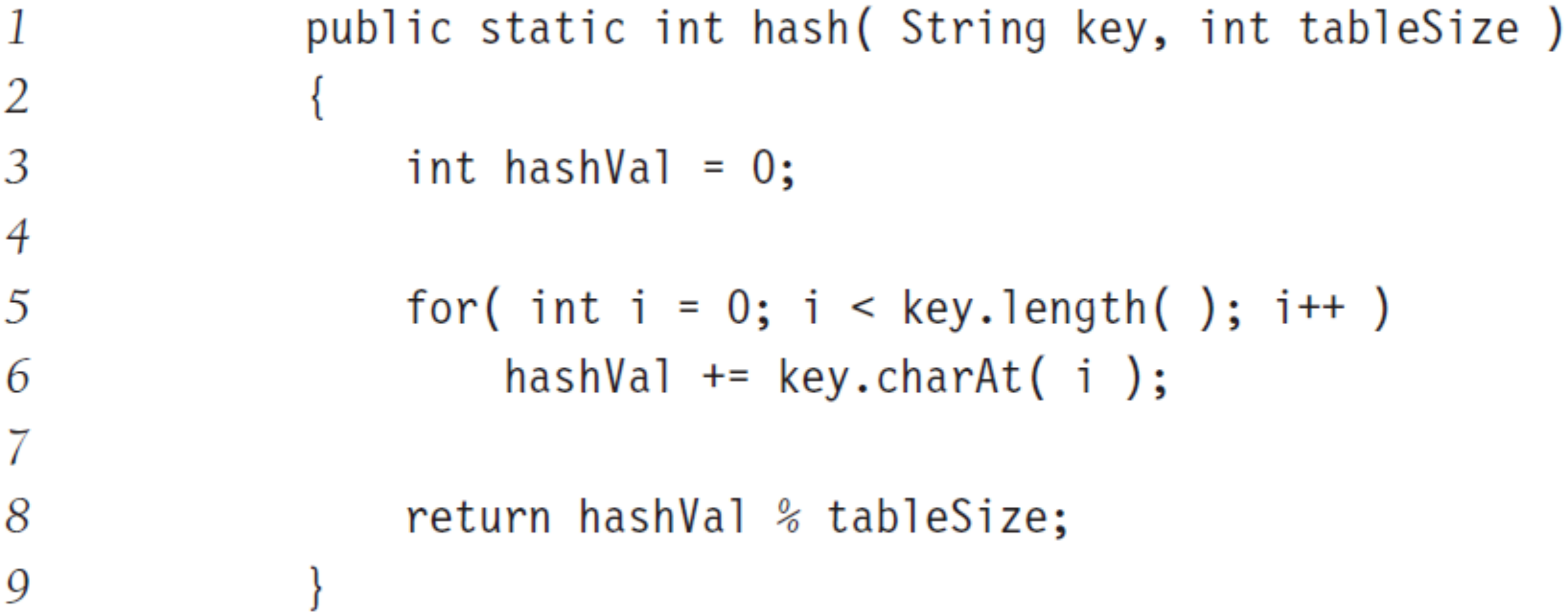
CSE 373
Review Session

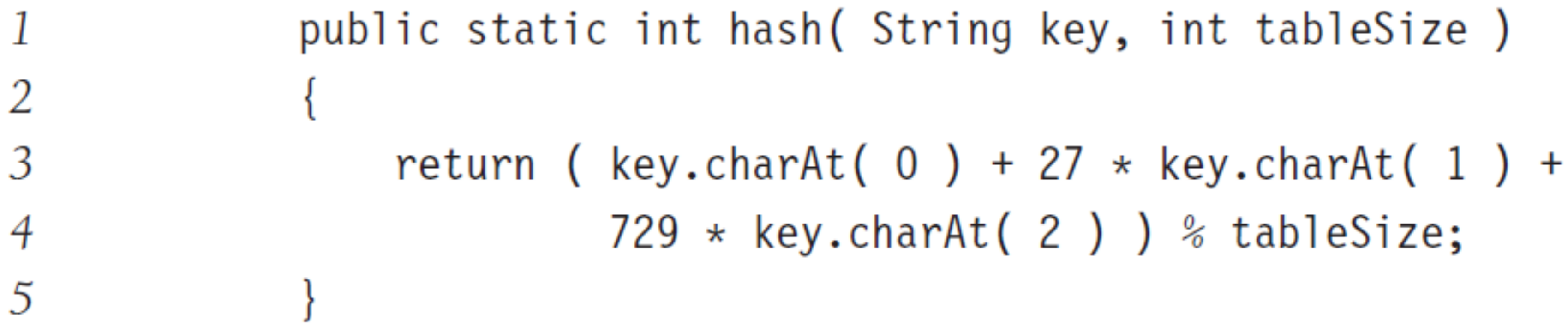
String Hash Functions
Splay Trees

String Hash Function #1



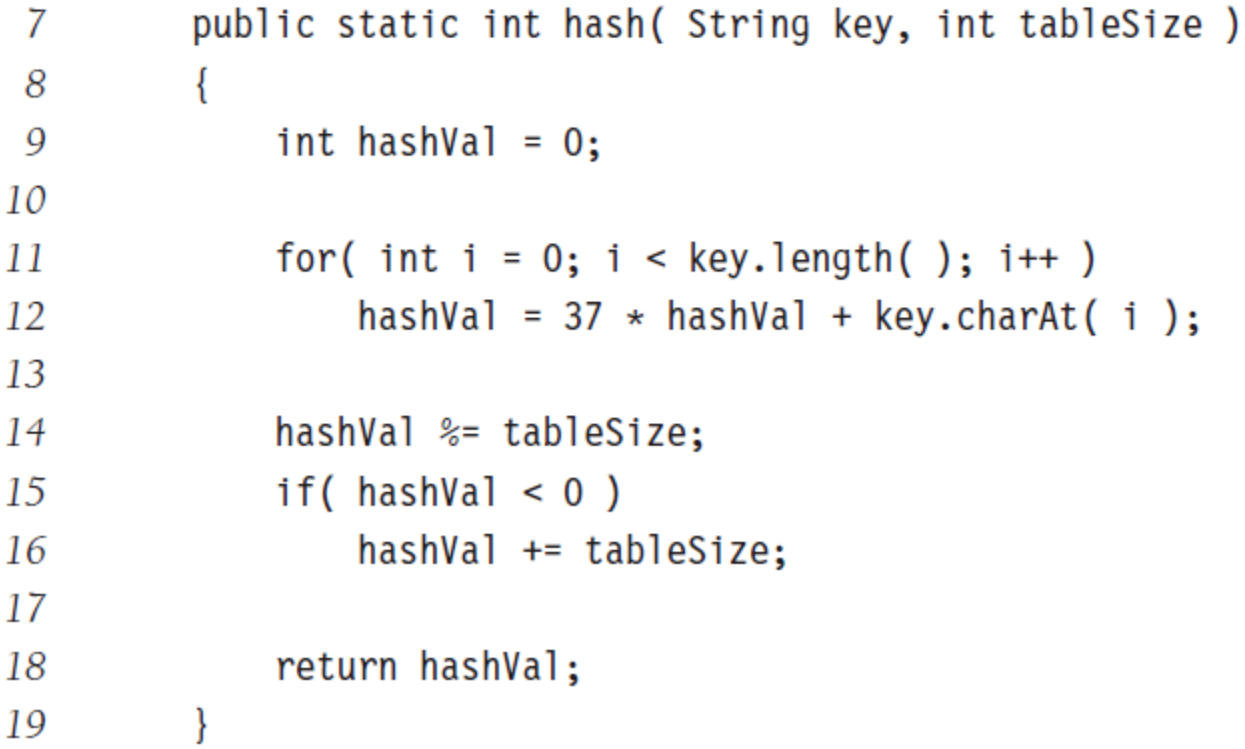
```java
public static int hash( String key, int tableSize )
{
    int hashVal = 0;

    for( int i = 0; i < key.length( ); i++ )
        hashVal += key.charAt( i );

    return hashVal % tableSize;
}
```

String Hash Function #2



```
public static int hash( String key, int tableSize )
{
    return ( key.charAt( 0 ) + 27 * key.charAt( 1 ) +
                  729 * key.charAt( 2 ) ) % tableSize;
}
```

