Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

Tarea #4: Árboles

1) Investiga como se construye el árbol de una expresión aritmética.

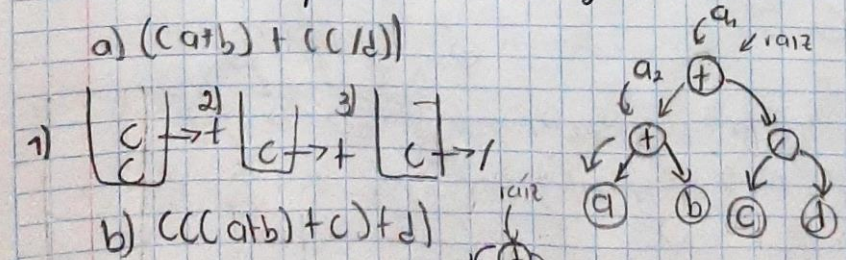
Método:

1.- Prioridad se determina solo por paréntesis
2).- La expresion completa se sitúa entre paréntesis
3.- Los operadores con mas prioridad: (*, /) y los ordenados aquí son (+, −)

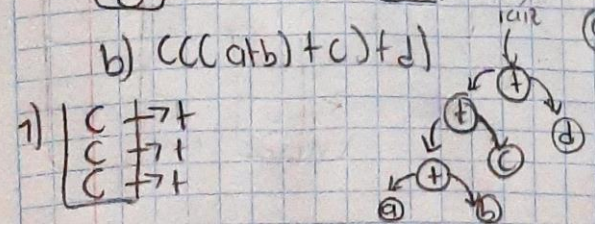
((((4*5)+(6/7))−(8+9))

1) La primera vez que se encuentra un paréntesis a izquierda, crea un nodo y lo hace en el raíz. se le llama nodo actual y se sitúa en la pila
2) Cada vez que se encuentre un nuevo paréntesis a izq. crear un nuevo nodo. Si el nodo actual no tiene un hijo izq. hacer el nuevo nodo el hijo izquierdo; en caso contrario, hacerlo el hijo derecho. Hacer el nuevo nodo el nodo actual y situar en una pila.
3) Cuando se encuentra un operando, crear un nuevo nodo y asignar el operando a su campo de datos. Si el nodo actual no tiene un hijo izquierdo, hacer el nuevo nodo el hijo izquierdo; en caso contrario, hacerlo el hijo derecho.
4) Cuando se encuentra un operador, sacar un puntero de la pila y situar el operador en el campo datos del nodo del puntero.
5) Ignorar paréntesis derecho y blancos.

a) ((a+b)+(c/d))



b) (((a+b)+c)+d)



Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

2.- Investiga en que consiste o como se realiza el balanceo de árboles binarios de búsqueda (Árboles AVL) realiza ejemplos de inserción y eliminación siguiendo el algoritmo.

Un arbol totalmente equilibrado se caracteriza porque la altura de la rama izquierda es igual que la altura de la rama derecha para cada uno de los nodos del árbol y difieren máximo en 1.

La condicion de equilibrio de cada nodo implica restricciones en las alturas de los subarboles, tal que la rama izquierda, los valores de la altura sea h y la altura de la derecha sea h-1, h, h+1 y a ello a cada nodo agrega el balance del nodo.

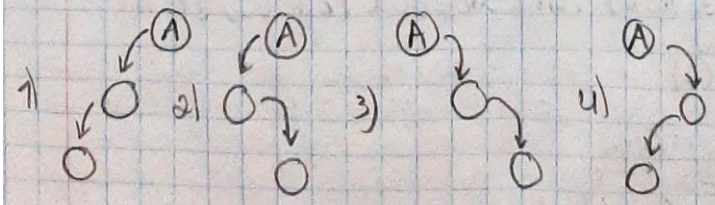
La altura del subarbol derecho menos la altura del subizq, el factor de equilibrio puede tomar : -1, 0, 1.

- Inserción

Aplica el algoritmo de insercion en un arbol de busqueda; este algoritmo sigue el camino de busqueda hasta llegar al fondo del árbol y se enlaza como nodo hoja y con factor de equilibrio 0. Es necesario recorrer el camino de búsqueda en sentido contrario, hacia la raiz, para actualizar el campo factor de equilibrio. Si los nodos pueden haber cambiado.

