Obscure Binary Search Trees

ScapeGoat Tree:

Search Time is O(logN)

Insertion and Deletion is O(logN) amortized

A self-balancing tree like AVL, RBTree, etc., with partial rebuilding

The balancing idea is to make sure that nodes are a- size balanced, meaning size of left and right subtree are at most a- (size of node).

An a-weight balanced node is defined as meeting a relaxed weight criterion:

Size(left) <= a*size(node)

Size(right) <= a*size(node)

The value of the a- can be decided based on the requirement, the higher the a- = fewer balances, and insertion faster, but deletion and search slower.

A BST that is a-weightbalanced must also be a-height balanced

$$height(tree) \leq \lfloor \log_{1/a}(size(tree)) \rfloor$$

ScapeGoat tree doesn't require extra space per node, e.g:

RBTree nodes are required to have color, in ScapeGoat Tree, we only have left, right, and parent pointer in the struct of the Node. Use of parent is use for simplicity and can be omitted.

Rebuilding the subtree:

- ➔ Convert the subtree to the most possible balanced BST
- ➔ Store in-order traversal of BST in an array
- ➔ Build a new BST from array by recursively dividing it into ½

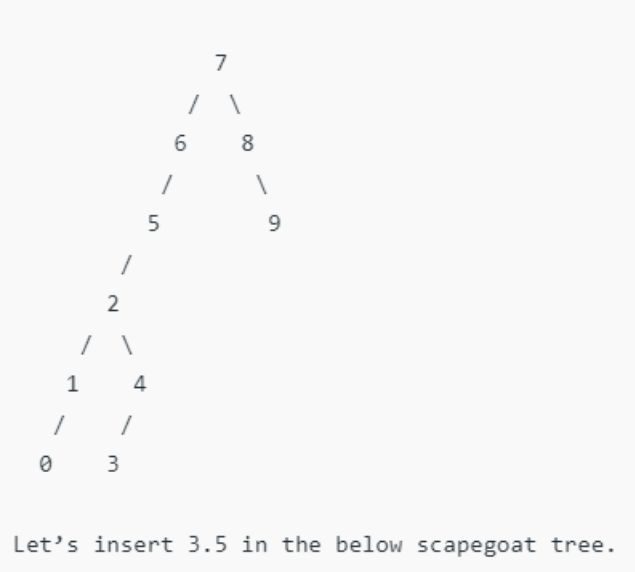
```
void rebuild(Node<T> u) {
  int ns = size(u);
  Node<T> p = u.parent;
  Node<T>[] a = (Node<T>[]) Array.newInstance(Node.class, ns);
  packIntoArray(u, a, 0);
  if (p == nil) {
    r = buildBalanced(a, 0, ns);
    r.parent = nil;
  } else if (p.right == u) {
    p.right = buildBalanced(a, 0, ns);
    p.right.parent = p;
  } else {
    p.left = buildBalanced(a, 0, ns);
    p.left.parent = p;
  }
}
int packIntoArray(Node<T> u, Node<T>[] a, int i) {
  if (u == nil) {
    return i;
  }
  i = packIntoArray(u.left, a, i);
  a[i++] = u;
  return packIntoArray(u.right, a, i);
}
Node<T> buildBalanced(Node<T>[] a, int i, int ns) {
  if (ns == 0)
    return nil;
  int m = ns / 2;
  a[i + m].left = buildBalanced(a, i, m);
  if (a[i + m].left != nil)
    a[i + m].left.parent = a[i + m];
  a[i + m].right = buildBalanced(a, i + m + 1, ns - m - 1);
  if (a[i + m].right != nil)
    a[i + m].right.parent = a[i + m];
  return a[i + m];
}
```

Example of Insertion:

Functions used:

```
boolean add(T x) {
  // first do basic insertion keeping track of depth
  Node<T> u = newNode(x);
  int d = addWithDepth(u);
  if (d > log32(q)) {
    // depth exceeded, find scapegoat
    Node<T> w = u.parent;
    while (3*size(w) <= 2*size(w.parent))
      w = w.parent;
    rebuild(w.parent);
  }
  return d >= 0;
}
```

```
boolean remove(T x) {
  if (super.remove(x)) {
    if (2*n < q) {
      rebuild(r);
      q = n;
    }
    return true;
  }
  return false;
}
```



Let's insert 3.5 in the below scapegoat tree.

ScapeGoat tree with 10 nodes, and height of 5, Assuming a- is 2/3

$$q/2 \leq n \leq q \ .$$

q = counter(maintains upper bound of nodes), n = number of node

Equation 8.1:

$$\log_{3/2} q \leq \log_{3/2} 2n < \log_{3/2} n + 2 \ .$$

Tree yg diatas memiliki equation seperti ini

$$q = n = 10 \text{ and height } 5 < \log_{3/2} 10 \approx 5.679.$$

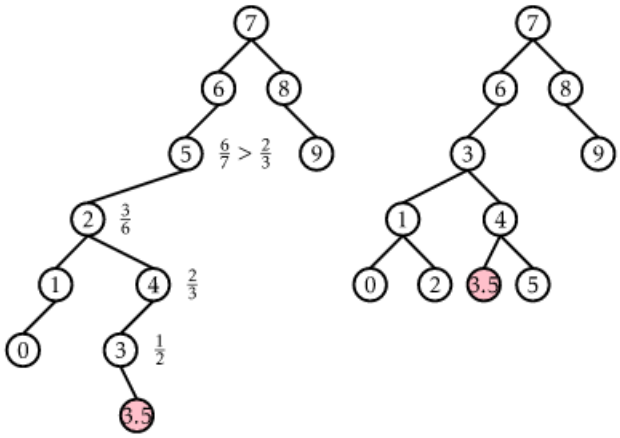


Figure 8.2: Inserting 3.5 into a `ScapegoatTree` increases its height to 6, which violates Equation 8.1 since $6 > \log_{3/2} 11 \approx 5.914$. A scapegoat is found at the node containing 5.

To implement add(x) , we increment n & q then use the usual algorithm, search for x then add new leaf u with u.x = x, if u < log3/2 q, we leave and don't do anything else,

Else, we need to reduce the height just like in figure 8.2, to fix it we need to walk from u back up to root looking for scapegoat (w), w is a very unbalance node with a property thats

$$\frac{\texttt{size(w.child)}}{\texttt{size(w)}} > \frac{2}{3}$$

Once we found the scapegoat w, we destroy the subtree rooted at w, and rebuild to a balanced BST, therefore the hight decreases by at least I so that the height of the tree is once again at most log3/2 q.

The implementation of remove(x), we search for x and remove it using the usual algo for removing node from a BST, then we decrement n, but leave q unchanged, finally we check if q > 2n, and if so , we rebuild the entire tree into perfectly balanced binary tree and set q = n.

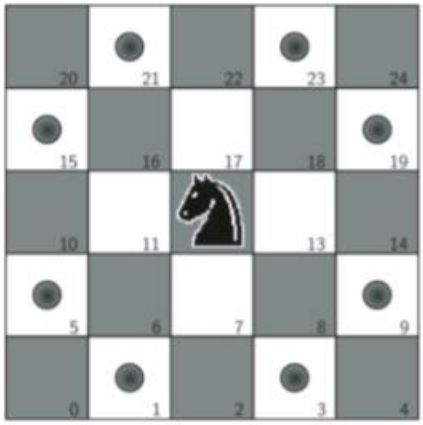
Comparison to other self-balancing BST:

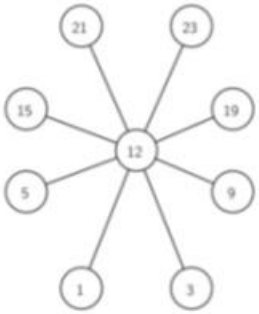
RBTree & AVL: Time complexity of search, insert and delete is 0(logN)

Splay Tree: Worst case of search ,insert and delete is O(n), but amortized time complexity is O(log n)

ScapeGoat Tree: like splay tree, it is easy to implement, worst case of O(log N). Worst case and amortized time complexity of insert and delete are same as Splay tree.

Knight's Travails

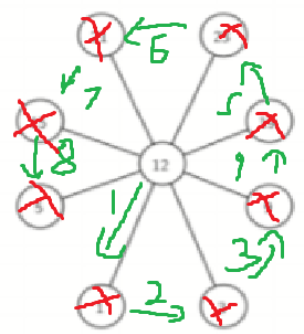




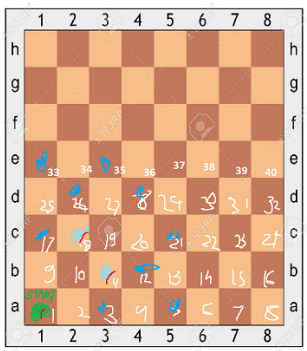
For this problem, we can see it as shortest path in unweighted graph, therefore we use BFS to solve it.

