CMPT307: Binary Search Trees

Week 2-3

Xian Qiu

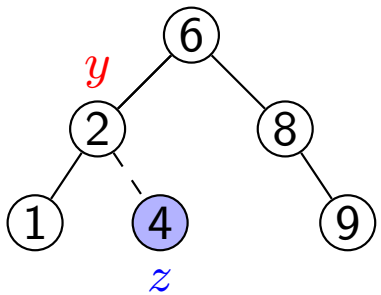
Simon Fraser University

xianq@sfu.ca

Insertion

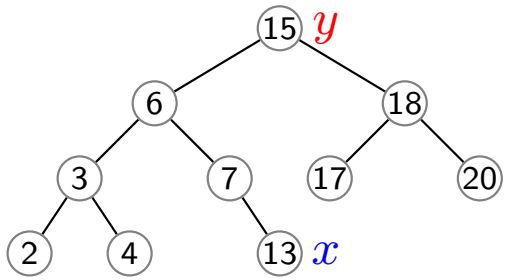
TREE-INSERT(T, z)

```
1  y = nil ;            // to record the parent of z
2  x = T.root;
3  while x ≠ nil do
4      y = x;
5      if z.key < x.key then
6          x = x.left;
7      else x = x.right;
8  z.p = y;             // do not forget z's parent
9  if y == nil then
10     T.root = z ;                  // T was empty
11 else if z.key < y.key then
12     y.left = z;
13 else
14     y.right = z;
```



Successor

successor of x: node of smallest key $\geq x$.key



case 1: 15, 17 (y inserted after x)

case 2: 13, 15 (y inserted before x)

TREE-SUCCESSOR(x, k)

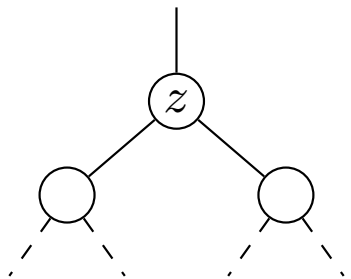
```
1 if x.right ≠ nil then
2     return TREE-MINIMUM(x.right);                    // case 1
3 y = y.p;                                             // case 2
4 while x ≠ nil and x ≠ y.left do
5     x = y;
6     y = y.p;
7 return y;
```

predecessor of x: node of largest key $\leq x$.key

Deletion

to delete z from T:

- z has no children
- z has only one child
- z has two children



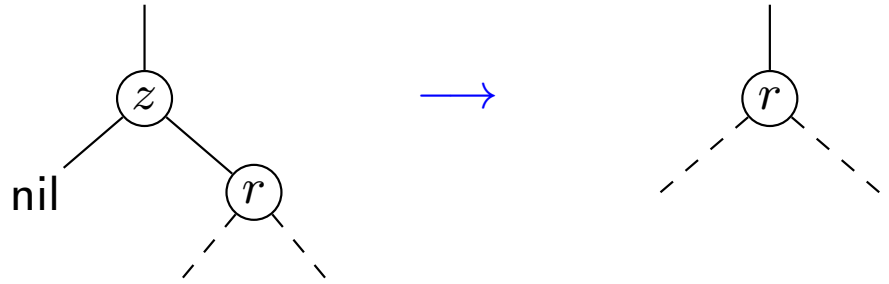
observation: z's successor $y =$ TREE-MINIMUM(z.right)

$\Rightarrow y$.left = nil

Deletion

(a) no left child

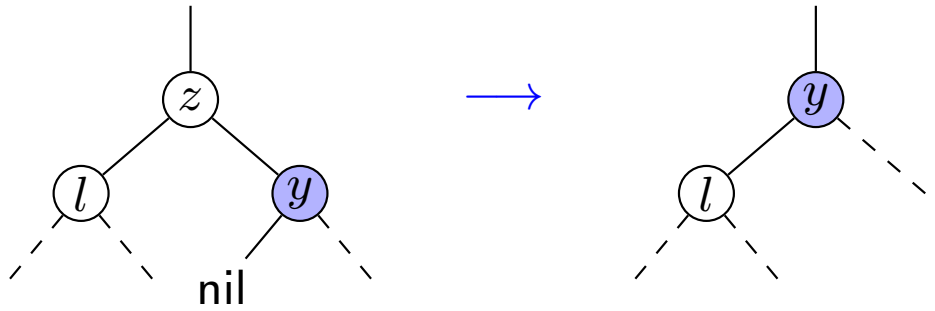
(b) only left child



(c) two children[1]

