UNIVERSIDAD NACIONAL
GUILLERMO BROWN

CLASE 5 - Unidad 3

Árboles Binarios

ESTRUCTURAS DE DATOS (271)

Clase N. 5. Unidad 3.

Clase 5: AGENDA

AGENDA

- **Temario:**
  - Arboles binarios. Arboles de expresión.
  - Construcción de arboles. Búsquedas.
  - Recorridos ordenados (InOrden, PostOrden, PreOrden).
  - Actualización: inserción y borrado. Análisis de tiempo de ejecución de estas operaciones.

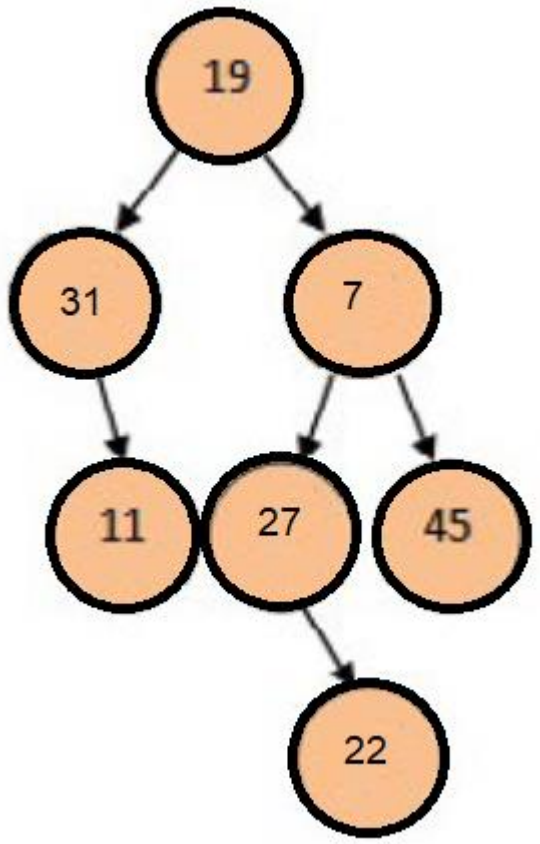
- **Ejemplos** en Lenguajes **Python**
- **Temas relacionados y links de interés**
- **Práctica**
- **Cierre de la clase**

Clase 5: Árboles Binarios

Arboles Binarios:

Un **árbol** es una estructura ramificada donde cada elemento puede relacionarse con muchos elementos.

El **árbol binario** es un caso particular de árbol en el que cada nodo puede tener como máximo dos hijos, los cuales se denominan hijo izquierdo e hijo derecho.



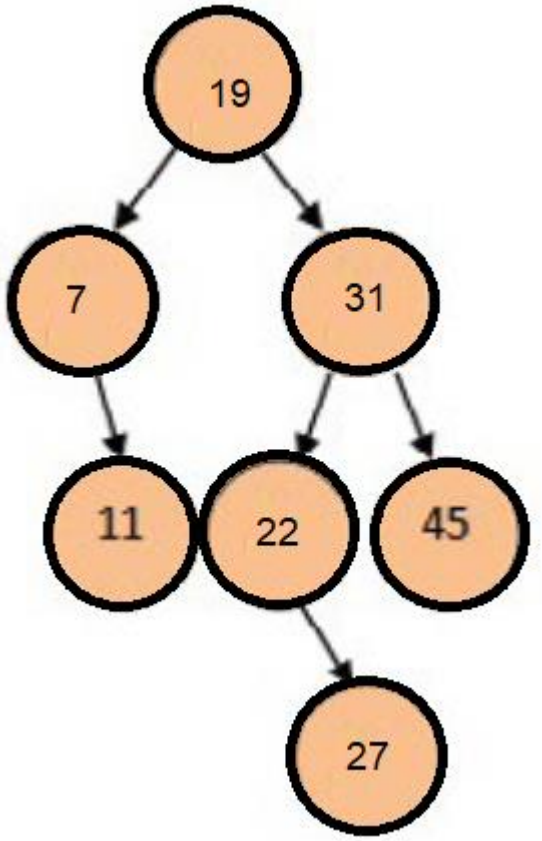
Arboles Binarios propiedades :

