CSE 5311 Notes 5: Trees

(Last updated 6/4/13 4:12 PM)

What is the *optimal* way to organize a *static* tree for searching?

An *optimal (static) binary search tree* is significantly more complicated to construct than an optimal list.

1. Assume access probabilities are known:

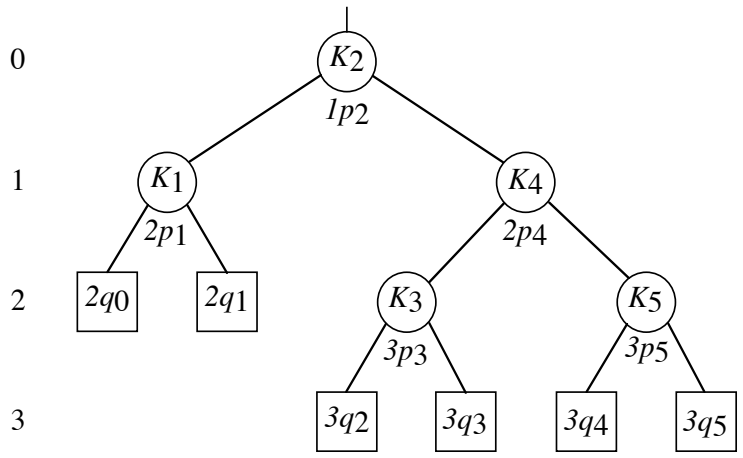
   keys are $K_1 < K_2 < \cdots < K_n$

   p_i = probability of request for K_i
   q_i = probability of request with $K_i <$ request $< K_{i+1}$
   q_0 = probability of request $< K_1$
   q_n = probability of request $> K_n$

2. Assume that levels are numbered with root at level 0. Minimize the expected number of comparisons to complete a search:

$$\sum_{j=1}^{n} p_j\left(KeyLevel(j)+1\right)+\sum_{j=0}^{n} q_j MissLevel(j)$$

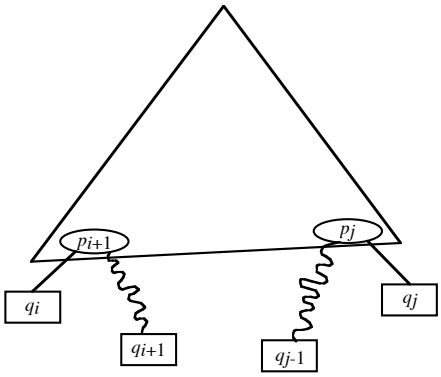
3. Example tree:



4. Solution is by dynamic programming:

   Principle of optimality - solution is not optimal unless the subtrees are optimal.

   Base case - empty tree, costs nothing to search.



$c(i,j)$ – cost of subtree with keys $K_{i+1},\cdots,K_j$

$c(i,j)$ always includes exactly $p_{i+1},\cdots,p_j$ and $q_i,\cdots,q_j$

$c(i,i)=0$ – Base case, no keys, just misses for q_i (request between K_i and K_{i+1})

Recurrence for finding optimal subtree:

$$c(i,j)=w(i,j)+\min_{i<k\le j}\left(c(i,k-1)+c(k,j)\right)$$

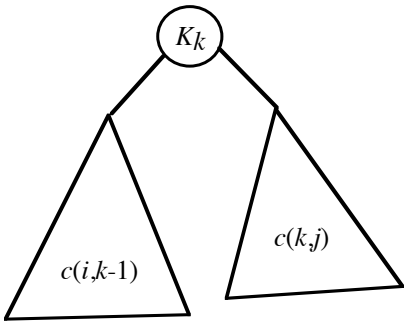
tries every possible root ("k") for the subtree with keys $K_{i+1},\cdots,K_j$

$w(i,j)=p_{i+1}+\cdots+p_j+q_i+\cdots+q_j$ accounts for adding another probe for all keys in subtree:

Left: $p_{i+1}+\cdots+p_{k-1}+q_i+\cdots q_{k-1}$

Right: $p_{k+1}+\cdots+p_j+q_k+\cdots q_j$

Root: p_k



5. Implementation: A k-family is all cases for $c(i,i+k)$. k-families are computed in ascending order from 1 to n. Suppose $n=5$:

