Árboles

EEDD - GRADO EN ING. INFORMATICA - UCO

Árboles de búsqueda multicamino

Motivación

- Motivación.
  - Queremos optimizar al máximo la búsqueda y para ello contamos con un árbol AVL que se asegura encontrar un valor con O(Log N).
  - Si estimamos que tendremos un conjunto de 8 millone de claves, ¿cuál será la altura el árbol AVL?
  - ¿Cómo podríamos mejorar esto?

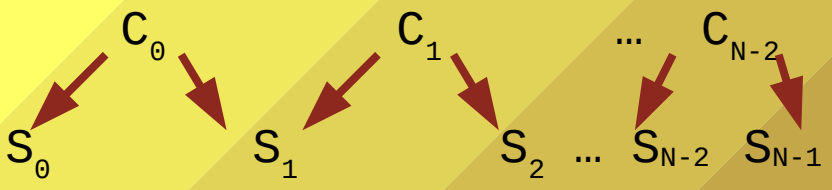
Contenidos

- Concepto de árbol de búsqueda multicamino.
- Implementación enlazada.
- Algoritmo de búsqueda.

EEDD - GRADO EN ING. INFORMATICA - UCO

Árboles de búsqueda multicamino

- Árbol de búsqueda multicamino de Grado N:
  - Es un árbol donde:
    - Cada subárbol, no vacío, almacena hasta N-1 claves en la raíz: $c_0,c_1,\ldots,c_{n-2}$.
    - Cada subárbol, no vacío, puede tener un máximo de N subárboles: $s_0,s_1,\ldots,s_{n-1}$.
    - Dada una clave c_j, $0<=j<N-1$, el subárbol s_j es su hijo izquierdo s_{j+1} el derecho.
    - En cada subárbol se cumple el invariante:
      - claves{s_0}<c_0<claves{s_1}<c_1< … claves{s_{n-2}}<c_{n-2}<claves{s_{n-1}}



