Binary Search Trees

CIS2520, F11

Lab 8

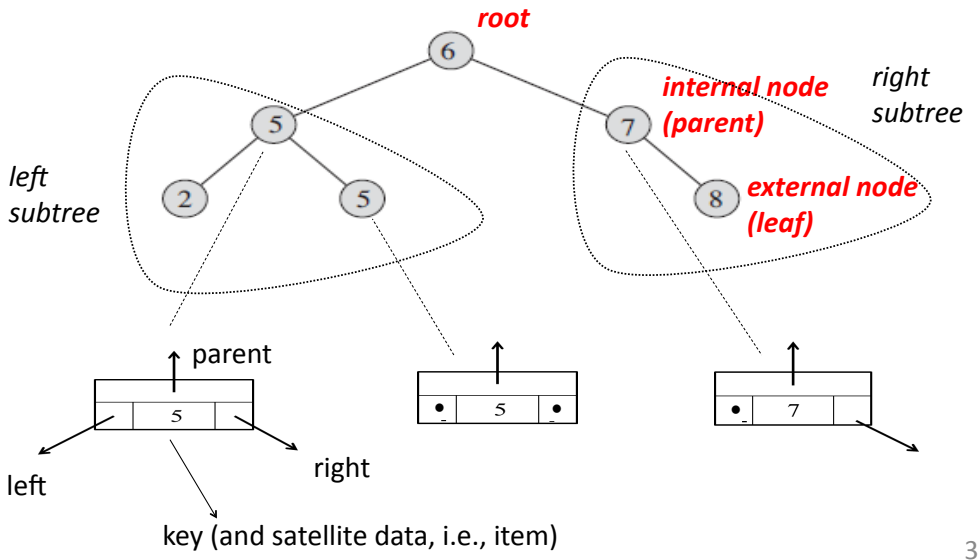
Mohammad Naeem

mnaeem@uoguelph.ca

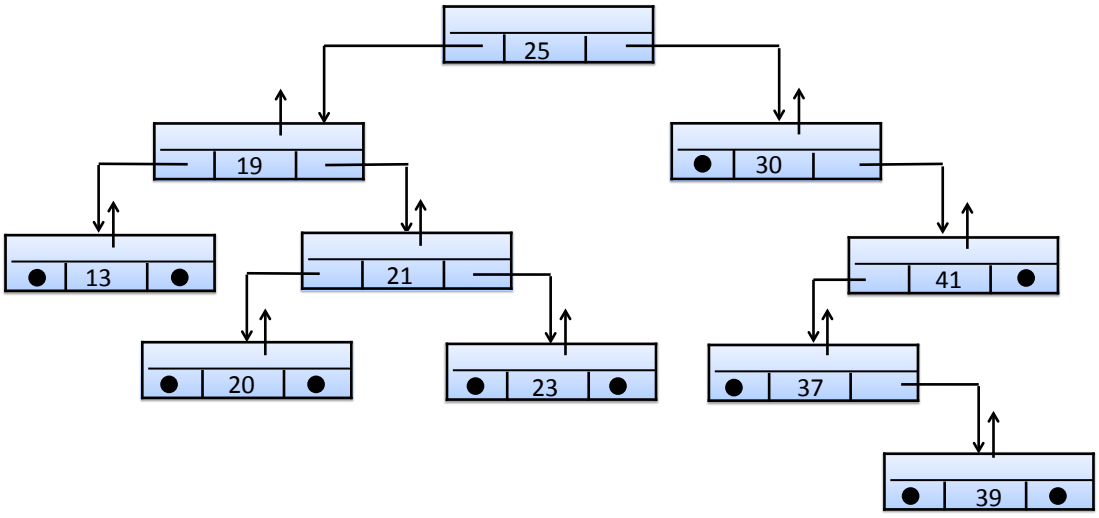
Topics

- Binary Search Trees
  - introduction
  - inorder traversal
  - search
  - min/max
  - successor/predecessor
  - insertion
  - removal

Introduction



Introduction



Inorder Traversal

```
function Inorder (N)
   if N.left≠nil then Inorder(N.left)
   visit N
   if N.right≠nil then Inorder(N.right)
```

