Estructuras de Datos y Algoritmos

Jose Luis Tavera - 201821999
Juan Diego Yepes - 202022391

Laboratorio VIII

1 ¿Qué relación encuentra entre el número de elementos en el árbol y la altura del árbol?

En principio, es evidente que el árbol no está balanceado ya que podemos determinar el número de elementos máximo de un árbol (de un árbol perfectamente balanceado). Con la fórmula:

$$\sum_{n=0}^{n} 2^n$$

Asimismo, para $n = 29$:

$$\sum_{n=0}^{29} 2^{29} > 1177$$

Es por ello que no podemos hallar relación alguna entre el número de elementos en el árbol y su altura, ya que hay demasiadas ramificaciones y combinaciones dentro del mismo.

2 ¿Si tuviera que responder esa misma consulta y la información estuviera en tablas de hash y no en un BST, cree que el tiempo de respuesta sería mayor o menor? ¿Por qué?

Si tuviéramos que hacer esta consulta con tablas de hash, tendríamos que revisar llave por llave y verificar si se encuentra en el rango, lo cual es mucho más costoso en términos de tiempo, ya que si hay n llaves el ordenamiento sería O(n). Precisamente esta es la ventaja que nos otorgan los árboles BST, ya que se encuentran organizados por llaves y encontrar los valores en un rango determinado es una operación O(1).

3 ¿Qué operación del TAD se utiliza para retornar una lista con la información encontrada en un rango de fechas?

```
from DISClib.ADT import orderedmap as om
```

```
def getCrimesByRange(analyzer, initialDate, finalDate):
    """
    Retorna el numero de crimenes en un rango de fechas.
    """
    lst = om.values(analyzer['dateIndex'], initialDate, finalDate)
    totcrimes = 0
    for lstdate in lt.iterator(lst):
        totcrimes += lt.size(lstdate['lstcrimes'])
    return totcrimes
```

La función getCrimesByRange() utiliza la función values() del TAD de orderedmap. Esta a su vez llama a la función values() del datastructure orderedmapstructure. Finalmente, esta ultima función vuelve a referenciar otra función values pero esta se encuentra en el datastructure de BST, la cual se muestra a continuación:

```
def values(bst, keylo, keyhi):
    """
    Retorna todas los valores del arbol que se encuentren entre
    [keylo, keyhi]

    Args:
        bst: La tabla de simbolos
        keylo: limite inferior
        keylohi: limite superiorr
    Returns:
        Las llaves en el rago especificado
    Raises:
        Exception
    """
    try:
        lstvalues = lt.newList('SINGLE_LINKED', bst['cmpfunction'])
        lstvalues = valuesRange(bst['root'], keylo, keyhi, lstvalues,
                                bst['cmpfunction'])
        return lstvalues
    except Exception as exp:
        error.reraise(exp, 'BST:Values')
```

Esta función a su vez referencia a valuesRange(), la cual se muestra a continuación:

```
def valuesRange(root, keylo, keyhi, lstvalues, cmpfunction):
    """
    Retorna todas los valores del arbol en un rango dado por
    [keylo, keyhi]
    Args:
        bst: La tabla de simbolos
        keylo: limite inferior
        keylohi: limite superior
    Returns:
        Las llaves en el rago especificado
    Raises:
        Excep
    """
    try:
        if (root is not None):
            complo = cmpfunction(keylo, root['key'])
            comphi = cmpfunction(keyhi, root['key'])

            if (complo < 0):
                valuesRange(root['left'], keylo, keyhi, lstvalues,
                            cmpfunction)
            if ((complo <= 0) and (comphi >= 0)):
                lt.addLast(lstvalues, root['value'])
            if (comphi > 0):
                valuesRange(root['right'], keylo, keyhi, lstvalues,
                            cmpfunction)
        return lstvalues
    except Exception as exp:
        error.reraise(exp, 'BST:valuesrange')
```

Esta funcion agrega recursivamente los valores que se encuentran en el rango especificado.