- el numero máximo de nodos en el nivel **i** de un árbol binario es $\mathbf{2^i-1}$,
- el numero máximo de nodos en un árbol binario de altura **k** es $\mathbf{2^k-1.}$
- Se dice que un árbol binario de altura **k** esta lleno si tiene $\mathbf{2^k-1}$ nodos.
- se dice que un árbol binario de altura **k** esta completo si esta lleno hasta la altura **k-1** y su ultimo nivel esta ocupado de izquierda a derecha.

Clase 5: Árboles Binarios

Arboles Binarios de búsqueda:

El **árbol binario de búsqueda** es un caso particular de un árbol binario. Estos árboles están ordenados, es decir que todos los nodos del subarbol izquierdo son menores que el padre y todos los del subarbol derecho mayores, a partir de esto se puede establecer la siguiente regla: para cada nodo del árbol se establece que el nodo izquierdo es menor y el nodo derecho es mayor.



Clase 5: Arboles binarios

Operaciones:

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

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

3. reemplazar(raiz). Determina el nodo que reemplazara 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–, sino devuelve None

Clase 5: Arboles binarios

Operaciones:

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 5: Arboles Binarios

Implementación:

```
#estructura del nodo arbol
class nodo_arbol(object):
    #clase nodo arbol
    def __init__(self,info):
        #crea un nodo con la informacion
        self.izquierda = None
        self.derecha = None
        self.informacion = info

class NodoCola(object):
    #clase nodo cola
    info = None
    siguiente = None

class Cola(object):
    #clase cola
    def __init__(self):
        #crea una cola vacia
        self.frente = None
        self.final = None
        self.tamaño = 0
```

Usaremos lo visto en la teoría de la clase 3.
Importaremos las estructuras y las funciones necesarias para trabajar con el árbol binario
Esto lo hacemos usando **from** modulo **import** funciones

Para el recorrido por nivel usaremos el tad_cola y el tad_árbol
Por lo que deberemos agregar la importación de los tad
Al comienzo de nuestro programa

```
from tad_arbol import insertar_nodo,arbol_vacio,imprimir_arbol
from tad_cola import Cola,push,pop,cola_vacia
```

Implementación:

```
def inorden(raiz):
    #realiza el recorrido en orden del arbol
    if(raiz is not None):
        inorden(raiz.izquierda)
        print(raiz.informacion)
        inorden(raiz.derecha)

def preorden(raiz):
    #realiza el recorrido preorden del arbol
    if(raiz is not None):
        print(raiz.informacion)
        preorden(raiz.izquierda)
        preorden(raiz.derecha)


def postorden(raiz):
    #realiza el recorrido postorden del arbol
    if(raiz is not None):
        postorden(raiz.derecha)
        print(raiz.informacion)
        postorden(raiz.izquierda)
```

Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho

Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.

Se procesan primero los hijos, izquierdo y derecho, y luego la raíz

Clase 5: Arboles Binarios

Implementación:

```
def por_nivel(raiz):
    #recorre el arbol por niveles
    pendientes = Cola()
    push(pendientes,raiz)
    while(not cola_vacia(pendientes)):
        nodo = pop(pendientes)
        print(nodo.informacion)
        if(nodo.izquierda is not None):
            push(pendientes,nodo.izquierda)
        if(nodo.derecha is not None):
            push(pendientes,nodo.derecha)
```

