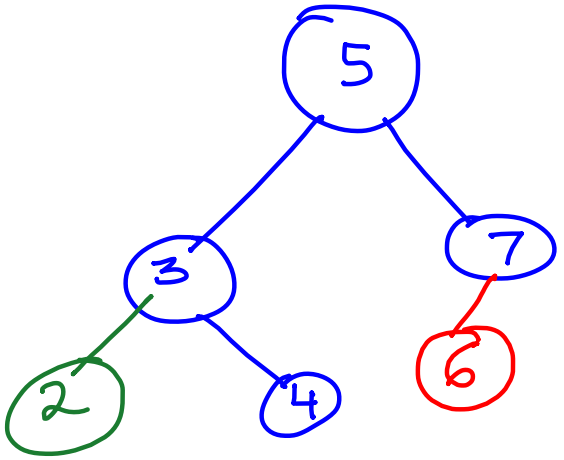
Insertion and Deletion Operations in BST

Example



T

T_1

Insert(T, 2)

$2 < 5$

$3 < 5$

Insert(T_1, 6)

$6 > 5$

$6 < 7$

Pseudo Code

Succ(x) — x is a node in the tree

min(x) — x is a node in the tree. Finds min value in T_x

Notation: For x, a node in T, let T_x stand for tree rooted at x.

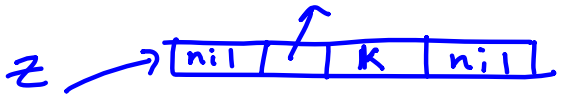
functions like succ, min etc. return a value

Insert/Delete do not return any value, just modify the tree

In these operations T is part of the input

We represent tree as a record (or tuple) one of whose field is root T.root

```
Insert(T, z)
if (T.root == nil)
    T.root = z
    z.p = nil
    return
else Insert1(T.root, z)
```




```
Insert1(x, z)   // x is a node in T, z is to be inserted in T_x
if (z.key ≤ x.key)
    and (x.left == nil)
    x.left = z
    (x.left).p = x
    return
if (z.key ≤ x.key) and (x.left ≠ nil)
    Insert1(x.left, z)
    return
if (z.key > x.key) and (x.right == nil)
    x.right = z
    (x.right).p = x
    return
if (z.key > x.key) and (x.right ≠ nil)
    Insert1(x.right, z)
    return
```

Correctnen of Insert1(x, z)

By induction on no. of nodes in T_x.

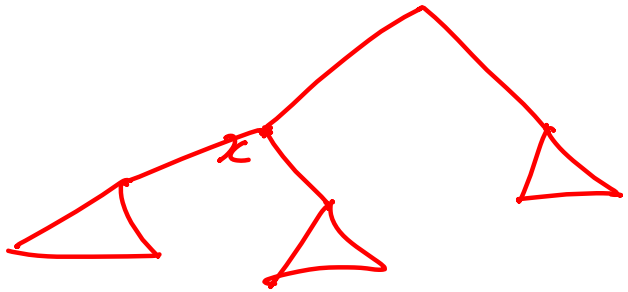
Time Complexity $O(h_x)$

Can be proved by substitution method.

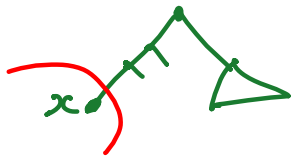
Exercise Convert insertI(x,z) into iterative procedure.

Delete(T, x) / x is a node in the tree

Returns a BST which does not contain x, but contains all other nodes of T.

