ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives. Inventing the Future. www.iit.edu

CS430

Introduction to Algorithms

Lec 11 & 12

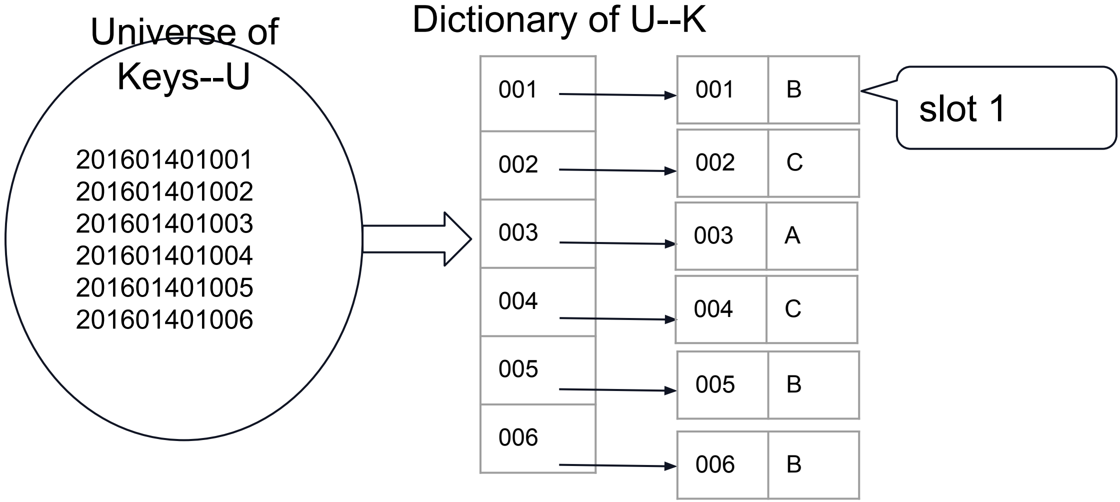
Lan Yao

Outlines

- **Hash Tables**
- **Binary Search Tree**

- Hash Tables-- An Example

ID#	Last Name	First Name	Midterm Exam	Final Exam	Final Grade
201601401001	Baker	Lane	86	89	B
201601401002	Diebold	Cormac	75	80	C
201601401003	Green	Bob	89	92	A
201601401004	Nowoj	Michael	66	78	C
201601401005	Ocon	Diann	88	87	B
201601401006	Wong	Madison	82	80	B

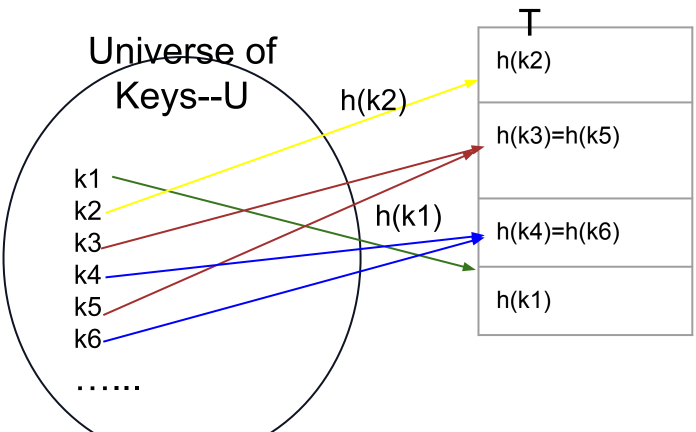


Direct-addressing Table:

- Works well when |U| is small.
- Complexity: O(1)
- What if |U| is large?

Our Goal:

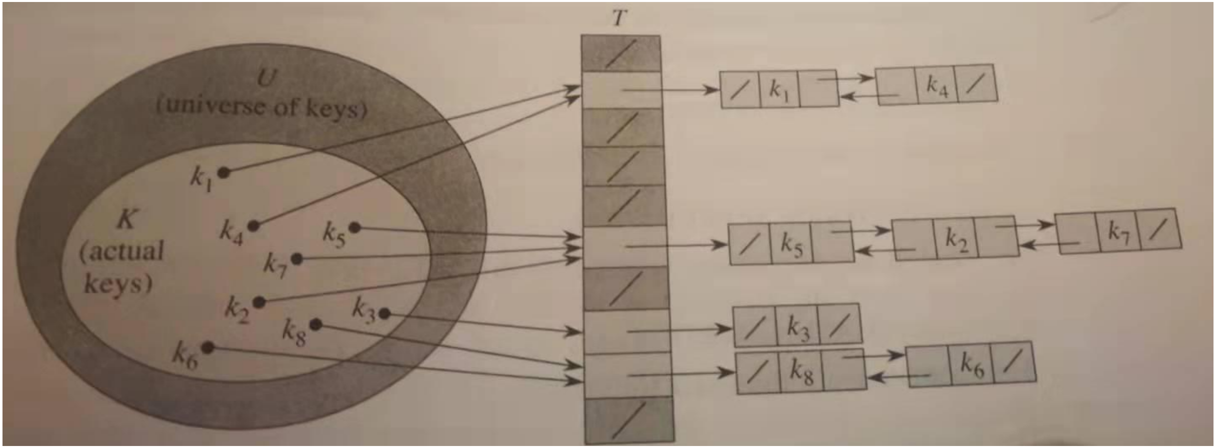
- Reduce |K| to be much smaller than |U| to require much less storage Θ(K);
- maintain the benefit that searching for an element still requires O(1) time.
- We have to design a function h to map the universe U of keys into the slots of a table T[0,1,...,m-1]



Definitions

- The function *h* that is designed to compute the slot from key k is a **Hash Function**. h must be **deterministic**.
- The table *T* that contains all slots is Hash Table.
- h(k) is the **hash value** of key k or key k hashes to slot h(k).
- When multiple keys hash to the same slot, they have **Collision**.
  - How to avoid collision?
    - Because |U|>m, at least two keys have the same hash values. ----impossible.
  - How to resolve collision?

Collision Resolution--Chaining



Dictionary Operations on Chained Hash Table T

CHAINED-HASH-INSERT (T, x)

insert x at the head of list T [h(x.key)]

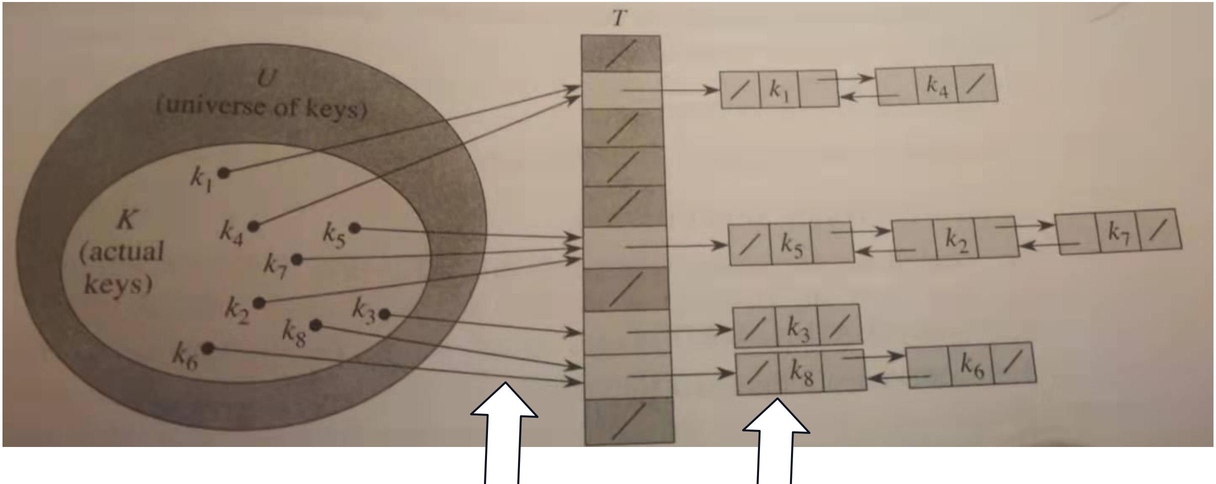
CHAINED-HASH-SEARCH(t, k)

search for an element with key k in list T[h(k)]

CHAINED-HASH-DELETE

delete x from the list T[h(x.key)]

Analysis of Chained Hash



O(1)--h(ki)

O(α)-- α is the expected # of elements examined in T[h(ki)] link in a unsuccessful search

Analysis of Chained Hash

There are m slots in T, which are T[0]...T[n

chain if collision happens. We denote the

nj. Then $n_0+n_1+...+n_{m-1}=n$. The expected l

The expected number of elements to exar

search is:

$$1+\alpha/2-\alpha/2n$$

[Xij]=I{h(ki)=h(kj)} and Pr {Xij}=1/m

The total required complexity is Θ (1+1+α/2-α/2n)=Θ(1+α)

Since n and m are proportional. n=O(m).