Para el recorrido por nivel usaremos el tad_cola y el tad_árbol

Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

```
arbol_binario = insertar_nodo(None, 45)
arbol_binario  = insertar_nodo(arbol_binario , 11)
arbol_binario  = insertar_nodo(arbol_binario , 85)
arbol_binario  = insertar_nodo(arbol_binario , 1)
arbol_binario  = insertar_nodo(arbol_binario , 185)
arbol_binario  = insertar_nodo(arbol_binario , 121)
arbol_binario  = insertar_nodo(arbol_binario , 44)
#imprimir_arbol(arbol_binario)
print("Recorrido por niveles: ")
por_nivel(arbol_binario)
print("Recorrido InOrden: ")
inorden(arbol_binario)
print("Recorrido PreOrden: ")
preorden(arbol_binario)
print("Recorrido PostOrden: ")
postorden(arbol_binario)
```

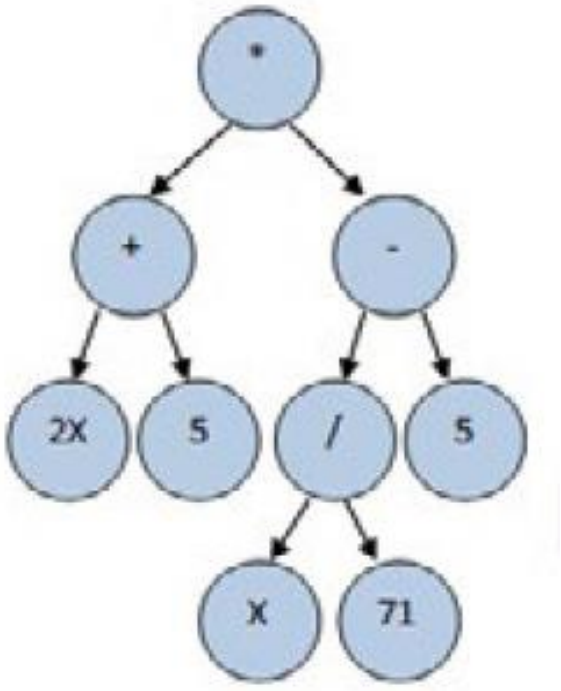
Construimos el árbol y probamos los recorridos

Clase 5: Arboles Binarios

Árbol de Expresión:

Es un árbol binario asociado a una operación aritmética.

- Nodos internos: representan operadores
- Nodos externos(hojas): representan valores