Existen 4 posibles casos:

1) insercion en el subarbol izquierdo de la rama izquierda de A
2) Insercion en el subarbol derecho de la rama izquierda de A
3) insercion en el subarbol derecho de la rama derecha de A
4) Insercion en el subarbol izquierdo de la rama derecha de A



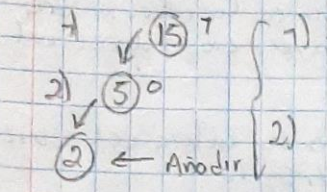
Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

- Rotacion simple: 1-3

Afectan a dos nodos, el tercero no modifica, solo se necesita una rotacion. Una vez realizado la rotacion los factores son cero

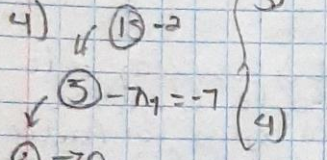
- Rotacion doble: 2-4

Se sube el nodo agregado como raiz del subarbol, como izq colocar la raiz original y a la derecha el nodo derecho o viceversa en base al caso.



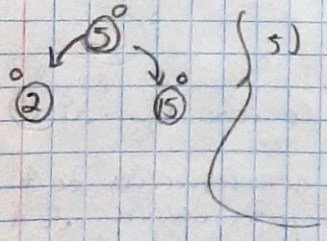
1) NodoAvl iz:
iz = insertarAvl(raiz.subarbol, d[,h)

2) NodoAvl iz:
iz = insertarAvl(raiz.subarbol, d[,h)



3) if (raiz == null)
raiz = new Nodo
h.set(true)

4) switch(raiz.fe)
case -7:
if(n7.fe == -7)
raiz = rotacionII(raiz, n7)



5) raiz.izq(n7.subarbol)
n7.ramaDerecha(raiz);

if(n7.fe == -7) *n = raiz
n.fe = 0
n7.fe = 0

NodoAvl iz:
iz = insertarAvl(NodoAvl) raiz.subarbol.izdo(), d[, h]
raiz.ramaIzdo(iz)
if(h.booleanValue())
switch:
case 7:
raiz.fe = 0
h.setLogical(false)

Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

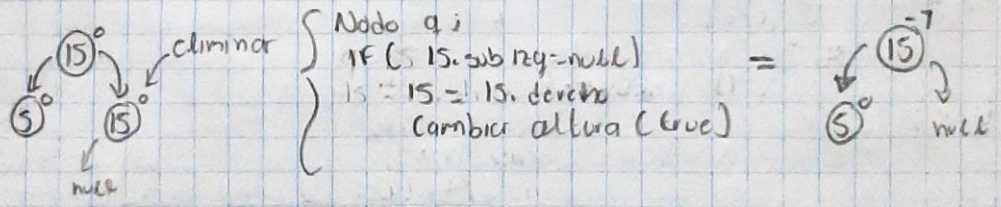
```
case 0:
    raiz.fe = -1

case -1:
    n1 = (NodoAvl) raiz.subarbolIzdo()
    IF( n1.fe == -1)
        raiz = rotacionII(raiz,n1)
    else
        raiz = rotacionID(raiz,n1)
    h.setLogical(false).
```

- Eliminación

1) Si el nodo es hoja solo se elimina
2) Se busca el nodo mas a la derecha del subarbol izquierdo, el de mayor clave en el subarbol de claves menores, se hace una copia y se elimina el copiado.

3) El algoritmo de ordenar los factores de equilibrio.