Árboles de búsqueda multicamino

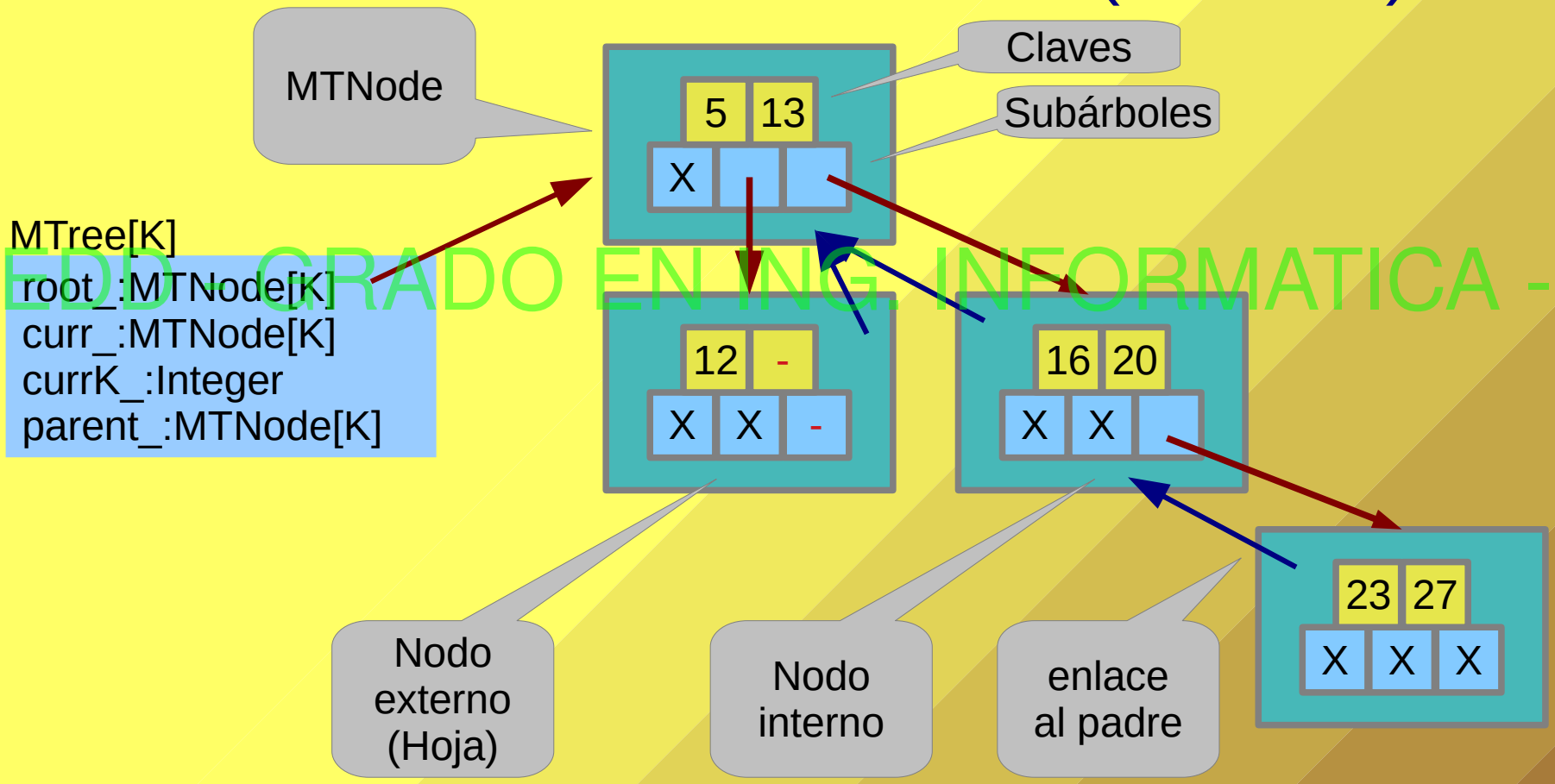
- Especificación del TAD MTree[K]
  - **Invariants:**
    - for each key i, {Keys of child i} < K_i < {keys of child i+1}
  - **Makers**:
    - create(d:Integer):MTree[K] //Makes an empty tree.
      - Pre-c: d>=2
      - Post-c: isEmpty()
      - Post-c: degree()==d
  - **Observers**:
    - isEmpty():Bool //is the tree empty?
    - degree():Integer //the maximum number of children.
    - nKeys():Integer //The number of root's keys.
      - Pre-c: not isEmpty()
      - Post: 0<nKeys()<degree()
    - nChild():Integer //The number of root's childs?
      - Pre-c: not isEmpty()
      - Post-c: retV=nKeys()+1
  - **Observers**:
    - key(i:Integer):K // Gets the i-th root's key.
      - Pre-c: not isEmpty()
      - Pre-c: 0<=i<nKeys()
    - child(i:Integer):MTree[K] //Gets the i-th subtree?.
      - Pre-c: not isEmpty()
      - Pre-c: 0<=i<nChild()
    - currentExists():Bool //Is there a valid current node?
      - Post-c: isEmpty() implies false
    - current():K //Gets current's key.
      - Pre-c: currentExists()
    - has(k:K):Bool //Is the key stored in the tree?
      - Pre-c: not isEmpty()

Árboles de búsqueda multicamino

- Especificación del TAD MTree[K]
  - **Modifiers**:
    - search(k:K): //search moving the cursor.
      - Post-c: not itemExists() or current()=K
    - insert(k:K) //ordered insertion of key in the tree.
      - Pre-c: isEmpty()
      - Post-c: not isEmpty()
      - Post-c: current()=k
      - Post-c: has(k)
    - remove() //remove the current key preserving the order.
      - Pre-c: currentExists()
      - Post-c: not currentExists()
      - Post-c: not has(old.current())

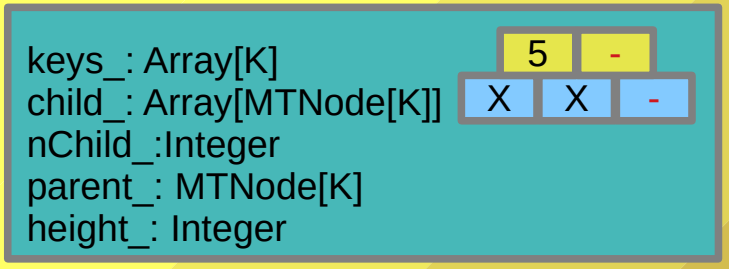
Árboles de búsqueda multicamino

- Diseño con nodos enlazados: (Grado 3).