String Hash Function #3



```
public static int hash( String key, int tableSize )
{
    int hashVal = 0;

    for( int i = 0; i < key.length( ); i++ )
        hashVal = 37 * hashVal + key.charAt( i );

    hashVal %= tableSize;
    if( hashVal < 0 )
        hashVal += tableSize;

    return hashVal;
}
```

String Hash Function #3

For a string $s = s_1 s_2 \cdots s_n$

$$h_3(s) = \sum_{i=1}^{n} s_i \cdot 37^{n-i}$$

Hash Tables Without Linked Lists

The ith probe for a key k and a hash table of size s:

Linear Probing:

Hash Tables Without Linked Lists

The ith probe for a key k and a hash table of size s:

Linear Probing: $h_l(k, i, s) = [h(k) + (i - 1)] \% s$

Quadratic Probing:

Hash Tables Without Linked Lists

The ith probe for a key k and a hash table of size s:

Linear Probing: $h_l(k, i, s) = [h(k) + (i - 1)] \% s$

Quadratic Probing: $h_q(k, i, s) = [h(k) + (i - 1)^2] \% s$

Double Hashing:

Hash Tables Without Linked Lists

The ith probe for a key k and a hash table of size s:

Linear Probing: $h_l(k, i, s) = [h(k) + (i - 1)] \,\%\, s$

Quadratic Probing: $h_q(k, i, s) = [h(k) + (i - 1)^2] \,\%\, s$

Double Hashing: $h_d(k, i, s) = [h(k) + (i - 1) \cdot g(k, s)] \,\%\, s$

Hash Tables Without Linked Lists

The ith probe for a key k and a hash table of size s:

Linear Probing: $h_l(k, i, s) = [h(k) + (i - 1)] \% s$

Quadratic Probing: $h_q(k, i, s) = [h(k) + (i - 1)^2] \% s$

Double Hashing: $h_d(k, i, s) = [h(k) + (i - 1) \cdot g(k, s)] \% s$

Where h and g are hash functions and g is never 0.
e.g. $g(k, s) = R - (R\%s)$ for a prime number $R < s$.

Splay Trees

Another type of binary search trees:

Structure property: every node has at most 2 children.

Order property: for every node r, every node in the *left subtree* of r is *smaller* than r, and every node in the *right subtree is bigger* than r.

Splay Trees

Runtime guarantee: every tree operation has an **amortized** runtime of $O(\log n)$ where n is the maximal size of the tree.
That is, starting from an empty tree, every sequence of M consecutive operations takes $O(M \log n)$ time.

Nice properties:

Automatically optimized, such that frequently accessed elements take less time.

Supports efficient (amortized $O(\log n)$) execution of additional operations, such as merging and splitting around a pivot.