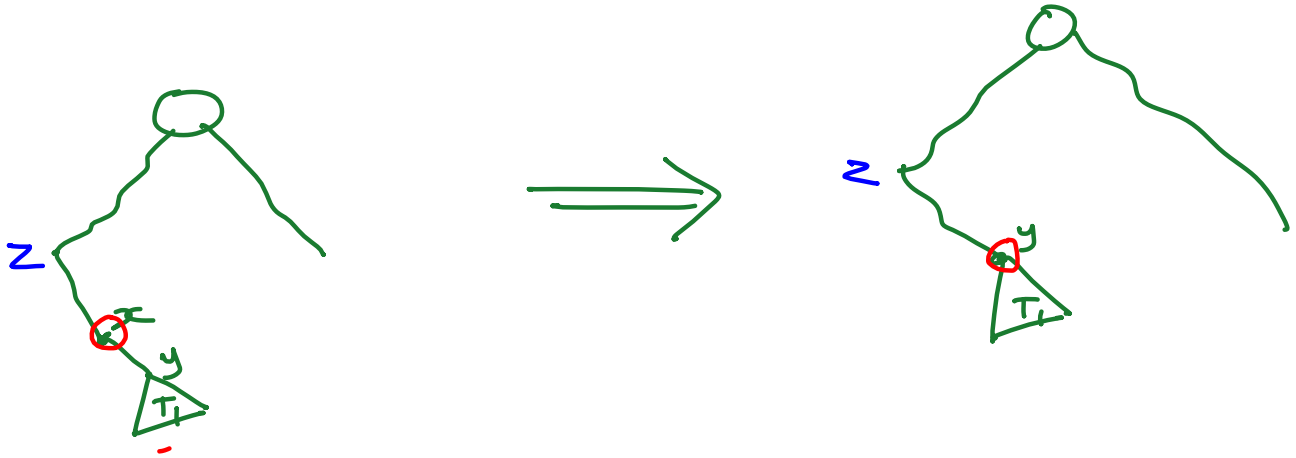


Easy cases

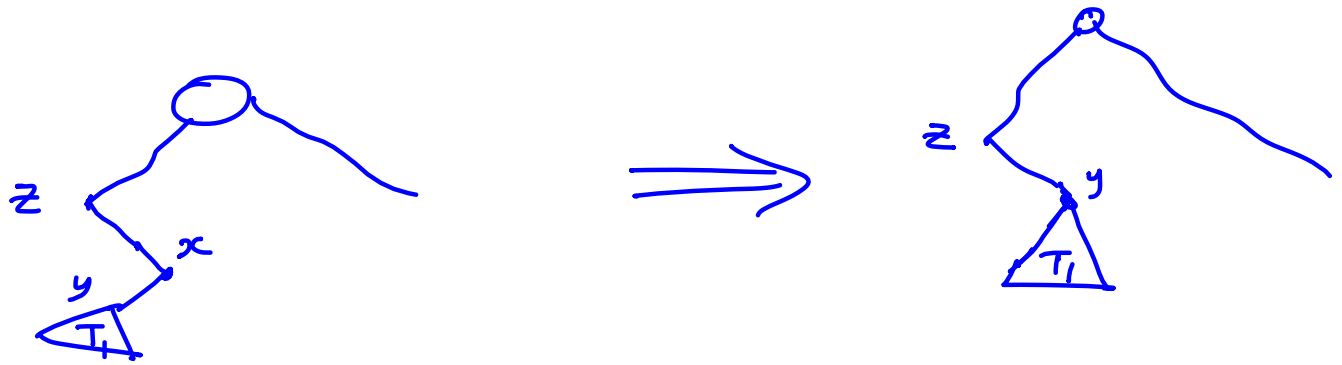


very easy

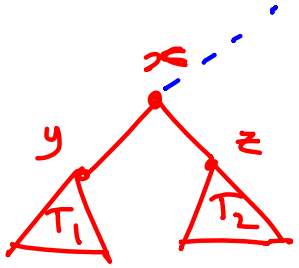
Case 2 x has exactly one of the left/right subtree which is non-empty.