Árboles de búsqueda multicamino

- ADT MTNode[K]:
  - **Makers**:
    - create(d:Integer, k:K):MTNode[K] // create a leaf node with one key.
      - Post-c: nKeys()==1 and nChild()==2
      - Post-c: key(0)==k
      - Post-c: degree()==d
      - Post-c: parent()==Void
      - Post-c: height()==0
  - **Observers**:
    - degree():Integer //The max number of childs.
    - nKeys():Integer //The number of stored keys.
      - Post-c: 0< nKeys() < degree()
    - nChild():Integer //How many subtrees are there?
      - Post-c: retV = nKeys()+1

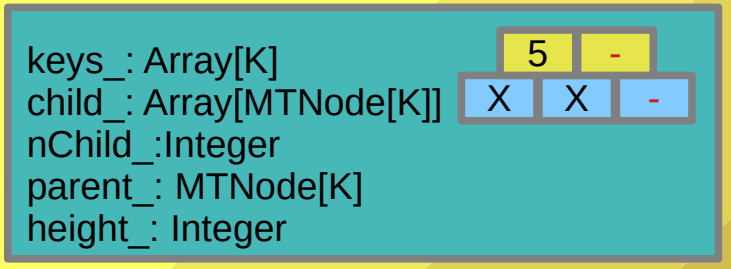


  - **Observers**:
    - height():Integer //The node height O(1).
    - parent():MTNode[K] //the node's parent.
    - locate(k:K):Integer //Get the key index where k should be //O(Log D).
      - Post-c: 0<= retV < nKeys()
    - has(k:K):Bool //Is key k stored?
    - key(i:Integer) //Gets key at index i.
      - Prec: 0<=i<nKeys()
    - child(i:Integer):MTNode[K] //Gets the child i.
      - Prec: 0<=i<nChild()

Árboles de búsqueda multicamino

ADT MTNode[K]:

- **Modifiers**:
  - setParent(p:MTNode[K]) //set the node's parent.
    - Post-c: parent()=p
  - insert(k:K) //Ordered insert of k.
    - Pre-c: nKeys()+1<degree()
    - Post-c: has(k)
    - Post-c: old.hash(k) OR nKeys()==old.nKeys()+1
  - setChild(i:Integer, n:MNode[K]) //set child at index i.
    - Pre-c: 0<=i<nChild()
    - Post-c: child(i)==n