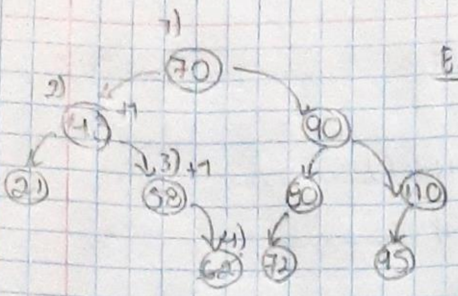



```
Nodo Avl drq:
q = r;
IF( q.subizq() == null)
    r = q.subder();
    cambia.altura (true)
else if( q.subdev() == null)
    r = q.subizq()
    cambioaltura.setLogical(true)
else
    Nodo Avl izq;
        iz = reemplazar(r, r.subizq, cambiaraltura)
        r.raizIzq(iz)
        IF( cambiarAltura.booleano())
            r = equilibrar1(r, cambiarAltura).
```

Díaz Hernández Marcos Bryan

Tarea N°4

Grupo: 09



E. 62

1) NodoAvl iz
iz = borrarAlv(sub.izq, clav, ch)

2) NodoAvl dr
dr = borrarAlv(sub.drc, clav, ch)

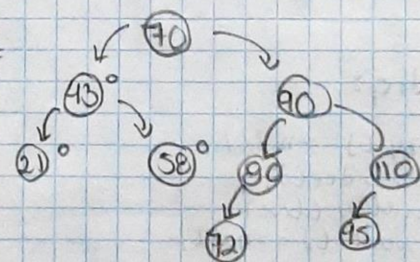
3) NodoAvl drc
dr = borrarAlv(sub.drc, clave.ch)

4) NodoAvl dr
dr = borrarAlv(sub.dr, clave.ch)

3) if (Abdoizq = null)
r = q.subdr() = null
cambiaAltura(True)

6) equilibrar (r, cambiaAltura) } Rotacion derecha
NodoAvl n1
switch (r.Fe)
case +1:
n1 = r.subdr
if (n1.fe == 0)
cambiaAltura(false)
n = rotacionDD(n, n1)

7) if (n1.fe == +1) }
n.fe = 0;
n1.fe = 0;



3). ¿Qué es un árbol B^* y cuales son sus principales diferencias con un arbol B+?

Los arboles B^* tienen un comportamiento similar a los arboles B, la diferencia se aplica en la redistribucion en el alto logrando detener hasta último momento la necesidad de los split con el consecuente mayor espacio.

Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

Lo anterior hace referencia a que se pueden almacenar mas datos tal que el split permita que se divida en 3 nodos 2/3 llenos. A partir de 2 nodos

Otra ventaja derivada de lo anterior es la altura ya que no se hacen redistribuciones de forma constante y cada vez que se realiza, un nodo menos se crea, a comparación de un arbol B.

- Caracteristicas:

  - Todos los ramos tienen igual profundidad
  - Si un nodo tiene k claves, tiene m+1 descendientes.
  - Se respeta a la izq menor y der mayor

- Propiedades

  Claves max: 11
  Descendientes max: m
  Cantidad minima claves: $\left[\frac{m-2}{3}\right]^3$

  Cantidad minima de descendientes: $\left[\frac{(2m-1)}{3}\right]^4$

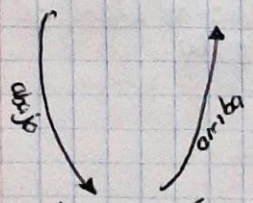
Raiz:

Como se parte la separación de los nodos desde dos de estos, la raiz al ser uno, se comporta como sigue:

1) Nodo de arbol B
2) Raiz con mayor tamaño y split en dos hojas 2/3 llenos
3) Raiz con mayor tamaño y split en dos hojas 2/3 llenas

