

Apunte de Sistemas Operativos FIUBA

lcondoriz

May 2023

Índice

1. Introducción	3
2. Lista Enlazada	3
2.1. Operaciones	3
2.2. Características	3
3. Pilas	3
3.1. Operaciones	3
4. Cola	4
4.1. Operaciones	4
5. TDA conjunto	4
5.1. Árbol binario AB	4
5.2. Árbol binario de búsqueda ABB	5
5.2.1. Propiedad de ABB:	5
5.2.2. Operaciones	5
5.2.3. Insertar	5
5.2.4. Eliminar o Borrado	5
5.2.5. Búsqueda	6
5.2.6. Recorridos	6
5.3. Árboles balanceados	6
5.3.1. ABB balanceado por su altura	6
5.3.2. ABB balanceado por el peso	6
5.4. Árboles AVL	6
5.5. Árboles multivía o de m vías	7
5.6. Árbol B	7
5.6.1. Borrado	8
5.7. Estructura B+	8
6. Cola de Prioridad	8
6.1. Heap: Representación en arreglo - Posiciones	8
6.2. Operaciones sobre Heaps: Upheap y Downheap	9
6.3. Heapify: Cómo construir un heap desde un arreglo	9
6.4. Heapsort	9
7. Grafos	9
7.1. Propiedades	10
7.2. Estructuras para implementar grafos	11
7.2.1. Matriz de Adyacencia	11
7.2.2. Matriz de Incidencia	12
7.2.3. Complejidad	13
7.3. Recorridos Grafos	13
7.3.1. BFS: Recorrido en Anchura	14
7.3.2. DFS: Recorrido en Profundidad	14
7.3.3. Complejidad BFS y DFS	15
7.3.4. Aplicaciones	15
7.4. Problemas sobre caminos en un grafo	15
7.4.1. Árbol de expansión mínimo	16
7.4.2. Algoritmo de Dijkstra	16
7.4.3. Algoritmo de Floyd	16
7.4.4. Algoritmo de Warshal - Cerradura transitiva	16
7.5. Algunos problemas sobre Grafos no dirigidos	16

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. Características

- **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.

La única condición que debe cumplir un árbol binario es que cada nodo puede tener como máximo dos hijos. No hay ninguna restricción sobre el orden de los elementos, ni sobre la forma del árbol.

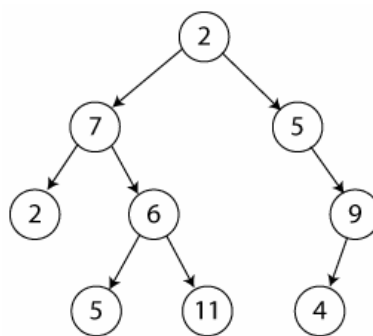


Figura 1: Árbol binario

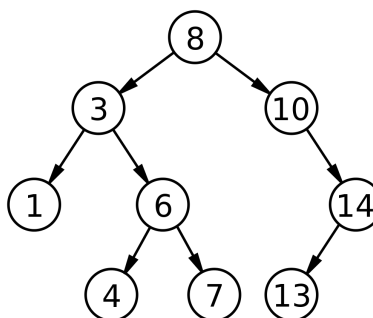


Figura 2: Árbol binario de búsqueda

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 [5] :

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 implementación 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. Búsqueda

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 [6]:

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

5.3. Árboles 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 si para todo nodo x del árbol se verifica:

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

5.3.2. ABB balanceado por el peso

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

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

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.

