Fakultas Informatika
School of Computing
Telkom University
Think Big! Go Forward!

Telkom University

ANALISIS ALGORITMA

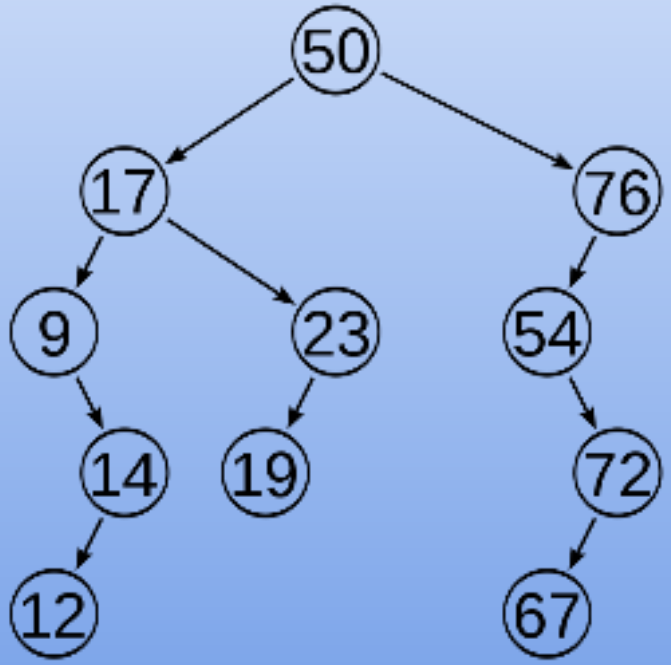
Week 05: Pengantar Struktur Data

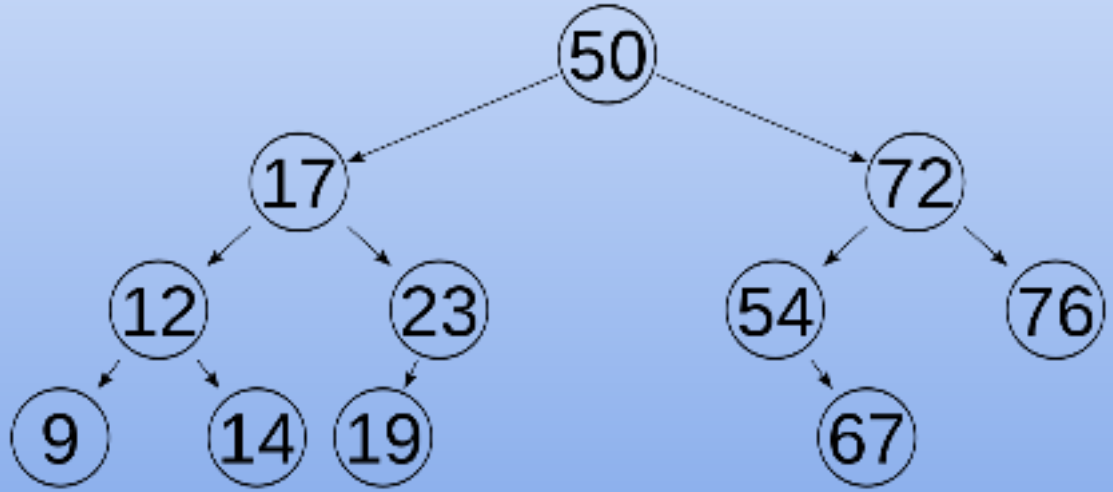
PROGRAM PASCA SARJANA INFORMATIKA
FAKULTAS TEKNIK INFORMATIKA
UNIVERSITAS TELKOM

2022/2023

Struktur Data: Binary Search Tree

Binary Search Tree, contoh:





Binary Search Tree, properti

- Properti: `leftchild^value ≤ node^value ≤ rightchild^value`
- Informasi dalam setiap node `{ value, ^parent, ^left, ^right }`
- Operasi dasar: add, delete, search, findMax, and findMin in the tree
- Mengunjungi (traversal) setiap node:
  - Preorder : the node, left subtree, then right subtree
  - Postorder : left subtree, right subtree, then the node itself
  - Inorder : left subtree, the node, then right subtree
  - Same level first (BFS) : the root, its children (left, right), the grand children (left to right), etc
- Traversal secara inorder akan memberikan data secara terurut!*

