Splay trees

CS 202 – Fundamental Structures of Computer Science II

Bilkent University

Computer Engineering Department

Spay Trees

- So far we have seen:
  - BST: binary search trees
    - Worst-case running time per operation = O(N)
    - Worst case average running time = O(N)
      - Think about inserting a sorted item list
  - AVL tree:
    - Worst-case running time per operation = O(logN)
    - Worst case average running time = O(logN)
- We will now wee a new data structure, called *splay trees,* for which:
    - Worst-case running time of one operation = O(N)
    - Worst case running time of *M* operations = O(*M*log*N)*
      - O(log*N*) amortized running time.
- A spay tree is a *binary **search** tree.*

Splay trees

- A splay tree guarantees that, for *M* consecutive operations, the total running time is O(*M*log*N*).
- A *single* operation on a splay tree can take O(N) time.
  - So the bound is not as strong as O(log*N*) worst-case bound in AVL trees.

Amortized running time

- **Definition**: For a series of *M* consecutive operations:
  - If the total running time is O(*M**f(*N*)), we say that the amortized running time (per operation) is O(f(*N*)).
- Using this definition:
  - A splay tree has O(log*N*) amortized cost (running time) per operation.

Splay tree idea:

- Try to make the worst-case situation occur less frequently.
  - In a Binary search tree, the worst case situation can occur with every operation. (while inserting a sorted item list).
  - In a splay tree, when a worst-case situation occurs for an operation:
    - The tree is re-structured (during or after the operation), so that the subsequent operations do not cause the worst-case situation to occur again.
- The basic idea of splay tree is:
  - After a node is accessed, it is pushed to the root by a series of AVL tree-like operations (rotations).
  - For most applications, when a node is access, it is likely that it will be accessed again in the near future.

Splay tree idea:

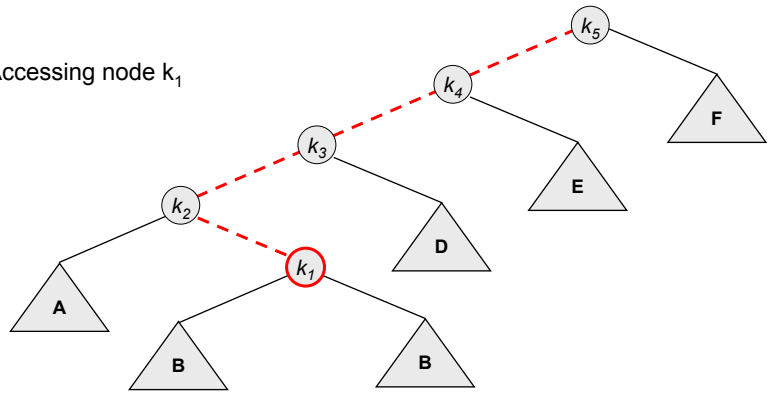
- By pushing the accessed node to the root the tree:
  - 1) If the accessed node is accessed again, the future accesses will be much less costly.
  - 2) During the *push to the root* operation, the tree might be more balanced than the previous tree.
    - Accesses to other nodes can also be less costly.

Splay tree: a bad method for *pushing to the root*

- A simple idea, which is not working, is:
  - When a node K is accessed, push it towards the root by the following algorithm:
    - On the path from k to root:
      - Do a singe rotation between node k's parent and node k itself.
        - Every single rotation will push k to one level higher.
        - Finally, it will reach to the root.

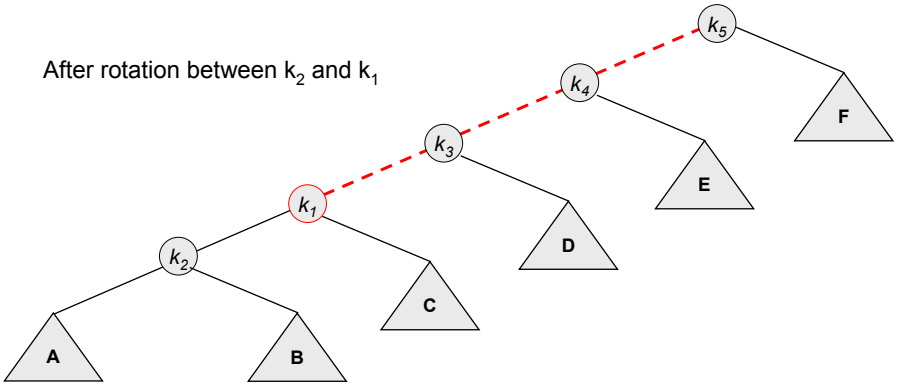
Splay tree: a bad method for *pushing to the root*