0	1	2	3	4	5
$c(0,0)$	$c(0,1)$	$c(0,2)$	$c(0,3)$	$c(0,4)$	$c(0,5)$
$c(1,1)$	$c(1,2)$	$c(1,3)$	$c(1,4)$	$c(1,5)$	
$c(2,2)$	$c(2,3)$	$c(2,4)$	$c(2,5)$		
$c(3,3)$	$c(3,4)$	$c(3,5)$			
$c(4,4)$	$c(4,5)$				
$c(5,5)$					

Complexity: $O(n^2)$ space is obvious. $O(n^3)$ time from:

$$\sum_{k=1}^{n} k(n+1-k)$$

where k is the number of roots for each $c(i,i+k)$ and $n+1-k$ is the number of $c(i,i+k)$ cases in family k.

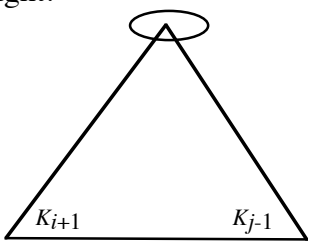
6. Traceback - besides having the minimum value for each $c(i,j)$, it is necessary to save the subscript for the optimal root for $c(i,j)$ as r[i][j].

   This also leads to Knuth's improvement:

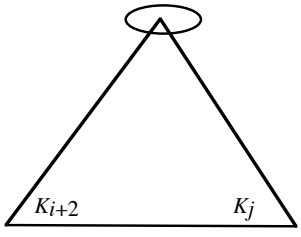
   Theorem: The root for the optimal tree $c(i,j)$ must have a key with subscript no less than the key subscript for the root of the optimal tree for $c(i,j-1)$ and no greater than the key subscript for the root of optimal tree $c(i+1,j)$. (These roots are computed in the preceding family.)

   Proof:

   1. Consider adding p_j and q_j to tree for $c(i,j-1)$. Optimal tree for $c(i,j)$ must keep the same key at the root or use one further to the right.



   2. Consider adding p_{i+1} and q_i to tree for $c(i+1,j)$. Optimal tree for $c(i,j)$ must keep the same key at the root or use one further to the left.



7. Analysis of Knuth's improvement.

Each $c(i,j)$ case for *k*-family will vary in the number of roots to try, but overall time is reduced to $O(n^2)$ by using a telescoping sum:

$$\sum_{k=2}^{n}\sum_{i=0}^{n-k}\left(r[i+1][i+k]-r[i][i+k-1]+1\right)=\sum_{k=2}^{n}\begin{pmatrix} r[1][k]-r[0][k-1]+1 \\ + \\ r[2][1+k]-r[1][k]+1 \\ + \\ r[3][2+k]-r[2][1+k]+1 \\ +\cdots+ \\ r[n-k+1][n]-r[n-k][n-1]+1 \end{pmatrix}$$

$$=\sum_{k=2}^{n}\left(r[n-k+1][n]-r[0][k-1]+n-k+1\right)$$

$$\le\sum_{k=2}^{n}\left(n-0+n-k+1\right)=\sum_{k=2}^{n}\left(2n-k+1\right)=O(n^2)$$

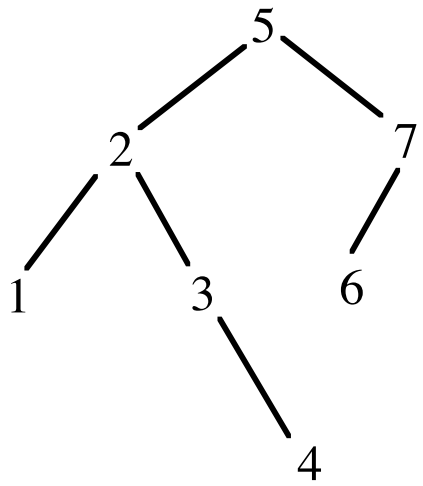
```
n=7;
q[0]=0.06;
p[1]=0.04;
q[1]=0.06;
p[2]=0.06;
q[2]=0.06;
p[3]=0.08;
q[3]=0.06;
p[4]=0.02;
q[4]=0.05;
p[5]=0.10;
q[5]=0.05;
p[6]=0.12;
q[6]=0.05;
p[7]=0.14;
q[7]=0.05;
for (i=1;i<=n;i++)
  key[i]=i;
```

```
w[0][0]=0.060000
w[0][1]=0.160000
w[0][2]=0.280000
w[0][3]=0.420000
w[0][4]=0.490000
w[0][5]=0.640000
w[0][6]=0.810000
w[0][7]=1.000000
w[1][1]=0.060000
w[1][2]=0.180000
w[1][3]=0.320000
w[1][4]=0.390000
w[1][5]=0.540000
w[1][6]=0.710000
w[1][7]=0.900000
w[2][2]=0.060000
w[2][3]=0.200000
w[2][4]=0.270000
w[2][5]=0.420000
w[2][6]=0.590000
w[2][7]=0.780000
w[3][3]=0.060000
w[3][4]=0.130000
w[3][5]=0.280000
w[3][6]=0.450000
w[3][7]=0.640000
w[4][4]=0.050000
w[4][5]=0.200000
w[4][6]=0.370000
w[4][7]=0.560000
w[5][5]=0.050000
w[5][6]=0.220000
w[5][7]=0.410000
w[6][6]=0.050000
w[6][7]=0.240000
w[7][7]=0.050000
Counts - root trick 44 without root
   trick 77
Average probe length is 2.680000
trees in parenthesized prefix
c(0,0) cost 0.000000
c(1,1) cost 0.000000
c(2,2) cost 0.000000
c(3,3) cost 0.000000
c(4,4) cost 0.000000
c(5,5) cost 0.000000
c(6,6) cost 0.000000
c(7,7) cost 0.000000
c(0,1) cost 0.160000 1
c(1,2) cost 0.180000 2
c(2,3) cost 0.200000 3
c(3,4) cost 0.130000 4
c(4,5) cost 0.200000 5
c(5,6) cost 0.220000 6
c(6,7) cost 0.240000 7
c(0,2) cost 0.440000 2(1,)
c(1,3) cost 0.500000 3(2,)
c(2,4) cost 0.400000 3(,4)
c(3,5) cost 0.410000 5(4,)
c(4,6) cost 0.570000 6(5,)
c(5,7) cost 0.630000 7(6,)
c(0,3) cost 0.780000 2(1,3)
c(1,4) cost 0.700000 3(2,4)
c(2,5) cost 0.820000 4(3,5)
c(3,6) cost 0.800000 5(4,6)
c(4,7) cost 1.000000 6(5,7)
c(0,4) cost 1.050000 2(1,3(,4))
c(1,5) cost 1.130000 3(2,5(4,))
c(2,6) cost 1.210000 5(3(,4),6)
c(3,7) cost 1.290000 6(5(4,),7)
c(0,5) cost 1.490000 3(2(1,),5(4,))
c(1,6) cost 1.630000 5(3(2,4),6)
c(2,7) cost 1.810000 5(3(,4),7(6,))
c(0,6) cost 2.050000 3(2(1,),5(4,6))
c(1,7) cost 2.230000 5(3(2,4),7(6,))
c(0,7) cost 2.680000 5(2(1,3(,4)),7(6,))

3: c(0,2) + c(3,7) + w[0][7]
   0.44     1.29     1.0    = 2.73

4: c(0,3) + c(4,7) + w[0][7]
   0.78     1.0      1.0    = 2.78

5: c(0,4) + c(5,7) + w[0][7]
   1.05     0.63     1.0    = 2.68
```



SPLAY TREES

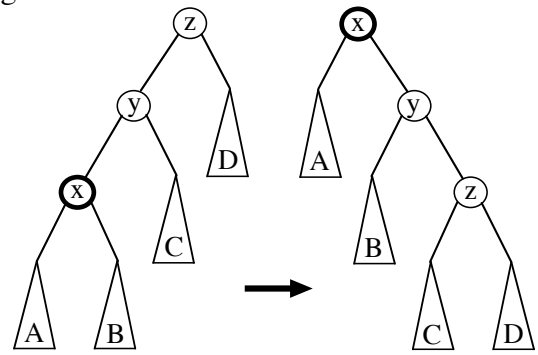
Self-adjusting counterpart to AVL and red-black trees

Advantages - 1) no balance bits, 2) some help with locality of reference, 3) amortized complexity is same as AVL and red-black trees

Disadvantage - worst-case for operation is O(*n*)

Algorithms are based on rotations to *splay* the last node processed (x) to root position.

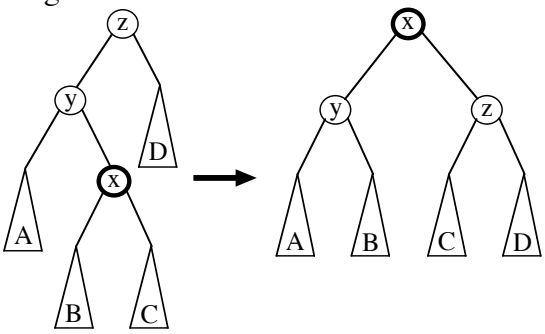
Zig-Zig:



1. Single right rotation at z.
2. Single right rotation at y.

(+ symmetric case)

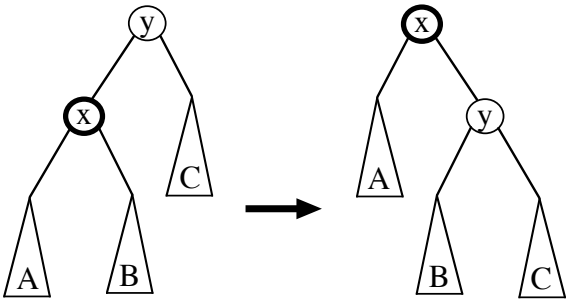
Zig-Zag:



Double right rotation at z.

(+ symmetric case)

Zig: Applies ONLY at the root



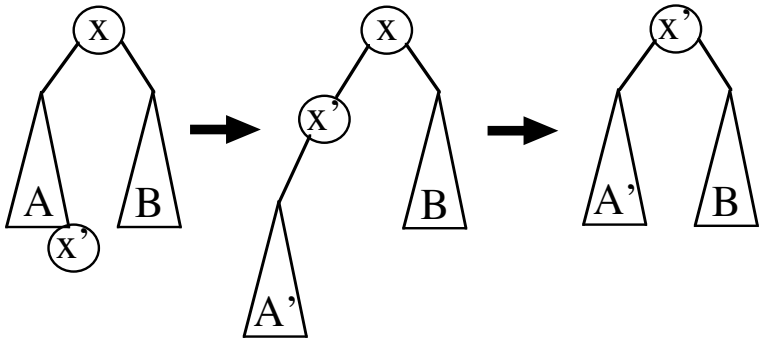
Single right rotation at y.

(+ symmetric case)

Insertion: Attach new leaf and then splay to root.

Deletion:

1. Access node x to delete, including splay to root.



2. Access predecessor x' in left subtree A and splay to root of left subtree.

3. Take right subtree of x and make it the right subtree of x'.

Amortized Analysis of Splaying for Retrieval (aside):

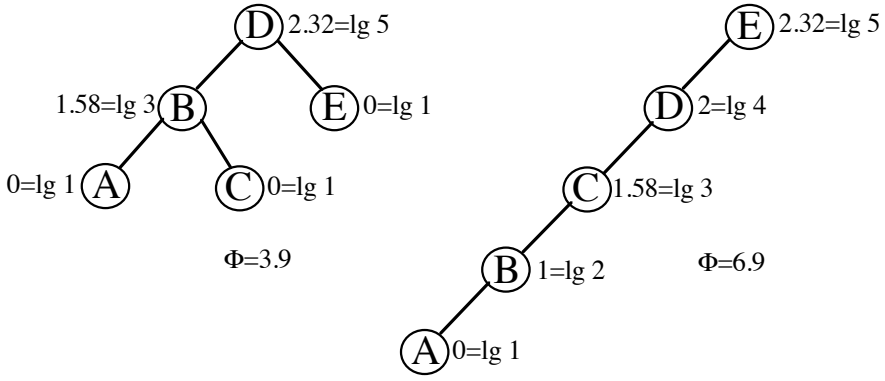
Actual cost (rotations) is 2 for zig-zig and zig-zag, but 1 for zig.

$S(x)$ = number of nodes in subtree with x as root ("size")

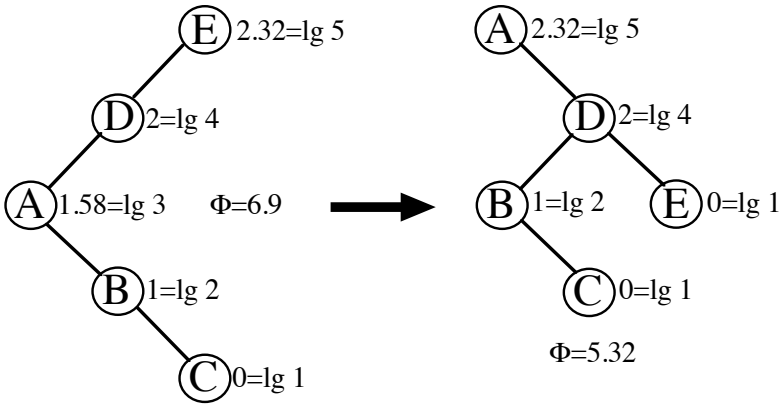
$r(x) = \lg S(x)$ ("rank")

$$\Phi(T) = \sum_{x \in T} r(x)$$

Examples:

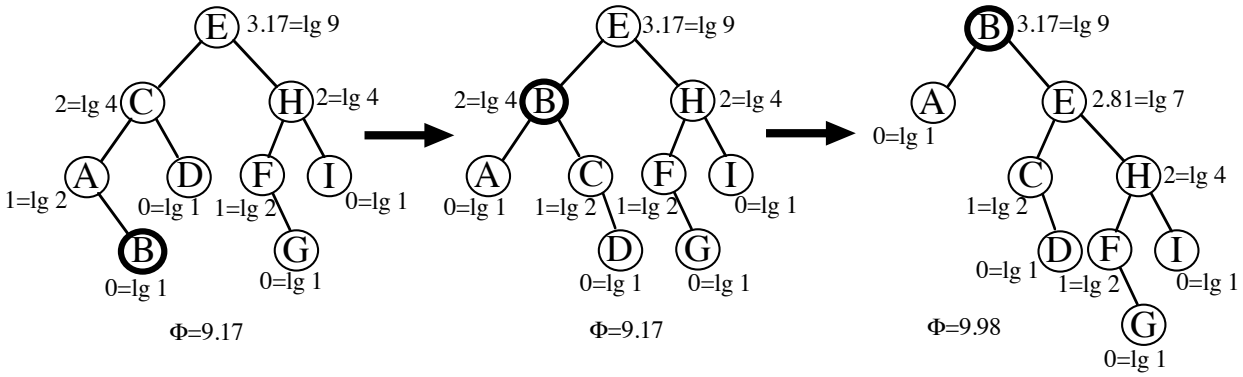


Now suppose that the leaf in the second example is retrieved. Two zig-zigs occur.



$$\sum \hat{C}_i = \sum C_i + \Phi(After) - \Phi(Before) = 4 + 5.32 - 6.9 = 2.42 \le 1 + 3\lg n = 7.96$$

Another example of splaying. There will be a zig-zag and a zig.



$$\sum \hat{C}_i = \sum C_i + \Phi(After) - \Phi(Before) = 3 + 9.98 - 9.17 = 3.81 \le 1 + 3\lg n = 10.51$$

Compute amortized complexity of individual steps and then the complete splaying sequence:

Lemma: If $\alpha > 0$, $\beta > 0$, $\alpha + \beta \le 1$, then $\lg\alpha + \lg\beta \le -2$.

Proof: $\lg\alpha + \lg\beta = \lg\alpha\beta$. $\alpha\beta$ is maximized when $\alpha = \beta = \frac{1}{2}$, so $\max(\lg\alpha + \lg\beta) = -2$.

Access Lemma:

Suppose 1) x is node being splayed

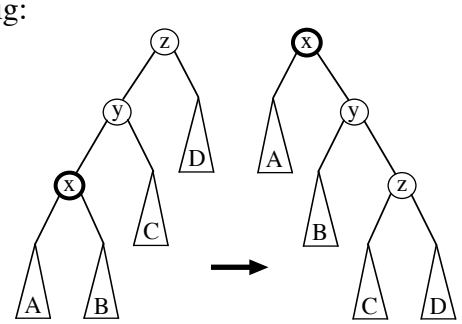
2) subtree rooted by x has

$S_{i-1}(x)$ and $r_{i-1}(x)$ before ith step
$S_i(x)$ and $r_i(x)$ after ith step

then $\hat{C}_i \le 3r_i(x) - 3r_{i-1}(x)$, except last step which has $\hat{C}_i \le 1 + 3r_i(x) - 3r_{i-1}(x)$.