Rotaciones

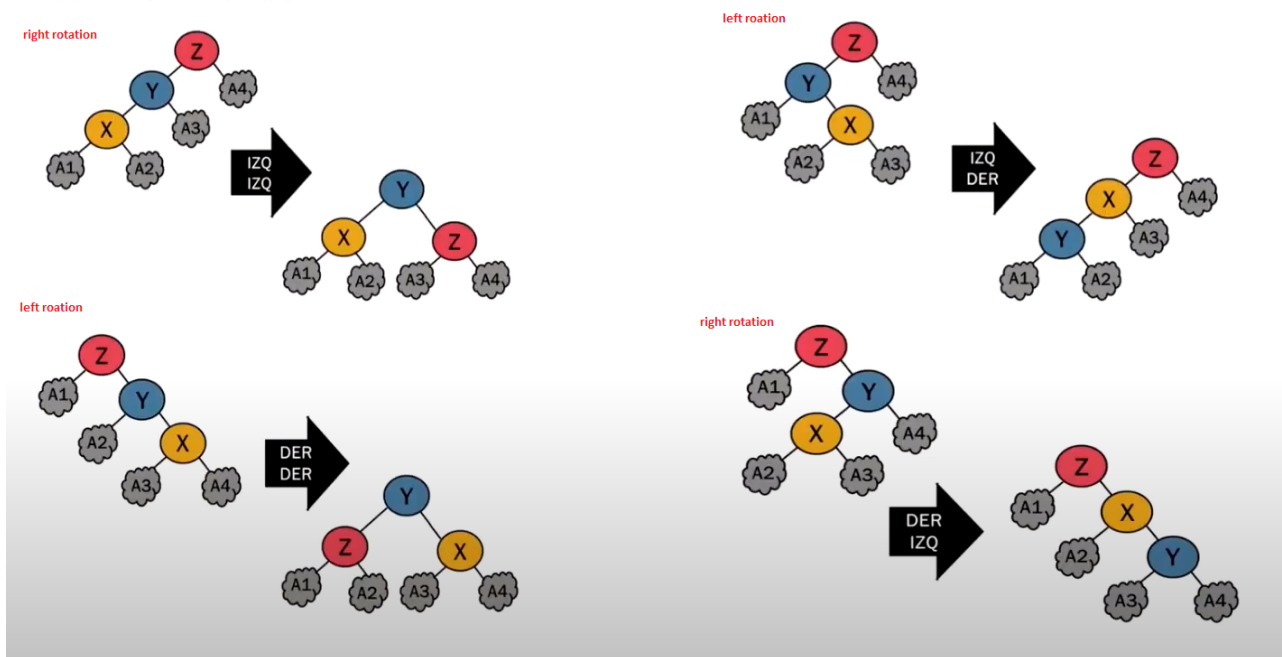


Figura 3: Rotaciones Árbol AVL

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 ($k_0, k_1, k_2, \dots, k_{m-2}$).

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 i ésima clave son menores que la i ésima clave.
4. Las claves de los subárboles que están a la derecha de la i ésima clave son mayores que la i ésima clave.
5. No está balanceado.

5.6. Árbol B

Se trata de un árbol de m vías balanceado. Las primeras cuatro condiciones representan un árbol de m-vías. Video YouTube [7]

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 i ésima clave son menores que la i ésima clave.
4. Las claves de los subárboles que están a la derecha de la i ésima clave son mayores que la i ésima clave.
5. Está 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 árbol B, puede adicionarse una lista ligada con esos datos, la cual es accesible 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 [8] y Video YouTube Ejercicios [9].

Heaps: Propiedad de heap (de máximos) Un árbol binario AB tiene propiedad de heap si la raíz 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 implementación de una cola de prioridad se hace a través de un arreglo pero se usa la representación de árbol como interpretación.

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 [\[13\]](#).

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 [\[14\]](#).

