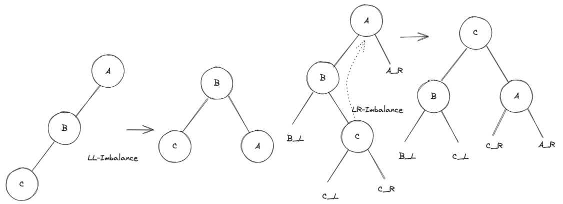
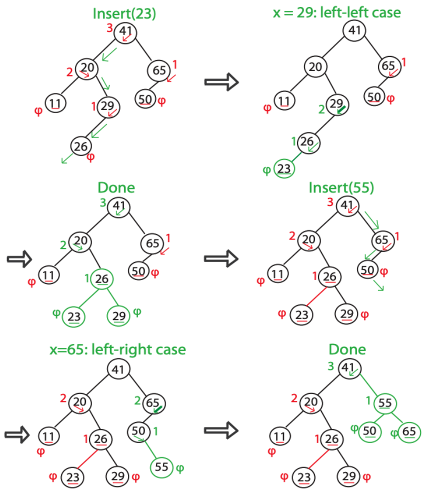
AVL Tree Balancing

- Let x be the lowest "violating" node
  - We will fix the subtree of x and move up
- Assume the right child of x is deeper than the left child of x (x is right-heavy)



Conclusions

- Can maintain balanced BSTs in $O(\log n)$ time per insertion
- Search etc take $O(\log n)$ time



Red-Black Trees

Red-Black properties:

1. Every node is either red or black
2. The root and leaves (NIL's) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node x to a descendant leaf have the same number of black nodes $= black-height(x)$.

Theorem. A red-black tree with n keys has height

$$h \leq 2lg(n+1)$$

Proof. Merge red nodes into their black parents → node has 2, 3 or 4 children. The 2-3-4 has uniform depth h' of leaves.

$$n+1 \implies n+1 \geq 2^{h'} \implies lg(n+1) \geq h' \geq \frac{h}{2}$$

$$\implies h \leq 2\ lg(n+1)$$

Red-Black Trees

Corollary. The queries $SEARCH$, MIN, MAX, $SUCCESSOR$, and $PREDECESSOR$ all run in $O(lg\ n)$ time on a red-black tree with n nodes

0: Z= root ; 1: Z.uncle and parent = red
2. Z.parent = red, uncle = black(triangle) ; 3. parent = red, Z.uncle = black(line)

- 1: recolor Z's parent, grandparent, and uncle
- 2: rotate Z.parent
- 3: rotate Z.grandpa and recolor original parent and grandpa

Analysis

- Go up the tree performing Case 1, which only recolors nodes
- If Case 2 or Case 3 occurs, perform 1 or 2 rotations, and terminate

Running time: $O(lg\ n)$ with $O(1)$ rotations.

Hash Tables

Worst case:

- Every key hashes to the same slot.
- Access time $= O(n)$ if $|S| = n$

Assume *simple uniform hashing*

Let n be the number of keys in the table, and m be the number of slots.

Define the *load factor* of T to be:

$$\alpha = \frac{n}{m} = \text{average number of keys per slot}$$

Search Cost: Expected time of *unsuccessful* search for record with given key $= \Theta(1+\alpha)$

Graphs - Intro

DFS starting from node v Running Time (without recursion): $\Theta(deg^+ v)$

DFS for all nodes of a graph. Running time: $\Theta(|V| + \sum_{v \in V}(deg^+(v)+1)) = \Theta(|V|+|E|)$

BFS for all nodes of a graph. Running time: $\Theta(|V| + |E|)$

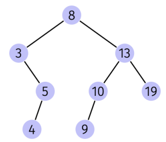
Graphs - Topological Sorting

TOPOLOGICAL-SORT(G)

```
1  call DFS(G) to compute finish times v.f for each vertex v
2  as each vertex is finished, insert it onto the front of a linked list
3  return the linked list of vertices
```

The procedure runs in $\Theta(V+E)$ time.
($O(1)$ to insert nodes in linked list)

Binary Search Trees



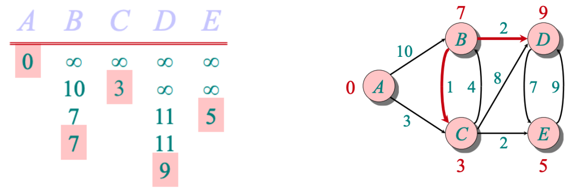
- preorder: v, then $T_{left}(v)$, then $T_{right}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- postorder: $T_{left}(v)$, then $T_{right}(v)$, then v.
  4, 5, 3, 9, 10, 19, 13, 8
- inorder: $T_{left}(v)$, then v, then $T_{right}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19

Strongly Connected Components

1. call DFS(G) from any vertex v $O(V+E)$
2. Insert each finished vertex in a stack $O(V)$
3. "Calculate G^R $O(E)$
4. Apply DFS on G^R following the stack in 2) $O(V+E)$

The trees of the DFS in 4) are the SCC of G.

Shortest Paths - Dijkstra



$S : \{A, C, E, B, D\}$

$|V|$ times

while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    $degree(u)$ times
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$

Handshaking Lemma $\Rightarrow \Theta(E)$ implicit DECREASE-KEY's.

$$\text{Time} = \Theta(V.T_{EXTR-MIN} + E.T_{DECR-KEY})$$

Q	T_{E-N}	T_{D-K}	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(lgV)$	$O(lgV)$	$O(ElgV)$

MST - Prim's and Kruskal's Algo

Prim's:

1. Start at a vertex, each time go to the unvisited vertex with the lowest edge weight

Kruskal's:

1. Start at lowest weighted edge, and create one single tree with all the lowest weighted edges in order

Rolling Hash

In the Pattern Matching, input is text string T length n and pattern string P of length m < n. Goal to determine if text has substring exactly equal to pattern.
T:CMPLbSdGCMPSNEDQCMP
P:CMPS

Algorithm 1 PatternMatch1(T,P)

```
                                                    ▷ Tot: O(mn)
for i = 0 to n − m do
    if T[i...i + m − 1] == P then                   ▷ O(m)
        return True
    end if
end for
return False
```

Algorithm 2 PatternMatch2(T,P)

```
h_p = hash(P)                                       ▷ O(m)
for i = 0 to n − m do                               ▷ O(n)
    h = hash(T[i...i + m − 1])                      ▷ O(m)
    if h_p == h then                                ▷ O(n)
        return True
    end if
end for
return False
```

Algorithm 3 PatternMatch2(T,P)

```
                                                    ▷ Tot: O(mn)
h_p = hash(P)                                       ▷ O(m)
for i = 0 to n − m do                               ▷ O(n)

    if i = 0 then
        h = hash(T[i...i + m − 1])                  ▷ O(m) Once
    else
        h = h − h(T[i − 1]) + h(T[1 + m − 1])
    end if
    if h_p == h then                                ▷ O(n)
        return True
    end if
end for
return False
```