Árboles

Árbol Multicamino: Trie

Contenidos

- Concepto Trie.
- Operaciones de inserción y búsqueda y obtención de claves.

EEDD - GRADO EN ING. INFORMATICA - UCO

Motivación

Queremos implementar un campo de texto de forma que en función de lo que se vaya escribiendo se vaya generando una lista de posibles coincidencias de entre un número prefijado de cadenas de texto.

Aplicaciones:

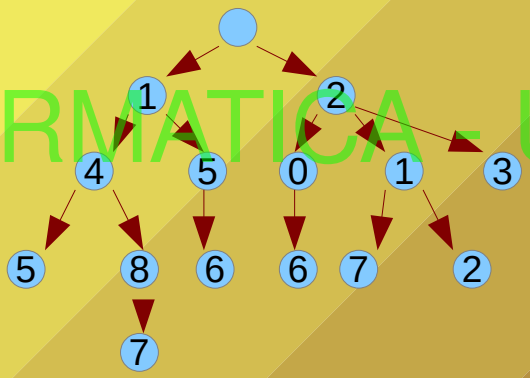
- En un aplicación para recetar medicamentos, la lista de todos los medicamentos.
- En un aplicación para gestionar envíos, la lista de localidades.
- …

¿Cómo podríamos almacenar todas las posibles cadenas de forma que se recuperen eficientemente al mismo tiempo que se está escribiendo?

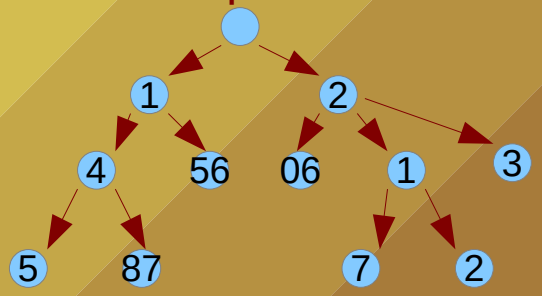
Trie

- **Concepto**:
  - Árboles multivía para almacenar y recuperar (re**TRIE**ve) claves de forma eficiente.
  - **Clave**: concatenación de símbolos de un alfabeto (si digital, tenemos **árboles dígitales**).
  - Cada nodo interno representa un símbolo del alfabeto.
  - Claves con igual prefijo se agrupan por subárboles.
- **Ventajas**:
  - Menor tiempo de búsqueda: O(M) frente a O(Log(N)) en ABO. M es long. clave más larga posible.
  - Menor espacio ocupado (claves comparten prefijos) compacto
- **Inconveniente**:
  - Desperdicio de espacio si poco denso. Solución si un nodo tiene un sólo hijo, permitir compactar nodos.

Calcula el Trie para las claves: {14, 145, 1487, 156, 206, 217, 212, 23}.



Calcula la versión compactada



Trie

- ADT Trie:

Makers:
- make():Trie //makes an empty trie.
  - Post-c: isEmtpy()

Observers:
- isEmpty():Bool //Is it empty?
- prefix():String //Root's prefix.
  - Pre-c: not isEmpty()
- isKey():Bool //Is the root's prefix a key?
  - Pre-c: not isEmpty()
- has(sufix:String):Bool //Is the prefix prefix()+sufix stored?
  - Pre-c: not isEmpty()
- retrieve(keys:DArray[String]) //Retrieve the stored keys.
  - Pre-c: not isEmpty()
  - Post-c: for all k in retV, prefix() is the prefix of k.
- child(sufix:String):Trie //gets the subtrie with prefix prefix()+sufix
  - Pre-c: not isEmpty()
  - Post-c: retV.isEmpty() or retV.prefix()==prefix()+sufix
  - Post-c: not retV.isEmpty() or retV.prefix()==""
- currentExists():Bool //Is there a current?.
  - Pre-c: not isEmpty()
  - Post-c: not isEmpty() or not currentExist()
- current():Trie //Gets the current sub-trie representing the keys with prefix prefix()+currentSymbol()
  - Pre-c: currentExits()
- currentSymbol():Char //Gets the current next symbol.
  - Pre-c: currentExists()

Trie

- ADT Trie:

Modifiers:

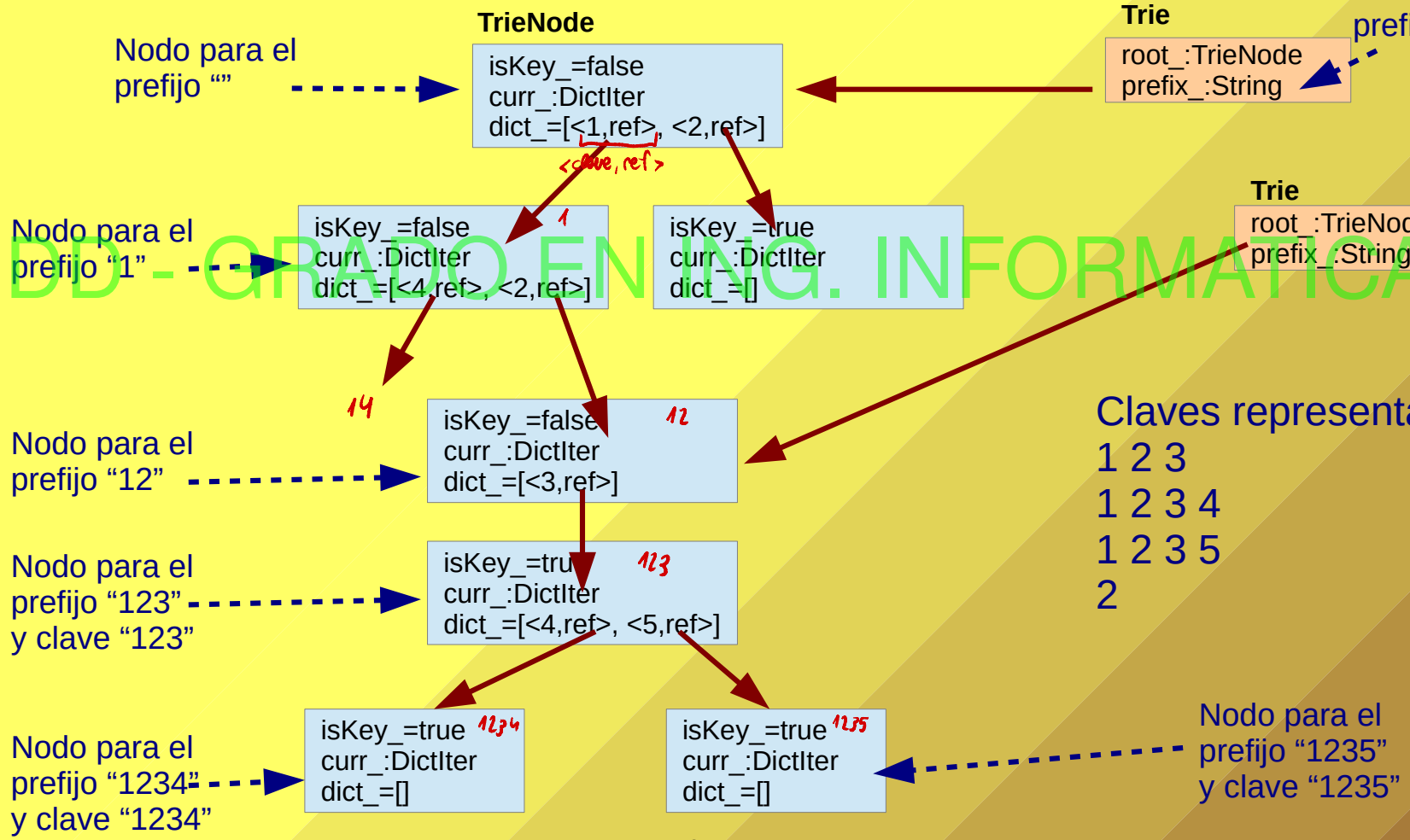
- insert(sufix:String) //Insert the key prefix()+sufix into the trie.
  - Pre-c: sufix <> “”
  - Pos-c: not isEmpty()
  - Post-c: has(sufix)
- findSymbol(s:Char) //Move current to the subtree representing the prefix prefix()+s
  - Pre-c: not isEmpty()
  - Post-c: !currentExists() or currentSymbol() == s
- gotoFirstSymbol() //Move current to the subtree prefix()+’first symbol’
  - Pre-c: not isEmpty()
  - Post-c: !currentExists() or current().prefix() == prefix()+currentSymbol()
- gotoNextSymbol() //Move current to the next subtree prefix()+’next symbol’ if it exists.
  - Pre-c: currentExists()
  - Post-c: not currentExists() or currentSymbol()==”next of old.currentSymbol()”.

