UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 4 - Unidad 2

Estructuras de Datos Recursivas

ESTRUCTURAS DE DATOS (271)

Clase N. 4. Unidad 2.

Clase 4: AGENDA

AGENDA

- **Temario:**
  - Revisión de conceptos: Arboles y Grafos.
  - Representación y estrategias de implementación de cada estructura, accesos y recorridos**.**

- **Ejemplos** en Lenguajes **Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

Clase 4: TEMARIO. Arboles

Arboles:

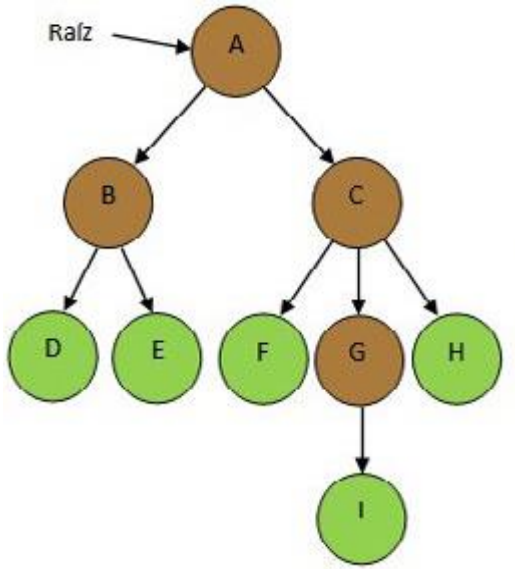
- Un **árbol** es una estructura ramificada donde cada elemento puede relacionarse con muchos elementos. Cuando el tamaño de la entrada de nuestro problema es muy grande es necesario recurrir a estructuras más complejas que nos otorgan en promedio un orden de ejecución **O(log n)**.
- Los **árboles** son **eficientes** para realizar operaciones de búsqueda: se almacena un campo clave –sobre el que se realizara la búsqueda– y la posición (número relativo de registro nrr), donde se encuentra el resto de la información en el archivo. ***Esto reduce el tamaño de la representación del árbol en memoria mejorando su rendimiento***.

Cap. X del libro Algoritmos y Estructuras de Datos en Python

Clase 4: TEMARIO. Arboles

Un **árbol** es una colección de datos jerárquica, en el que cada **nodo** tiene un padre a excepción del **nodo raíz**, que representa la jerarquía máxima sobre el resto de los elementos.

Arboles terminología:



Raíz: es el nodo que se encuentra en la parte superior del árbol. En este caso el nodo A.
Hijo: es un nodo que depende directamente de otro, el nodo esta conectados por una flecha que llega.
Padre: es aquel nodo del que depende al menos un nodo hijo, los hijos son aquellos nodos a los que apuntan las flechas que salen de este. Ej. A, B, C, G.
Hermanos: son el conjunto de nodos que tienen el mismo padre. Ej. grupos de hermanos “B-C”, “D-E”, “F-G-H”.

Clase 4: TEMARIO. Arboles

Hoja: son todos los nodos que no tienen hijos.

Grado de un nodo: es el número de subárboles de dicho nodo. Ej. el grado de A es dos y el de G es uno.

Rama: es la conexión entre dos nodos, también es llamado enlace.

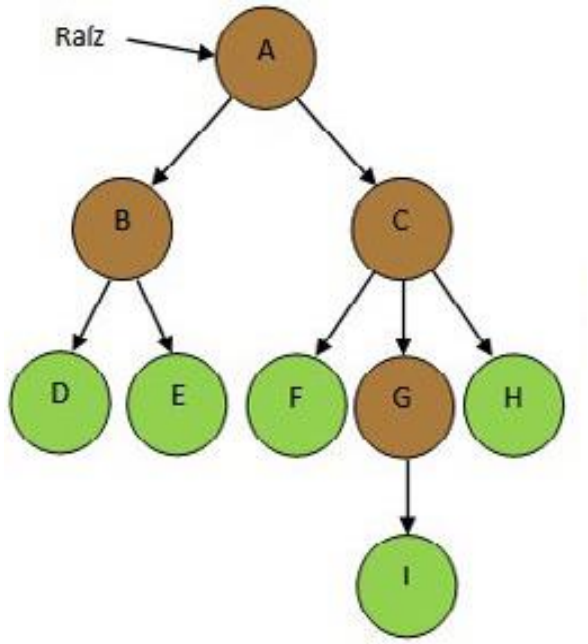
Grado de un nodo: es el número de subárboles de dicho nodo. Ej. el grado de A es dos y el de G es uno.