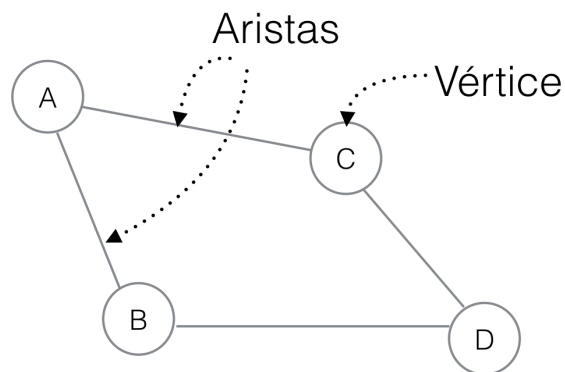


Figura 4: Aristas = Edge y Vértices o Nodos = Vertex

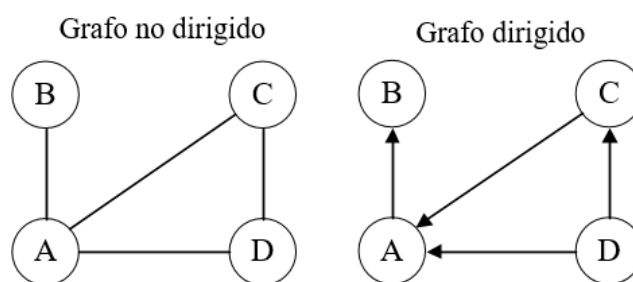


Figura 5: Grafos Dirigido y No Dirigido

7.1. Propiedades

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

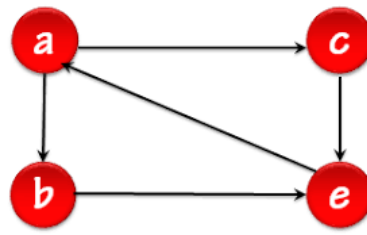


Figura 6: Grafos Ciclo: $a \rightarrow b \rightarrow e \rightarrow a$, longitud 3.

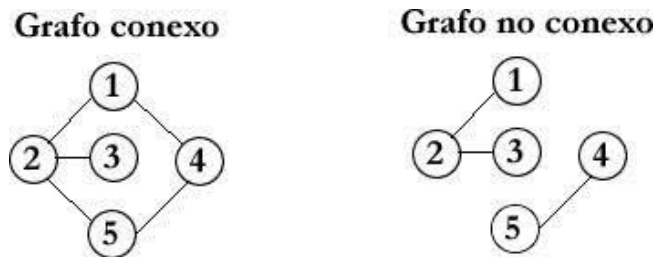


Figura 7: Grafos Conexos y No Conexos

7.2. Estructuras para implementar grafos

7.2.1. Matriz de Adyacencia

Propiedades de la matriz de adyacencia:

- **Grafo no dirigido:** La matriz de adyacencia es simétrica.