O(n) with n number of nodes

Search

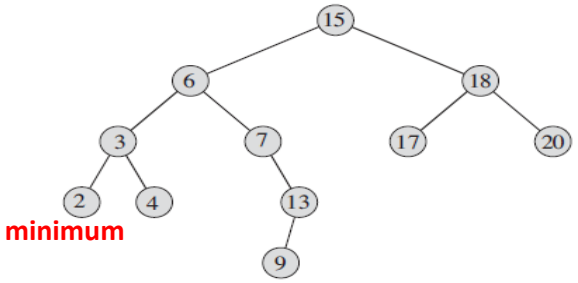
```
function Search (N, key)
   if key<N.key and N.left≠nil
      return Search(N.left,key)
   if key=N.key
      return N
   if key>N.key and N.right≠nil
      return Search(N.right,key)
   return nil
```

O(h) with h=height

Minimum

```
function Minimum (N)
   while N.left≠nil
      N=N.left
   return N
```

O(h) with h=height

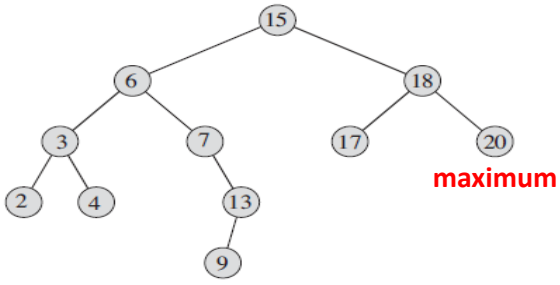


starting from root
follow N.left to leaf

Maximum

```
function Maximum (N)
   while N.right≠nil
      N=N.right
   return N
```

O(h) with h=height



starting from root
follow N.right to leaf

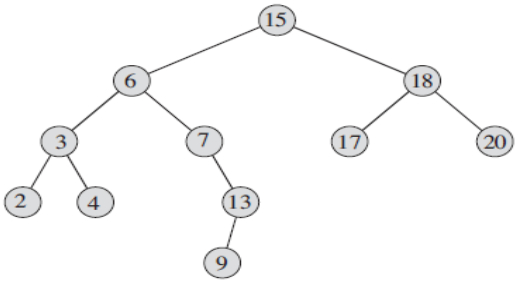
Successor

```
function Successor (N)
   if N.right≠nil
      return Minimum(N.right)
   P=N.parent
   while P≠nil and N=P.right
      N=P
      P=P.parent
   return P
```

case 1

case 2

O(h) with h=height



15 has successor 17 **(case 1)**
13 has successor 15 **(case 2)**

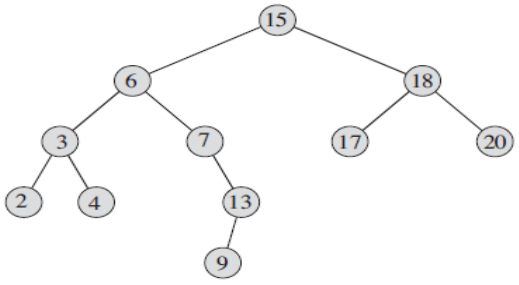
Predecessor

```
function Predecessor (N)
   if N.left≠nil
      return Maximum(N.left)
   P=N.parent
   while P≠nil and N=P.left
      N=P
      P=P.parent
   return P
```