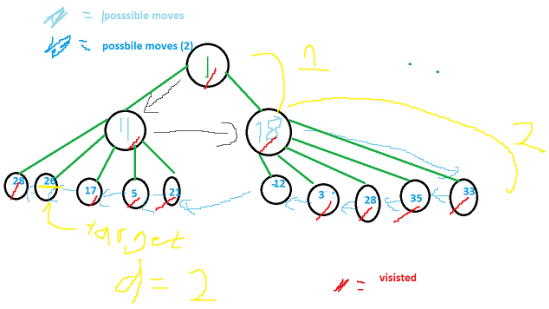
We try all 8 possible positions reachable, if reachable position is not already visited and is inside the board, we push this state into queue with distance 1 more than its parent state.

Then we return distance of target position when it gets pop out from queue.



e.g: horse is at 1, want to get to 26





Text Editor

Link Code:

https://github.com/kevicebryan/ProjectFinale-BootcampDS/blob/main/texteditor.cpp

Tree Simulation

a) **Insert 80, 35, 20, 100, 25, 30, 45, 40, 50, 37**
Remove 35, 25, 30, 45, 80

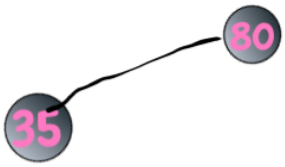
AVL Tree

Insert 80

Karena tree kosong, langsung create node

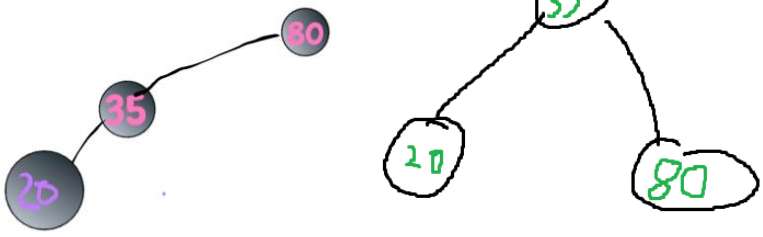
Insert 35

35 < 80, insert di kirinya 80, balance factor masih <= 1 , >= -1 tidak perlu rotasi



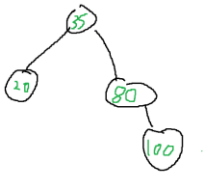
Insert 20

20 < 80 ke kiri, 20 < 35, insert di kiri 35, sudah bukan complete tree alias degenerate, maka perlu di rotate, karena dia condong kiri (saat cek 80 BFnya <-1), maka kita akan right rotation dengan 80



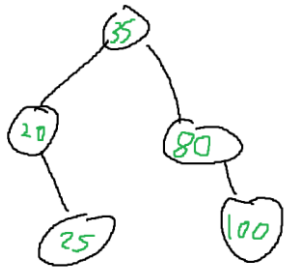
Insert 100

100 > 35 ke kanan, 100 > 80 , insert 100 dikanan 80, balance factor masih <= 1 , >= -1, tidak perlu rotasi



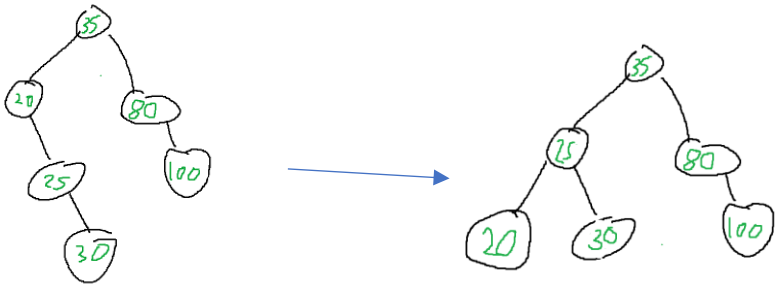
Insert 25

20 <25 ke kiri, 25 > 20, tarok dikanannya 20, balance factor masih blm ada yg masuk ke IF , tidak perlu rotate



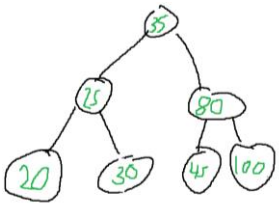
Insert 30

30 < 35 ke kiri, 30 > 25 insert di kanannya 25, karen condong ke kanan(saat cek BF 20 dia >1), maka left rotation dengan 20



Insert 45

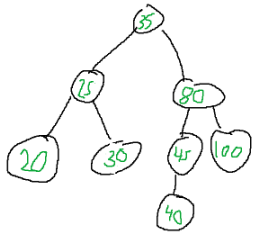
45 > 35 ke kanan, 45 < 80, 45 > 40, insert 45 di kiri 80, balance factor blm menyimpang AVL tidak perlu rotate



Kevin Bryan – Final Project

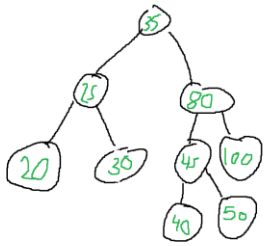
Insert 40

40 > 35 ke kanan, 40 < 80 ke kiri, 40 < 45, insert di kiri 45, belum menyimpang balance factor tidak perlu rotate



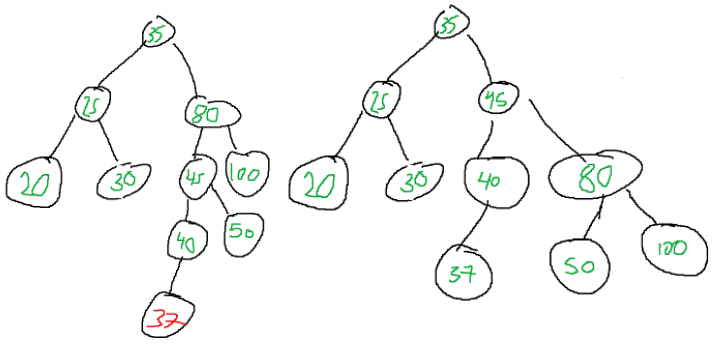
Insert 50

50 > 35 ke kanan, 50 < 80, ke kiri, 50 > 45 insert di kanan 45, balance factor blm menyimpang AVL , tdk perlu rotate



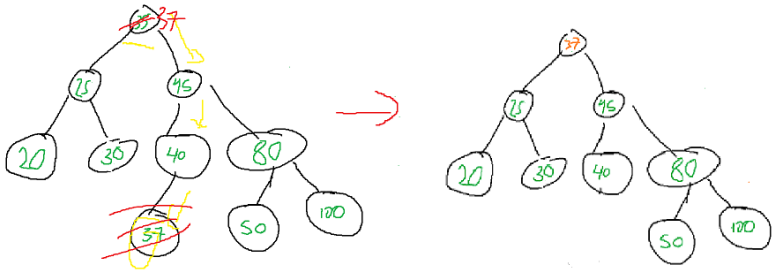
Insert 37

37 > 35, ke kanan, 37 < 80, ke kiri, 37 < 45, ke kiri, 37 < 40 insert di kiri 40, karena dia condong kiri (saat cek 80 bfnya < -1), maka right rotation 80, dan anak kanan dari 45 yaitu 50 menjadi anak kiri 80



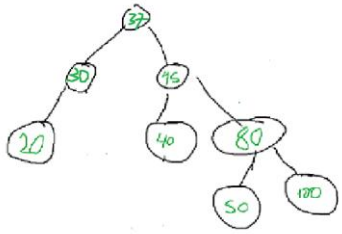
Remove 35

Search 35, ketemu 35, anaknya ada 2, cari successor dari 35, ketemu 37, hapus 35, gantikan dengan 37, lalu hapus node successor, cek balance factor, masih tidak ada yg menimpang, tidak perlu rotate



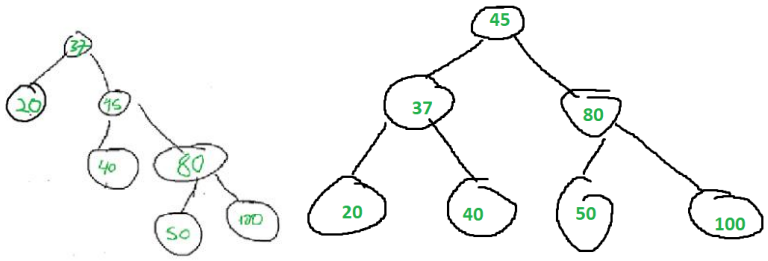
Remove 25

Search 25, ketemu 25, anaknya ada 2, cari successor dari 25, ketemu 30, timpah 30 dengan 25, lalu hapus node successor, cek bf , masih tidak ada yg menyimpang, tidak perlu rotate