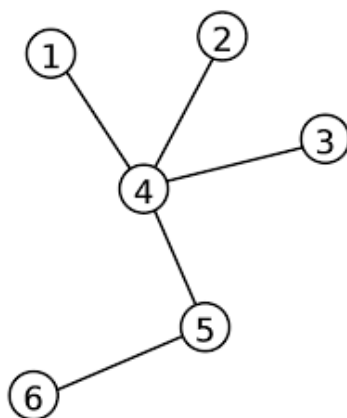


Figura 8: Grafos Árbol Libre

	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

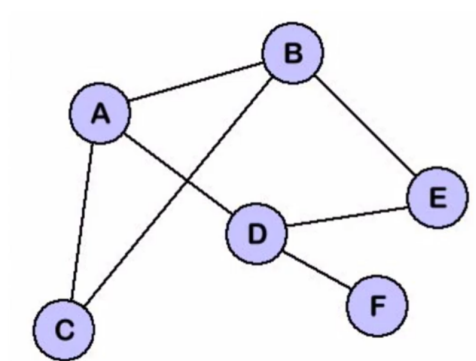


Figura 9: Grafo no dirigido y no pesado Matriz de Adyacencia

7.2.2. Matriz de Incidencia

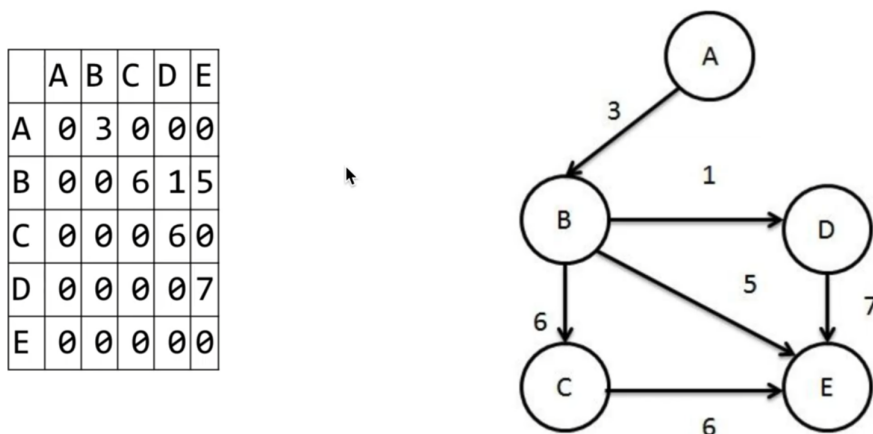


Figura 10: Grafo dirigido y pesado Matriz de Adyacencia

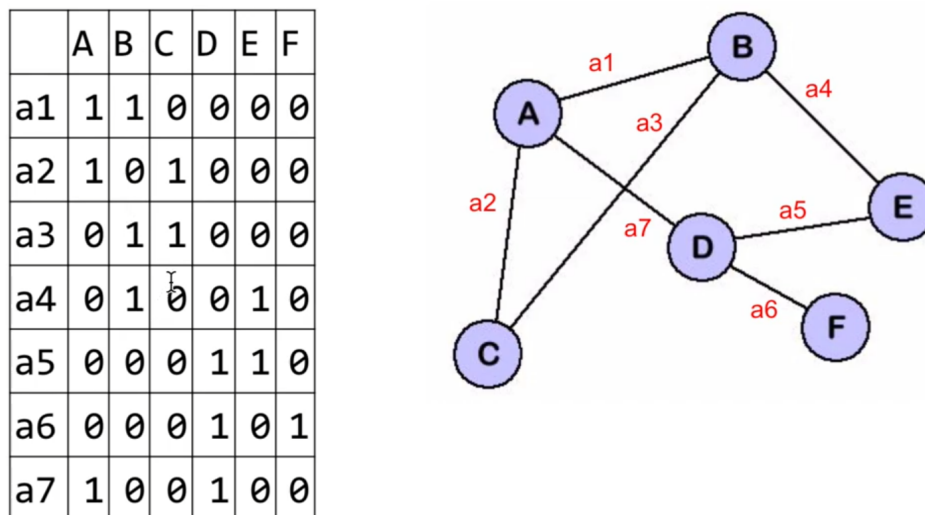


Figura 11: Grafo no dirigido y no pesado

7.2.3. Complejidad

7.3. Recorridos Grafos

Recorridos Grafos Dirigidos

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

BFS: Se lo puede implementar con una cola (es mas fácil verlo de esta manera).

DFS: Se lo puede implementar con recursividad (es mas fácil verlo de esta manera) o con una pila. Cuando se trabaja con grafos No conexos se puede implementar el algoritmo para que recorra todos los nodos no visitados.

Referencias:

- Video YouTube - Recorrido DFS Recursividad FIUBA [10].
- Video YouTube - Recorrido BFS y DFS con Pilas y Colas [1].
- Video YouTube - Recorrido BFS y DFS con Pilas y Colas [2].
- Página simula recorrido BFS y DFS tiene errores [4].

	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

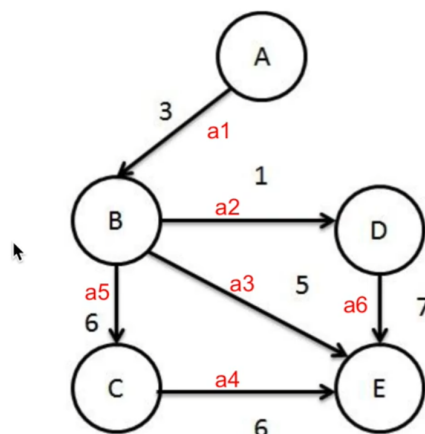


Figura 12: Grafo dirigido y pesado Matriz de Incidencia

	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 adyacentes de un vértice:	$O(E)$	$O(V)$	$O(V)$

Cuadro 1: Complejidad.

- Página simula recorrido BFS y DFS SIN errores [3].

7.3.1. BFS: Recorrido en Anchura

Para grafos dirigidos y no dirigidos.

Procedimiento:

- Seleccionar un vértice inicial.
- Marcarlo como visitado.
- Encolarlo.
- Mientras la **cola** (FIFO) no esté vacía :
 - Desencolar vértice.
 - Mostrarlo.
 - Marcar como visitados.
 - Los vertices adyacentes no visitado.
 - Encolar.

7.3.2. DFS: Recorrido en Profundidad

Para grafos dirigidos y no dirigidos.

Procedimiento:

- Seleccionar un vértice inicial.
- Marcarlo como visitado.
- Apilarlo.
- Mientras la **pila** (LIFO) no esté vacía :

- Desapilar vértice.
- Mostrarlo.
- Recorrer todos los vértices adyacentes del vértice desapilado
 - Si el vértice adyacente no ha sido visitado, marcarlo como visitado y apilarlo.
 - Si el vértice adyacente ya ha sido visitado, continúa con el siguiente vértice adyacente.

7.3.3. Complejidad BFS y DFS

	Matriz Adyacencia	Matriz Incidencia
Anchura BFS	$O(V^2)$	$O(V + E)$
Profundidad DFS	$O(V^2)$	$O(V + E)$

Cuadro 2: Complejidad.

7.3.4. Aplicaciones

DFS: Recorrido Profundidad

1. **Test de Aciclicidad (Ciclos):** Se recorre el grafo teniendo en cuenta los recorridos parciales. El coste del algoritmo es el mismo que la Tabla 2.
2. **Puntos de Articulación:** Un punto de articulación (o vértice de corte) de un grafo **no dirigido** y **conexo** es un vértice que verifica que al ser eliminado del grafo éste deja de ser conexo.
3. **Obtención de las componentes fuertemente conexas en un grafo dirigido:** Una componente fuertemente conexa es un subgrafo en el que para cada par de vértices existe un camino de uno a otro.

BFS: Recorrido Anchura

1. **Camino mínimo:** Si el grafo es no pesado, el camino mínimo entre dos vértices es el camino que tiene menos aristas.
2. **Árbol de expansión mínimo:** Si el grafo es pesado, el árbol de expansión mínimo es el subgrafo que tiene todos los vértices del grafo original y la suma de los pesos de sus aristas es la mínima posible.

Puntos de Articulación: Propiedades:

- Si r es raíz es el árbol DFS y tiene más de un hijo en el árbol, entonces r es un punto de articulación.
- $\text{bajo}(u)$ = mínimo número asignado en el recorrido en profundidad.
- Para todo vértice u que no sea raíz, es punto de articulación si y sólo si tiene al menos un hijo x tal que $\text{bajo}(x) \geq \text{numero asignado al vértice } u \text{ en el recorrido en profundidad}$.

Procedimiento:

1. Recorrer el grafo en profundidad y numerar los vértices en el orden en que se los visita.
2. Asignar el valor $\text{bajo}(u)$ a cada vértice u comenzando desde el último hasta el primero de los visitados.

7.4. Problemas sobre caminos en un grafo

- **Camino más corto entre un vértice y todos los demás:** Consiste en determinar el costo del camino más corto desde el vértice considerado origen a todos los otros vértices. Para grafos dirigidos y no dirigidos con aristas ponderadas no negativas.
 - **Estrategia greedy:**
- **Camino más corto entre todos los pares de vértices:** Si el grafo es no pesado, el camino más corto entre todos los pares de vértices es el camino que tiene menos aristas.
 - **Programación Dinámica:**

7.4.1. Árbol de expansión mínimo

- **Árbol de expansión mínimo:** Si el grafo es pesado, el árbol de expansión mínimo es el subgrafo que tiene todos los vértices del grafo original y la suma de los pesos de sus aristas es la mínima posible.

7.4.2. Algoritmo de Dijkstra

- **Algoritmo de Dijkstra:** El algoritmo de Dijkstra es un algoritmo para la determinación del camino más corto desde un vértice origen, hacia el resto de los vértices en un grafo dirigido no dirigidos que tiene pesos (NO NEGATIVOS) en cada arista.

7.4.3. Algoritmo de Floyd

- **Algoritmo de Floyd:** El algoritmo de Floyd es un algoritmo para la determinación del camino más corto entre todos los pares de vértices en un grafo DIRIGIDOS que tiene pesos (NO NEGATIVOS) en cada arista.

Ver video YouTube [11].

Se usa una matriz de adyacencia que contiene ceros y unos donde el cero significa que no hay camino y el uno significa que hay camino.

7.4.4. Algoritmo de Warshal - Cerradura transitiva

- **Algoritmo de Warshall:** El algoritmo de Warshall permite determinar qué pares de vertices están enlazados entre si por algún camino, sin importar la longitud (peso en la arista) de éste. Es para grafos dirigidos y no ponderados.

Ver video YouTube [12].

7.5. Algunos problemas sobre Grafos no dirigidos

Un **árbol libre** es un grafo no dirigido conexo sin ciclos.

Se verifica que

- En todo árbol libre con N vértices ($N > 1$), el árbol contiene $N - 1$ aristas.
- Si se agrega una arista a un árbol libre, aparece un ciclo.
- Si u y v son dos vertices distintos de un árbol, entonces hay un solo camino que los une.

Algoritmos que permiten obtener el árbol de expansión de coste mínimo

- **Algoritmo de Prim:** El algoritmo de Prim es un algoritmo para la determinación del árbol de expansión mínimo de un grafo no dirigido, conex y con aristas no negativas.
- **Algoritmo de Kruskal:** El algoritmo de Kruskal es un algoritmo para la determinación del árbol de expansión mínimo de un grafo conexo y no dirigido.

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. Muy buena explicación. En: 3 (). URL: https://www.youtube.com/watch?v=HelpfhBel_k.
- [3] Grafo BFS y DFS Simulación sin errores. En: 3 (). URL: <https://graphonline.ru/es/>.
- [4] Grafo BFS y DFS Simulación tiene errores. En: 3 (). URL: <https://visualgo.net/en/dfsdfs>.
- [5] ABB - Eliminar nodos. En: 1 (). URL: <https://slideplayer.es/slide/3617827/>.
- [6] ABB - Recorridos. En: 3 (). URL: <https://www.youtube.com/watch?v=cmTEaAh5Gpg&list=PLQXt15yGIG-dyRb2ivfYLArcqYtkhuOvN&index=8>.
- [7] ABB - Recorridos. En: 3 (). URL: <https://www.youtube.com/watch?v=R0TypAw0Ln0&t=2367s>.

- [8] ABB - Recorridos. En: 3 (). URL: <https://www.youtube.com/watch?v=BhDLf-Vm0ag&list=PLLfC2vEod54LyohAVsfZ3byiPOzHF15gg>.
- [9] ABB - Recorridos. En: 3 (). URL: https://www.youtube.com/watch?v=FV9_SdXIegY&list=PLQXt15yGIG-dyRb2ivfYLArcqYtkhuOvN&index=19.
- [10] Recorrido DFS. Usando Recursividad. En: 3 (). URL: <https://www.youtube.com/watch?v=SB7jaVUtFI4&list=PLLfC2vEod54KjquSG5N1w9KkaGM-3JTUG&index=5>.
- [11] Recorrido DFS. Usando Recursividad. En: 3 (). URL: <https://www.youtube.com/watch?v=h-nmexY9gtA&list=PLQXt15yGIG-dyRb2ivfYLArcqYtkhuOvN&index=23>.
- [12] grafo algoritmo warshall. En: 3 (). URL: <https://www.youtube.com/watch?v=P-xsjp9E1lk&list=PLQXt15yGIG-dyRb2ivfYLArcqYtkhuOvN&index=24>.
- [13] Heapsort YouTube. En: 3 (). URL: <https://www.youtube.com/watch?v=tBDrus4BpPQ>.
- [14] Heapsort YouTube. En: 3 (). URL: <https://www.youtube.com/watch?v=P45bRlCTqHI>.