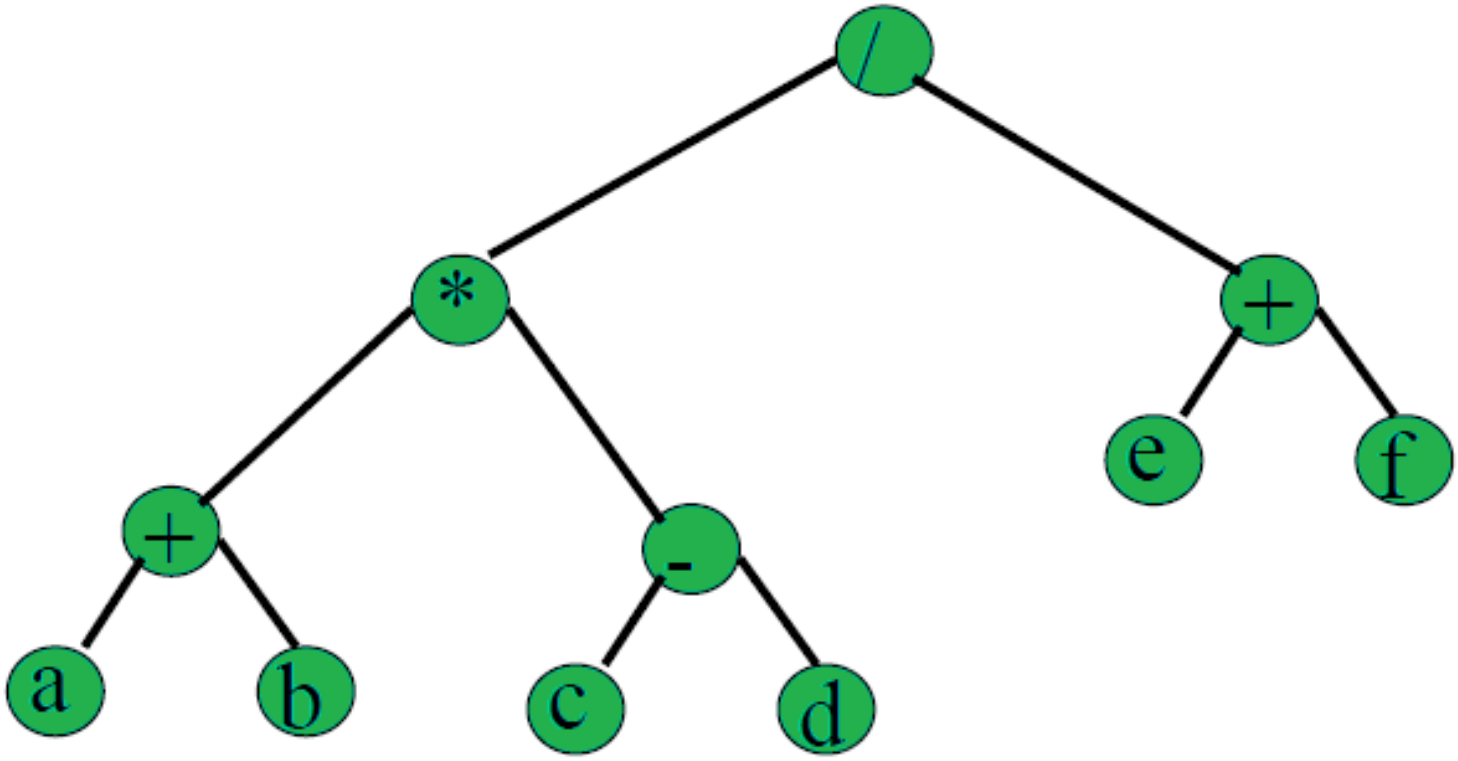
Usos:

- En compiladores para analizar, optimizar y traducir Programas.
- Evaluar expresiones algebraicas o lógicas. No es necesario usar paréntesis
- Traducir expresiones a notación sufija, prefija e infija

