Splay Trees

CSE 326
Data Structures
Lecture 8

Readings and References

- Reading
  - › Sections 4.5-4.7

Self adjustment for better living

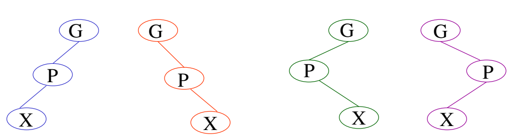
- Ordinary binary search trees have no balance conditions
  - › what you get from insertion order is it
- Balanced trees like AVL trees enforce a balance condition when nodes change
  - › tree is always balanced after an insert or delete
- Self-adjusting trees get reorganized over time as nodes are accessed

Splay Trees

- Splay trees are tree structures that:
  - › Are not perfectly balanced all the time
  - › Data most recently accessed is near the root.
- The procedure:
  - › After node X is accessed, perform “splaying” operations to bring X to the root of the tree.
  - › Do this in a way that leaves the tree more balanced as a whole

Splay Tree Terminology

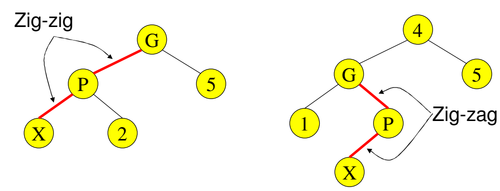
- Let X be a non-root node with $\geq$ 2 ancestors.
  - P is its parent node.
  - G is its grandparent node.



Zig-Zig and Zig-Zag

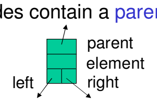
Parent and grandparent in same direction.

Parent and grandparent in different directions.



Splay Tree Operations

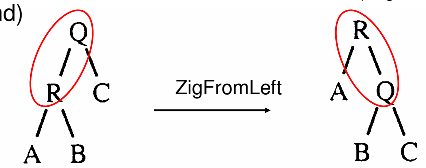
1. Helpful if nodes contain a parent pointer.



2. When X is accessed, apply one of six rotation routines.
- Single Rotations (X has a P (the root) but no G)
  ZigFromLeft, ZigFromRight
- Double Rotations (X has both a P and a G)
  ZigZigFromLeft, ZigZigFromRight
  ZigZagFromLeft, ZigZagFromRight

Zig at depth 1

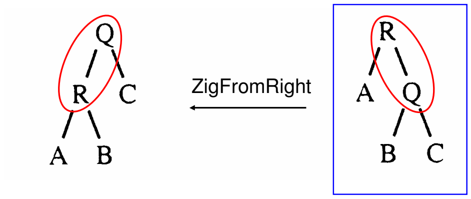
- “Zig” is just a single rotation, as in an AVL tree
- Let R be the node that was accessed (e.g. using Find)



- ZigFromLeft moves R to the top →faster access next time

Zig at depth 1

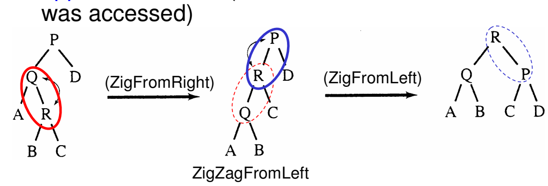
- Suppose Q is now accessed using Find



- ZigFromRight moves Q back to the top

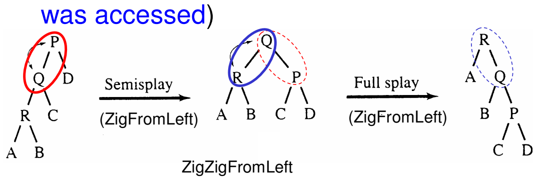
Zig-Zag operation

- “Zig-Zag” consists of two rotations of the opposite direction (assume R is the node that was accessed)



Zig-Zig operation

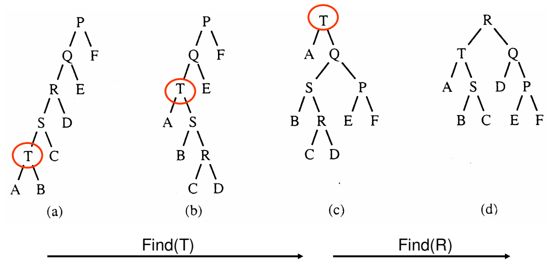
- “Zig-Zig” consists of two single rotations of the same direction (R is the node that was accessed)



Find Operation

- Find operation
  - Do a normal find in the binary search tree
  - Splay the the node found to the root by a series of zig-zig and zig-zag operations with an additional zig at the end if the length of the path to the node is odd.
  - If nothing found splay the last node visited to the root.