Accessing node k_1



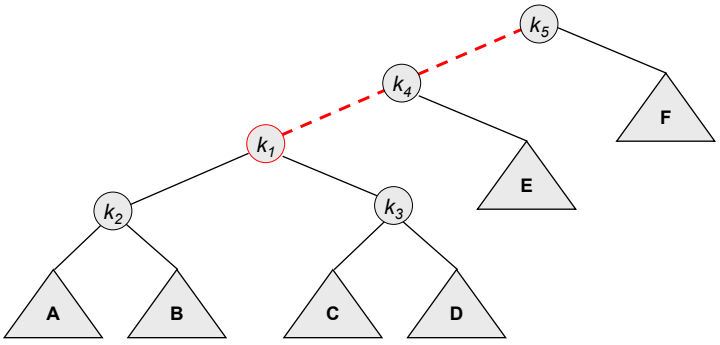
Pushing k_1 to the root

After rotation between k_2 and k_1



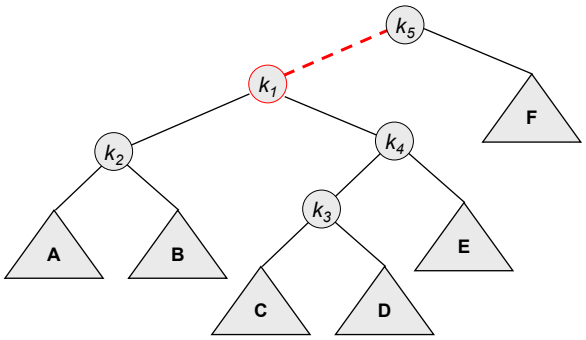
Pushing k_1 to the root

After rotation between k_3 and k_1



Pushing k_1 to the root

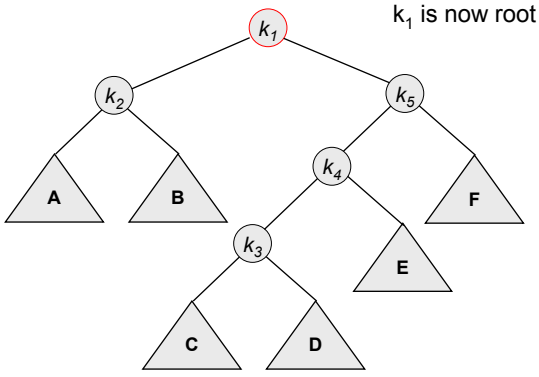
After rotation between k_4 and k_1



Pushing k_1 to the root

After rotation between k_5 and k_1

k_1 is now root



But k_3 is nearly as deep as k_1 was.
An access to k_3 will push some other node nearly as deep as k_3 is.
So, this method does not work!

Splaying – A method that works

- Now, we will give a method that works.
  - It will push the accessed node to the root.
  - With this pushing operation it will also balance the tree somewhat.
    - So that further operations on the new will be less costly compared to operations that would be done on the original tree.
- A deep tree will be spayed:
  - Will be less deep, more wide.

Splaying - algorithm

- Assume we access a node.
- We will splay along the path from access node to the root.
- At every splay step:
  - We will selectively rotate the tree.
  - Selective operation will depend on the structure of the tree around the node around which rotation will be performed

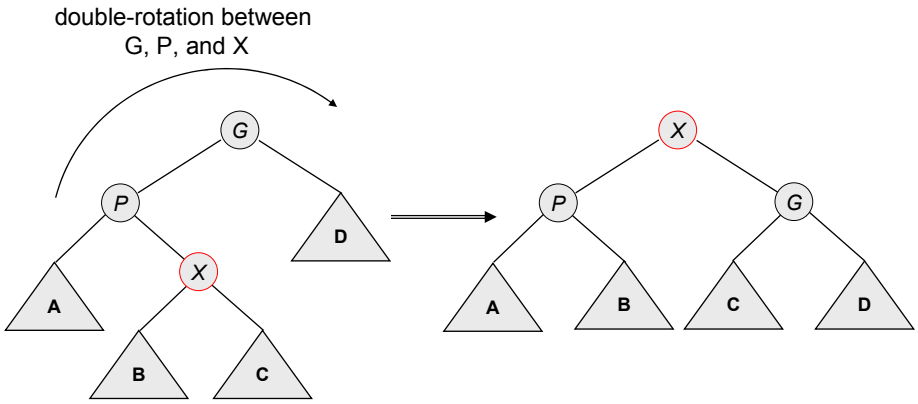
Splaying - algorithm

- Let X be a non-root node on the access path at which we are rotating.
  - If parent of X is root,
    - We rotate between X and root. X becomes root. And we are done!
  - Otherwise, X has a parent P which is non-root.
    - X certainly has also a grand-parent G.
  - There two cases to consider
    - Zig-zag case: P is left child of G and X is right child of P
    - Zig-zig case: P is left child of G and X is left child of P
  - We will have also symmetric cases of the two cases above. The operations for these symmetric cases will be very similar to the operations for the two cases above. Symetric cases are:
    - P is right child of G and X is left child of P
    - P is right child of G and X is right child of P.