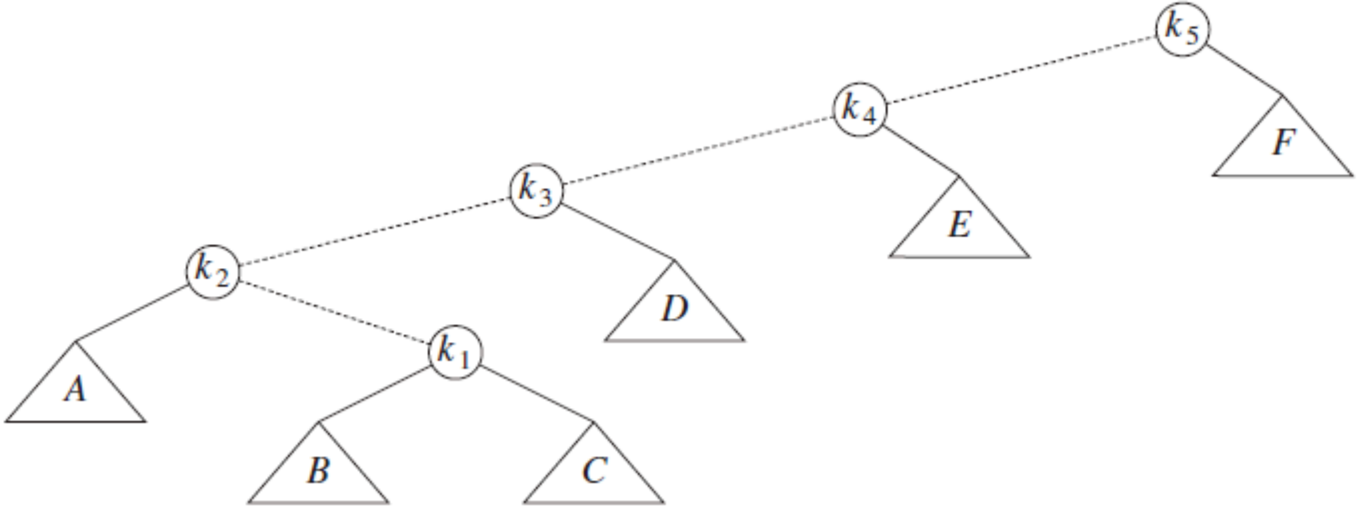
Splay Trees - Operations

Find: same as a regular BST.

But(!!), in order to maintain runtime guarantees, once the element is found propagate (“splay”) it up to the root.

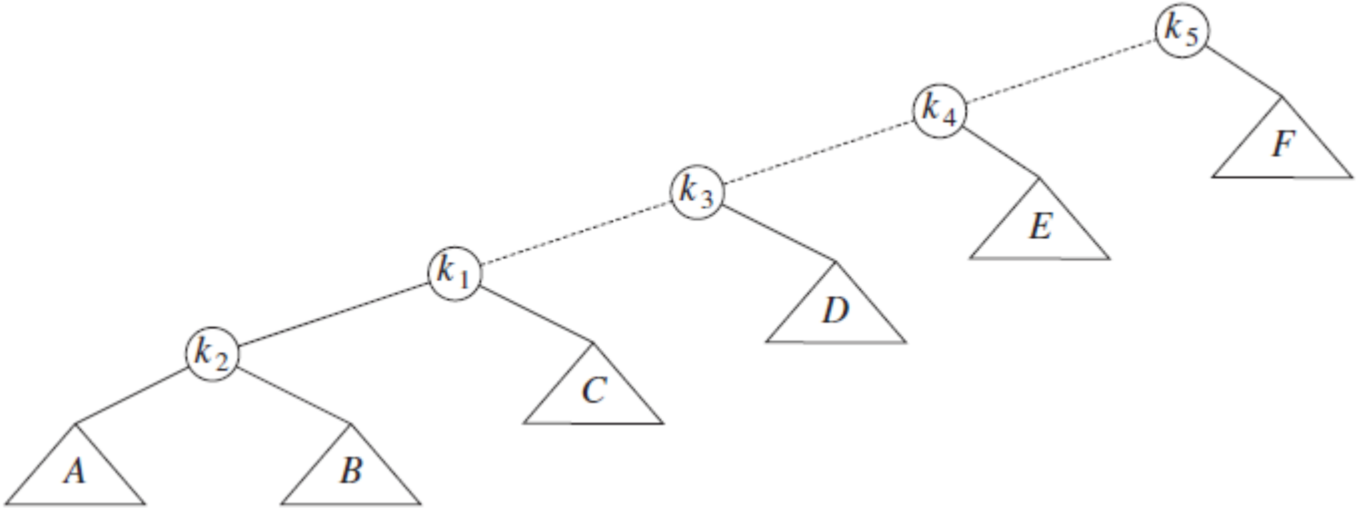
Find - **Bad** Implementation (doesn’t work)

Propagate the found node to the root using single rotations:



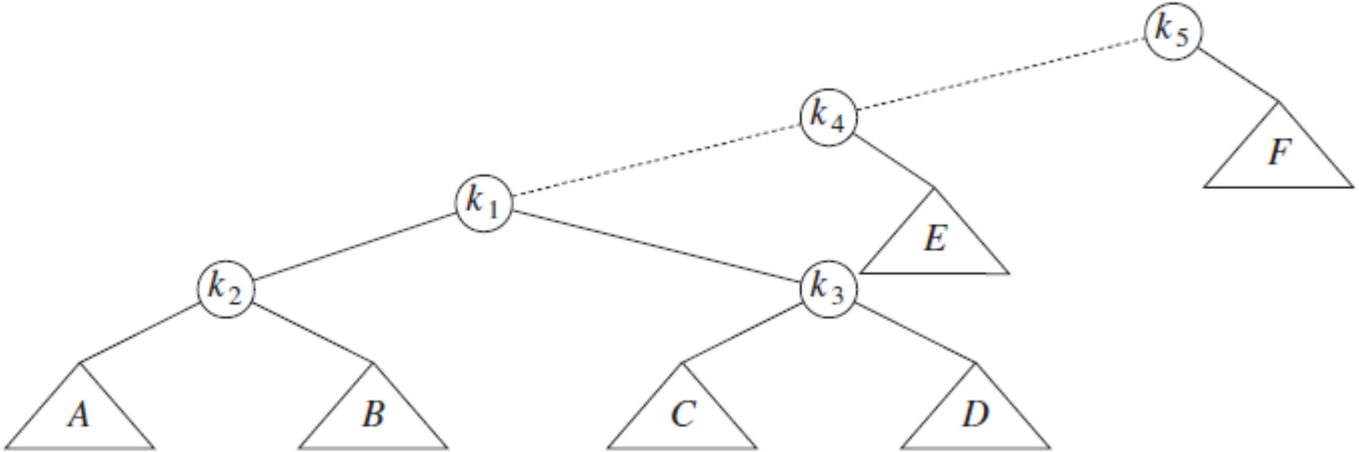
Find - **Bad** Implementation (doesn’t work)

Propagate the found node to the root using single rotations:



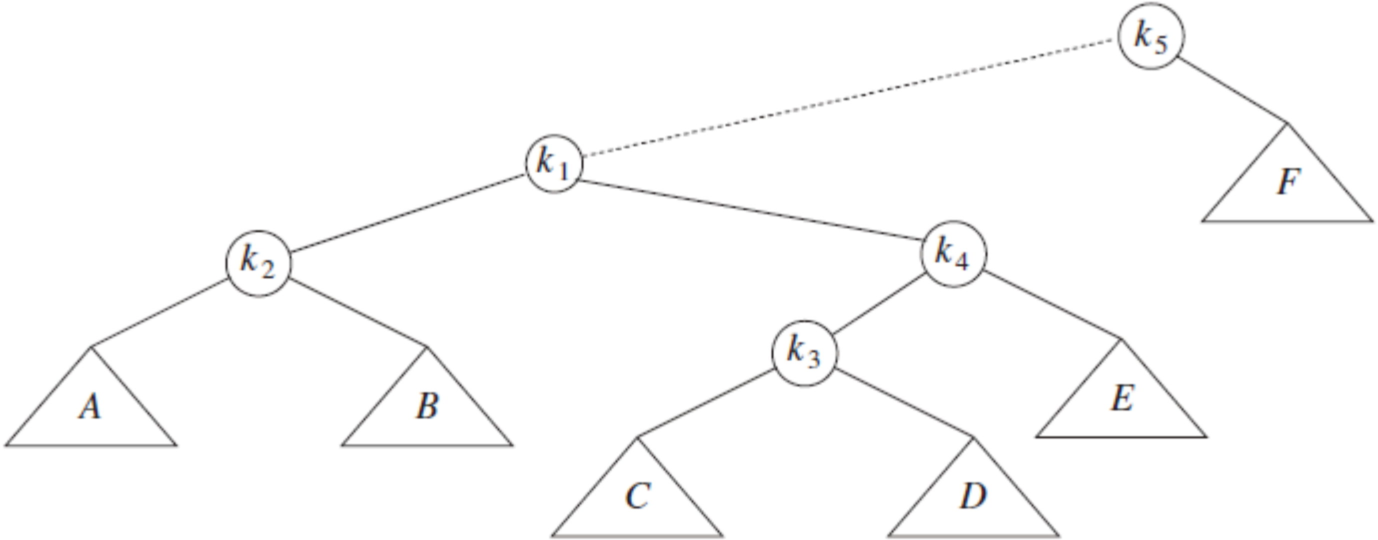
Find - **Bad** Implementation (doesn’t work)