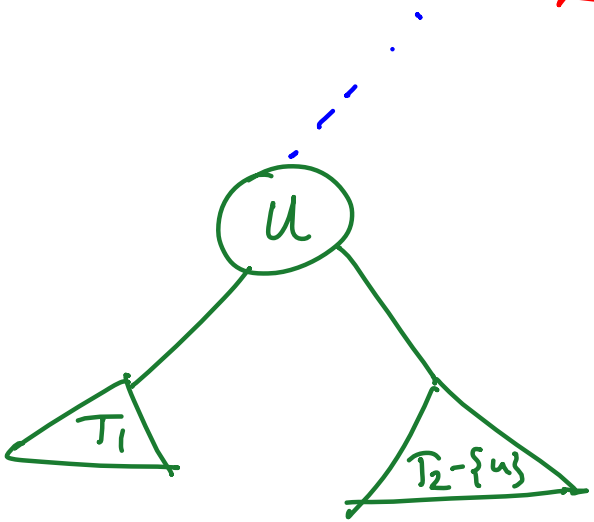


Case 2'



Case 3





Minimum element in T_2 = u

$T_1 \leq u$

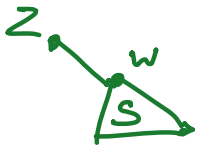
$T_2 - \{u\} \geq u$

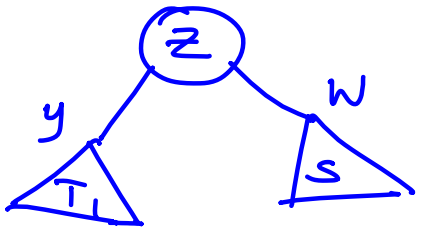
= delete(T_2, u)

How to compute delete(T_2, u)