Binary Search Tree: Traversal

```
proc BSTInorder( T )
    if T != NIL then
        BSTInorder(T^left)
        process(T)
        BSTInorder(T^right)
    endif
endproc
```

```
proc BSTPreorder( T )
    if T != NIL then
        process(T)
        BSTPreorder(T^left)
        BSTPreorder(T^right)
    endif
endproc
```

```
proc BSTPostorder( T )
    if T != NIL then
        BSTPostorder(T^left)
        BSTPostorder(T^right)
        process(T)
    endif
endproc
```

```
proc BSTAltorder( T )
    clear(S)
    push(S, T)
    while not empty(S) do
        pv = pop(S)
        process(pv)
        push(S, pv^right)
        push(S, pv^left)
    endwhile
endproc
```

```
proc BST_BFS( T )
    clear(Q)
    enqueue(Q,T)
    while not empty(Q) do
        pv = dequeue(Q)
        process(pv)
        enqueue(Q, pv^left)
        enqueue(Q, pv^right)
    endwhile
endproc
```

Binary Search Tree: Search

```
func BSTSearch( T, v ) → pv
    pv := T

    found := False
    while pv != NIL and not found do

        if pv^value < v then
            pv := pv^right
        elif pv^value > v then
            pv := pv^left
        else
            found := True
        endif
    endwhile

    return pv
endfunc
```

```
func BinarySearch( A[1..n], v ) → ipos
    i := 1
    j := n
    found := False
    while i <= j and not found do
        ipos := (i+j) div 2
        if A[ipos] < v then
            i := ipos + 1
        elif A[ipos] > v then
            j := ipos - 1
        else
            found := True
        endif
    endwhile
    if not found then ipos := -1
    return ipos
endfunc
```

Binary Search Tree: Search dan FindMax

```
func Search( T, v ) → pv
    pv := T
    found := False
    while pv != NIL and not found do
        if pv^value < v then
            pv := pv^right
        elif pv^value > v then
            pv := pv^left
        else
            found := True
        endif
    endwhile
    return pv
endfunc
```

```
func FindMax( T ) → pmax
    p := T
    pmax := NIL
    while p != NIL do
        pmax := p
        p := p^right
    endwhile
    return pmax
endfunc
```

Binary Search Tree: Insert

```
proc Insert( T, v )
    q := new()
    q^value := v
    q^left := q^right = NIL
    pv := SearchParent(T, v)
    if pv == NIL then
        q^parent := NIL
        T := q
    else
        q^parent := pv
        if pv^value > v then
            pv^left := q
        else
            pv^right := q
        endif
    endif
endproc
```

```
func SearchParent( T, v ) → pv
    p := T
    pv := NIL
    while p != NIL do
        pv := p
        if p^value < v then
            p := p^right
        else
            p := p^left
        endif
    endwhile
    return pv
endfunc
```

Binary Search Tree: Delete

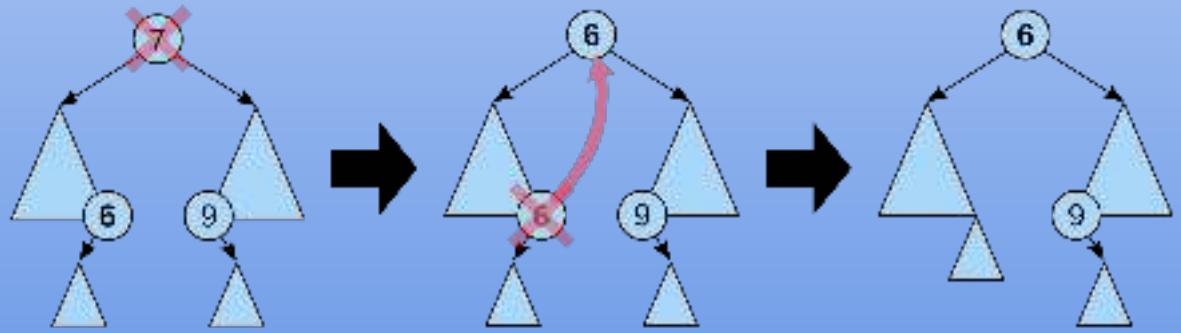
```
proc Delete( T, p )
    repl := p^left
    q := FindMax(repl)

    q^parent^right := q^left
    q^left^parent := q^parent

    q^right := p^right
    q^left := p^left
    q^parent := p^parent

    p^right^parent := q
    p^left^parent := q
    remove( p )
endproc
```