α=n/m=O(m)/m =O(1)

then O(1+**α**)=O(1)

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+E\left[\sum_{j=i+1}^{n} X_{ij}\right]\right)$$

$$=\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}E(X_{ij})\right)$$

$$=\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)$$

$$=\frac{1}{n}\sum_{i=1}^{n}1+\frac{1}{n}\sum_{i=1}^{n}\sum_{j=i+1}^{n}\frac{1}{m}$$

$$=1+\frac{1}{nm}\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n}1\right)\cancel{m}$$

$$=1+\frac{1}{nm}\cdot\sum_{i=1}^{n}(n-i)$$

$$=1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)$$

$$=1+\frac{1}{nm}\left[n^2-\frac{n(n+1)}{2}\right]$$

$$=1+\frac{n-1}{2m}=1+\frac{n}{2m}-\frac{1}{2m}$$

$$=1+\frac{\alpha}{2}-\frac{\alpha}{2n}$$

Binary Search Tree

- Each node contains a quintuple: index, key (satellite) , pointers to its left child, right child and parent;
- all keys in x’s left subtree <=x.key; all keys in x’s right subtree>=x.key.

- search, insert, delete, predecessor, successor, minimum, maximum operations are all O(h) where "h" is height of BST.
- with standard BST, "h" is determined by the order the "n" items are inserted into the BST and in the worst case h=n (best case h=lgn)

Tree Traversals

InOrder(root) visits nodes in the following order:

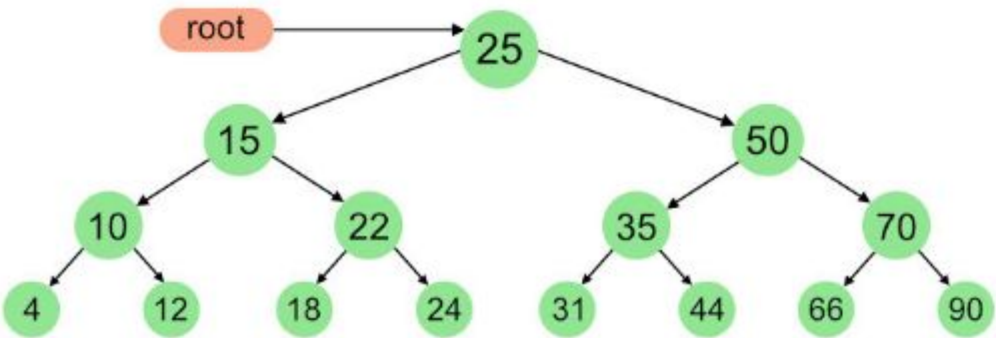
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

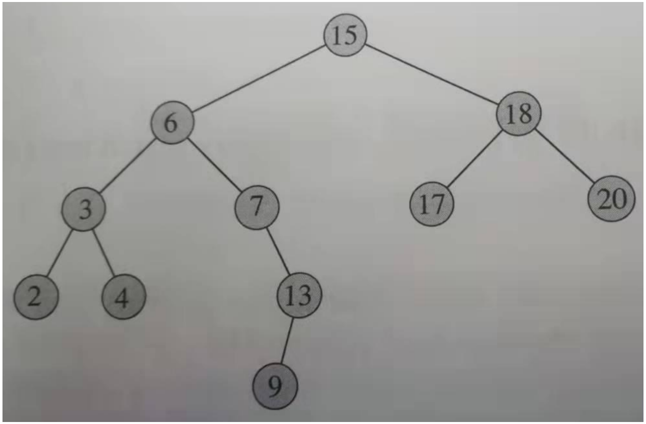
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:

4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Tree-Order



output：2，3，4，6，7，9，
13，15，17，18，20

INORDER-TREE-WALK (x)

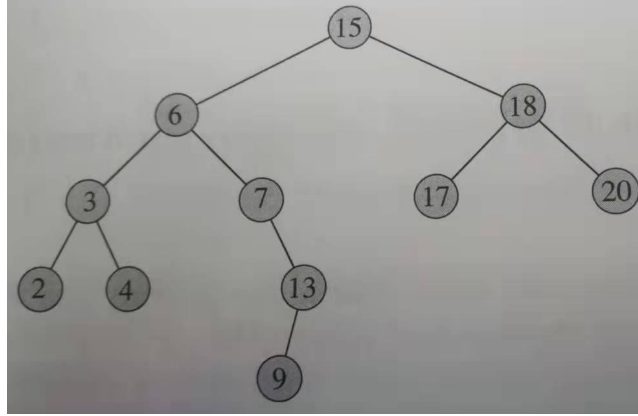
if x≠NIL

INORDER-TREE-WALK (x. left)

print x. key

INORDER-TREE-WALK (x. right)

Tree Search




```
TREE-SEARCH (x, k)
    ifx＝＝NIL  or  k＝＝x. key
        return（x）
    ifk＜x. key
        return  TREE－SEARCH（x. left,   k）
    else
        returnTREE－SEARCH（x. right,   k）
```

How to extract the MAX／MIN from BST？

```
TREE－MAX(x)
    While  x. right≠NIL
        x＝x. right
    return x
```

```
TREE－MIN(x)
    While  x. left≠NIL
        x＝x. left
    return x
```

Review of BST Successor

TREE-SUCCESSOR(x)

if $right[x] \neq$ NIL

  then return TREE-MINIMUM ($right[x]$)

case 1: x has a right child/subtree

$y = parent[x]$

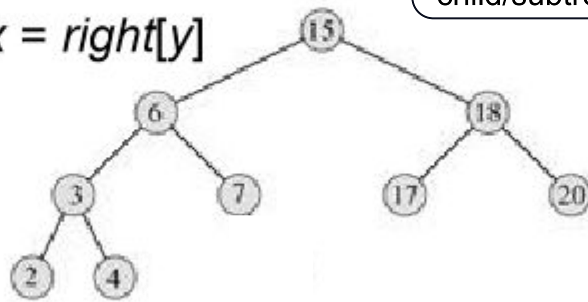
case 2: x does not have a right child/subtree