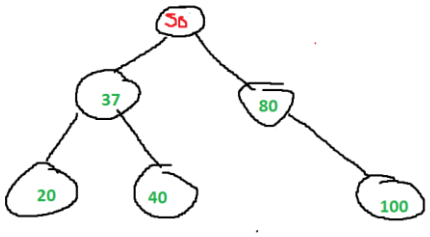
Remove 30

Search 30, ketemu 30, anaknya hanya 1 dikiri yaitu 20, timpah 30 dengan 20 dan delete node 20 yg sblmnya, cek BF factor, BF factornya menyimpang di 37, condong kanan, maka kita left rotation dengan 45 sebagai root baru, dan anak kiri si 45 menjadi anak kanan 37



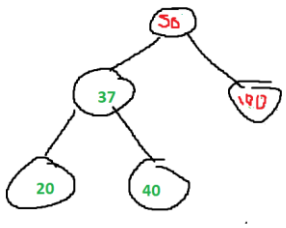
Remove 45

Search 45, ketemu 45, anakny 2, cari successor, ketemu 50, timpah 45 dengan 50, lalu hapus node successor, cek BF, masih tidak ada yg menyimpang jdi tidak perlu rotasi



Remove 80

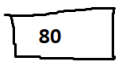
Search 80, anakny hanya 1 , timpah 80 dengan 100, lalu hapus 100 yg lama, cek BF, blm menyimpang jdi tidak perlu rotate



2-3 Tree

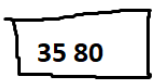
Insert 80,

Tree masih kosong, createnode 80 langsung



Insert 35,

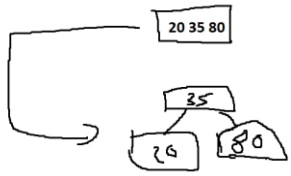
35 < 80, insert 30 dikirinya 80, karena masih ada ruangan masukan dalam 1 node



Kevin Bryan – Final Project

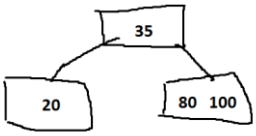
Insert 20,

20 < 35, masukan ke kiri 35, karena room sudah full, maka median kita push ke atas, mediannya 35, 35 kita push ke atas



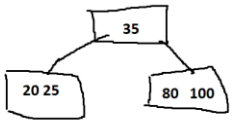
Insert 100,

100 > 35 , ke kanan, 100 > 80, masukan 100 ke kanan 80, masih ada ruangan langsung tarok



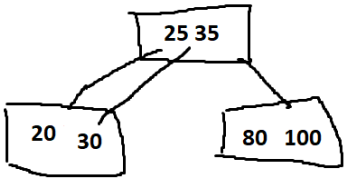
Insert 25,

25 < 25 ke kiri, 25 > 20, masukan 25 di kanan 20



Insert 30,

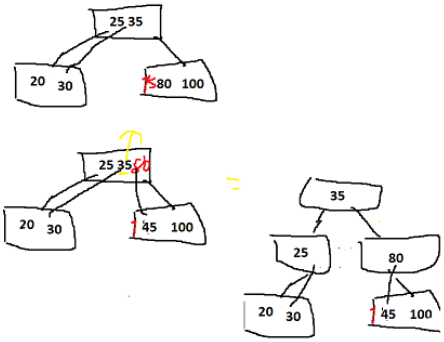
30 < 35, ke kiri, 35 > 25, ke kanandari 25, ruangannya penuh, push median , mediannya adalah 25, push 25 ke atas



Kevin Bryan – Final Project

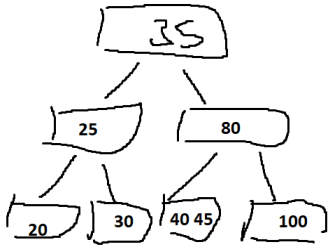
Insert 45,

40 > 35, masukan ke kanan, 45 < 80 insert di kirinya 80, ruangannya penuh, push median ke atas, 80 adalah median push 80 ke atas,di node atas ada 25 35 80, sudah penuh push median lgi ke atas, mediannya 35 push 35 ke atas lagi



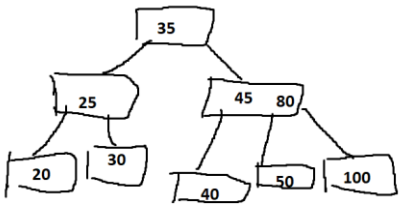
Insert40,

40 > 35, kekanan, 40 < 80, ke kiri, 40 < 45, insert 40 di kirinya 45



Insert 50,

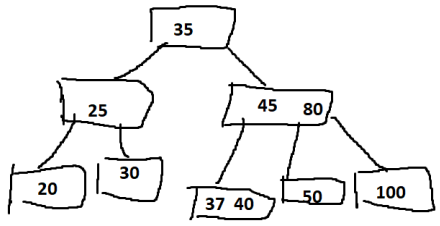
50 > 35, ke kanan, 50 < 80 ke kiri, 50 > 45 masukan ke kanan 45, karena ruangan penuh, makah push median ke atas, mediannya 45, 45 push ke atas



Kevin Bryan – Final Project

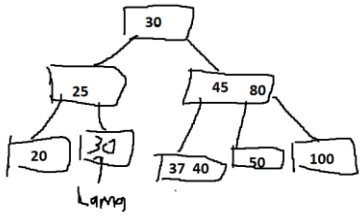
Insert 37

37 > 35, ke kanan, 37 < 45, ke kirinya 45, 37 < 40, masukan 37 di kirinya 40

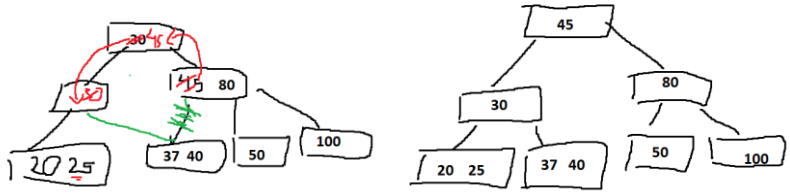


Remove 35,

Search 35, ketemu 35, 35 merupakan internal, sehingga kita cari predecessor dari 35, yaitu 30, timpah 35 dengan 30 dan delete node si 30,

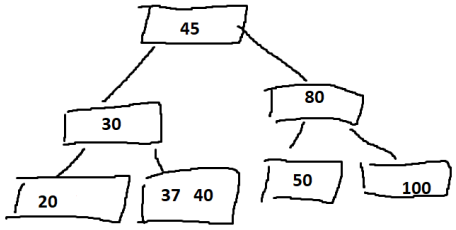


saat 30 yg lama ingin di delete ia berada di leaf dan merupakan node dengan minimal key, kita delete si 30 anggap ksoong, sehingga dia cek sibling kiri, kiri sudah minimum tidak bisa, kanan tidak ada, jadi dia merge dengan sibling kiri yaitu 20 , dan 25 pun akan turun kebawah. Node yg tdinya berisi 25, akan pinjam sibling kanan terkiri , ketemu 45, parent dari node tsb (30) dijadikan dirinya, dan 45 dijadikan parent, lalu anak kiri dari yg yg dipinjam menjadi akan kanannya



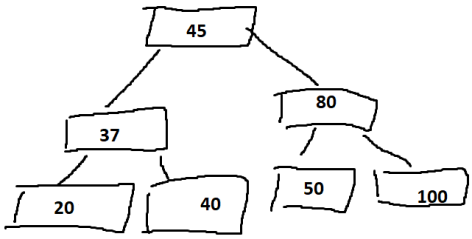
Remove 25,

Search 25, 25 berada di leaf node yg lebih dari minimum, sehingga langsung didelete saja



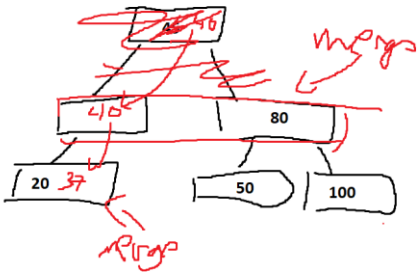
Remove 30,

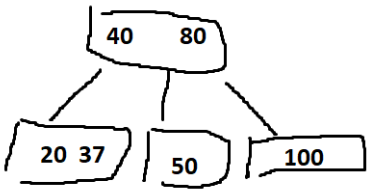
Search 30, 30 merupakan internal node, cari predecessor, ketemu 20, timpah 30 dengan 20, 20 adalah minimal key, pinjam sibling kanan terkiri yaitu 37, 37 akan dijadikan parent, dan parent (20) turun menggantikan node yg tdinya berisi 20, dan setelah itu kita hapus 37 yg ada di sibling kanan



Remove 45,

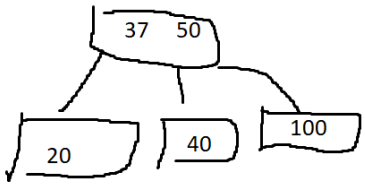
Search 45, 45 ada di root dan merupakan internal, cari predecessor, ketemu 40, 40 timpah node yg berisi 45, node asal (predecessor yg berisi 40) merupakan minimal key, cek sibiling kiri/kanan semuanya minimal, sehingga merge dengan sibling kiri, cek ke parent parent juga memiliki kondisi yg sama sehingga merge lagi





Remove 80

Search 80, 80 merupakan internal, cari predecessor ketemu 50, timpah 80 dengan 50, delete 50, saat delete50 cek sibling kiri ketemu yg tidak minimal jadi pinjam sibling kiri terkanan yaitu 37, 37 menggantikan parent terkiri yaitu 40, dan 40 turun ke node yg seblumny berisi 50.