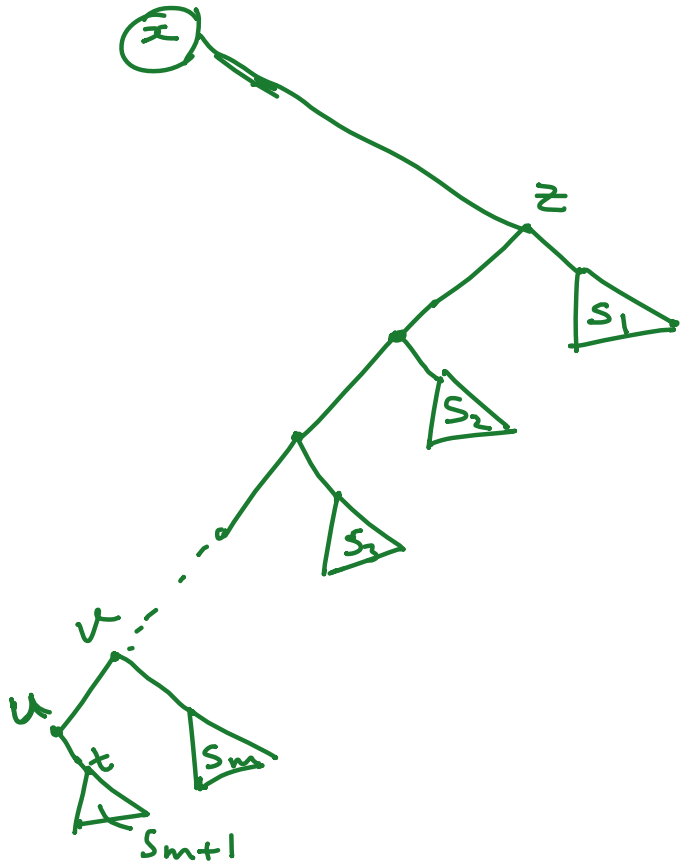
Easy case — z has no left child

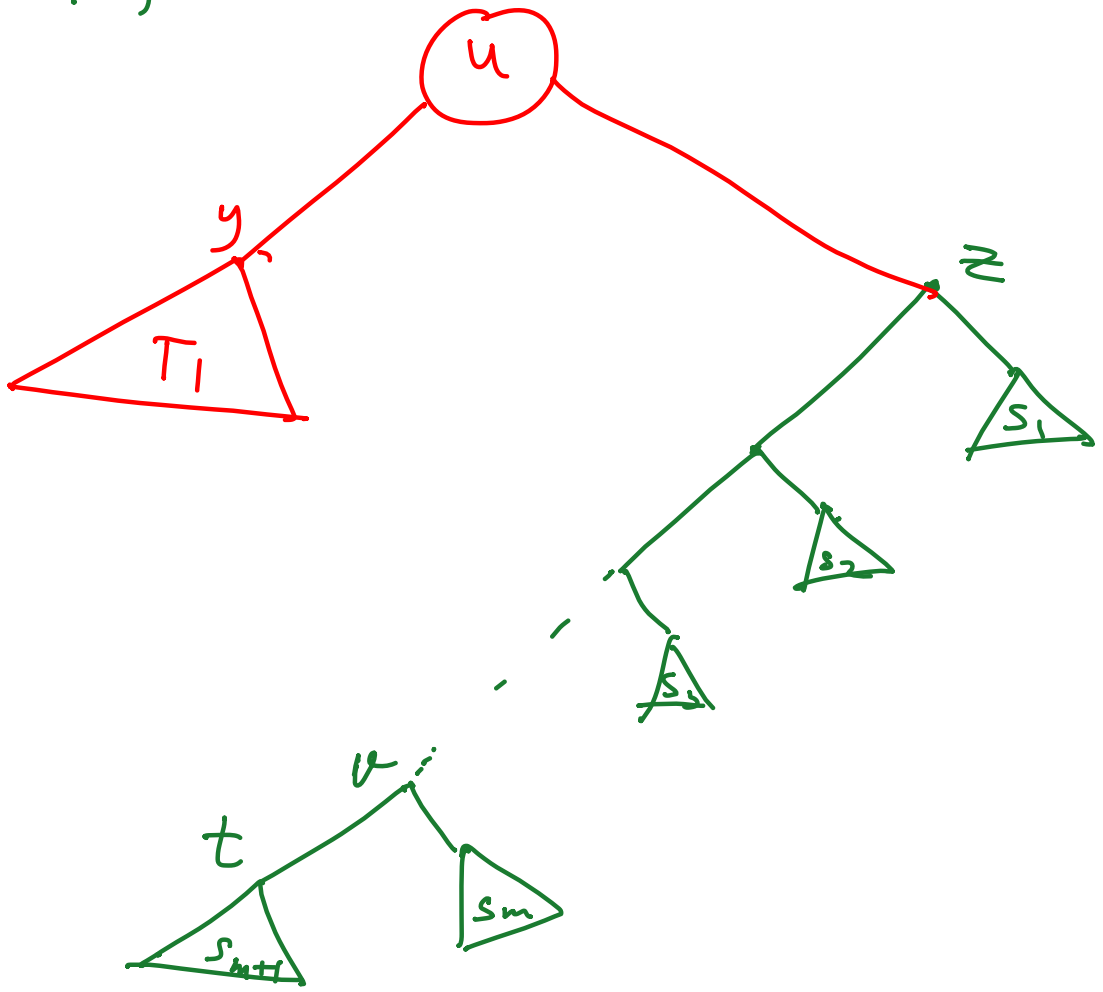
$u = z$





General Case — z has a left child

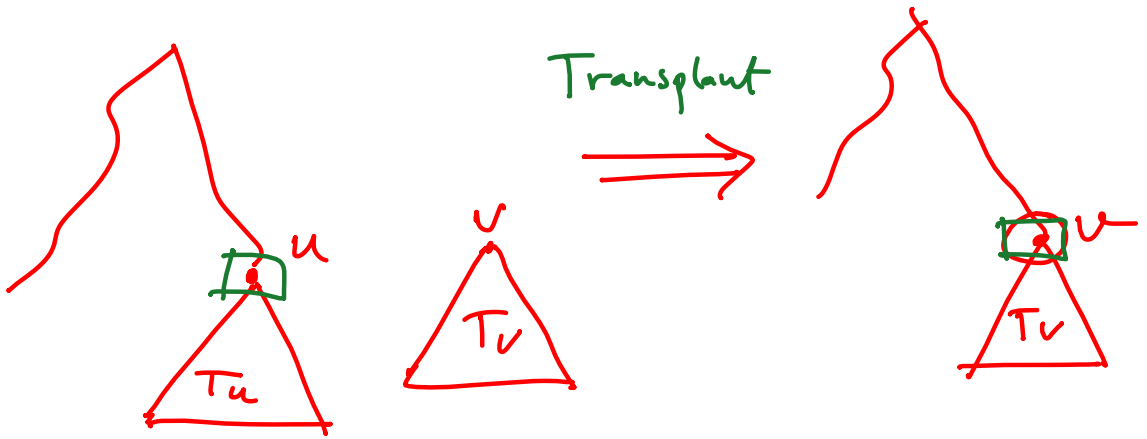




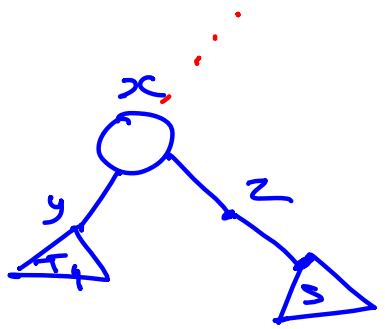
Pseudocode for delete(T, x)

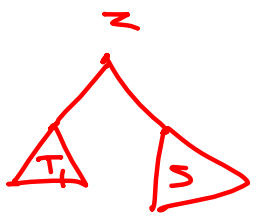
It is helpful to write an auxiliary procedure to replace a tree in T by another tree

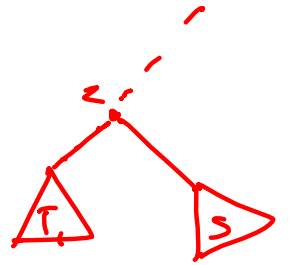
```
Transplant(T, u, v)   // u is a node in T. Replace Tu by Tv.
                      // v may be nil also.
if (u.p == nil)
        T.root = v
        v.p = nil
        return
if (u == (u.p).left)
        (u.p).left = v
else (u.p).right = v
if v ≠ nil
        v.p = u.p
```



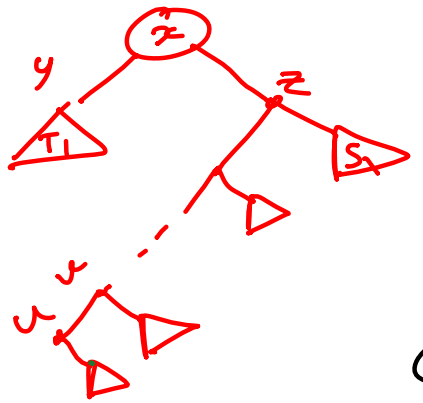
```
Delete(T, x)
  if(x.left == nil)
      Transplant(T, x, x.right)
       return
  if(x.right == nil)
       Transplant(T, x, x.left)
        return
  if (x.right).left == nil
       Transplant(T, x, x.right)
       z.left = x.left
       (z.left).p = z
        return
  u = min(x.right)
  Transplant(T, u, u.right)
  (u.right) = x.right
  (u.right).p = u
  Transplant(T, x, u)
  u.left = x.left
  (u.left).p = u
```







u

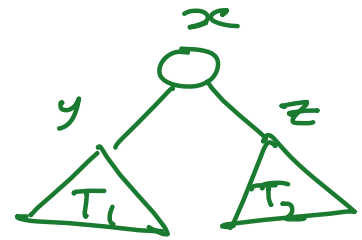


Correctness

Consider various cases and prove correct in each case.

[This is exactly what we have done before writing pseudocode]

Time Complexity $O(h)$ h = height of bst T



Exercise In case 3 of delete operation we may take maximum node in T_1 and consider partition of $T_1 \cup T_2$ wrt that node. Give algorithm and pseudo code with this scheme.