while $y \neq$ NIL and $x = right[y]$

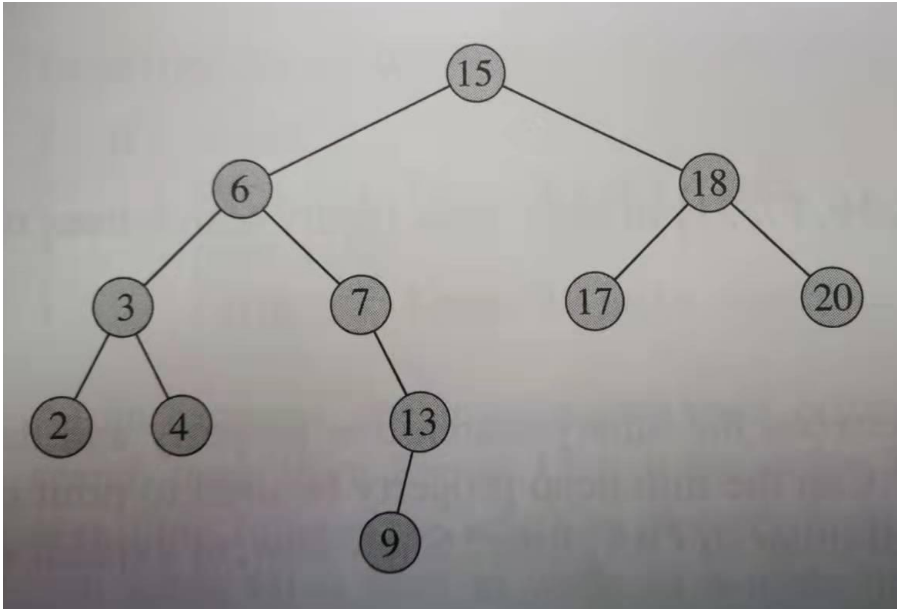
  $x = y$

  $y = parent[y]$

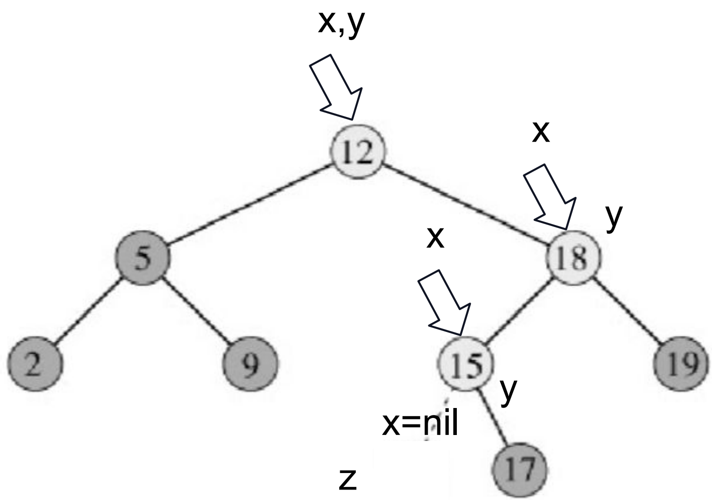
return y



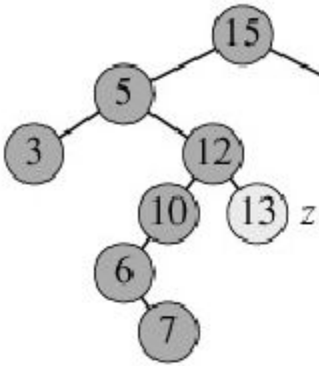
y is x’s successor if y is the lowest ancestor of x, whose left child is also x’s ancestor.

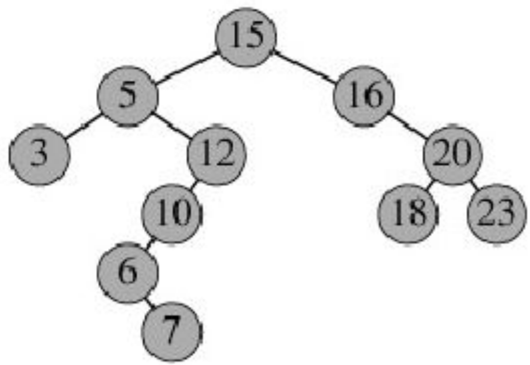


```
TREE-INSERT(T, z)
    y=NIL
    x=T.root
    while x≠NIL
        y = x
        if z.key<x.key
            x=x.left
        else   x=x.right
    z.p=y
    if y==NIL
        T.root=z
    elseif z.key <y.key
        y.left=z
    else   y.right=z
```

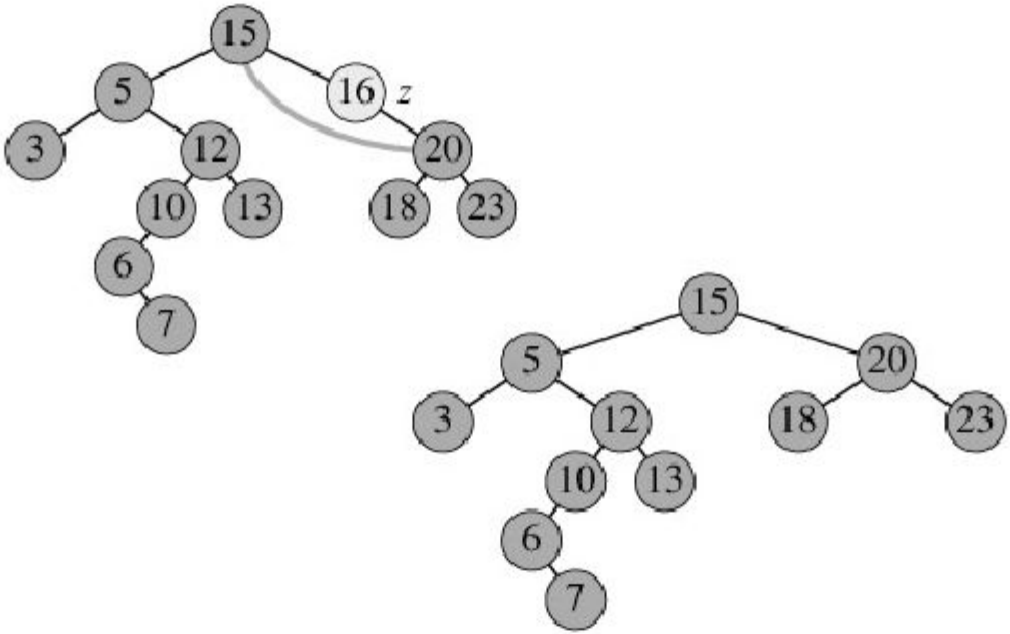


BST Delete (case A)

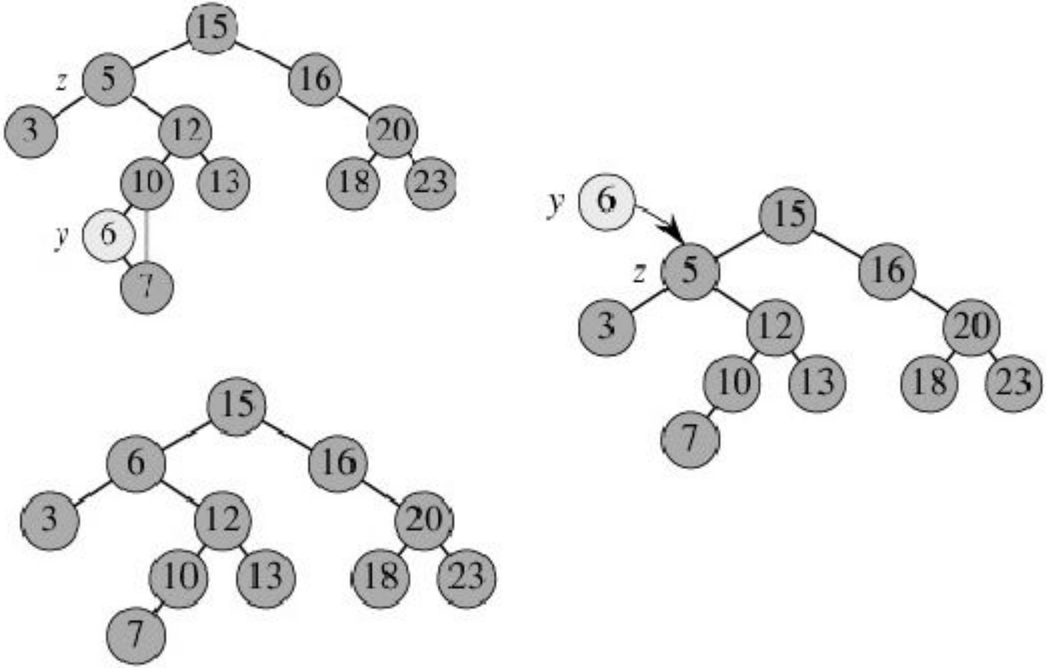




BST Delete (case B)

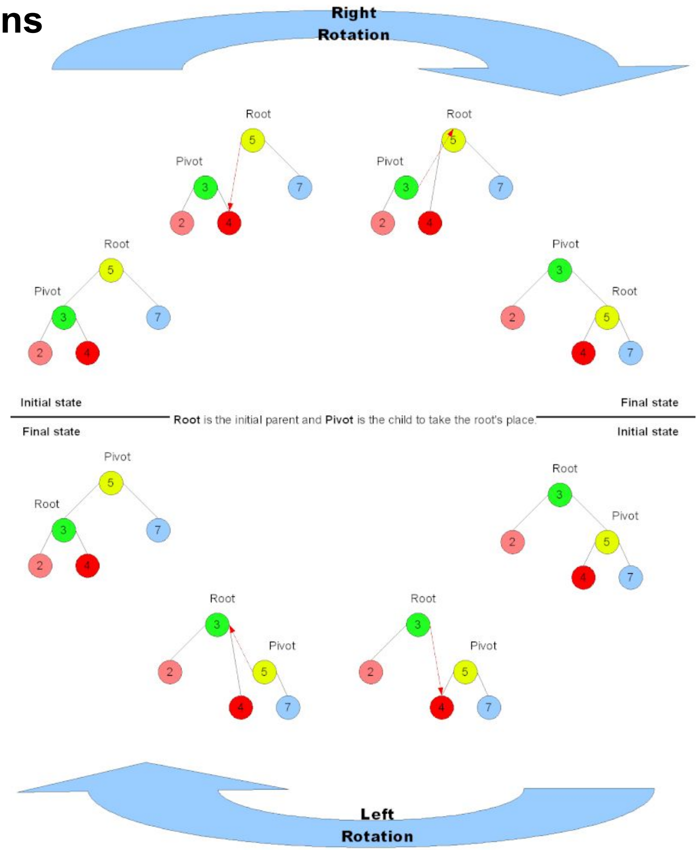


BST Delete (case C)



```
TREE-DELETE(T, z)  // assumes z points to a node to delete
 if left[z] = NIL or right[z] = NIL
     then y = z
 else y = TREE-SUCCESSOR(z)  // O(h) and z has two kids. Its
                                                   successor y must be the leftmost
                                                   node in its right subtree. And, y’s
                                                   left kid MUST be nil.
 if left[y] ≠ NIL
     then x = left[y] // x = y’s left kid under only one circumstance:
                         z.right=nil.
 else x = right[y]  // x = y’s right kid if: either z.left=nil or y is z’s
                         successor
 if x ≠ NIL            // x may be nil when z does not have either kids.
     then p[x] = p[y]
 if p[y] = NIL
     then root[T] = x
 else if y = left[p[y]] // if y is its father’s left child
            then left[p[y]] = x
            else right[p[y]] = x
 if y ≠ z  // y may be the same to z due to the repetition.
     then key[z] = key[y]
          copy y's satellite data into z
 return y
```

BST Rotations



Binary Search Tree Rotations*

Rotations - local operation in a search tree that maintains the BST property.

x and y are nodes; a, b, c are sub trees

```
      p [x]
       /
   x
  /  \        left-rotate(x)--->
a    y
    / \
   b   c
```

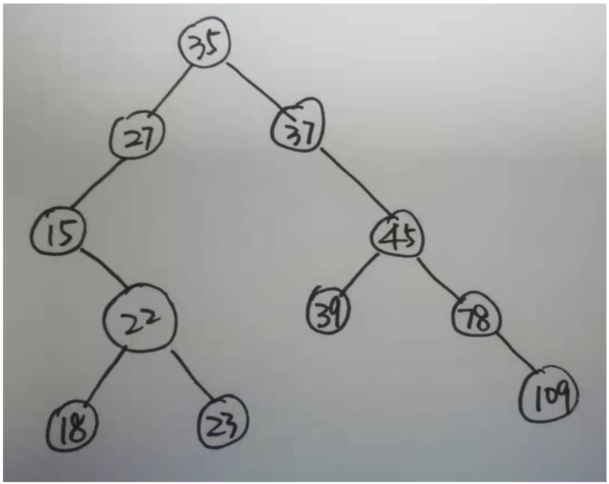
Left Rotation on x

1. x’s right child y (pivot) will replace x’s position;
2. y’s left subtree will be x’s right subtree;
3. x will be y’s left child;
4. x’s parent will be y’s parent;

```
LEFT-ROTATE(T, x)
y ← right[x]    // Set y
right[x] ← left[y]
//Turn y's left subtree into x's right subtree
p[left[y]] ← x
p[y] ← p[x]    //Link x's parent to y
if p[x] = nil[T]
    root[T] ← y
else
    if x = left[p[x]]
        left[p[x]] ← y
    else right[p[x]] ← y
left[y] ← x   // Put x on y's left
p[x] ← y
```

Classwork

(1) Explain how Tree-delete works to delete key 45 from the tree.



Classwork

(2) Go to **desmos.com** at the link below and answer the question “BST Right Rotation”.

https://student.desmos.com/activitybuilder/student-greeting/6257851d8ca41d5ca8a65c68