Trie

- Diseño: consideraciones previas.
  - El nodo raíz del Trie representa el prefijo universal “” y no es una clave.
  - Cada trie hijo representa el siguiente símbolo de una clave cuyo prefijo es el representado por el árbol (padre).
  - Cada trie hijo representa un subconjunto de claves que comparten el prefijo: “prefijo del padre” + “siguiente símbolo asociado al subárbol”.
  - La raíz de un trie puede representar también una clave: por ejemplo podemos tener las claves “123”, “1234” y “1235”.
  - Las hojas representarán claves.

Trie

- Diseño: como nodos enlazados.



Trie

- ADT TrieNode.

Makers:
- create(isKeyState:Bool):TrieNode
  - Post-c: isKey()==isKeyState

Observes:
- isKey():Bool //This node represent a key.
- hasChild(c:Char):Bool //Has a child associated with symbol c?.
- child(c:Char):TrieNode //The child node associated with symbol c.
  - Pre-c: hasChild(c)
- currentExists():Bool// Is Current valid?
- currentSymbol():Char //Symbol associated to current.
  - Pre-c: currentExists()
- currentChild():TrieNode //Node associated to current symbol.
  - Pre-c: currentExists()

Modifiers:
- setIsKey(newState:Bool) //Set the isKey property value.
  - Post-c: isKey()==state
- setChild(c:Char, n:TrieNode) //set/add a new child.
  - Pos-c: has(c) and child(c)==n
- gotoFirstChild()//Move current to the first child.
- gotoNextChild()//Move current to the next child.
  - pre-c: currentExists()
- findChild(c:Char) //Move current to the child.
  - Post-c: not currentExists() or currentSymbol()==c

Trie

- Insertar una clave nueva.

```
Algorithm Trie::insert(k:String) O (  )
Var
  i:Integer
  node:TrieNode
Begin
  If root_ == Void Then
     root_ ← TrieNode::create(False)
  End-If
  node ← root_
  For i ← 0 To k.size() Inc 1 Do
    If node.has(k[i]) Then
      node ← node.child(k[i])
    Else
      newNode ← TrieNode::create(False)
      node.setChild(k[i], newNode)
      node ← newNode
    End-If
  End-For
  node.setIskeyState(true)
End.
```

Trie

- Buscar una clave.

```
Algorithm Trie::has(k:String):Bool
Var node:TrieNode
Begin
  node ← findNode(k)
  found ← node<>Void And node.isKey()
  Return found
End.
```



```
Algorithm Trie::findNode(pref:String):TrieNode  // O( m )
Var node:TrieNode, i:Integer
Begin
  node ← root_
  i ← 0
  While i<pref.size() And node<>Void Do
    If node.has(pref[i]) Then
       node ← node.child(pref[i])
       i ← i + 1
    Else
       node ← Void
    End-If
  End-While
  Return node
End.
```

Cantidad de dígitos max

Trie

- Operación de recuperación de claves (retrieve).

```
Algorithm Trie::retrieve(Var keys:DArray[String])
Begin
  preorderTraversal(root_, prefix(), keys)
End.
```



```
Algorithm preorderTraversal(n:TrieNode;
                            prefix:String,
                            Var keys:DArray[String])
Var c:Char
Begin
  If n.isKey() Then
    keys.pushBack(prefix)
  End-If
  n.gotoFirstChild()
  While n.currentExist() Do
    preorderTraversal(n.currentChild(),
          prefix+n.currentSymbol(), keys)
    n.gotoNext()
  End-For
End.
```

Trie

- Resumen:
  - Son árboles multi camino donde cada nodo representa una clave y/o un prefijo almacenado.
  - Las ramas del árbol representan prefijos de clave.
  - Todas las claves con un mismo prefijo se almacenan en el mismo subárbol.
  - En un trie con N claves, las operaciones de inserción/búsqueda de una clave son O(M) donde M es la longitud de la clave más larga.

Trie

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