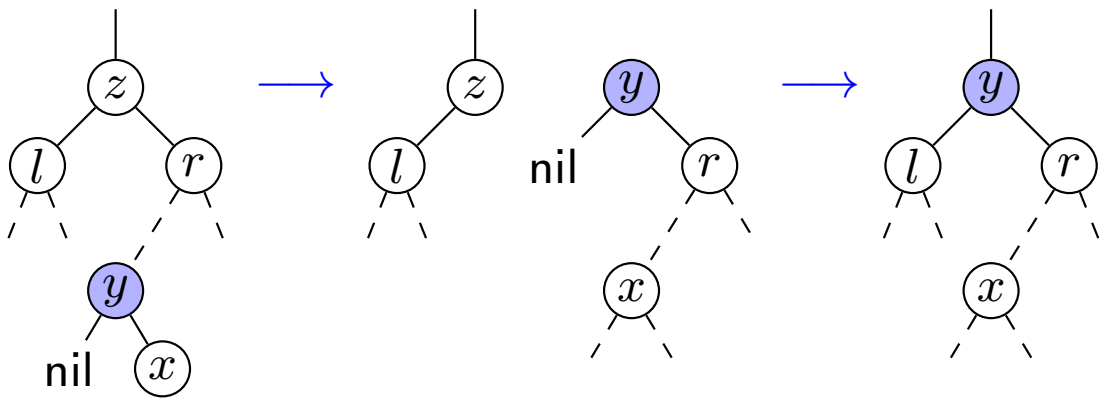
y is z's child

$y = z$'s successor



(d) two children[2]

y is not z's child



Transplant

replace u with the subtree rooted at v:

- u's parent becomes v's parent
- u's parent has v as its appropriate child

TRANSPLANT(T, u, v)

```
1 if u.p == nil then
2     T.root = v;
3 else if u == u.p.left then
4     u.p.left = v;
5 else
6     u.p.right = v;
7 if v ≠ nil then
8     v.p = u.p;
```

Implementation of Delete

TREE-DELETE(T, z)

```
1  if z.left == nil then
2      TRANSPLANT(T, z, z.right);                          // case (a)
3  else if z.right == nil then
4      TRANSPLANT(T, z, z.left);                           // case (b)
5  else
6      y = TREE-MINIMUM(z.right);                  // find z's successor
7      if y.p ≠ z then
8          TRANSPLANT(T, y, y.right);                      // case (d)
9          y.right = z.right;
10         y.right.p = y;                  // y.right.p was pointed to z
11     TRANSPLANT(T, z, y);
12     y.left = z.left;
13     y.left.p = y;
```

Running Time

Theorem

Each of the dynamic-set operations

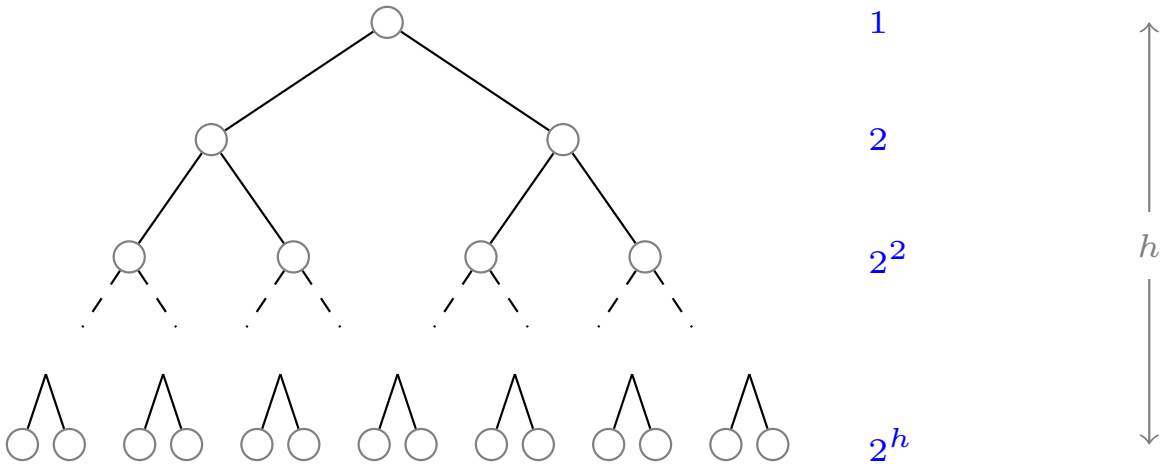
- ▷ Search
- ▷ Minimum, Maximum
- ▷ Successor, Predecessor
- ▷ Insert, Delete

on a binary search tree runs in $O(h)$, where h is the tree height.

how large can h be?

We Expect ...

a balanced binary tree



$$n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1 \quad \Rightarrow h = \Theta(\log n)$$

idea: to maintain an approximately balanced binary search tree