degree=3
nKeys=1
nChild=2

Árboles de búsqueda multicamino

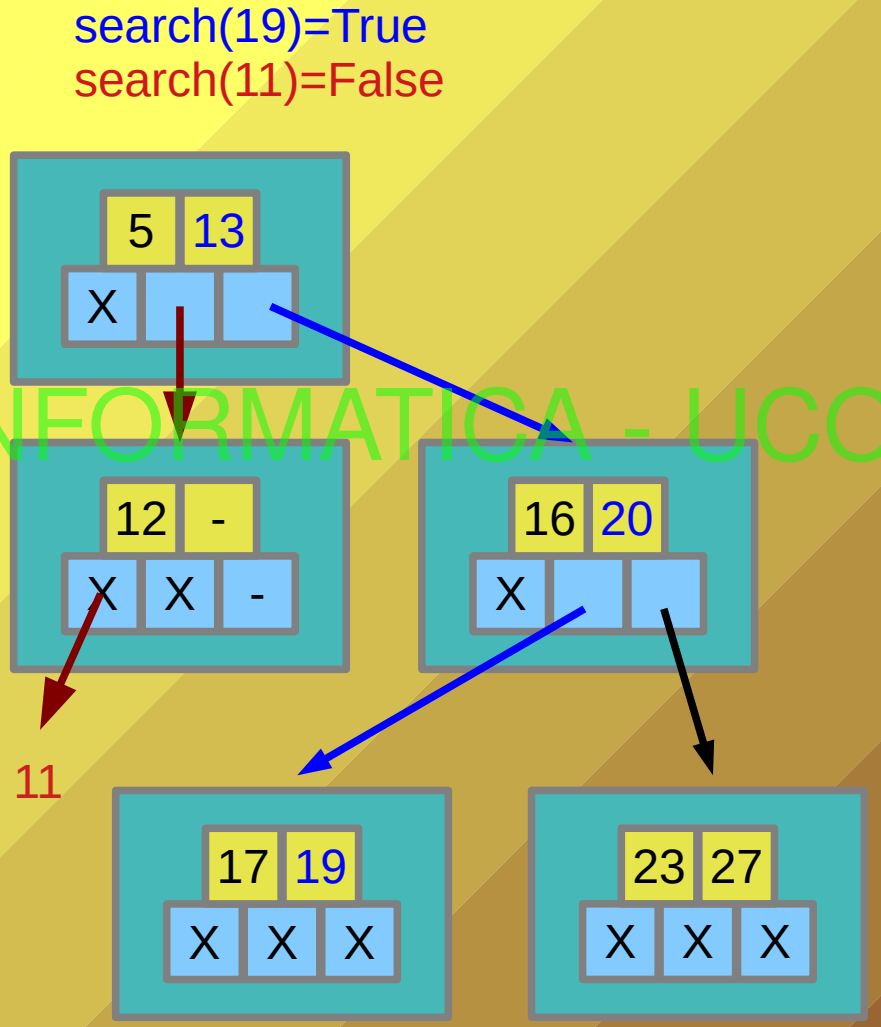
- Notas sobre su diseño:
  - Usados para almacenamiento externo (DB's).
  - Se suele almacenar de forma separada el campo clave (fichero índice) de los registros (fichero apilo estructurado que es un DArray sobre un fichero (push_back() O(1)).
  - Para reducir el tiempo de búsqueda/accesos a disco se utilizan órdenes muy altos (200 o más):
    - 1 nivel: 200 claves
    - 2 niveles 200^2= 40.000 claves
    - 3 niveles=200^3 = 8.000.000 claves.

¿Cuál sería la altura mínima de un árbol AVL para almacenar 8.000.000 de claves?

Árboles de búsqueda multicamino

- Búsqueda

```
Mtree::search(k:K):Bool
Var found: Bool
Begin
  found ← False
  parent_ ← Void
  curr_ ← _root
  currK_ ← -1
  while curr_ <> Void and not found do
    currK_ ← curr_.locate(k)//O(Log D)
    if curr_.key(currK_) == k then
      found ← True
    else if k<curr_.key(currK_) then
      parent_ ← curr_
      curr_ ← curr_.child(currK_)
    else
      parent_ ← curr_
      curr_ ← curr_.child(currK_+1)
  end-while
  return found
End.
```

Árboles de búsqueda multicamino

- Algoritmo Locate.

```
Algorithm locate(data:Array[T]; size:Integer; k:T):Integer
Prec:
  0 < size <= data.size()
  data[0:size] in order.
Var:
  a,b,loc:Integer
Begin
  If k <= data[0] then
    loc ← 0
  Else If k >= data[size-1] Then
    loc ← size-1
  Else
    a ← 0
    b ← size-1
    loc ← (a+b) div 2
    While data[loc] != k And (b-a)>1 Do
      If k<data[loc] Then
        b ← loc
      Else
        a ← loc
      End-If
      loc ← (a+b) div 2
    End-While
  End-If
  Return loc
End.
```

O( )

Árboles de búsqueda multicamino

- Resumen:
  - En grado de un árbol es el número máximo de sub-árboles que puede un cualquier sub-árbol nodo.
  - Mejoran el desempeño en la localización de una clave reduciendo la altura del árbol al aumentar el número de hijos por nodo (G»2).
  - ¿Cómo insertar/borrar para mantener el orden y el equilibrio?

Referencias

- Lecturas recomendadas:
  - Cap. 13 de “Estructuras de Datos”, A. Carmona y otros. U. de Córdoba. 1999.
  - Wikipedia.
    - https://en.wikipedia.org/wiki/Search_tree