- Busqueda: igual al B

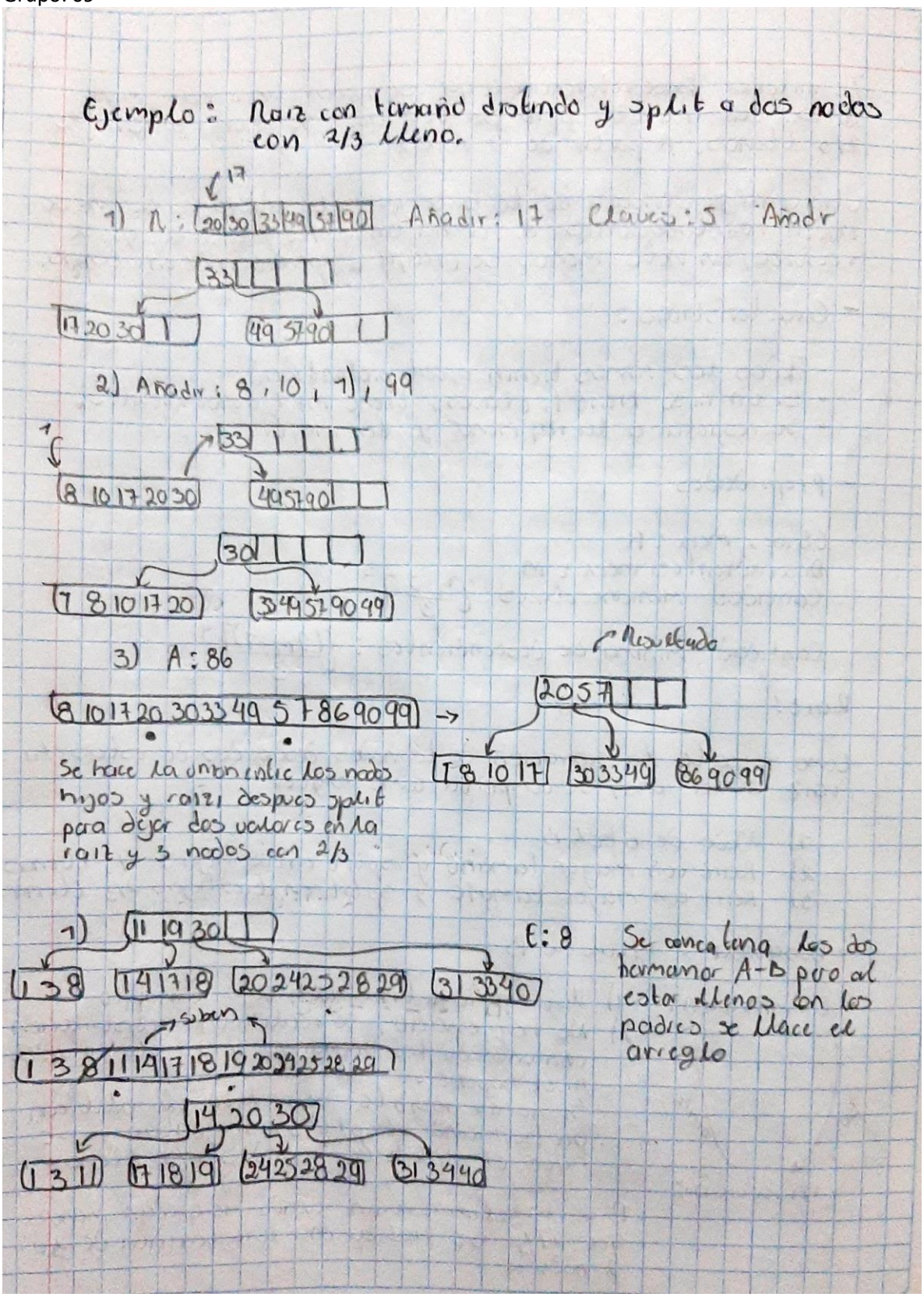
- Eliminacion:
  1) Hay espacio en las claves, solo se inserta
  2) No hay espacio, se realiza una redistribucion con uno de los hermanos y el elemento del nivel superior.
     Si no es posible se realiza el split con un hermano completo, se juntan y se hace el split a 3 nodos 2/3 llenos



- Insercion:

Para la eliminacion cuando se quedan menos que 2/3 se realiza una concatenacion de 3 a 2 nodos.

Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09



Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

4) En que consisten los árboles RedBlack y donde se utilizan.