Proof: Proceeds by considering each of the three cases for splaying:

Zig-Zig:



$$\begin{aligned}
\hat{C}_i &= C_i + \Phi(T_i) - \Phi(T_{i-1}) \\
&= 2 + r_i(x) + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) - r_{i-1}(z) && \text{Potential changes only in this subtree} \\
&= 2 + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) && r_i(x) = r_{i-1}(z) \\
&\le 2 + r_i(y) + r_i(z) - 2r_{i-1}(x) && r_{i-1}(x) \le r_{i-1}(y) \\
(*) \quad &\le 2 + r_i(x) + r_i(z) - 2r_{i-1}(x) && r_i(y) \le r_i(x)
\end{aligned}$$

Let $\alpha = \dfrac{S_{i-1}(x)}{S_i(x)}$, $\beta = \dfrac{S_i(z)}{S_i(x)}$, $\alpha > 0$, $\beta > 0$, $\alpha + \beta = \dfrac{S_{i-1}(x) + S_i(z)}{S_i(x)} \le 1$. ($y$ is absent from numerator)

Lemma conditions are satisfied, so $\lg\alpha + \lg\beta \le -2$. Applying logs to α and β gives:

$$r_{i-1}(x) + r_i(z) - 2r_i(x) \le -2$$

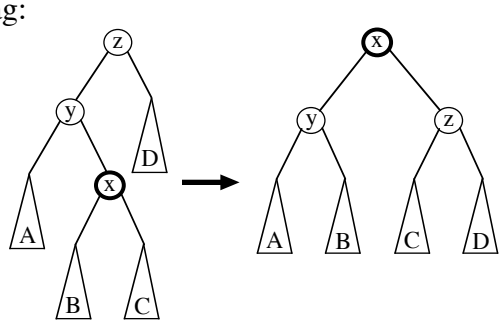
which may be rearranged as:

$$0 \le 2r_i(x) - r_{i-1}(x) - r_i(z) - 2$$

Add this to (*) to obtain:

$$\hat{C}_i \le 3r_i(x) - 3r_{i-1}(x)$$

Zig-Zag:



$$\begin{aligned}
\hat{C}_i &= C_i + \Phi(T_i) - \Phi(T_{i-1}) \\
&= 2 + r_i(x) + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) - r_{i-1}(z) && \text{Potential changes only in this subtree} \\
&= 2 + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) && r_i(x) = r_{i-1}(z) \\
(**) \quad &\le 2 + r_i(y) + r_i(z) - 2r_{i-1}(x) && r_{i-1}(x) \le r_{i-1}(y)
\end{aligned}$$

Lemma may be applied by observing that $S_i(y) + S_i(z) \le S_i(x)$ and thus

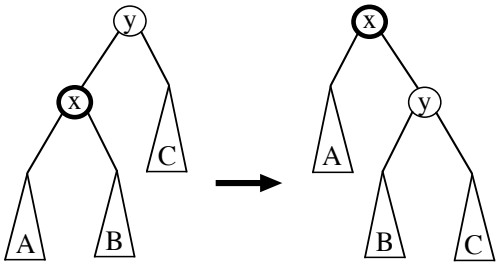
$$\frac{S_i(y)}{S_i(x)} + \frac{S_i(z)}{S_i(x)} \le 1$$

By lemma, $\lg\left(\frac{S_i(y)}{S_i(x)}\right) + \lg\left(\frac{S_i(z)}{S_i(x)}\right) \le -2$

$$r_i(y) + r_i(z) - 2r_i(x) \le -2$$
$$r_i(y) + r_i(z) \le 2r_i(x) - 2 \quad \text{which can substitute into (**)}$$

$$\begin{aligned}
\hat{C}_i &\le 2 + 2r_i(x) - 2 - 2r_{i-1}(x) \\
&= 2r_i(x) - 2r_{i-1}(x) \\
&\le 3r_i(x) - 3r_{i-1}(x) && \text{Since } r_{i-1}(x) \le r_i(x)
\end{aligned}$$