case 1

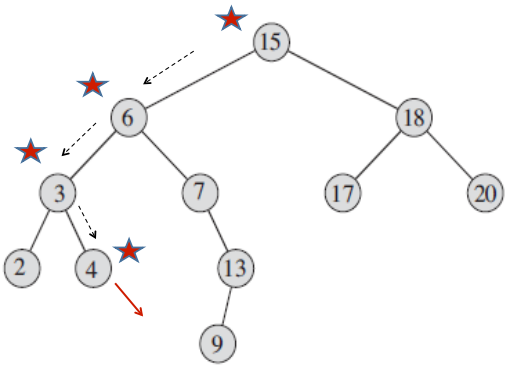
case 2

O(h) with h=height

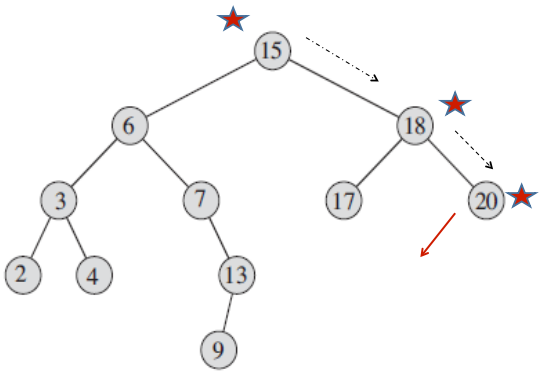


6 has successor 4 **(case 1)**
9 has successor 7 **(case 2)**

Insertion



Insert 5 in T



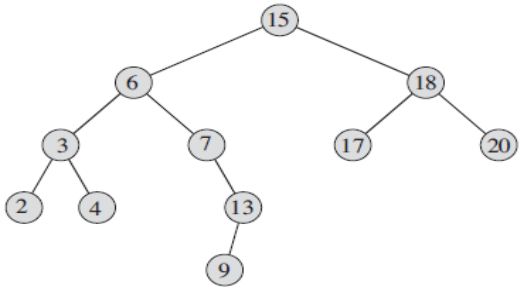
Insert 19 in T

Insertion

```
function Insert (T, key, item)
   N.key=key
   N.item=item
   N.right=N.left=nil
   P=T.root
   Q=nil
   while P≠nil
      Q=P
      if N.key<P.key then P=P.left
      else P=P.right
   N.parent=Q
   if Q=nil then T.root=N
   elseif N.key<Q.key then Q.left=N
   else Q.right=N
```

O(h) with h=height

Removal



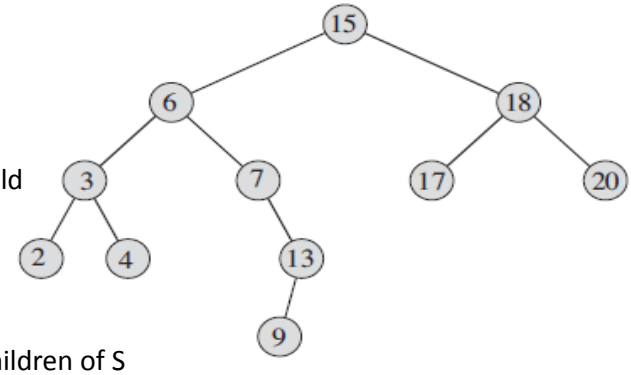
example 1: remove 9 (simplest case)
example 2: remove 13
example 3: remove 7
example 4: remove 6
example 5: remove 15

Removal

case 1: **N has no children**
set the left or right
pointer of N's parent to nil

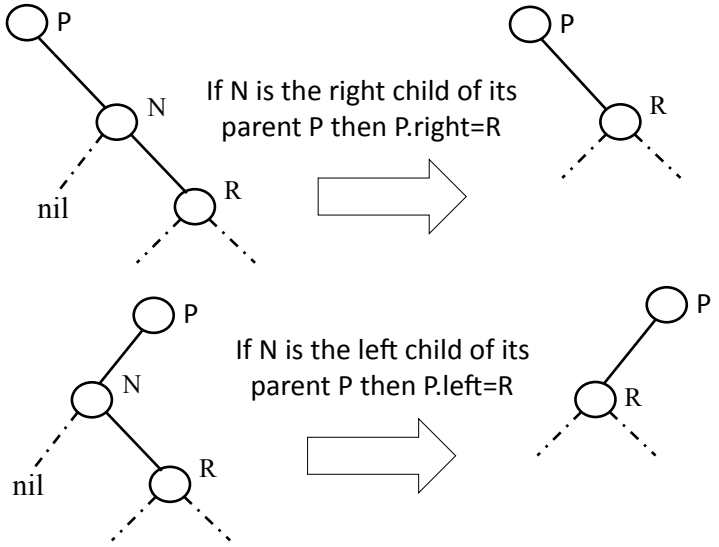
case 2: **N has just one child**
set the left or right
pointer of N's parent to N's child

case 3: **N has two children**
-the tricky case
-N replaced by its successor S
with N*'s* children becoming children of S
-has subcases



