Lecture 14: Binary Search Trees 3

COSC242: Algorithms and Data Structures

Brendan McCane

Department of Computer Science, University of Otago

Delete

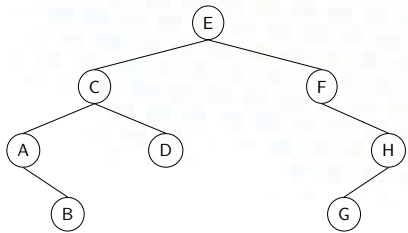
The one key operation we have not yet discussed is delete. It's a bit more complicated than the others.

We'll need a few helper operations first:

- minimum, maximum
- predecessor, successor

Minimum and maximum

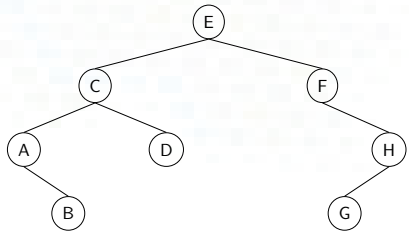
```
1: function BST_FIND_MIN(T)
2:     if T == NIL then
3:         return 'Not Found'
4:     else if T→left == NIL then
5:         return T→key
6:     else
7:         return BST_find_min(T→left)
8:     end if
9: end function
```



BST_find_max left as an exercise.

Predecessor and successor

```
1:  function BST_SUCCESSOR(T)
2:      if T→right ≠ NIL then
3:          return BST_find_min(T→right)
4:      else
5:          parent = T→parent
6:          while parent ≠ NIL and T→key>parent→key do
7:              parent = parent→parent
8:          end while
9:          return parent
10:     end if
11: end function
```

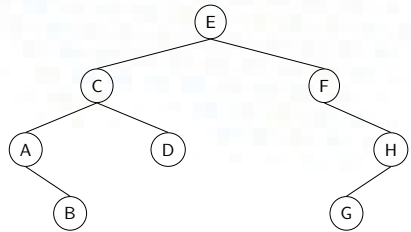


BST_predecessor is left as an exercise.

Delete

Let's consider our example BST again.
What do we do if we want to:

1. delete a node with no children? E.g. D?
2. delete a node with one child? E.g. A?
3. delete a node with two children? E.g. C? or E?



Delete

In the labs, you will develop a delete based on key values (which basically incorporates a search into delete). Here we are going to assume the node to delete is already found.

```
1:  procedure BST_DELETE(BST T)
2:      if T→left==NIL and T→right==NIL then
3:          T→parent→[left or right] = NIL
4:          delete T
5:      else if T has one child then /* splice out T */
6:          T→parent→[left or right] = T→[left or right]
7:          delete T
8:      else if T has two children then
9:          BST_replace_with_successor(T)
10:     end if
11: end procedure
```


Splicing out

When T has two children, then we can replace T (or rather its contents) by its successor (or predecessor), and recursively delete the successor:

```
1: procedure BST_REPLACE_WITH_SUCCESSOR(BST T)
2:     successor = BST_successor(T)
3:     successor_key = successor→key
4:     BST_delete(successor)
5:     T→key = successor_key
6: end procedure
```

We are going to prove that BST_delete always terminates ...

Lemma

If T has two children, then T's successor must be a right descendant of T.

Proof

If T has two children then T's successor must be a descendant in the right subtree of T:

- if T is a right descendant of some node, A, then T is greater than A, and therefore A cannot be a successor;
- if T is a left descendant of some node, A, then T's right descendants lie between T and A and therefore A cannot be a successor;
- therefore T's successor must be a descendant of T;
- all of T's left descendants are less than T, therefore T's successor must be a right descendant. □

Lemma

If T has two children, then T's successor has no left child.

Proof

- From the lemma in the previous slide, we know that T's successor is a right descendant.
- All of T's right descendants are greater than T.
- By definition, T's successor is the smallest value in T's right subtree.
- Let's assume that T's successor has a left child. If the successor has a left child, then that child's value is smaller than the successor, but since it is in T's right subtree, it must be greater than T. In which case we have found a node that is greater than T, but smaller than the successor, which is a contradiction.
- Therefore we conclude that T's successor has no left child. □

Theorem

The algorithm BST_delete will always terminate.

Proof

- The only loop in BST_delete is via a call to BST_replace_with_successor which subsequently calls BST_delete with T's successor (call T's successor S).
- BST_replace_with_successor is only ever called when T has two children.
- S has at most one child by the lemma on the previous page.
- Therefore, when BST_delete is called with S, one of the first two branches of the if statement are taken and neither of them include a loop.
- Therefore BST_delete will always terminate. □

Relevant parts of the textbook

Iterative versions of minimum, maximum, predecessor and successor are given in section 12.2

Delete is discussed in section 12.3, and although similar in spirit, is quite different to the delete algorithm discussed here.

The two lemmas and theorems at the end of the lecture are not discussed in the textbook or anywhere that I am aware of. The particular knowledge is not that important, but the idea of proving something about an algorithm is. It is also important to note that not all proofs contain equations!