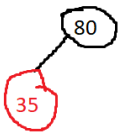
Red Black Tree

Insert 80,

Tree masih kosong, sehingga langsung createnode dengan warna hitam

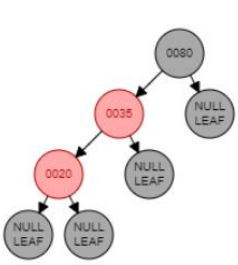
Insert 35,

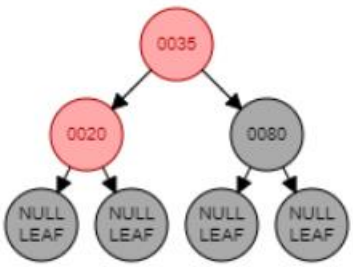
35 < 80, insert di kiri 80, set 35 ke merah, karena parentnya hitam tidak masalah

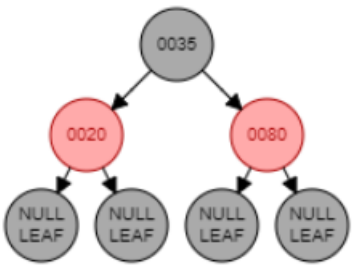


Insert 20,

20 < 80, ke kiri, 20 < 35, insert di kiri 35, set ke merah, karena parentnya merah dan node merah,, node adalah anak kiri, dan parent adalah anak kiri,cuk unclenya, unclenya hitam, maka right rotate, lalu recolor

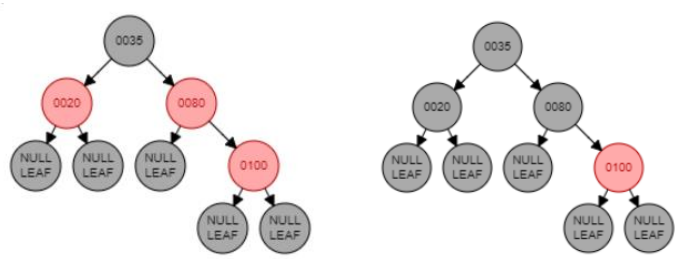






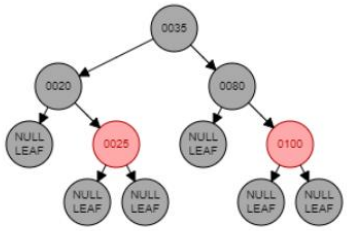
Insert 100,

100 > 35 ke kanan, 100 > 80 insert di kanan 80, set ke merah, karena node merah dan parent merah, cek unclenya node, karena unclenya merah juga, maka warnakan parent menjadi hitam, dan grandparent jdi merah, karena grandparent root, dihitamkan



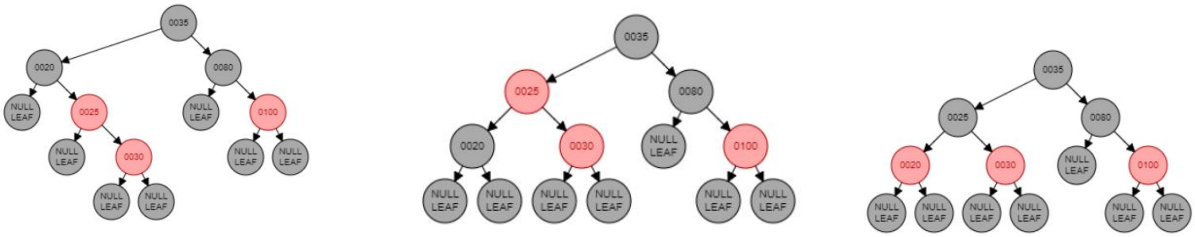
Insert 25,

25< 35, ke kiri , 25 > 20 , insert di kanannya 20, set ke merah, karena parent hitam, tidak perlu apa2



Insert 30,

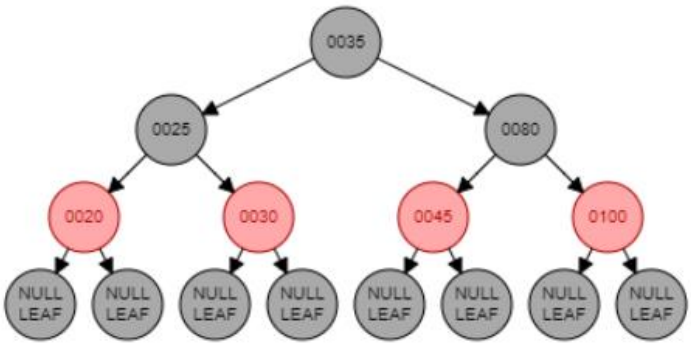
30 < 35, ke kiri, 30>20, ke kanan, 30 > 25, insert di kanannya 25 , set ke merah, node dan parent keduanya merah, unclenya hitam, karena keduanya merupakan anak kanan, maka left rotate, dan recolor



Kevin Bryan – Final Project

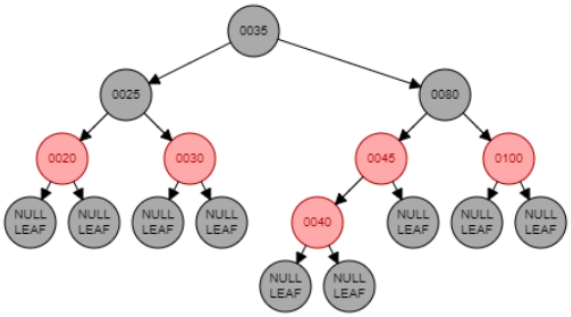
Insert 45,

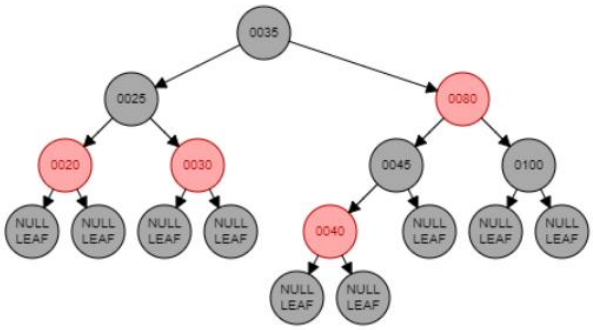
45 > 35, ke kanan, 45 < 80 insert di kiri 80, set 45 ke merah karena parent hitam tidak perlu diapa-apakan



Insert40,

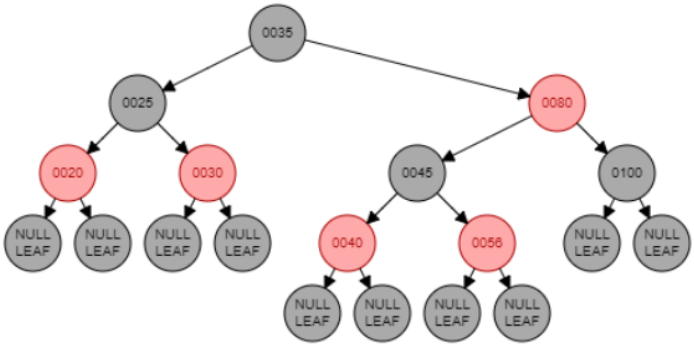
40 > 35, ke kanan, 40 < 80 ke kiri, 40 < 45, insert di kiri 45, set ke merah, karena parent dan nodenya merah keduanya, maka cek uncle dari node, uncle merah, maka cukup recolor grandparent jadi merah, dan kedua child grandparent jadi hitam