- Hanya sebagian, bagian utamanya saja
- Kasus khususnya:
  - p adalah root (e.g. node 7)
  - p adalah anak kiri/kanan dari root
  - p tidak mempunyai anak kiri/kanan
  - anak kiri terbesar(e.g. node 6) tidak mempunya ana kiri, atau sebaliknya



Binary Search Tree: Search* dan FindM*

- Semua algoritma diatas bergantung lurus pada ketinggian pohon
- Jika ada n nodes dalam pohon, kemungkinan biaya terkait tinggi pohon:
  - Pohon sangat setimbang, sehingga tingginya adalah …
  - Pohon sangat condong, sehingga maksimum tinggi menjadi …
  - Rasio tinggi sub-tree (taller/shorter) **h/s=c**, dimana **c** konstan …
  - Rasio tersebut polinomial, $\mathbf{h = s^c}$, dimana c konstan (pecahan) …

Binary Search Tree: Search* dan FindM*

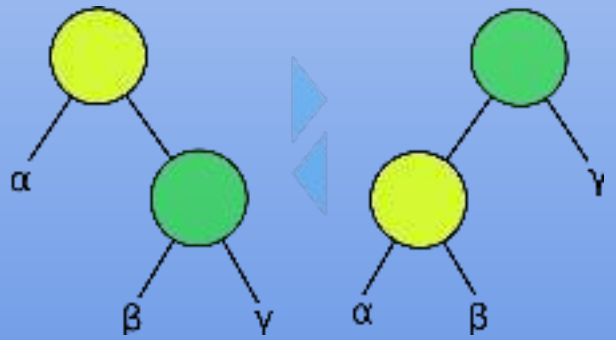
- Pohon sangat setimbang, leafs ada ½n sehingga tingginya adalah h=$O(\lg n)$
- Pohon sedikit condong, rasio subtree **s=c.h**, where **0 < c < 1**
  Jumlah node dalam pohon, $n \geq 2^s = 2^{ch}$
  → $\lg(n) \geq \lg(2^{ch})$, so $ch \leq \lg(n)$ or $h = O(\lg n)$
- Pohon sangat condong, hanya satu sisi yang ada anak
  → seperti list sederhana, ketinggian maksimum $h = O(n)$
- Pohon sangat condong, setiap level pohon paling banyak ada c nodes
  → nodes mengelompok, dan tinggi pohon $h = n/c$ or $h = O(n)$

Binary Search Tree: Rotasi, Kiri dan Kanan

```
proc RotateLeft( T, p )
    parent := p^parent
    lchild := p^left

    p^parent := parent^parent
    p^left := parent

    parent^parent := p
    parent^right := lchild
endproc
```



- Prosedur dikiri hanya ringkasan, tidak lengkap
- Kasus khusus yang harus diperhitungkan
  - p adalah root
  - p tidak mempunyai anak kiri atau kanan
  - p tidak punya saudara
- Efek rotasi: merubah tinggi (lht. gb kiri ke gb kanan)
  - Tinggi p dan anak kanan γ **berkurang satu**
  - Tinggi anak kiri β tidak berubah
  - Tinggi parent dan saudara p α **bertambah satu**