Grado de un árbol: es la cantidad máxima de hijos que tiene alguno de los nodos del árbol. Ej. el grado del árbol es tres.

Altura de un nodo: es el número de ramas del camino más largo entre ese nodo y una hoja. Ej. la altura de C es dos y la de B es uno.

Altura de un árbol: es la altura de su nodo raíz, Ej. la altura del árbol es tres.

Profundidad de un nodo: es el número de ramas desde la raíz del árbol hasta dicho nodo. Ej. la profundidad de I es tres y la de B es uno.



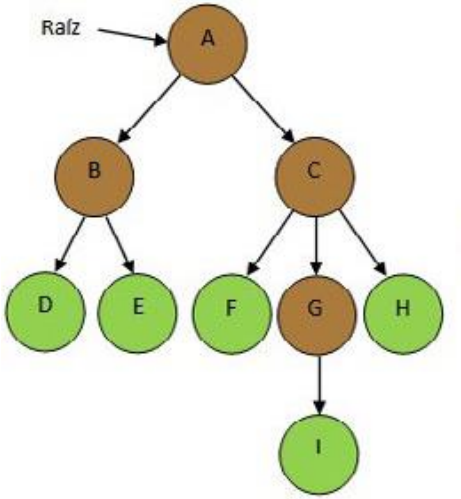
Clase 4: TEMARIO. Arboles

Descendientes: los descendientes de un nodo son todos aquellos nodos que pertenecen a los subárboles de los hijos de dicho nodo. Ej. los descendientes de B son D y E, mientras que los de C son F, G, H, I.

Ancestros: los ancestros de un nodo, son todos aquellos nodos en el camino desde la raíz a dicho nodo. Ej. los ancestros de E son A y B, mientras que los de I son A, C, G.

Nodo interno: son todos aquellos nodos que tienen al menos un hijo, a excepción de la raíz, es decir B, C, G.

Camino: es una secuencia de nodos desde un **nodoj** a un **nodok** –*distintos entre si*– de modo que exista una rama que conecte cada par de nodos –*siempre en sentido descendente*–,su longitud es el número de ramas que contiene. Por ejemplo el camino de C hasta I es “C-G-I” y su longitud es dos. Además, cada nodo del árbol a excepción del nodo raíz puede ser accedido desde este último mediante un camino único a través de una secuencia de ramas. Si existe un camino entre un **nodoj** y un **nodok**, entonces **nodoj** es un ancestro de **nodok** y **nodok** es un descendiente de **nodoj**.



Clase 4: TEMARIO. Arboles

Operaciones:

1. insertar_nodo(raíz, elemento). Agrega el elemento al árbol;

2. eliminar_nodo(raíz, clave). Elimina y devuelve del árbol si encuentra un elemento que coincida con la clave dada –el primero que encuentre–, devuelve None caso contrario.

3. reemplazar(raíz). Determina el nodo que reemplazará al que se va a eliminar, esta es una función interna que solo es utilizada por la función eliminar

4. arbol_vacio(raíz). Devuelve verdadero si el árbol no contiene elementos;

5. buscar(raíz, clave). Devuelve un puntero que apunta al nodo que contiene un elemento que coincida con la clave –el primero que encuentra–, si devuelve None significa que no se encontró la clave en el árbol;

6. preorden(raíz). Realiza un recorrido de orden previo del árbol mostrando la información de los elementos almacenados en el árbol;

7. inorden(raíz). Realiza un recorrido en orden del árbol mostrando la información de los elementos almacenados en el árbol;

8. postorden(raíz). Realiza un recorrido de orden posterior del árbol mostrando la información de los elementos almacenados en el árbol.