Propagate the found node to the root using single rotations:



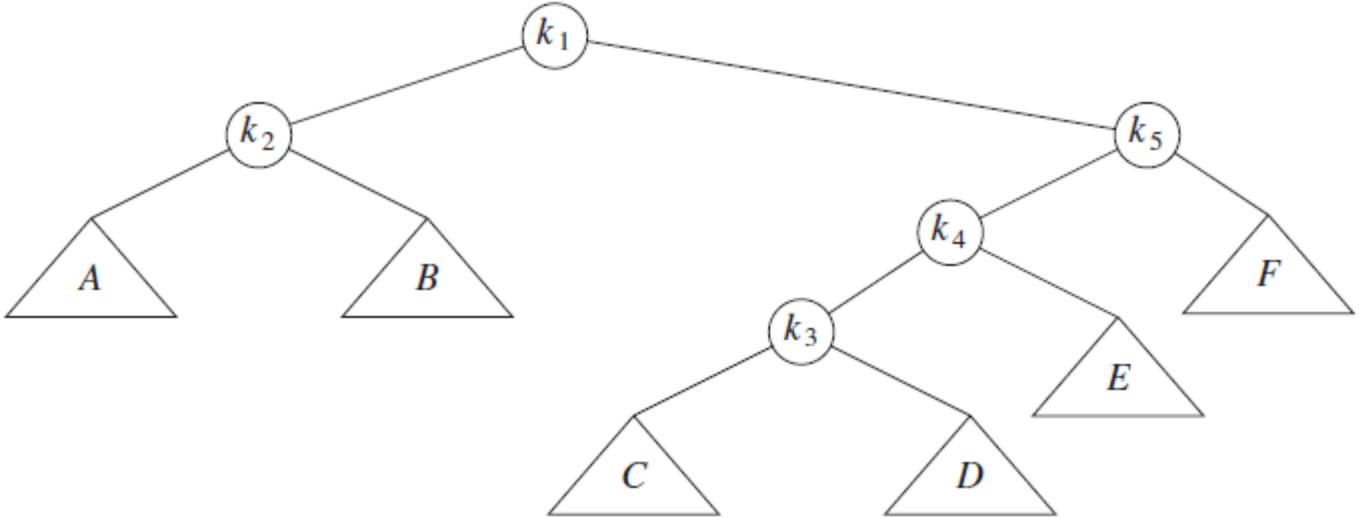
Find - **Bad** Implementation (doesn't work)

Propagate the found node to the root using single rotations:



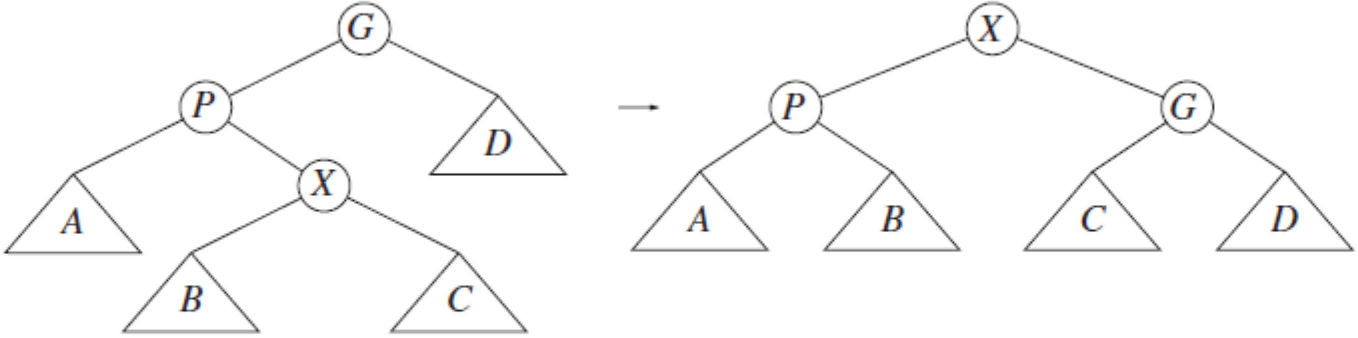
Find - **Bad** Implementation (doesn’t work)

