. Se lleva a cabo el intercambio del primer elemento (posición inicial del heap) con el último, y la reconstrucción del heap en una zona que se disminuye en uno en cada etapa. Esta parte concluye cuando la zona del heap queda reducida a 1.

Complejidad: Se termina haciendo n veces Downheap (desde la raíz siempre) $O(n \cdot log(n))$

7. Grafos

Lista de reproducción YouTube [9].

7.1. Propiedades

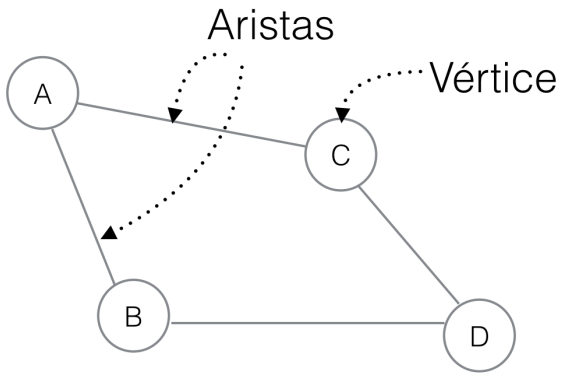

Figura 2: Grafos Aristas = Edge y Vértices o Nodos = Vertex

1. **Grafos orientados** (o dirigidos o digrafos) si las aristas (o arcos) que conectan sus vertices (también llamados nodos) están orientadas.
2. **Grafos no orientados** (o no dirigidos) si las aristas que conectan sus vertices no están orientadas.

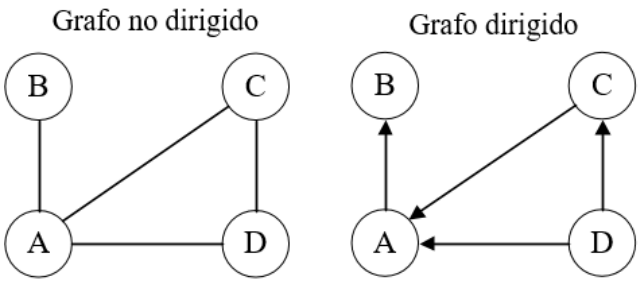

Figura 3: Grafos Dirigido y No Dirigido

3. **Ciclo:** camino que conteniendo vertices distintos, excepto el primero que coincide con el ultimo.
4. **Grafo no dirigido conexo:** Grafo no orientado es conexo si para todo vértice del grafo hay un camino que lo conecte con otro vertice cualquiera del grafo.

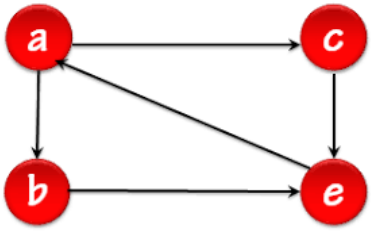

Figura 4: Grafos Ciclo: $a \to b \to e \to a$, longitud 3.

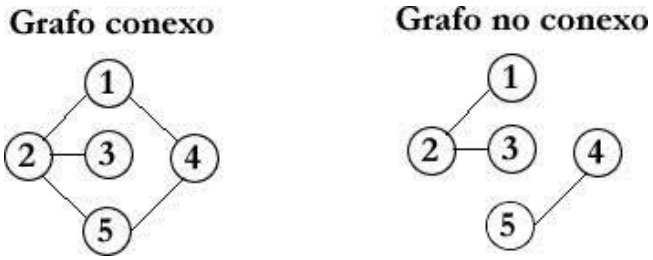

Figura 5: Grafos Conexo y No Conexo

5. **Grafo dirigido fuertemente conexo:** Grafo dirigido es fuertemente conexo sii entre cualquier par de vértices hay un camino que los une. Ver Figura 4.
6. **Árbol libre:** Grafo no dirigido conexo sin ciclos.

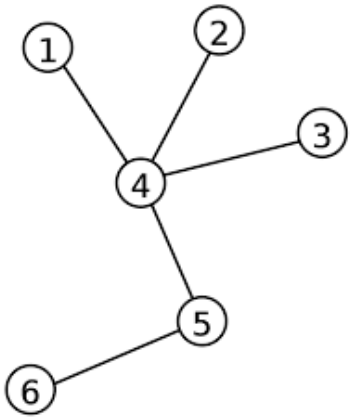

Figura 6: Grafos Árbol Libre

7.2. Estructuras para implementar grafos

7.2.1. Matriz de Adyacencia

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	1	0	1	0
C	1	1	0	0	0	0
D	1	0	0	0	1	1
E	0	1	0	1	0	0
F	0	0	0	1	0	0