Decreasing depth - "autobalance"

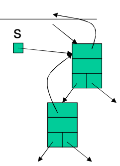


Details of SplayFind

```
SplayFind(p: node pointer,x: key): node pointer {
r,s : node pointer;
r := Find(p,x); //if x is not in the tree then
                //the last node visited is returned
while r.parent ≠ null do {
  s := r.parent.parent;
  case {
  s = nil:
    if r.parent.right = r then ZigFromRight(r.parent) ;
    else ZigFromLeft(r.parent);
  s.right.right = r: ZigZigFromRight(s);
  s.left.left = r: ZigZigFromLeft(s);
  s.right.left = r: ZigZagFromRight(s);
  s.left.right = r: ZigZagFromLeft(s);
  }
return r //r contains x if it is in the tree
}
```


ZigFromLeft

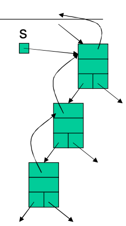
```
ZigFromLeft(s: node pointer): {
c: node pointer;
c := s.left;
s.left := c.right;
if s.left ≠ null then s.left.parent := s;
c.parent := s.parent;
if c.parent ≠ null then
   if c.parent.right = s then c.parent.right := c;
   else c.parent.left := c;
s.parent := c;
c.right := s;
}
```



Try ZigZigFromLeft

- Design ZigZigFromLeft

```
ZigZigFromLeft(s: node pointer) {
???
}
```

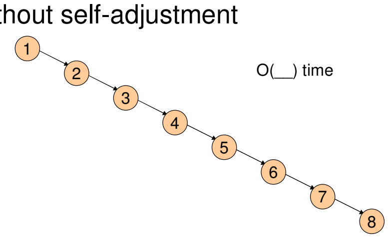


Splay Tree Insert

- Insert x
  - Insert x as normal then splay x to root.

Example Insert

- Inserting in order 1,2,3,…,8
- Without self-adjustment



With Self-Adjustment

1

2

3

With Self-Adjustment

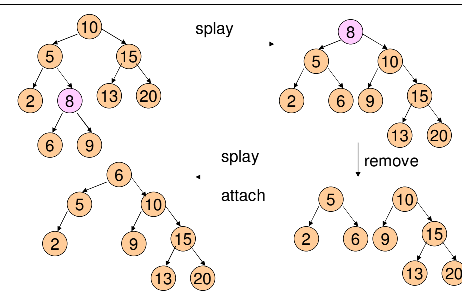
4

O(__) time!!

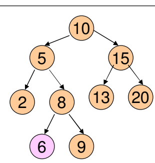
Splay Tree Deletion

- Delete
  - › Splay x to root and remove it. Two trees remain, right subtree and left subtree.
  - › Splay the max in the left subtree to the root
  - › Attach its right subtree to the new root of the left subtree and return it. The predecessor of x becomes the root.

Example Deletion



Practice Delete

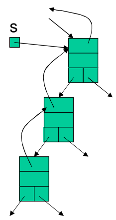


Analysis of Splay Trees

- Splay trees tend to be balanced
  - › M operations takes time O(M log N) for M $\geq$ N operations on N items.
  - › Amortized O(log n) time.
- Splay trees have good “locality” properties
  - › Recently accessed items are near the root of the tree.
  - › Items near an accessed node are pulled toward the root.

Solution to First Exercise

```
ZigZigFromLeft(s: node pointer) {
c: node pointer;
c := s.left;
ZigFromLeft(s);
ZigFromLeft(c);
}
```



Solution to Second Exercise