Balance Binary Search Tree

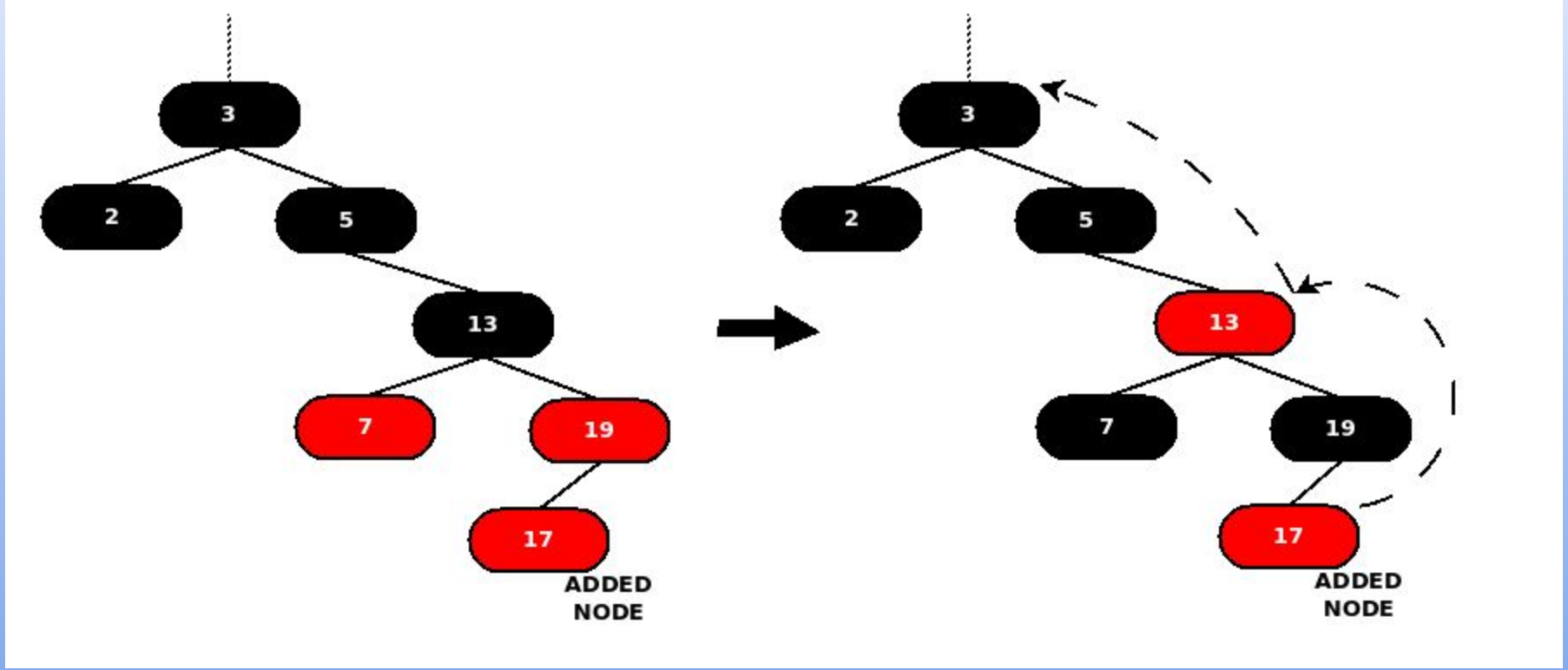
- Memanfaatkan operasi RotateLeft dan RotateRight u/ mengubah ketinggian
- Tetapi menghasilkan kesetimbangan sempurna sangat sulit dan tidak perlu
- Rebalancing diterapkan saat melakukan operasi Insertion dan Deletion
- Ada banyak variasi dalam tema ini (cari di Google atau lihat Wikipedia)
  - 2-3 tree
  - AA tree
  - AVL tree
  - Red-black tree
  - Scapegoat tree
  - Splay tree
  - Treap

Contoh: Red-Black BST

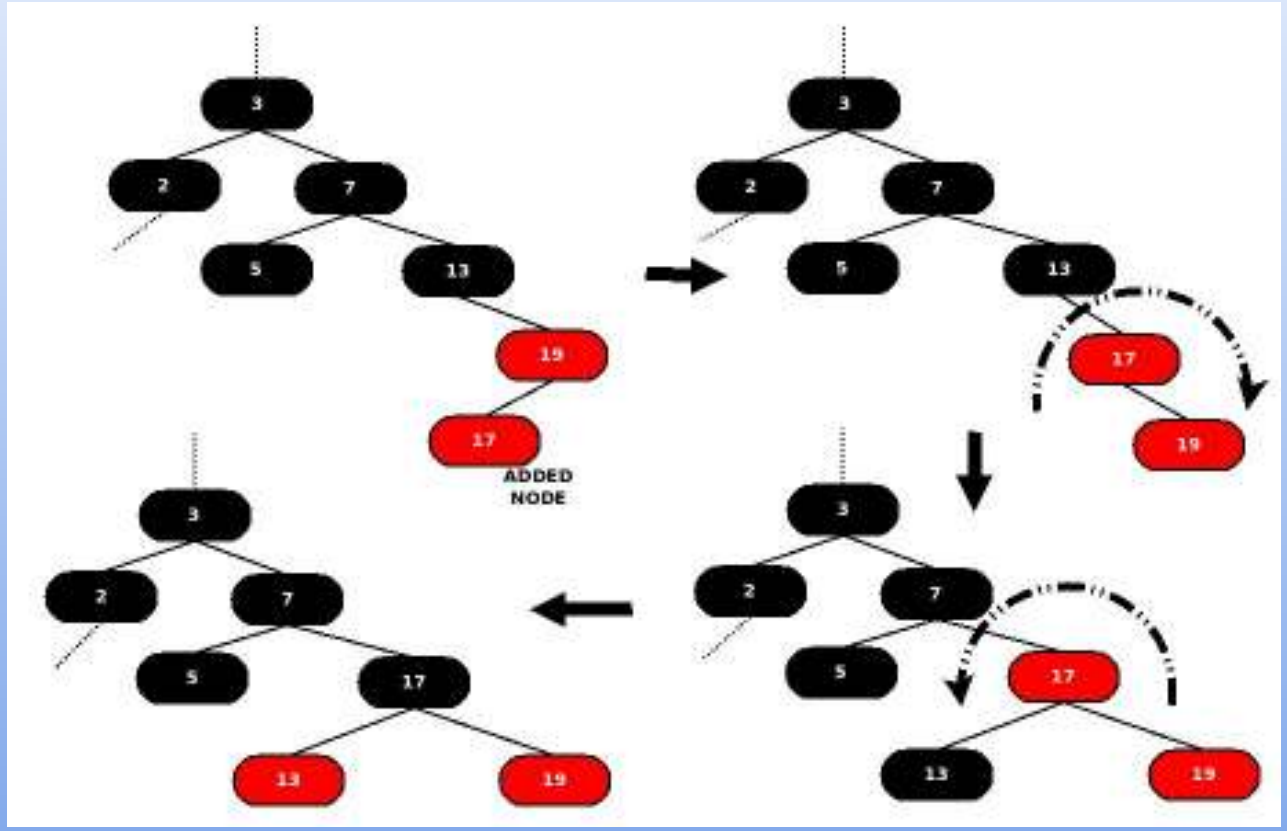
1. Setiap node di T ditandai red atau black.
2. Root dari T selalu black.
3. Setiap node NIL adalah black. (NIL merupakan ujung atau node kosong dari node leaf)
4. Jika node adalah red, kedua anaknya harus black. Karenanya tidak ada dua node red berdampingan dalam satu jalur dari root ke suatu node NIL
5. Setiap jalur dari root ke NIL mempunya jumlah node black yang sama

Kasus: Red-Black BST (Tinggi Pohon)

- Jarak black dari setiap node ke leaf manapun dibawahnya akan sama(prop 5)
  → Dalam suatu pohon, jumlah node setidaknya ada $2^{bh}-1$
- Tidak ada dua red bersebelahan dalam suatu path (prop 4)
  → Maksimum tinggi pohon (dengan red dan black) $h \leq 2bh$, atau $bh \geq \frac{1}{2}h$
- Jumlah node $n \geq 2^{\frac{1}{2}h}-1$ atau $n+1 \geq 2^{\frac{1}{2}h}$ atau
  tinggi pohon $h \leq 2\lg(n+1)$



Insert pada Red Black, kasus Bapak dan Paman adalah red

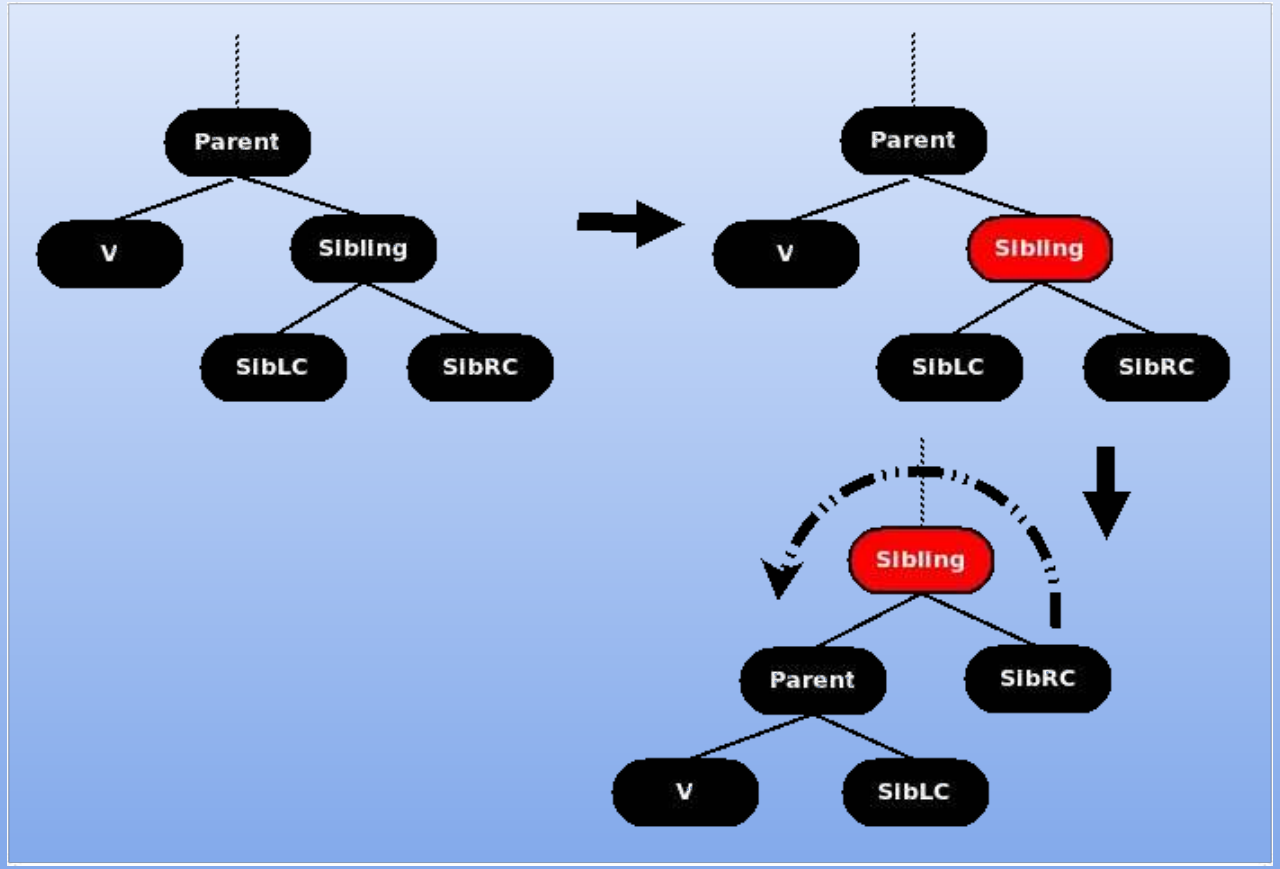


Insert pada Red Black, Bapak red dan Paman black

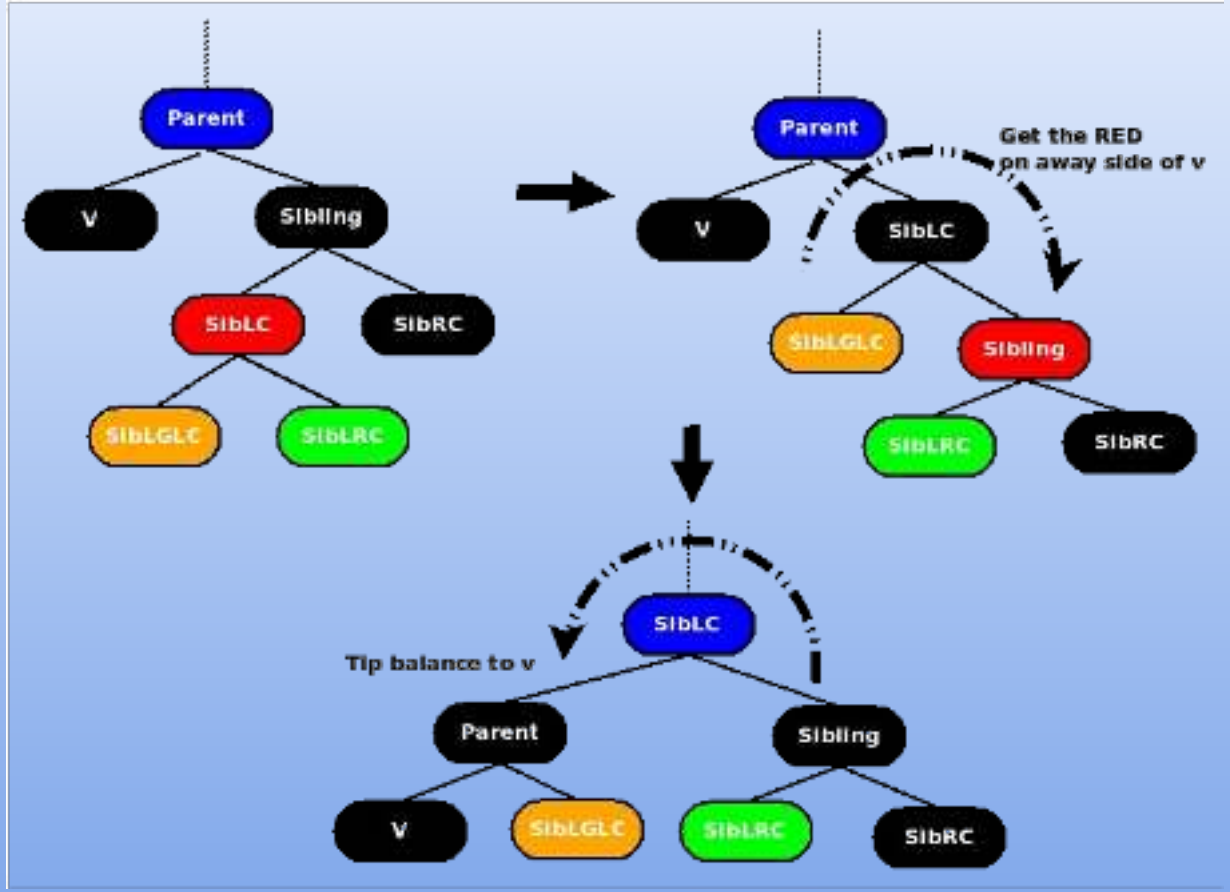
Binary Search Tree: Red Black Insert

```
proc RedBlackInsert( T, v )
    BST_Insert(T, v)
    v^color = RED
    while parentColor == RED do
        if uncleColor == RED then
            ganti warna bapak, paman, dan kakek
        else
            rotasi dg sumbu bapak jika perlu, agar
                v, bapak, dan kakek pada sisi sama
                sesuaikan peran bapak dan v
            rotasi kebalikan dg sumbu kakek
            ganti warna bapak dan kakek
        endif
        lanjutkan dengan kakek sebagai v
    endwhile
    T^color = BLACK
endproc
```

- pada sisi yang sama berarti v anak kiri bapak dan bapak anak kiri kakek, atau sebaliknya
- Root selalu **BLACK**



Delete pada Red Black, Saudara red atau kedua anak adalah black
Reorientasi untuk menggantikan ketinggian dari node v yang dihapus



Delete pada Red Black, Salah satu saudara anak adalah red ,
Reorientasi agar pohon dicondongkan dulu ke v

Binary Search Tree: Red Black Delete

```
proc RedBlackDelete( T, v )
    BST_Delete(T, v)
    while v != T and v^color == BLACK then
        if siblingColor == RED then
            rotasiMenjauhiSaudara
            ganti warna saudara dan bapak
        endif
        if anak saudara semua BLACK then
            ganti saudara menjadi RED
            lanjut ke node v^parent
        else
            rotasi menjauh pada saudara, jika perlu
                agar semua RED jauh dari v
            rotasi pada bapak kesisi yang dihapus
            ambil warna bapak untuk saudara
            warnai bapak dan saudara jauh anak BLACK
    endwhile
    T^color = BLACK
endproc
```

Intinya pada sisi yang dihapus, tinggi akan berkurang:

- Jika yang dihapus node is red, tidak masalah
- Jika anaknya red dia akan menggantikan yang dihapus (manjadi black)
- Jika saudara red, rotasi pada bapak kearah yang dihapus
- Jika kedua keponakan black, set saudara red untuk mengurangi tinggi sisi lawan, dan update keatas agar tidak ada red berdampingan
- Sisanya, reorientasi pohon ke arah yang akan dihapus

Perbandingan Binary Search Tree

Primitives	U-Array	O-Array	Linked List	O Linked List	D Linked List	(Balance) BST	
Search	O(n)	O(n)/O(lg n)	O(n)	O(n)	O(n)	**(**O(lg n)**)** O(n)	
FindMax	O(n)	O(1)	O(n)	O(n)/O(1)	O(1)	**(**O(lg n)**)** O(n)	
Insert	O(1)	O(n)	O(1)	O(n)	O(n)/O(1)	**(**O(lg n)**)** O(1)*	
Delete	O(n)	O(n)	O(n)	O(n)	O(1)	**(**O(lg n)**)** O(1)*	

*) Diluar proses SearchParent atau FindMax