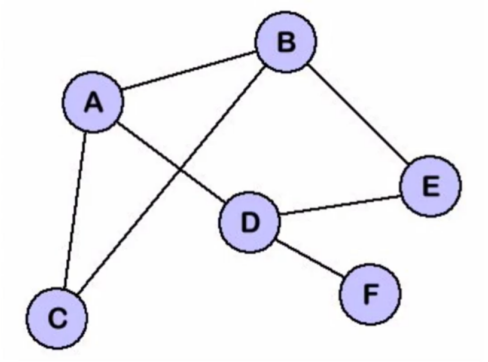

Figura 7: Grafo no dirigido y no pesado Matriz de Adyacencia

	A	B	C	D	E
A	0	3	0	0	0
B	0	0	6	1	5
C	0	0	0	6	0
D	0	0	0	0	7
E	0	0	0	0	0

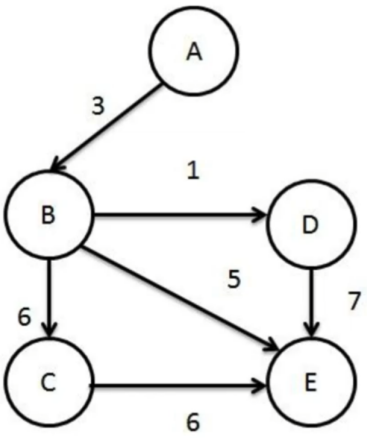

Figura 8: Grafo dirigido y pesado Matriz de Adyacencia

7.2.2. Matriz de Incidencia

	A	B	C	D	E	F
a1	1	1	0	0	0	0
a2	1	0	1	0	0	0
a3	0	1	1	0	0	0
a4	0	1	0	0	1	0
a5	0	0	0	1	1	0
a6	0	0	0	1	0	1
a7	1	0	0	1	0	0

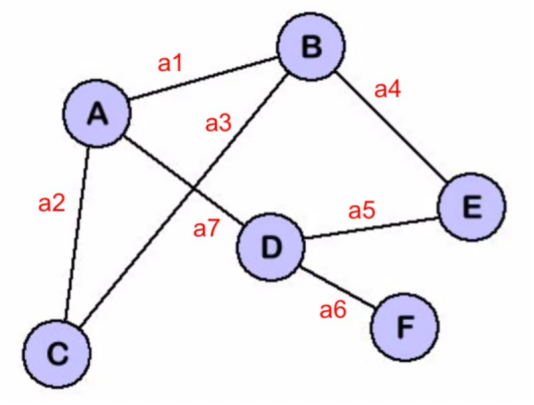

Figura 9: Grafo no dirigido y no pesado Matriz de Incidencia

	A	B	C	D	E
a1	-3	3	0	0	0
a2	0	-1	0	1	0
a3	0	-5	0	0	5
a4	0	0	-6	0	6
a5	0	-6	6	0	0
a6	0	0	0	-7	7

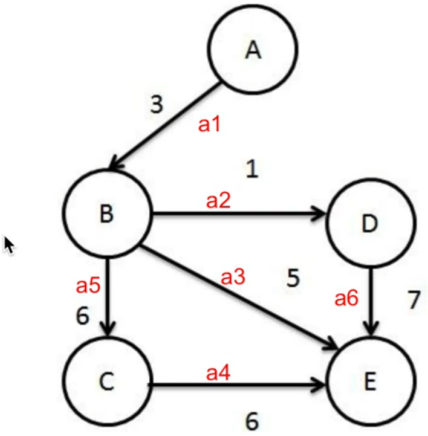

Figura 10: Grafo dirigido y pesado Matriz de Incidencia

7.2.3. Complejidad

	M. Incidencia	M. Adyacencia	Lista Adyacencia
Espacio:	$O(V \cdot E)$	$O(V^2)$	$O(V + E)$
Agregar un vértice:	$O(V \cdot E)$	$O(V^2)$	$O(1)$ o $O(V)$
Agregar una arista:	$O(V \cdot E)$	$O(1)$	$O(V)$
Si dos vértices son adyacentes:	$O(E)$	$O(1)$	$O(V)$
Obtener los abyacentes de un vértices:	$O(E)$	$O(V)$	$O(V)$

Cuadro 1: Coplejidad.

7.3. Recorridos Grafos Dirigidos

Recorridos Grafos Dirigidos

- Anchura: BFS (Breadth First Search)
- Profundidad: DFS (Depth First Search)

7.3.1. BFS: Recorrido en Anchura

Para grafos dirigidos y no dirigidos. Videos YouTube [1] y [2].

- Se recorre el grafo por niveles.
- Se utiliza una cola.
- Se marca cada vértice como visitado al encolarlo.
- Se encolan los vértices adyacentes no visitados.

Referencias

[1] Grafo BFS y DFS con pilas y colas. En: 3 (). URL: https://www.youtube.com/watch?v=P45bRlCTqHI.

[2] Grafo BFS y DFS con pilas y colas. En: 3 (). URL: https://www.youtube.com/watch?v=HelpfhBel_k.

[3] ABB - Eliminar nodos. En: 1 (). URL: https://slideplayer.es/slide/3617827/.

[4] ABB - Recorridos. En: 3 (). URL: https://www.youtube.com/watch?v=cmTEaAh5Gpg&list=PLQXt15yGlG-dyRb2ivfYLArcqYtkhuOvN&index=8.

[5] ABB - Recorridos. En: 3 (). URL: https://www.youtube.com/watch?v=R0TypAw0Ln0&t=2367s.

[6] ABB - Recorridos. En: 3 (). URL: https://www.youtube.com/watch?v=BhDLf-Vm0ag&list=PLLfC2vEod54LyohAVsfZ3h

[7] ABB - Recorridos. En: 3 (). URL: https://www.youtube.com/watch?v=FV9_SdXIegY&list=PLQXt15yGlG-dyRb2ivfYLArcqYtkhuOvN&index=19.

[8] Heapsort YouTube. En: 3 (). URL: https://www.youtube.com/watch?v=tBDrus4BpPQ.

[9] Heapsort YouTube. En: 3 (). URL: https://www.youtube.com/watch?v=P45bRlCTqHI.