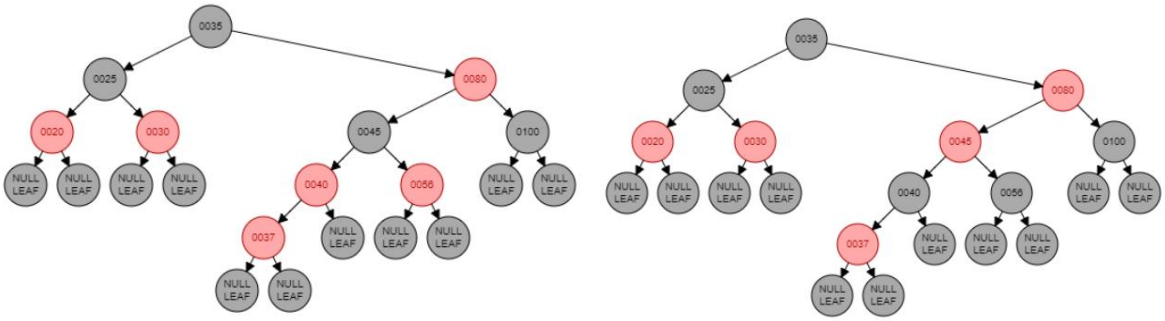
Insert 50,

50 > 35, ke kanan, 50 < 80 ke kiri, 50> 45, insert di kanannya 45, set ke merah, karena parentnya hitam , tidak perlu diapa-apakan

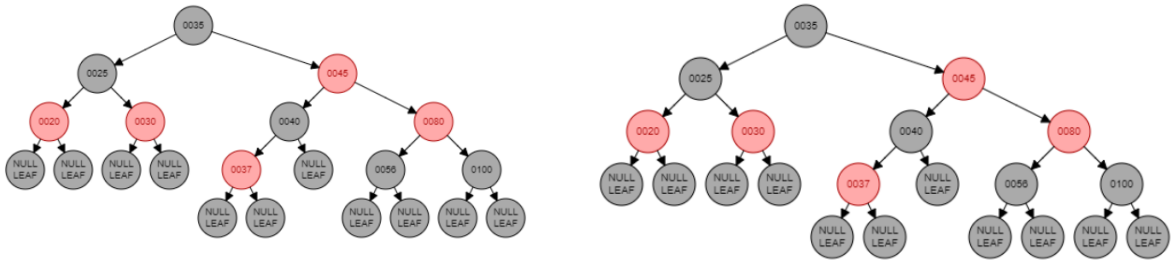


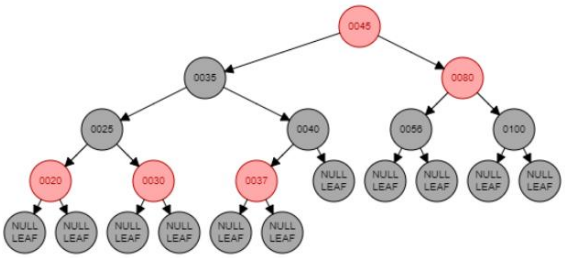
Insert 37

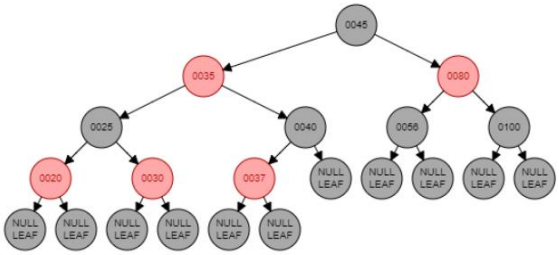
37 > 35 , ke kanan, 37 < 80, ke kiri, 37 < 45 ke kiri, 37 < 40, insert di kiri 40, set ke merah, karena parent merah dan dia merah, maka cek uncle, unclenya merah, maka cukup recolor grandparent menjadi merah dan kedua child grandparent jadi hitam.



Solve untuk grandparentnya, dan karena node grandparent merah, dan parent dari node tsb merah juga, dan unclenya hitam, maka karena node adalah anak kiri, dan parent anak kanan, right-left rotation, dan lalu recolor

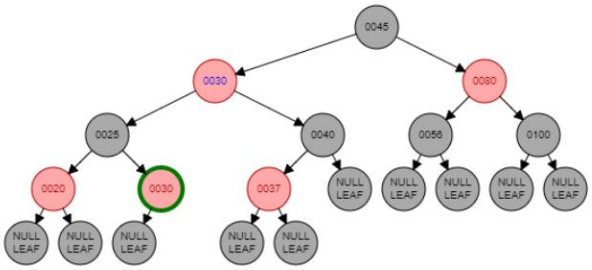


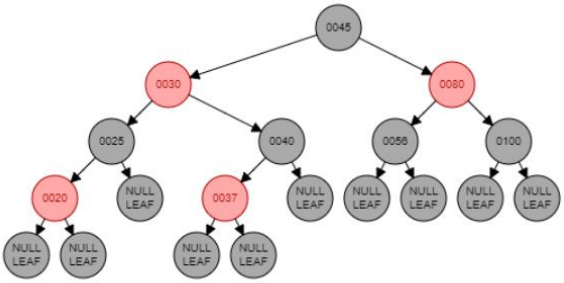




Remove 35,

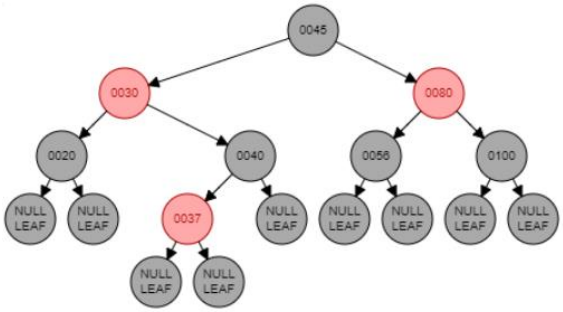
Search 35, 35 memiiki 2 child, maka cari predecessor keetemu 30, timpah 35 dengan 30, lalu hapus 30 yg lama, karena merah tidak perlu rotation





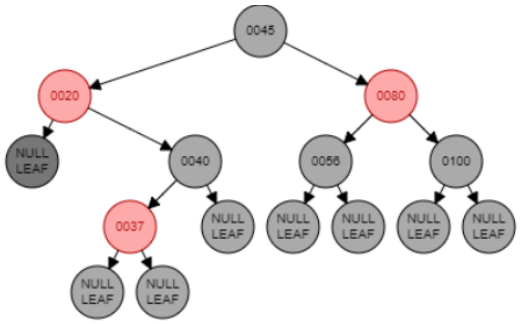
Remove 25,

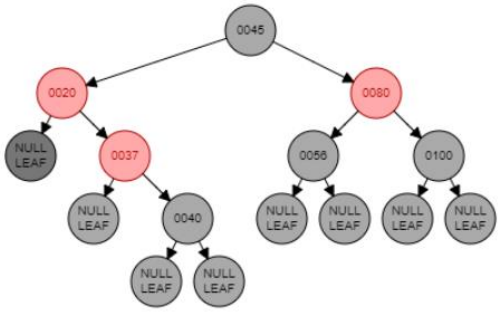
Search 25, 25 tidak punya anak kanan, cukup timpah 25 dengan 20 dan delete 20 yg lama



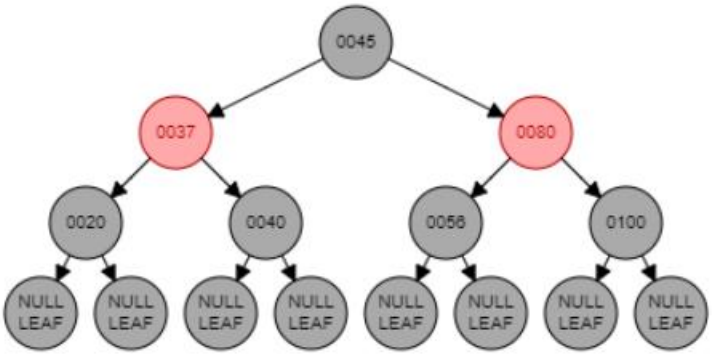
Remove 30,

Search 30, 30 memiliki 2 anak, cari predecessor, ketemu 20, timpah 30 dengan 20 , dan hapus 20 yg lama, recolor node yg di delete menjadi hitam, saat recolor menjadi double black, double black adalah anak kiri, dan nephew kanannya hitam, rotate treenya supaya nephew kanan menjadi merah, maka kita akan melakukan right left rotation



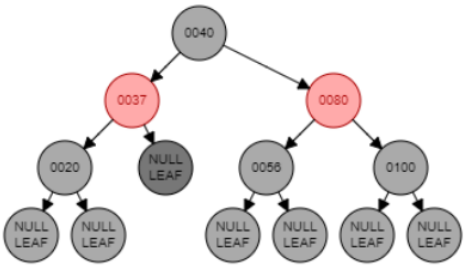


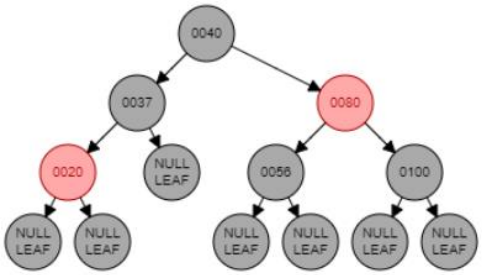
Setelah rotation ini recolor 37 menjadi hitam, dan 40menjadi merah, lalu left rotate, dan ubah double black menjadi black biasa



Remove 45,

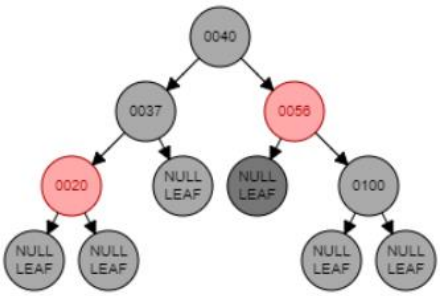
Search 45, 45 memiliki 2 anak, maka cari predecessor ketemu 40, timpah 45 dengan 40, dan hapus node predecessor, dan saat kita mewarnai node yg dihapus menjadi double black, karena double blacknya memilki sibling hitam dan 2 nephew hitam, maka tinggal kita push blacknya ke atas dan warnai sibling menjadi merah

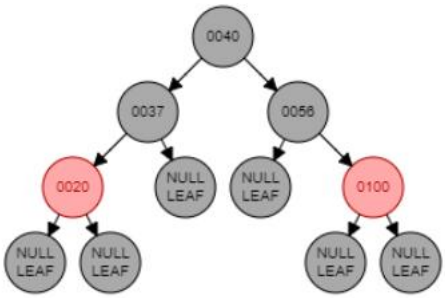




Remove 80

Search 80, 80 memiliki 2 anak, cari predecessor ketemu 56, timpah 80 dengan 56, dan delete node predecessor dan warnai hitam, menjadi double black, karena siblingnya hitam, dan memiliki 2 nephew yg hitam , maka kita push blacknya ke atas, dan recolor sibling menjadi merah





b) Insert 40, 75, 50, 90, 60, 65, 63, 100, 95, 30
Remove 63, 75, 40, 60, 95

AVL Tree

Insert 40

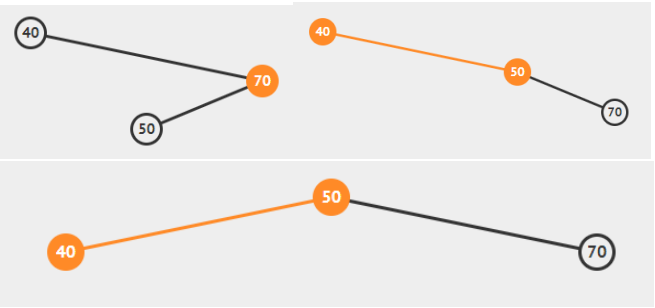
Tree masih kosong, masukin create 40 langsung

Insert 75,

75 > 40m isert di kanan 40, karena BF masih blom menyimpang AVL, tidak perlu rotate

Insert 50,

50 > 40, ke kanan, 50 < 75 , insert di kiri 70, karena BF sudah menyimpang, dia condong kiri lalu kanan(dari bawah), maka kita lakukan right left rotation

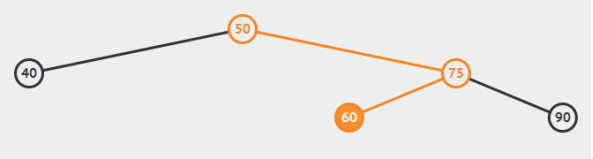


Insert 90

90 > 50, ke kanan, 90 > 75, insert di kanan 75, karena BF masih blm nyimpang tidak perlu rotate

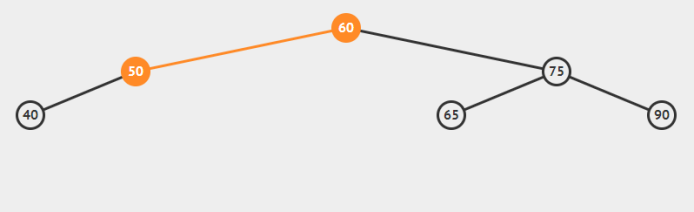
Insert 60

60 > 50, ke kanan, 60 < 70, insert di kiri 70, BF masih blm nyimpang jadi tidak perlu rotate



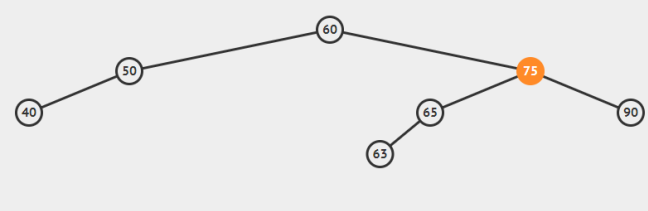
Insert 65

65 > 50 ke kanan, 65 < 75 ke kiri , 65 > 60 insert di kanan, karena BF menyimpang dan dia condong kanan lalu kiri maka kita lakukan right left rotaton



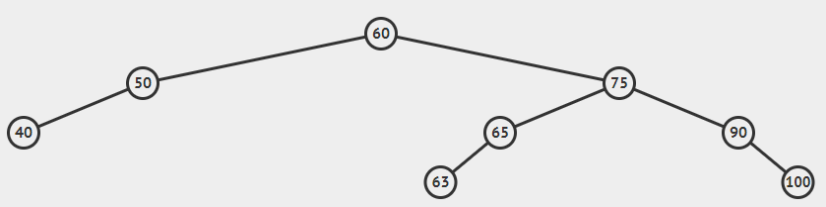
Insert 63

63 > 60, ke kanna, 63 < 70 ke kiri, 63 < 65 insert di kiri 65, BF tidak menyimpang maka tidak perlu rotate



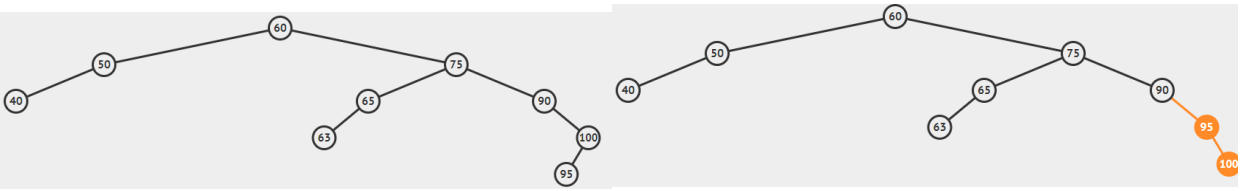
Insert 100

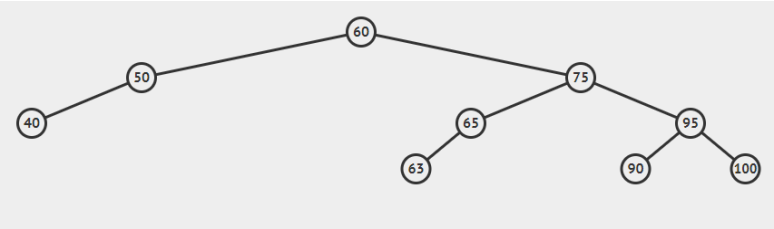
100 > 60 , ke kanan, 100 > 75 ke kanan, 100 > 90, insert di kanan 90



Insert 95

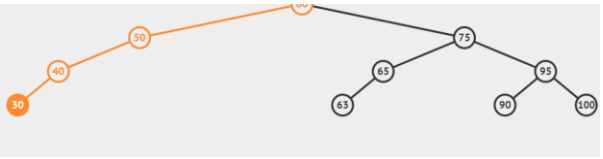
95 > 60 ke kanan , 95 > 75, ke kanan, 95 > 90, ke kanan, 95 < 100, insert di kiri 100, karena BF menyimpang dan dia condong kiri lalu kanan, maka kita lakukan right-left rotation

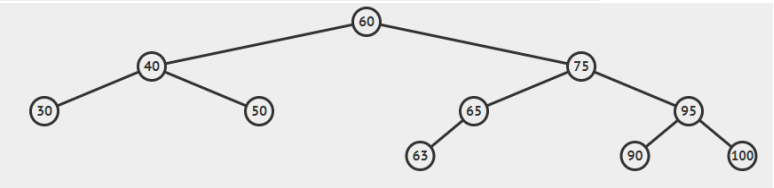




Insert 30

30 < 60, ke kiri, 30 < 50 ke kiri , 30 < 40 insert di kiri 40, karena BF menyimpang, lalu dia condong kiri , maka kita right rotation



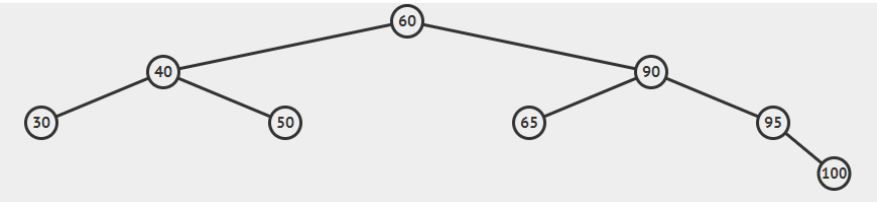


Remove 63

Search 63, 63 tidak memiliki anak, delete 63, recur cek balance factor dari parent sampai ke root, semuanya aman, tidak perlu rotate

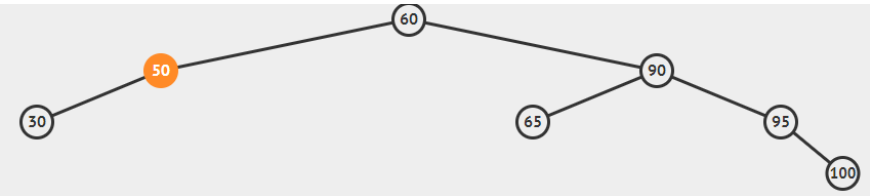
Remove 75

Search 75, 75 memiliki 2 anak, maka kita cari successor dari 75, ketemo 90, timpah 75 dengan 90 dan hapus 90 yg lama, lalu cek BF sampai root, karena aman, maka no rotation



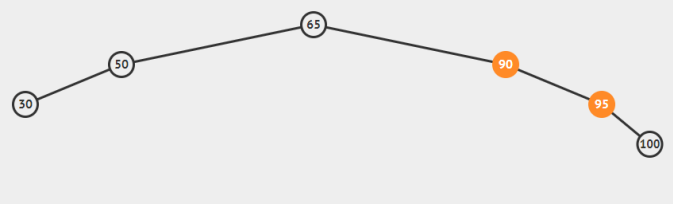
Remove 40

Search 40, 40 memilki 2 anak, cari successor ketemu 50, timpah 40 dengan 50, lalu hapus node successor, lalu recur cek BF ke root, karena aman maka tidak perlu rotate



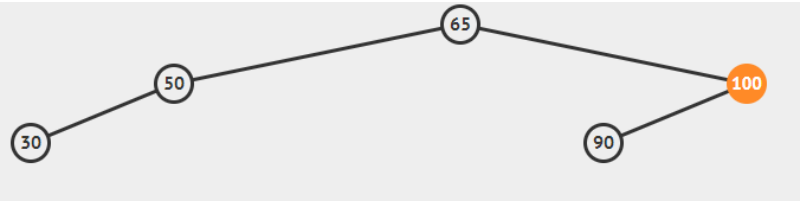
Remove 60

Search 60, ketemu 60 di root, 60 punya 2 anak, cari successor dari 60 ketemu 65, maka kita timpah 60 dengan 65, dan hapus node 65, lalu kita recur cek BF, BF menyimpang condong ke kanan, maka kita lakukan left rotation



Remove 95

Search 95, ketemu 95 memiliki 2 anak, cari successor dapat 100, timpah 95 dengan 100, dan delete 100 yg lama, karena BF tidak menyimpang tidak perlu rotations



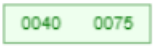
2-3 Tree

Insert 40

Tree kodong, maka langsung create node saja

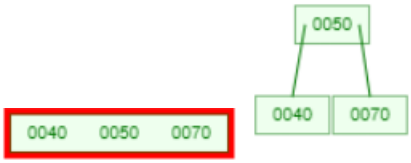
Insert 75,

75 > 40, masih ada ruangan insert di kanan 40



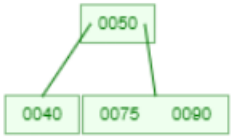
Insert 50,

50 > 40 , insert di kaan 40, nodenya penuh , maka push median, mediannya dalah 50, 50 kita push ke atas



Insert 90

90 > 50, ke kanan, 90 > 75 insert di kanan 75 karena masih ada ruangan tidak perlu diapa-apakan



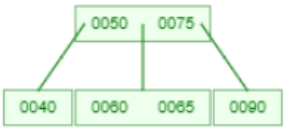
Insert 60

60 >50, ka kanan, 60 < 75 maka insert di kiri 75, karean sudah penuh kita push median ke atas, mediannya adalah 75

Kevin Bryan – Final Project

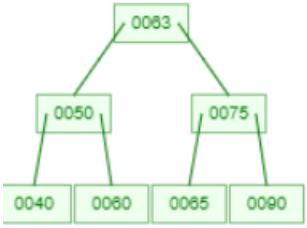
Insert 65

65 > 50 , ke kanan, 65 > 60 maka insert di kanan 60



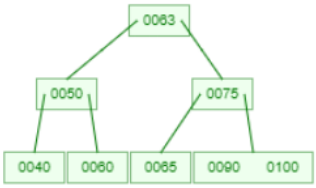
Insert 63

63 > 50 , ke kanan, 63 > 60 maka insert di kanan 60, karena sudah penuh maka kitapush median yaitu 63 maka kita push 63 ke atas, lalu di atas dia juga penuh, maka kita push mediannya yaitu 63 ke atas lagi



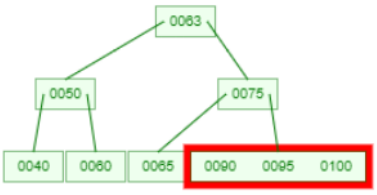
Insert 100

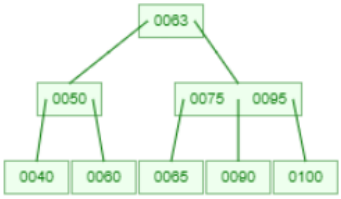
100 > 63, maka ke kanan, 100 > 75, maka ke kanan , 100 > 90 maka insert di akan 90, ada ruang jdi tidak perlu push



Insert 95

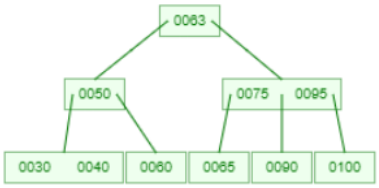
95 > 63, ke kanan, 95 > 75 ke kanan, 95 > 90 maka insert di tengaj 90 & 100, tree penuh push 95 ke atas, saat di push masih aman jdi tidak perlu push lagi





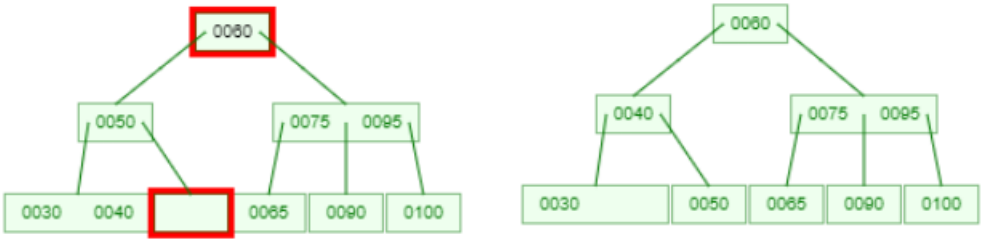
Insert 30

30 < 63 ke kiri, 30 < 50 ke kiri, 30 < 40 insert di kiri 40, karena masih aman tidak perlu diapa-apakan



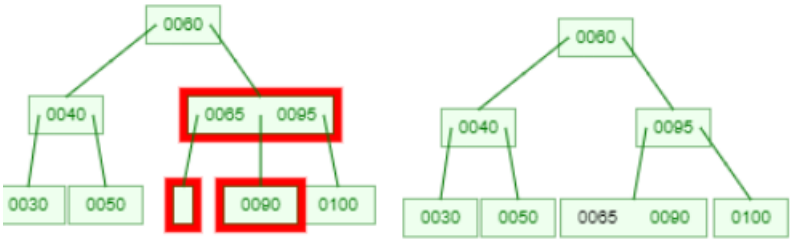
Remove 63

Search 63, ketemu 63 yaitu internal, maka cari predecessornya ketemu 60, timpah 63 dengan 60 dan proceed deleting 60 yg lama, 60 merupakan node minimum maka dia harus pinjam sibling kiri , cek sibling kiri ada yg tidak minimum maka element terkanan sibling kiri yaitu 40 akan dijadikan parent dan parentnya yaitu 50 menggantikan node yg 60 yg lama



