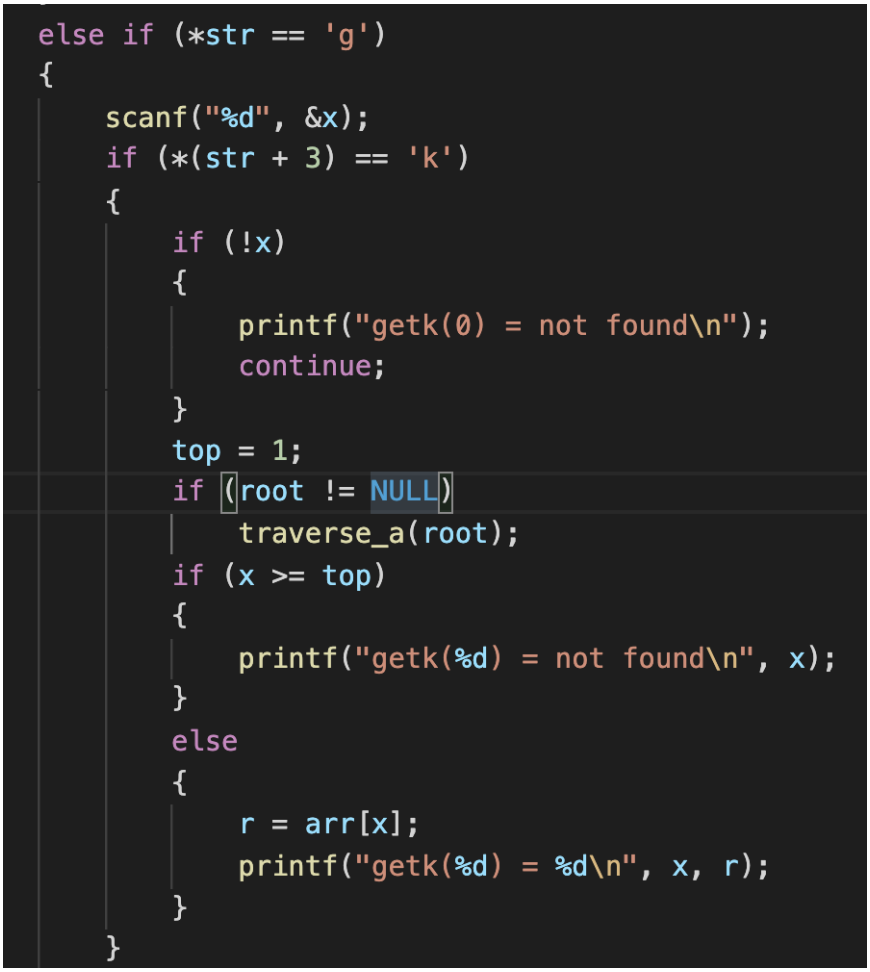
25 btree

worst-case time complexity of `getk` inplemetation is `O(n)`

因為如果getk到最後一個也就是最大的就會是O(n)，因為會把所有node traverse 一次



```
else if (*str == 'g')
{
    scanf("%d", &x);
    if (*(str + 3) == 'k')
    {
        if (!x)
        {
            printf("getk(0) = not found\n");
            continue;
        }
        top = 1;
        if (root != NULL)
            traverse_a(root);
        if (x >= top)
        {
            printf("getk(%d) = not found\n", x);
        }
        else
        {
            r = arr[x];
            printf("getk(%d) = %d\n", x, r);
        }
    }
```

worst-case time complexity of `removek` inplemetation is `O(n)`

因為如果removek到最後一個也就是最大的就會是O(n)，因為會把所有node traverse 一次，找到最大的node才能刪除，而delete node的時間也不會的時間也不會超過log n 因此 worst time complexity 也會是O(n)

```
void del_node(BTreeNode *node, int k)
{
    int idx = findKey(node, k);
    // The key to be removed is present in this node
    if (idx < node->n && node->keys[idx] == k)
    {
        // If the node is a leaf node - removeFromLeaf is called
        // Otherwise, removeFromNonLeaf function is called
        if (node->leaf)
            del_nodeFromLeaf(node, idx);
        else
            del_nodeFromNonLeaf(node, idx);
    }
    else
    {

        // If this node is a leaf node, then the key is not present in tree
        if (node->leaf)
        {
            printf("The key %d is does not exist in the tree\n", k);
            return;
        }
```

```
else
{

    // If this node is a leaf node, then the key is not present in tree
    if (node->leaf)
    {
        printf("The key %d is does not exist in the tree\n", k);
        return;
    }

    // The key to be removed is present in the sub-tree rooted with this node
    // The flag indicates whether the key is present in the sub-tree rooted
    // with the last child of this node
    int flag = ((idx == node->n) ? 1 : 0);

    // If the child where the key is supposed to exist has less that t keys,
    // we fill that child
    if (node->C[idx]->n < t)
        fill(node, idx);

    // If the last child has been merged, it must have merged with the previous
    // child and so we recurse on the (idx-1)th child. Else, we recurse on the
    // (idx)th child which now has atleast t keys
    if (flag && idx > node->n)
        del_node(node->C[idx - 1], k);
    else
        del_node(node->C[idx], k);
}
return;
```