Propagate the found node to the root using single rotations:



Find - **Good** Implementation

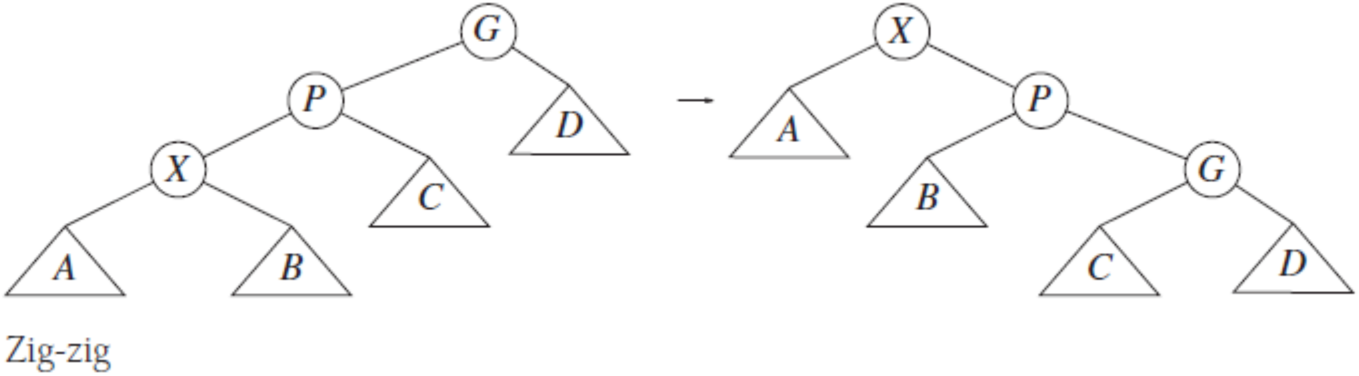
Propagate the found node to the root using **double** rotations:



Zig-zag

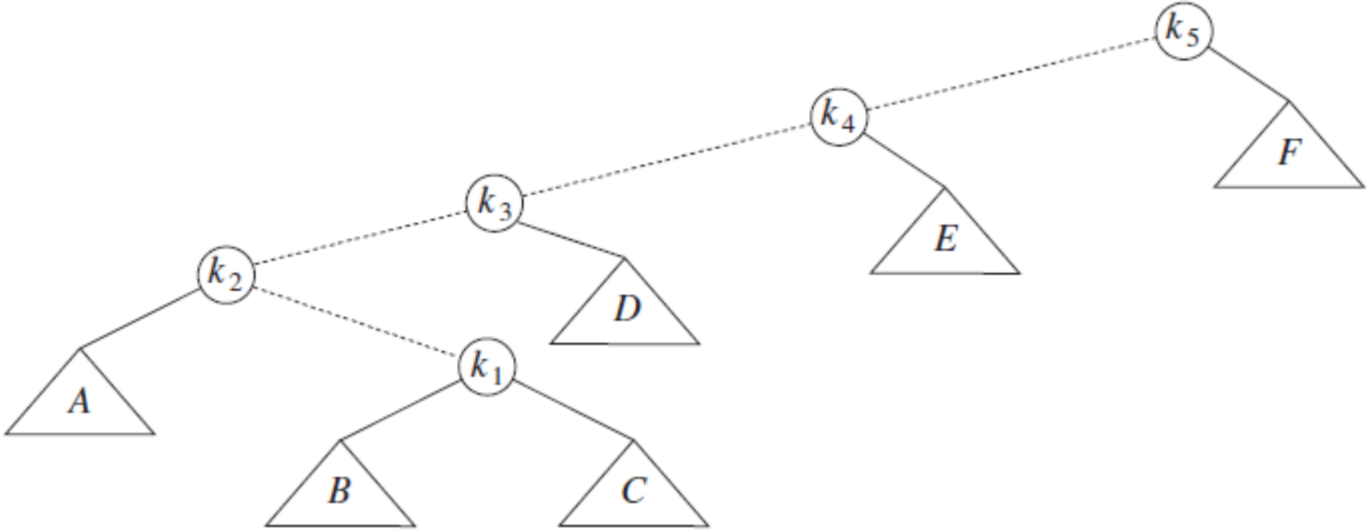
Find - **Good** Implementation

Propagate the found node to the root using **double** rotations:



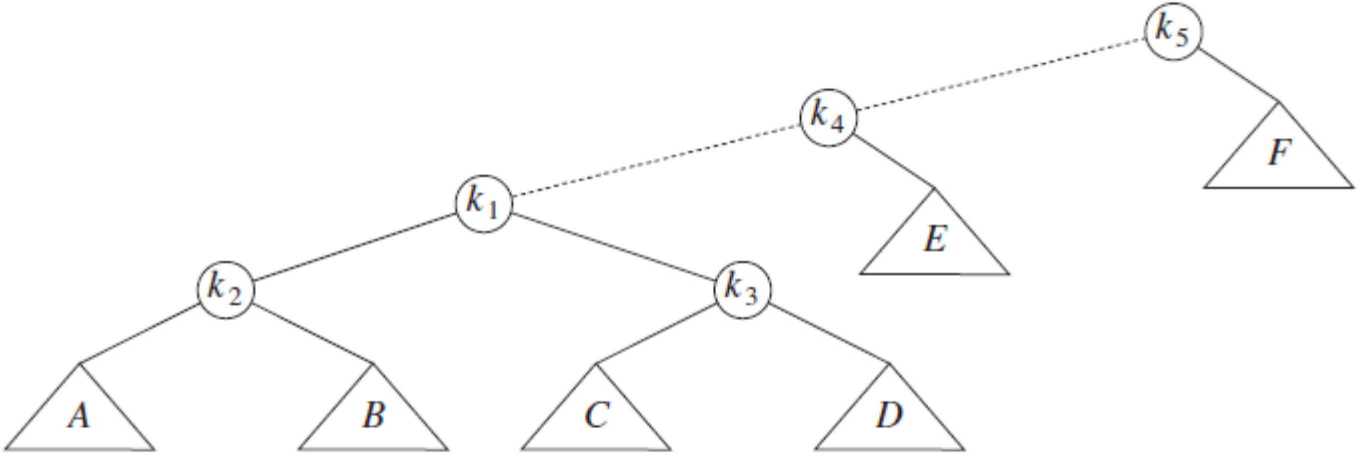
Find - **Good** Implementation

Propagate the found node to the root using **double** rotations:



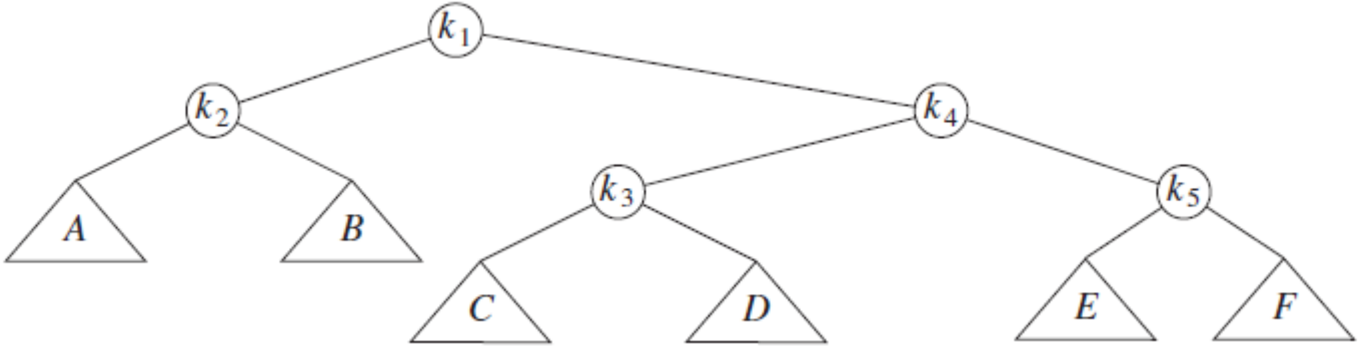
Find - **Good** Implementation

Propagate the found node to the root using **double** rotations:



Find - **Good** Implementation

Propagate the found node to the root using **double** rotations:



Find - Another Example

Propagate the found node to the root using **double** rotations: