Question 1, Morning lab

The task is to implement a Binary Search Tree of strings, supported by few operations.

The basic structure to be used is the following:

```
typedef struct _BST {
      char name[100];
      struct _BST* left;
      struct _BST* right;
      struct _BST* parent; //This is optional to include. Upto you.
} BST;
```

The functions to be supported are:

- **InitTree (char* name):** Creates a tree, and returns the pointer to the tree.
- **Insert (char *name, BST** T):** Inserts a node in the tree T. If the tree is empty (T is NULL, this might happen on enough deletions), then your function should internally call InitTree (name).
- **Delete (char* name, BST **T):** Deletes the node with the given name if it is present, or prints "Not Found" if tree is empty or if node is not present.
- **Find (char *name, BST *T):** Returns whether a node with the given name is present in T or not. Print it's **InOrder position** if present, or print 'Not Found' otherwise.
- **Height (BST* T):** Computes the height of T
- **PrintTree (BST* T, int order):** Prints the nodes in the tree in in-order manner if order = 0, pre-order manner if order = -1, and post-order(1) otherwise (Strings of nodes to be printed space-separated)

Input/Output
The first line of the input contains T, the number of tree operations to be performed.
Each of the Following T lines will contain one of the following:

-> InitTree -
Followed by a string (first string to initiate the tree) on the same line separated by a space. Print Nothing. InitTree would only be given as input as the first test case, and not anytime again.
-> Insert -
Followed by a string to be added to its appropriate position.
Print it's **InOrder Position**.
-> Delete -
Followed by a string whose corresponding node needs to be deleted in an inOrder fashion.

Print "Not Found" if no such element exists or if the tree is empty. If found, then print its **InOrder Position** and delete it.

-> Find -

Followed by a string whose corresponding node is to be searched in an inOrder Fashion. Print it's **InOrder Position** if node with given string exists in tree, or print 'Not Found' otherwise.

-> Height -

Prints the height/depth of the tree.
(Note that height of an empty tree is 0, while that of a tree containing just the root is 1)

-> PrintTree -

Followed by an integer i.
Prints the tree elements in a PREORDER single-space separated fashion if **i = -1**
Prints the tree elements in an INORDER single-space separated fashion. if **i = 0**
Prints the tree elements in a POSTORDER single-space separated fashion. if **i = 1**
If the tree is empty, blank line would be printed.

Note: InOrder Position refers to the index of the node when traversed in the inorder traversal (index starts from 1 and not 0).

Constraints

$1 \leq$ Length(String) ≤ 100
$1 \leq$ Number of Nodes $\leq$ 10^4

Sample Input

```
13
InitTree f
Insert a
Insert b
Delete g
Find m
Find b
Insert g
Insert m
Insert k
Height
PrintTree -1
PrintTree 0
PrintTree 1
```

Sample Output:

```
1
2
Not Found
```

Not Found
2
4
5
5
4
f a b g m k
a b f g k m
b a k m g f

Explanation:
#1 Initiates the Tree with a string 'f'
#2 Inserts a new String 'a'. Tree looks like:

```
    f
   /
  a
```

#3 Inserts a new String 'b'. Tree looks like:

```
    f
   /
  a
   \
    b
```

#4 Cannot find 'g'
#5 Cannot find 'm'
#6 InOrder of Tree Looks like: a b f. Finds index of b.
#7 Inserts a new String 'g'. Tree looks like

```
    f
   /  \
  a    g
   \
    b
```

#8 Inserts a new String 'm'. Tree looks like

```
    f
   /  \
  a    g
   \    \
    b    m
```

#9 Inserts a new String 'k'. Tree looks like

```
    f
   /  \
  a    g
   \    \
    b    m
```

/
k

#10 Height of tree is: 4
#11 PreOrder: f a b g m k
#12 InOrder: a b f g k m
#13 PostOrder: b a k m f g

Assumptions:
First operation will always be initTree.