DS WEEK-09

Start here ✕ *BST.c ✕

```c
#include <stdio.h>
#include <stdlib.h>
// Define the structure for a node
struct Node {
    int data;
    struct Node *left, *right;
};
// Function to create a new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->left = newNode->right = NULL;
    return newNode;
}
// Function to insert a node into the BST
struct Node* insert(struct Node* root, int value) {
    if (root == NULL) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
// In-order traversal (Left, Root, Right)
void inorder(struct Node* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
// Pre-order traversal (Root, Left, Right)
void preorder(struct Node* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
// Post-order traversal (Left, Right, Root)
void postorder(struct Node* root) {
    if (root != NULL) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
```

C:\Users\BMSCE\Documents\

```
--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 50

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 70

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 60

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 20

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 90
```

```
--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 10

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 40

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 1
Enter value to insert: 100
```

```
--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 2
In-order Traversal: 10 20 40 50 60 70 90 100

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 3
Pre-order Traversal: 50 20 10 40 70 60 90 100

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 4
Post-order Traversal: 10 40 20 60 100 90 70 50

--- Binary Search Tree Menu ---
1. Insert element
2. Display In-order traversal
3. Display Pre-order traversal
4. Display Post-order traversal
5. Exit
Enter your choice: 5

Process returned 0 (0x0)   execution time : 122.012 s
Press any key to continue.
```

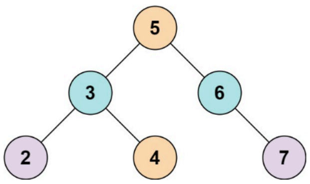
Binary Search Tree

Submit

Description | Editorial | Solutions | Submissions

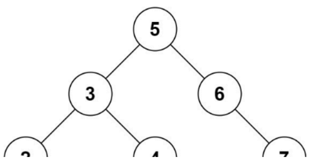
Given the `root` of a binary search tree and an integer `k`, return `true` *if there exist two elements in the BST such that their sum is equal to* `k`, *or* `false` *otherwise.*

Example 1:



Input: root = [5,3,6,2,4,null,7], k = 9
Output: true

Example 2:



Code

C Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

#include <stdbool.h>
#include <stdlib.h>

#define OFFSET 10000
#define SIZE 20001

bool visited[SIZE];

bool dfs(struct TreeNode* root, int k) {
    if (!root) return false;

    if (dfs(root->left, k)) return true;

    int complement = k - root->val;
    if (visited[complement + OFFSET]) return true;
    visited[root->val + OFFSET] = true;

    return dfs(root->right, k);
}

bool findTarget(struct TreeNode* root, int k) {
    for (int i = 0; i < SIZE; i++) visited[i] = false;
    return dfs(root, k);
}
```

Saved

Code

C ⌄ Auto

```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */

#include <stdbool.h>
#include <stdlib.h>

#define OFFSET 10000
#define SIZE 20001

bool visited[SIZE];

bool dfs(struct TreeNode* root, int k) {
    if (!root) return false;

    if (dfs(root->left, k)) return true;

    int complement = k - root->val;
    if (visited[complement + OFFSET]) return true;
    visited[root->val + OFFSET] = true;

    return dfs(root->right, k);
}

bool findTarget(struct TreeNode* root, int k) {
    for (int i = 0; i < SIZE; i++) visited[i] = false;
    return dfs(root, k);
}
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =

```
[5,3,6,2,4,null,7]
```

k =

```
28
```

Output

```
false
```

Expected

```
false
```

Testcase | >_ Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =

```
[5,3,6,2,4,null,7]
```

k =

```
9
```

Output

```
true
```

Expected

```
true
```