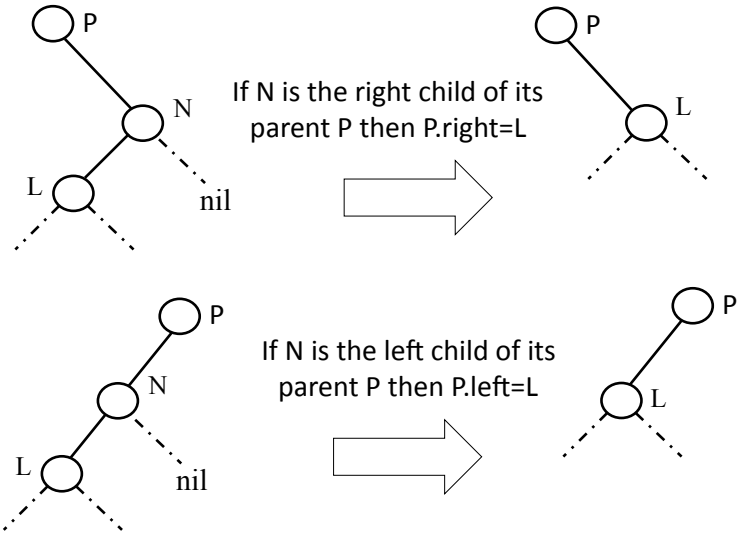
Removal

case 2.1
N only has a right child R

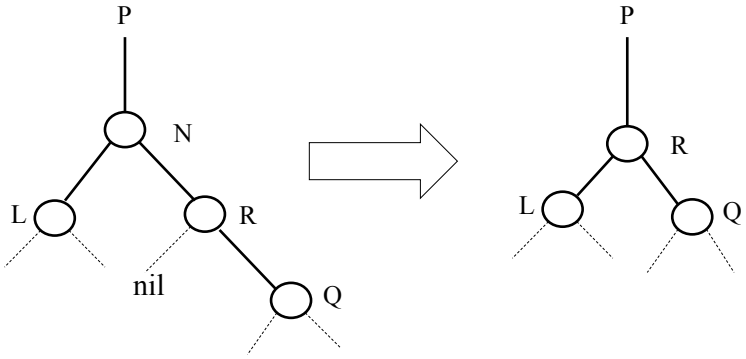


Removal

case 2.2
N only has a left child L



Removal

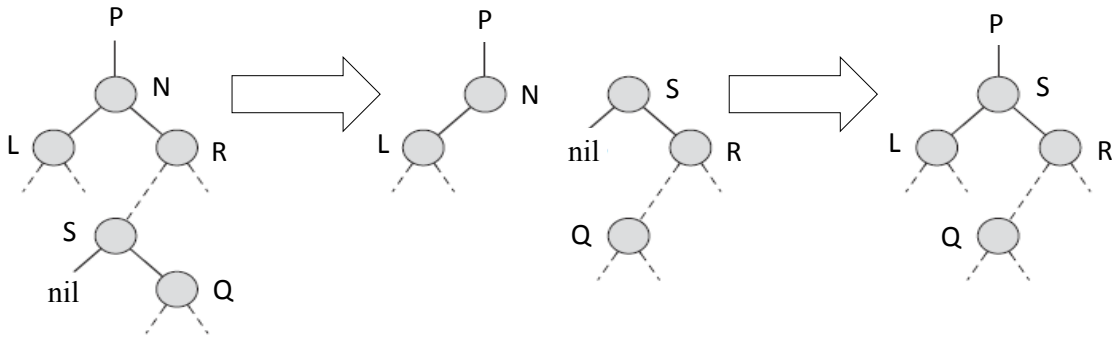


case 3, example 1

N has two children L and R but R has no left child (i.e., R is N's successor):

R.left=L, and set the (left or right) pointer of N's parent to R

Removal



case 3, example 2

N has two children L and R and R has a left child (i.e., N's successor is S≠R):

R.left=S.right, S.left=L, S.right=R, and set the (left or right) pointer of N's parent to S

Removal

```
function Transplant (T, N, M)
   if N.parent=nil then T.root=M
   elseif N=N.parent.left then N.parent.left=M
   else N.parent.right=M
   if M≠nil then M.parent=N.parent
```

Replaces the subtree rooted at node N
with the subtree rooted at node M

Removal

```
function Remove (T, N)
   if N.left=nil then Transplant(T,N,N.right)
   elseif N.right=nil then Transplant(T,N,N.left)
   else
      S=Minimum(N.right)
      if S.parent≠N
         Transplant(T,S,S.right)
         S.right=N.right
         S.right.parent=S
      Transplant(T,N,S)
      S.left=N.left
      S.left.parent=S
```

case 3

O(h) with h=height