Los arboles redblack son arboles binarios de busqueda con nodos con colores negros y rojos para satisfacer sus caracteristicas

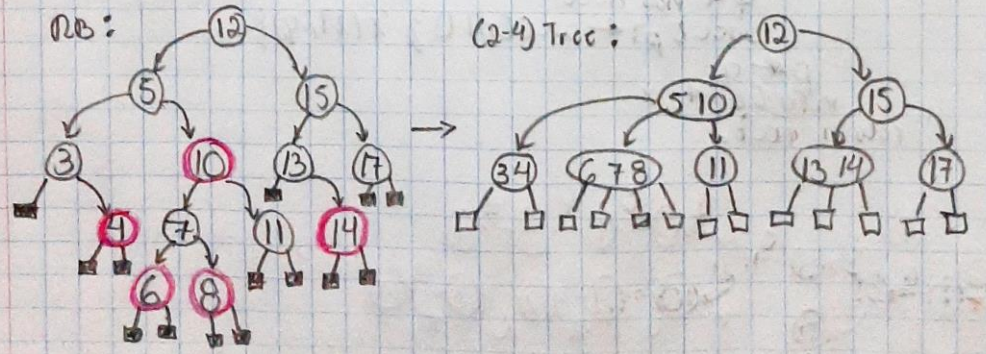
- Root: la root es negra
- Exterior: los nodos exteriores son negros
- Red: dos hijos de un nodo rojo son negros
- Depth: todos los nodos exteriores tienen la misma profundidad definida por los de sus padres.

Este tipo de arbol es balanceado por lo que se conservan las operaciones con respecto a la insercion, eliminacion y busqueda con una complejidad referidad como O(1) para los cambios en la estructura despues del ordenamiento del árbol.

En caso de la insercion, eliminacion y la busqueda se plantean para el peor caso con la complejidad O(log n)

Existe la posibilidad de crear de un (2-4) arbol a partir de un redblack uniendo cada nodo rojo con su padre.

Este tipo de nodos/arboles se utiliza para el kernel de Linux, un kernel como tal es el núcleo de Linux, tal que se puedan organizar los procesos, de hecho los kernel se usan para algo de M2.



Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09

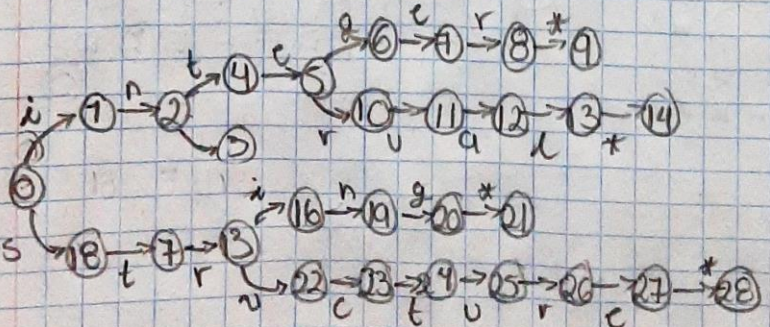
5) Explica que es un trie y cual es la diferencia con respecto a un arbol de prefijos simples

Un trie es un arbol que surge a partir de la raíz, que se va creando a partir de la raíz, hacia abajo, un arbol dirigido como un arbol hash, donde la raiz tiene una profundidad/altura 0 y donde un nodo con altura j puede acceder al j+1, el siguiente nodo.