Zig:



$$\begin{aligned}
\hat{C}_i &= C_i + \Phi(T_i) - \Phi(T_{i-1}) \\
&= 1 + r_i(x) + r_i(y) - r_{i-1}(x) - r_{i-1}(y) && \text{Potential changes only in this subtree} \\
&= 1 + r_i(y) - r_{i-1}(x) && r_{i-1}(y) = r_i(x) \\
&\le 1 + r_i(x) - r_{i-1}(x) && r_i(y) \le r_i(x) \\
&\le 1 + 3r_i(x) - 3r_{i-1}(x) && r_{i-1}(x) \le r_i(x)
\end{aligned}$$

Bound on total amortized cost for an entire splay sequence:

$$\begin{aligned}
\sum_{i=1}^{m} \hat{C}_i &= \sum_{i=1}^{m-1} \hat{C}_i + \hat{C}_m \\
&\le \sum_{i=1}^{m-1} \left(3r_i(x) - 3r_{i-1}(x)\right) + 1 + 3r_m(x) - 3r_{m-1}(x) \\
&= 3r_{m-1}(x) - 3r_0(x) + 1 + 3r_m(x) - 3r_{m-1}(x) \\
&= 1 + 3r_m(x) - 3r_0(x) \\
&\le 1 + 3r_m(x) \\
&= 1 + 3\lg n \qquad \text{since } x \text{ is the root after the final rotation}
\end{aligned}$$

Asides:

If each node is assigned a positive weight and the size of node x, $S(x)$, is the sum of the weights in the subtree, other results may be shown such as:

Static Optimality: Splay trees (online) perform within a constant factor of the optimal (static) binary search tree (offline) for a sequence of requests.

But the elusive goal remains:

Dynamic Optimality *Conjecture*: Splay trees (online) perform within a constant factor of the optimal (dynamic) binary search tree (offline) for a sequence of requests.

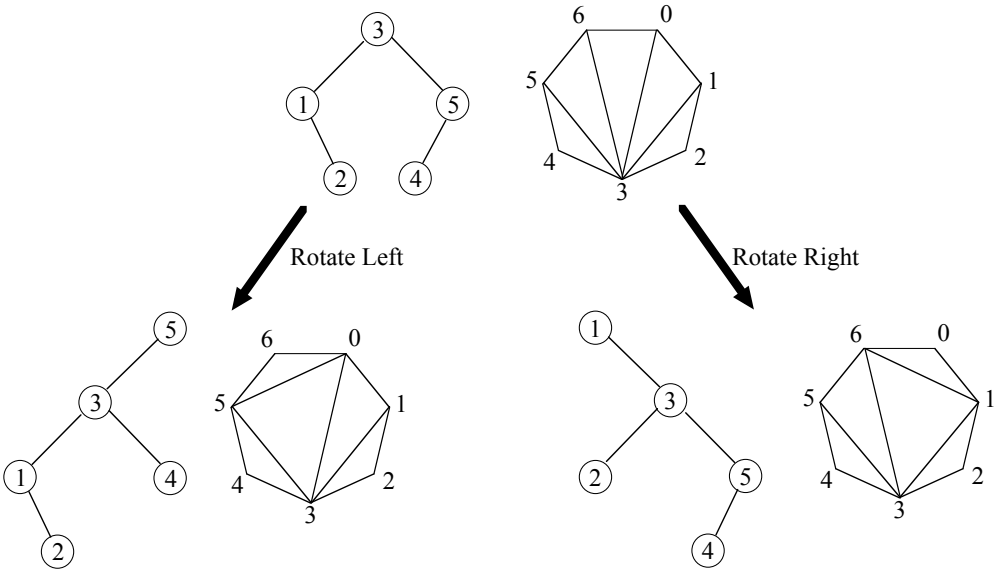
In principle, this may be attacked like "MTF is dynamically optimal". Many difficulties arise . . .

Potential Function: Minimum number of rotations to transform a BST into another BST?

Transformation: Suppose a BST with n nodes has keys $1 \ldots n$. Construct the regular $(n+2)$-gon with vertices $0, 1, 2, \ldots, n+1$. If the BST has a subtree with root i, minimum key $\min(i)$, and maximum key $\max(i)$, then include edges $\{i, \min(i)-1\}$ and $\{i, \max(i)+1\}$. Also, include edge $\{0, n+1\}$. These edges give a triangulated polygon.

