```
from algo1 import *
import linkedlist
from myqueue import enqueue, dequeue

class BinaryTree:
   root = None

class BinaryTreeNode:
   key = None
   value = None
   leftnode = None
   rightnode = None
   parent = None


#Buscamos un elemento (.value) en un arbol dado
def search(B, element):
   #Compara el elemento ingresado con el .value de un nodo
   #Funcion recursiva
   def search_node(node, element):
       if node is None:
           return None
       if node.value == element:
           return node.key

       # Buscar en el subárbol izquierdo
       left_result = search_node(node.leftnode, element)
       if left_result is not None:
           return left_result

       # Buscar en el subárbol derecho
       return search_node(node.rightnode, element)

   return search_node(B.root,element)


#Insertamos un elemento en un arbol de busqueda(BST)
def insert(B,element,key):
   new_node = BinaryTreeNode()
   new_node.value = element
   new_node.key = key

   #Caso con root vacio
    if B.root is None:
       B.root = new_node
       return key

   #Funcion recursiva que explora el arbol y devuelve verdadero si se pudo insertar el nodo
   def insert_node(node):
       #Verifica nodo izquierdo
       if new_node.key < node.key:
           if node.leftnode is None:
              node.leftnode = new_node
```

```
            new_node.parent = node
            return True
        else:
            insert_node(node.leftnode)
    elif new_node.key > node.key:
        if node.rightnode is None:
            node.rightnode = new_node
            new_node.parent = node
            return True
        else:
            insert_node(node.rightnode)
    else:
        return None

  #Llamamos a la funcion recursiva
  success = insert_node(B.root)
  if success:
      return key
  else:
      return None


#Funcion para borrar/desvincular un nodo con .key == element
def deleteKey(B,key):

  #Funcion que hace el reemplazo de un nuevo nodo en el lugar del nodo que queremos
reemplazar
  def replace(parent, replace_node, new_child):
      if parent is None:  #Eliminamos la raiz
          B.root = new_child
      elif parent.leftnode == replace_node:
          parent.leftnode = new_child             #Cambiamos el child node
      elif parent.rightnode == replace_node:
          parent.rightnode = new_child            #Cambiamos el child node
      if new_child is not None:
          new_child.parent = parent               #Cambiamos el parent del child node

  def delete_node_key(node):
      #Caso sin child nodes
      if node.leftnode is None and node.rightnode is None:
          replace(node.parent,node,None)
      #Casos 1 childnode
      elif node.leftnode is None:
          replace(node.parent, node, node.rightnode)
      elif node.rightnode is None:
          replace(node.parent, node, node.leftnode)
      #Caso 2 childnodes
      else:
          max_node = node.rightnode
          while max_node.leftnode is not None:
              max_node = max_node.leftnode

          node.key = max_node.key
```

```
            node.value = max_node.value
            delete_node_key(max_node)
            #En esta parte hacemos lo siguiente:
            #Ubicamos "el menor de los mayores"
            #Pasamos su key al nodo actual
            #Eliminamos el nodo que ocupa la posicion de "El menor de los mayores"

    #Buscamos el elemento que queremos borrar por su key
    def find_node_key(node,key):
        if node is None:
            return None
        if key < node.key:
            return find_node_key(node.leftnode, key)
        elif key > node.key:
            return find_node_key(node.rightnode, key)
        else:
            return node

    #Buscamos el elemento a borrar y si logramos borrarlo devolvemos su key
    target = find_node_key(B.root,key)

    if target is not None:
        delete_node_key(target)
        return key
    else:
        return None

#Funcion para borrar/desvincular un nodo con .value == element
def delete(B,element):
    #Buscamos la key del elemento a borrar
    key_to_find = search(B,element)
    if key_to_find is None:
        return None

    #Usamos la funcion deleteKey para borrar ese elemento
    return deleteKey(B,key_to_find)

#Nos permite acceder a un nodo del bt (ingresando su key devuelve su value)
def access(B,key):

    def search_key(node):
        if node is None:
            return None
        if (key == node.key):
            return node.value
        elif(key < node.key):
            return search_key(node.leftnode)
        else:
            return search_key(node.rightnode)

    result = search_key(B.root)
    if result is not None:
        return result
```

```
    else:
        return None

#Cambia el value de un nodo del bt
def update(B, element, key):

    def search_node_key(node):
        if node is None:
            return None
        if node.key == key:
            node.value = element
            return key
        elif key < node.key:
            return search_node_key(node.leftnode)
        else:
            return search_node_key(node.rightnode)

    result = search_node_key(B.root, element, key)
    if result is not None:
        return result
    else:
        return None

#Recorremos el bt en orden -> De izq a derecha (raiz en el medio)
def traverseInOrder(B):
    if B.root is None:
        return None

    R = linkedlist.LinkedList()

    def tio_r(node,index):
        if node is None:
            return index
        index = tio_r(node.leftnode, index)
        linkedlist.insert(R, node.key, index)
        index += 1
        index = tio_r(node.rightnode, index)
        return index

    tio_r(B.root, 0)
    return R

#Recorremos el bt en post orden -> raiz al final
def traverseInPostOrder(B):
    if B.root is None:
        return None

    R = linkedlist.LinkedList()

    def tiPostO_r(node,index):
        if node is None:
            return index
        index = tiPostO_r(node.leftnode, index)
```

```
        index = tiPostO_r(node.rightnode, index)
        linkedlist.insert(R, node.key, index)
        index  += 1
        return index


    tiPostO_r(B.root, 0)
    return R

#Recorremos el bt en pre orden -> raiz al principio
def traverseInPreOrder(B):
    if B.root is None:
        return None

    R = linkedlist.LinkedList()

    def tiPreO_r(node,index):
        if node is None:
            return index
        linkedlist.insert(R, node.key, index)
        index  += 1
        index = tiPreO_r(node.leftnode, index)
        index = tiPreO_r(node.rightnode, index)

        return index

    tiPreO_r(B.root, 0)
    return R

#Recorremos el bt con amplitud
def traverseBreadFirst(B):
    if B.root is None:
        return None

    R = linkedlist.LinkedList()
    Q = linkedlist.LinkedList()

    def dequeueBT(Q):
        current = Q.head
        if (current == None):
            return None
        elif (current.nextNode == None):
            Q.head = current.nextNode
            return current
        else:
            while (current.nextNode.nextNode != None):
                current = current.nextNode
            exitNode = current.nextNode
            current.nextNode = current.nextNode.nextNode
            return exitNode

    enqueue(Q, B.root)
    index = 0
```

```
    while(Q.head)is not None:
        current = dequeueBT(Q)
        node = current.value
        linkedlist.insert(R, node.key, index)
        index += 1

        if node.leftnode is not None:
            enqueue(Q, node.leftnode)
        if node.rightnode is not None:
            enqueue(Q, node.rightnode)

    return R
```

SALIDA UNITTEST

```
Insertar un elemento en un arbol y verificar su posicion ... ok
test_insert_first_element (__main__.TestBinaryTree.test_insert_first_element)
Insertar un solo elemento en un arbol vacio y verificar su valor ... ok
test_search (__main__.TestBinaryTree.test_search)
Busqueda de un elemento en un arbol y verificar su posicion y contenido ... ok
test_search_inexistent (__main__.TestBinaryTree.test_search_inexistent)
Busqueda de un elemento inexistente en un arbol y verificar None ... ok
test_traverse_bfs (__main__.TestBinaryTree.test_traverse_bfs)
Recorre un arbol en modo BFS ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.001s

OK
```