Clase 5: Arboles Binarios

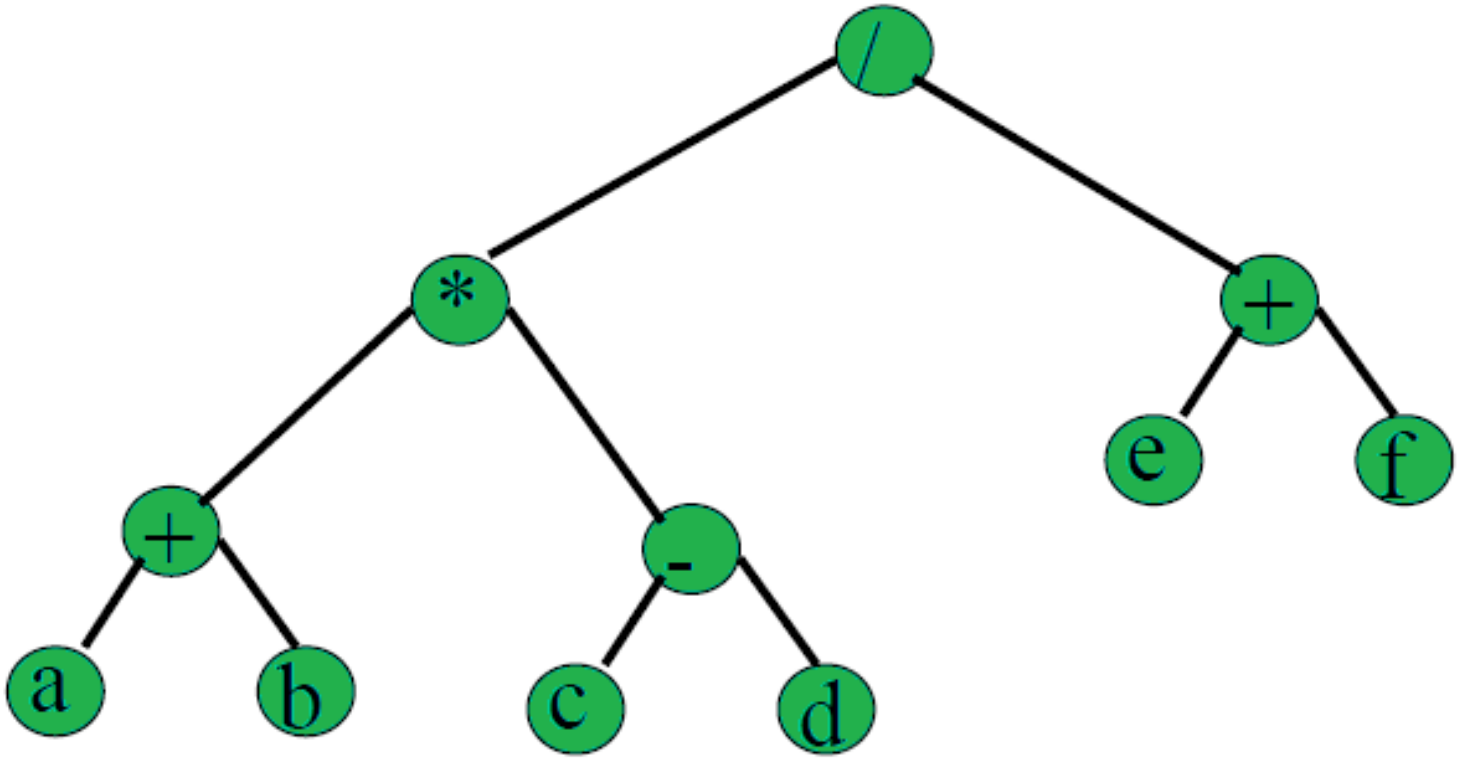
unab

Árbol de Expresión:



Clase 5: Arboles Binarios

Árbol de Expresión recorrido:

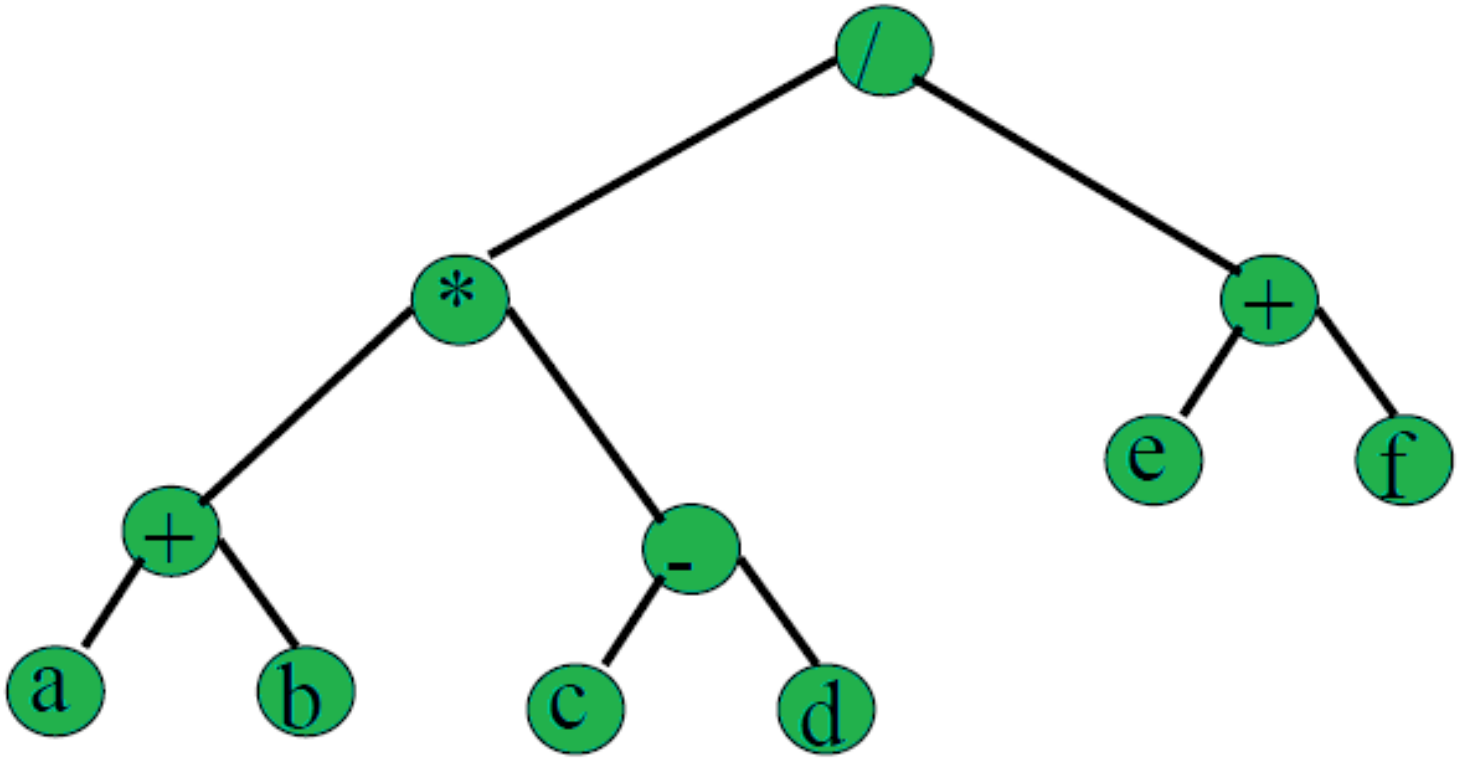


Inorden: (((a + b) * (c – d)) / (e + f))

Clase 5: Arboles Binarios

unab

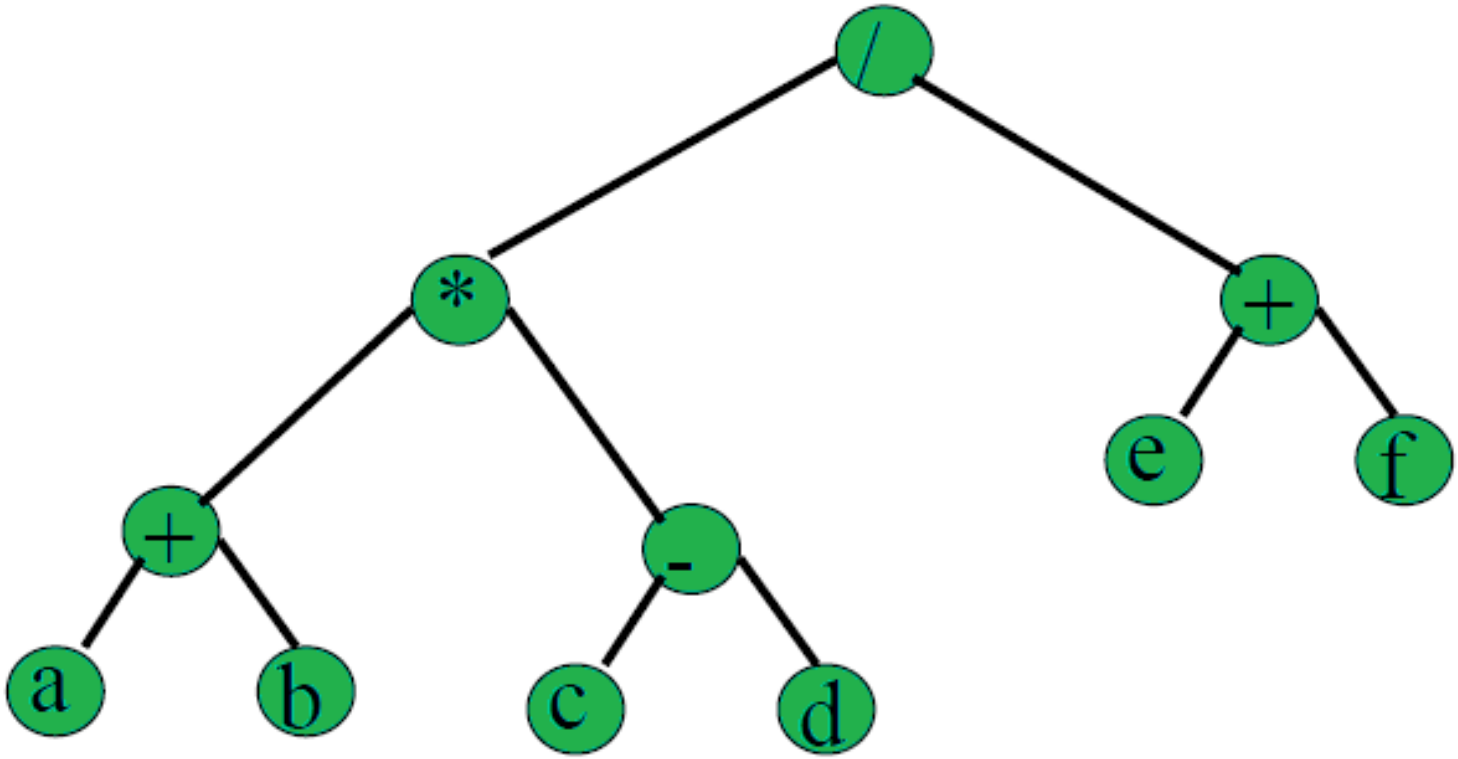
Árbol de Expresión recorrido:



Preorden: /*+ab-cd+ef

Clase 5: Arboles Binarios

Árbol de Expresión recorrido:



Postorden: ab+cd-*ef+/

Clase 5: Arboles Binarios

Árbol de Expresión:

Construcción de un árbol de expresión a partir de una expresión postfija

Algoritmo:

tomo un carácter de la expresión
mientras (existe carácter) hacer
si es un ***operando*** *creo un nodo y lo apilo.*
si es un ***operador*** *(lo tomo como la raíz de los dos últimos nodos creados)*
*- **creo** un nodo R,*
*- **desapilo** y lo agrego como hijo derecho de R*
*- **desapilo** y lo agrego como hijo izquierdo de R*
*- **apilo** R.*
tomo otro carácter
fin

Clase 5: Arboles Binarios

unab

Árbol de Expresión:

Construcción de un árbol de expresión a partir de una expresión prefija

Algoritmo:

ArbolExpresión (A: ArbolBin, exp: string)

si exp nulo nada.
si es un operador - creo un nodo raíz R
- ArbolExpresión (subArbIzq de R, exp (sin 1° carácter))
- ArbolExpresión (subArbDer de R, exp (sin 1° carácter))
si es un operando creo un nodo (hoja)

Árbol de Expresión:

Construcción de un árbol de expresión a partir de una expresión infija

Clase 5: Arboles Binarios

Árbol de Expresión:

Construcción de un árbol de expresión a partir de una expresión infija

(i) Estrategia del Algoritmo para convertir exp. infija en postfija :

a) si es un operando se coloca en la salida.
b) si es un operador se maneja una pila según la prioridad del operador en relación al tope de la pila

operador con > prioridad que el tope –> se apila
operador con <= prioridad que el tope –> se desapila elemento colocándolo en la salida.
Se vuelve a comparar el operador con el tope de la pila

c) si es un "(" , ")" "(" se apila
")" se desapila todo hasta el "(", incluído éste

d) cuando se llega al final de la expresión, se desapilan todos los elementos llevándolos a la salida, hasta que la pila quede vacía.

Clase 5

unab

Consultas

Temas a desarrollar la próxima clase

- Repaso general de las Unidades 1,2 y 3