(A bound of $O(\log\log n)$ as been shown:
`http://erikdemaine.org/papers/Tango_FOCS2004/`)

A BST rotation corresponds to “flipping” the diagonal of a quadrilateral:



May also rotate the original BST left at 1 and right at 5.

CSE 5311 Notes 6: Skip Lists

(Last updated 6/4/13 4:12 PM)

A randomized data structure with performance characteristics similar to treaps, i.e. another alternative to balanced binary search trees.

Paper available at `ftp://ftp.cs.umd.edu/pub/skipLists/skiplists.pdf`.

Advantages:

Simple algorithms, especially for ordered retrieval

Highly likely to perform as well as balanced trees.

Less space:

No balance bits.

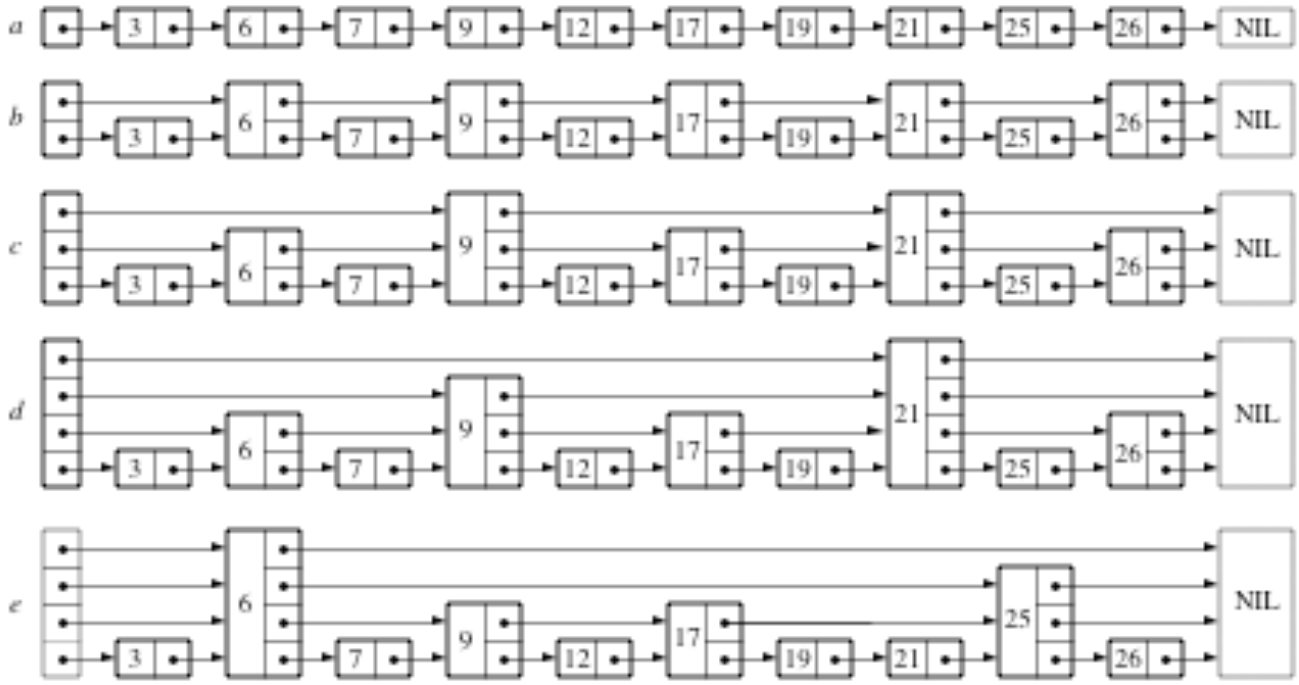
Can be tuned to use fewer pointers per node (on average).

Disadvantages:

Doesn't handle *extreme* changes in number of records well.

Bad worse case (similar to hashing), but depends on random generator rather than data.

Burden on memory management due to varying size nodes.



Design Decision :

Probability $0 < p \leq 0.5$ (0.25 typical) for determining number of pointers in a new node:

p^{k-1} = probability of a node having at least k pointers

$\frac{1}{1-p}$ = Expected number of pointers (used) in each node. (geometric sum)

Characteristics:

Low $p \Rightarrow$ Fewer pointers in structure

Increased search time (always expected to be logarithmic).

High $p \Rightarrow$ More pointers in structure

Decreased search time