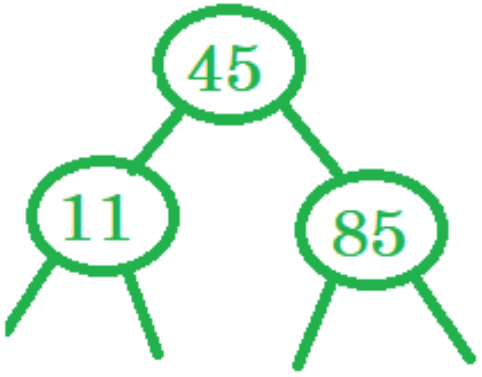
9. por_nivel(raíz). Realiza un recorrido del árbol por nivel mostrando la información de los elementos almacenados.

Clase 4: TEMARIO. Arboles


```
class nodo_arbol(object):
    #clase nodo arbol
    def __init__(self,info):
        #crea un nodo con la informacion cargada
        self.izquierda = None
        self.derecha =None
        self.informacion = info
def arbol_vacio(raiz):
    #retorna si el arbol esta vacio
    return raiz == None
def insertar_nodo(raiz,dato):
    #inserto el dato en el arbol
    if arbol_vacio(raiz):
        raiz = nodo_arbol(dato)
    elif(raiz.informacion <= dato):
        raiz.derecha = insertar_nodo(raiz.derecha,dato)
    else:
        raiz.izquierda = insertar_nodo(raiz.izquierda,dato)
    #una vez actualizado el dato se retorna la raiz
    return raiz

def imprimir_arbol(raiz):
    if arbol_vacio(raiz):
        print("-")
    else:
        print(raiz.informacion)
        imprimir_arbol(raiz.derecha)
        imprimir_arbol(raiz.izquierda)

arbol= insertar_nodo(None, 45)
arbol = insertar_nodo(arbol, 11)
arbol = insertar_nodo(arbol, 85)
imprimir_arbol(arbol)
```



Clase 4: TEMARIO. Grafos

Grafos:

Un **grafo** es una estructura ramificada, formada por una red de nodos conectados entre si.

Los **grafos** no tienen un nodo **raíz,** desde un nodo se puede acceder a muchos otros y un nodo puede ser accedido desde distintos nodos.

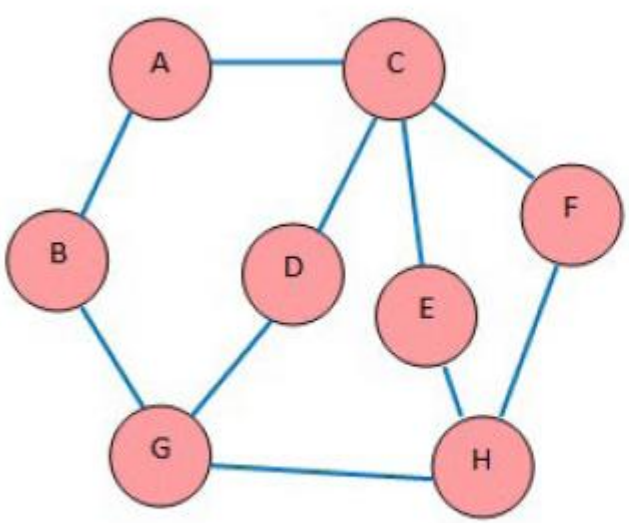
Existen dos tipos de grafos, dirigidos y no dirigidos

Podemos representar un grafo como un conjunto **G= (V, A)** que esta compuesto por un conjunto de **vertices** o nodos **V (v1, v2, v3,…, vn)** y un conjunto de **aristas** o arcos **A (a1, a2, a3,…,an).**

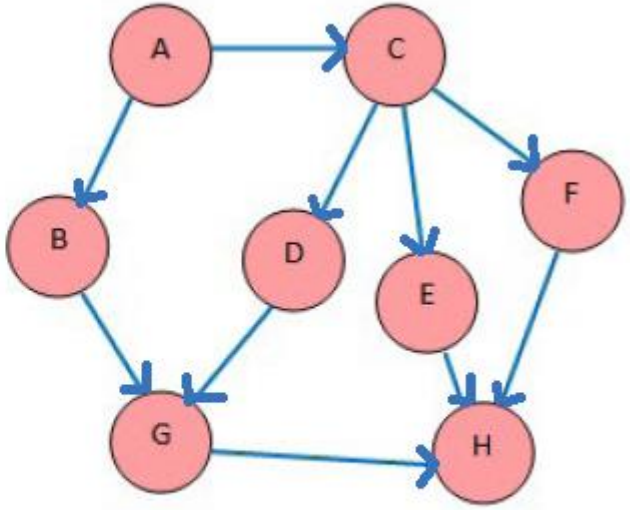
Una **arista** esta compuesta por un par ordenado **(a, b)**, *a* es el *vértice origen* y *b* el *vértice destino.*

Clase 4: TEMARIO. Grafos

Grafos:



Grafo no dirigido



Grafo dirigido

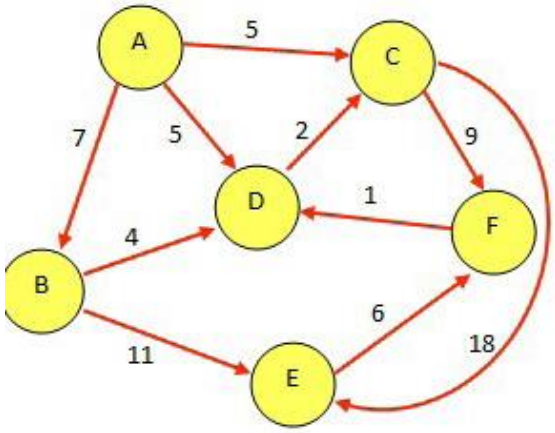
Grafos terminología:

Camino: en un grafo dirigido es un conjunto de vértices (v1, v2, v3) tal que existen los arcos del camino (v1-v2, v2-v3). Por ejemplo el camino de vértices *A*, *C*, *F*, *D*, esta compuesto de los arcos *AC*, *CF*, *FD*.

Longitud de un camino: es la cantidad de aristas del camino o la cantidad de vértices del camino menos uno. Siguiendo el ejemplo del camino anterior, la longitud es tres.

Etiqueta de una arista: es un valor que esta asociado a la relación entre dos vértices. Este suele ser un valor numérico que representa tiempo, distancia, cantidades que indican el valor de ir desde el vértice origen al destino, también se lo denomina peso. Por ejemplo la etiqueta de la arista *CF* es nueve.

Conexo: un grafo dirigido es conexo si desde cualquier vértice se puede acceder a cualquier otro, ya sea de manera directa o indirecta –es decir pasando por nodos intermedios–. Para el caso de la figura no es conexo dado que, por ejemplo, desde el vértice *C* no puedo acceder al vértice *B* de ninguna manera.



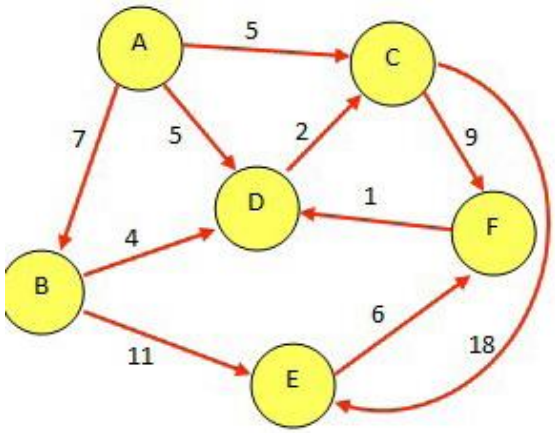
Clase 4: TEMARIO. Grafos

Grafos terminología:

Aciclico: un grafo dirigido es acíclico si no tiene ciclos, esto implica que para cada vértice del grafo no exista ningún camino que empiece en un vértice y termine en el mismo –un vértice podría tener un arco a si mismo y formar un ciclo–en el
ejemplo de la figura, el grafo no es aciclico dado que existe un ciclo entre *C-F-D*.

Camino hamiltoniano: es un camino que consiste en visitar todos los vértices de grafo pasando solo una vez por cada uno, si además el primer vértice es adyacente de el ultimo el camino también
es un ciclo hamiltoniano. En el ejemplo el camino *A-B-E-F-D-C* es un camino hamiltoniano.