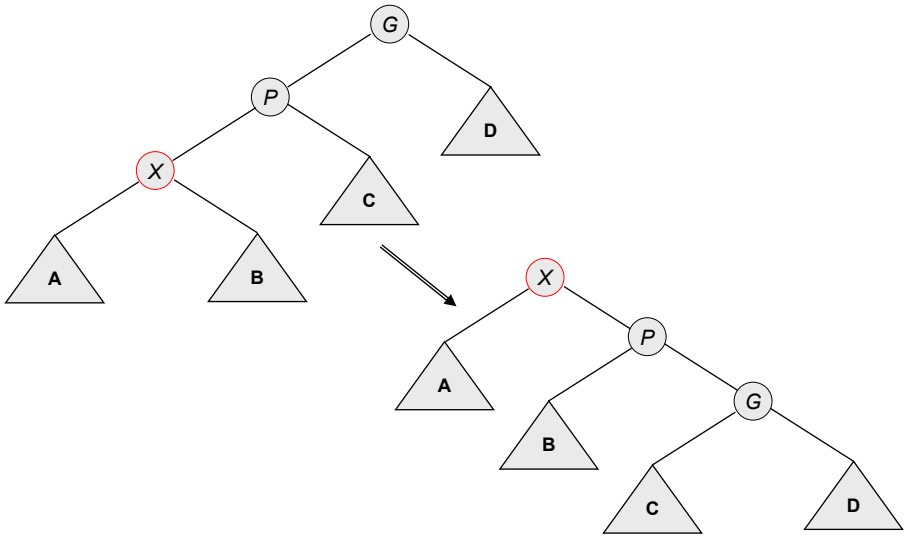
Splaying - algorithm

- Depending on the case that is determined from the interconnection of X, P, and G, we will perform one of the two operations we will describe below:
  - Zig-zag case:
    - Perform double rotation (an AVL tree operation).
  - Zig-zig case
    - Transform the tree to an other one as described below.

Case 1: Zig-zag case operation

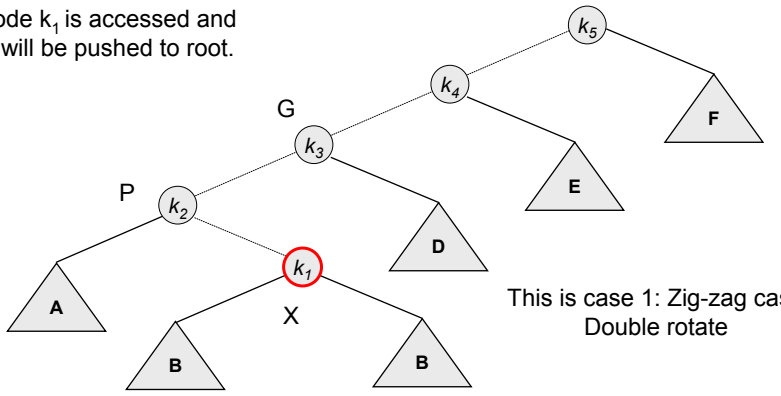


Case 2: Zig-zig case operation



A general example:

node k_1 is accessed and it will be pushed to root.

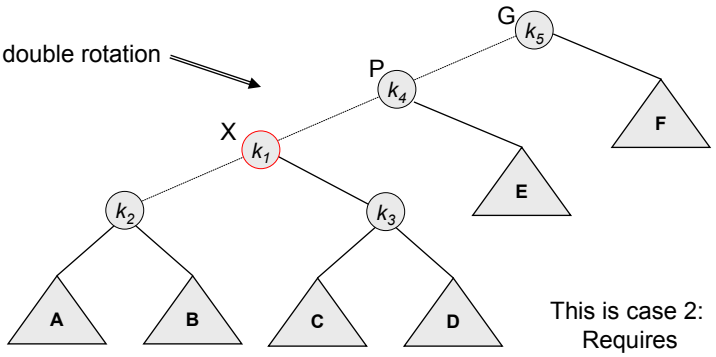


This is case 1: Zig-zag case
Double rotate

Original tree

A general example:

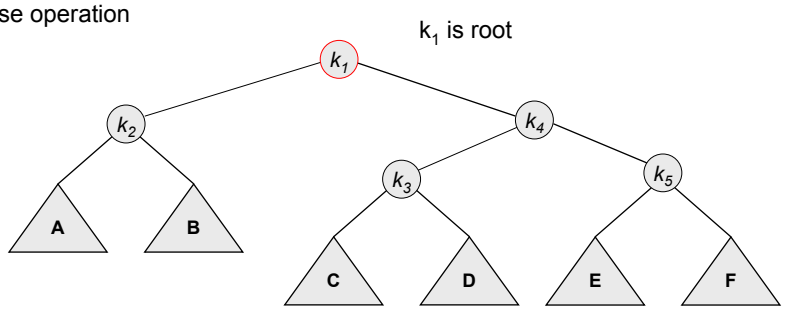
After double rotation



This is case 2:
Requires
zig-zig case
operation

A general example – final tree

After zig-zig case operation



New tree

The tree is more balanced than the original tree

The depth of the new tree is nearly half of the depth of the original tree

A specific example

- Assume 7,6,5,4,3,2,1 are inserted into an empty splay tree.