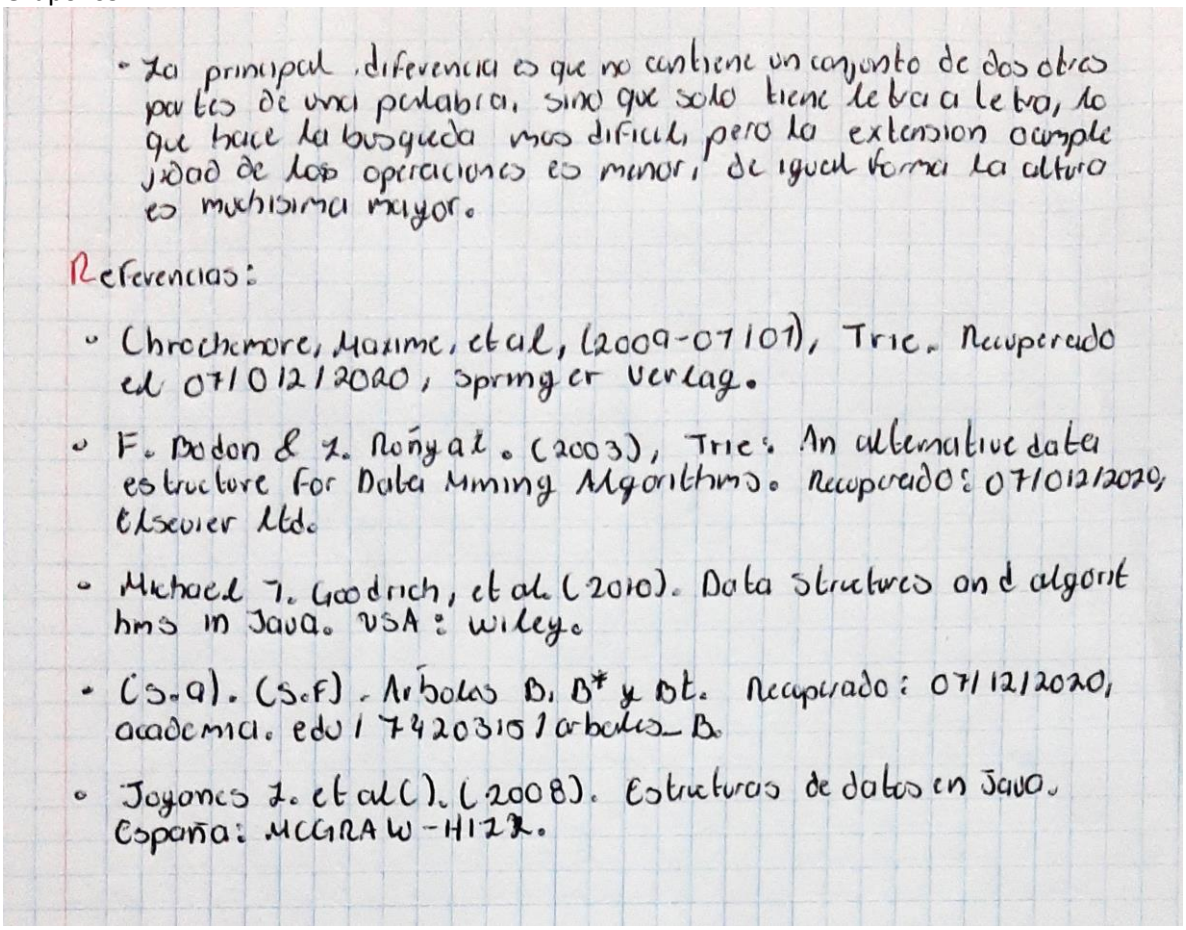
Existe la jerarquia del padre cuando un nodo i apunta a un nodo j y viceversa, cada una de las hojas representa el final de una oración, y esta palabra se forma desde la raiz hasta la hoja, por cierto que cada nodo contiene una letra y el nodo final se indica con (*)

La raiz cuenta como el simbolo o caracter vacio, ademas los vertices son marcados con una letra del abecedario, los nodos internos tienen dos hijos minimo.

```
Trie (x)                          x = { trie }
  root ← new node                 x = { int, integer, interval, string,
  For each string x ∈ X do                structure. }
      p ← root
      for i ← to |x| - 1 do
          q ← Target(p, x[i])
          IF q = NIL Then
              q ← new node
              Succ[p] ← succ[p] ∪ { (x[i], q) }
          p ← q
          info[p] ← x
      return root.
```



Díaz Hernández Marcos Bryan
Tarea N°4
Grupo: 09



- La principal diferencia es que no contiene un conjunto de dos otras partes de una palabra, sino que solo tiene letra a letra, lo que hace la busqueda mas dificil, pero la extension o complejidad de las operaciones es menor, de igual forma la altura es muchisima mayor.

Referencias:

- Chrochemore, Maxime, et al, (2009-07101), Trie. Recuperado el 07/012/2020, springer verlag.
- F. Bodon & L. Rónyai. (2003), Trie: An alternative data estructure for Data Mining Algorithms. Recuperado: 07/012/2020, Elsevier Ltd.
- Michael T. Goodrich, et al. (2010). Data structures and algorithms in Java. USA: wiley.
- (S.a). (S.f). Árboles B, B* y B+. Recuperado: 07/12/2020, academia.edu/7420315/arboles_B.
- Joyanes L. et al(). (2008). Estructuras de datos en Java. España: MCGRAW-HILL.