Camino euleriano: es un camino que consiste en pasar por cada arista del grafo solo una vez –no importa que visite los vértices mas de una vez–, asimismo si el vértice de partida es el vértice de llegada el camino es un ciclo euleriano (este ultimo es el famoso problema de los puentes de Konigsberg cap.XIII).

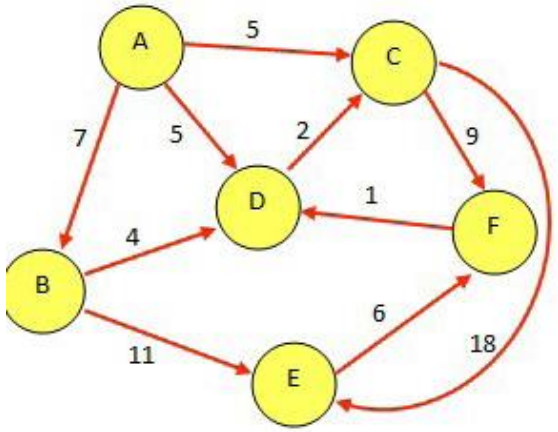


Clase 4: TEMARIO. Grafos

Grafos representación:

	A	B	C	D	E	F
A	0	7	5	5	0	0
B	0	0	0	4	11	0
C	0	0	0	0	18	9
D	0	0	2	0	0	0
E	0	0	0	0	0	6
F	0	0	0	1	0	0

matriz de adyacencias

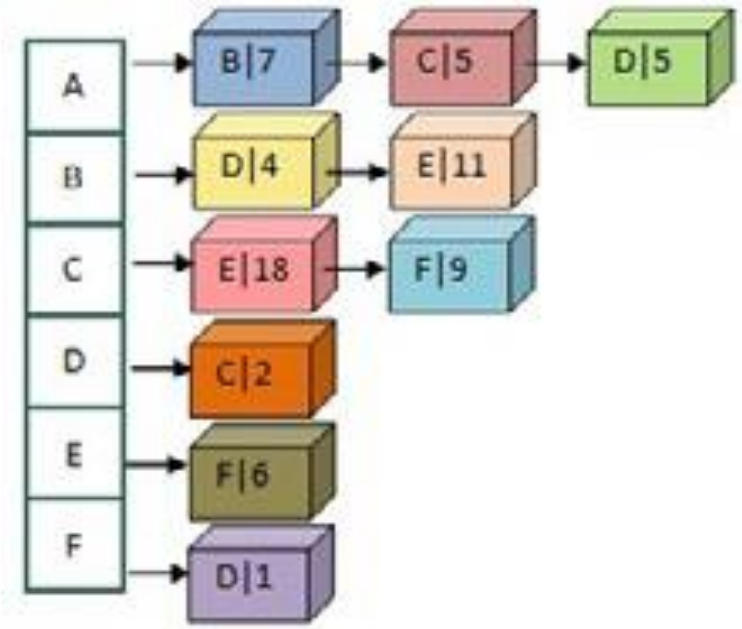


No es la representación mas adecuada si el grafo no es **denso**

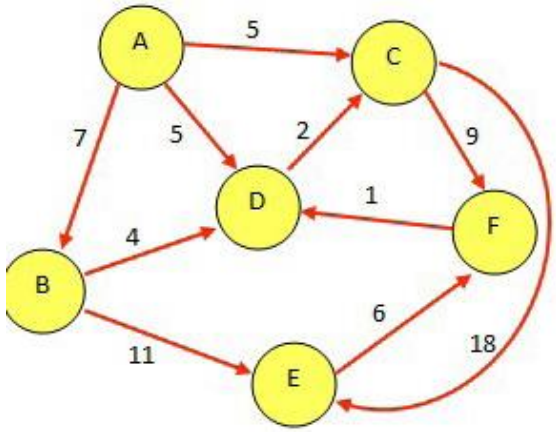
Grafo es denso cuando el número de aristas es mayor a la cantidad de vértices

Clase 4: TEMARIO. Grafos

Grafos representación:



Vector de lista de adyacencias



Independientemente del tipo de grafo que usemos deberemos marcar todos los vértices como no visitados y cuando pasamos por cada nodo lo marcamos como visitado, para evitar pasar dos veces por el mismo nodo y entrar en un ciclo infinito

Clase 4: TEMARIO. Grafos

TAD Grafo:

1. insertar_vértice(grafo, dato). Agrega el elemento como un vértice al grafo;
2. insertar_arista(grafo, dato, vértice origen, vértice destino). Agrega el elemento como una arista desde el vértice destino al vértice origen, si el grafo es dirigido y si es no dirigido también lo inserta desde el vértice destino al origen;
3. agregar_arista(vértice origen, dato, vértice destino). Agrega una arista a la lista de aristas del vértice origen al vértice destino;
4. eliminar_vértice(grafo, clave). Elimina y devuelve del grafo si encuentra un vértice que coincida con la clave dada –el primero que encuentre– y además recorre el resto de los vértices eliminando las aristas cuyo destino sea el vértice eliminado, si devuelve None significa que no se encontró la clave en el grafo;
5. Eliminar_arista(vértice, destino). Elimina y devuelve del vértice si encuentra una arista que coincida con el destino dado –el primero que encuentre–, sino devuelve None significa que no se encontró la arista destino en el vértice
6. buscar_vértice(grafo, clave). Devuelve un puntero que apunta al vértice que contiene un elemento que coincida con la clave –el primero que encuentra–, si no devuelve None;

Clase 4: TEMARIO. Grafos

TAD Grafo:

7. buscar_arista(grafo, vértice origen, vértice destino). Devuelve un puntero que apunta a la arista que contiene el elemento que coincida con el destino –el primero que encuentra–, en la lista de aristas del vértice origen, sino devuelve None

8. grafo_vacío(grafo). Devuelve verdadero si el grafo no contiene elementos;

9. tamaño(grafo). Devuelve la cantidad de vértices en la grafo;

10. barrido_vértices(grafo). Realiza un barrido de los vértices ordenados;

11. es_adyacente(vértice, destino). Devuelve verdadero si el destino es un nodo adyacente al vértice;

12. adyacentes(vértice). Realiza un barrido de los nodos adyacentes al vértice;

13. marcar_no_visitado(grafo). Marca todos los nodos vértices como no visitados poniendo el campo visitado con valor falso (false);

14. existepaso(grafo, vértice origen, vértice destino). Devuelve verdadero (true) si es posible ir desde el vértice origen hasta el vértice destino, caso contrario retornará falso;

15. recorrido_profundidad(grafo, vértice inicio). Realiza un recorrido en profundidad del grafo a partir del vértice de inicio;

Clase 1: TEMARIO. Grafos

TAD Grafo:

16. recorrido_amplitud(grafo, vértice inicio). Realiza un recorrido en amplitud del grafo a partir del vértice de inicio;

17. dijkstra(grafo, vértice origen, vértice destino). Devuelve el camino más corto desde el vértice origen al vértice destino;

Recorrido en profundidad: se comienza por un vértice arbitrariamente, se trata dicho vértice y se lo marca como visitado, luego se trata recursivamente todos los vértices adyacentes no visitados,
si al finalizar quedan vértices sin visitar se toma el siguiente vértice sin tratar y se repite el procedimiento. el grafo de ejemplo, se obtendrá como salida A, B, D, C, E, F.

Recorrido en amplitud: se crea una cola a la cual se le agrega un vértice arbitrariamente y se lo marca como visitado, luego mientras la cola no esté vacía se atiende el primer elemento de la cola y se trata el vértice. A continuación se agregan a la cola y se marcan como visitados todos los vértices adyacentes que aún no hayan sido visitados; cuando la cola queda vacía si aún quedan vértices sin visitar, se toma el siguiente vértice sin tratar y repite el procedimiento anterior.
La salida si aplicamos este barrido al grafo de ejemplo es A, B, C, D, E, F.

Clase 4

unab

Consultas

Clase 3: CIERRE DE LA CLASE

Temas a desarrollar la próxima clase

- ❏ Árboles Binarios
- ❏ Recorridos
- ❏ Análisis tiempo de ejecución