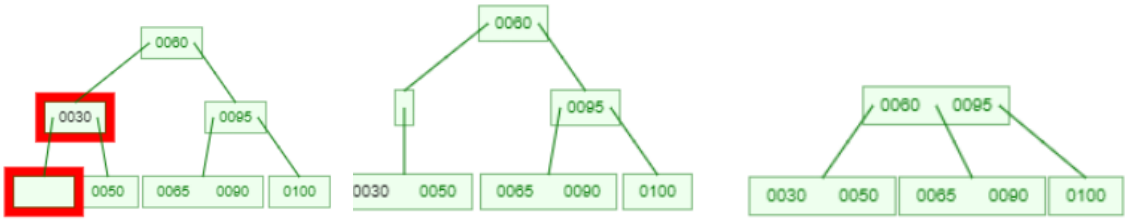
Remove 75

Search 75, 75 merupakan internal, maka kita harus cari predecessor ketemu 65, timpah 75 dengan 65, lalu delete 65 yg lama, karena dia minimal maka harus pinjam sibling, cek kiri kosong, kanan minimal jdi tidak bisa, maka lakukan merge sibling dengan parentnya



Remove 40

Search 40, ketemu 40 di internal, maka cari predecessor, ketemu 30, timpah 40 dengan 30, proceed in deleting 30 yg lama, karena 30 merupakan minimal key, maka cek kiri kosong, cek kanan minimal, maka merge dengan parent, sekarang node parent kosong dan dia cek sibling kiri/kanan tidak ada yg lebih dari minimal, maka merge lagi dengan paretnya dia



Remove 60

Search 60 , ketemu 60 sebagai internal, maka cari predecessor, dapat 50, timpah 60 dengan 50, dan karena 50 yg lama berada di node yg lebih dari minimal key cukup langsung di delete

Remove 95

Search 95, ketemu sebagai internal, maka cari predecessor ketemu 90, timpah 95 dengan 90, proceed to deleting 90 yg laama, karena 90 yg lama berada di node yg lebih dari minimal key, bisa langsung kita delete

Kevin Bryan – Final Project

Red Black Tree

Insert 40

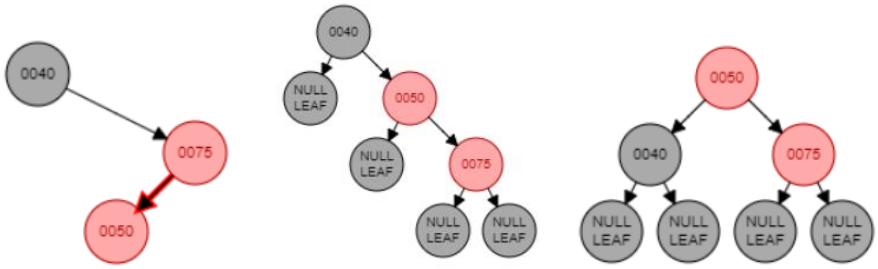
Tree masih kosong, maka insert langsung , karena root di recolor jadi hitam

Insert 75,

75 > 40 , insert di kanan 40, set ke red, karena parentnya hitam tidak perlu diapa-apakan

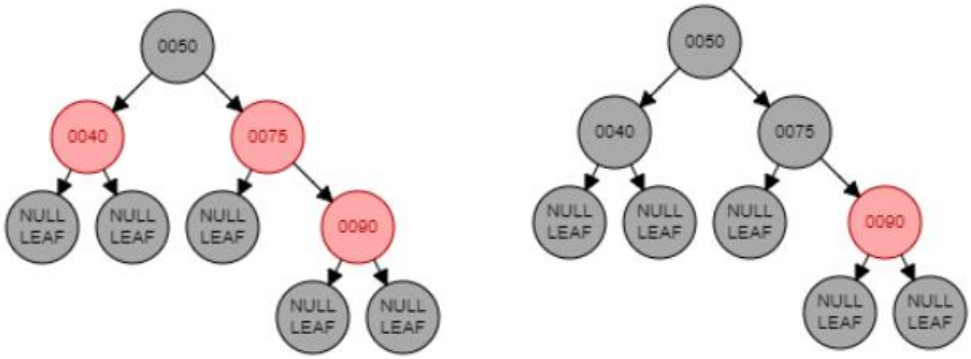
Insert 50,

50 > 40 , ke kanan, 50 < 75 insert di kiri 75, karena parentnya merah dan node di kiri, parent dikanan maka kita harus right left rotation, lalu recolor



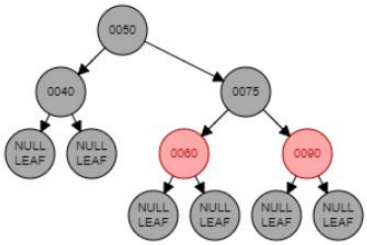
Insert 90

90 > 50 ke kanan, 90 > 75 insert di kanan 75, set ke merah, karena dia merah dan parent merah, cek uncle, unclenya merah juga, maka push black dari grandparent ke parents, dan karena grandparent adalah root recolor jadi hitam



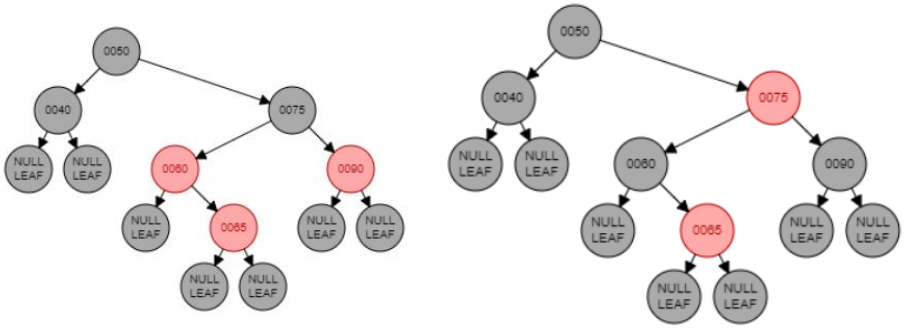
Insert 60

60 > 50 ke kanan, 60 < 75 insert di kiri 75, set ke merah, karena parent hitam tidak perlu diapa-apakan



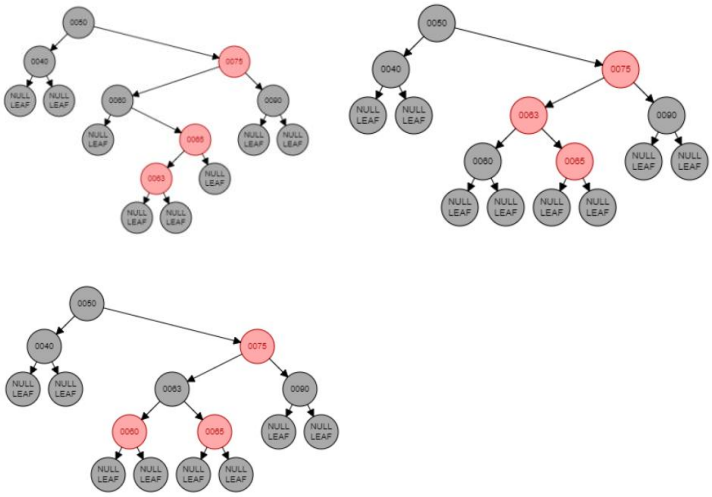
Insert 65

65 > 50, ke kanan, 65 < 75, ke kiri, 65 > 60 insert di kanan 60, set ke merah, karena dia merah dan paretnya merah cek unclenya, unclenya merah juga, karena uncle merah maka cukup push blackness dari grandparent



Insert 63

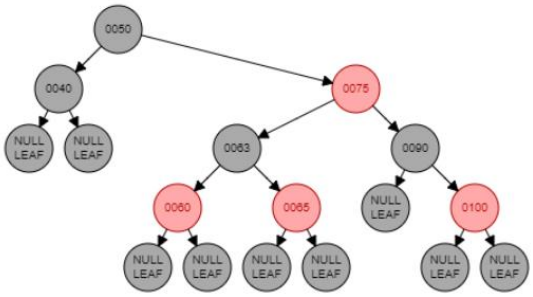
63 > 50 , ke kanan, 63 < 75 ke kiri, 63 > 60 ke kanan, 63 < 65 insert di kiri 65, set ke merah, karena dia merah dan parent merah, maka cek uncle, unclenya hitam, dan node adalah anak kiri, parent anak kanan, maka kita right-left rotation, lalu recolor



Kevin Bryan – Final Project

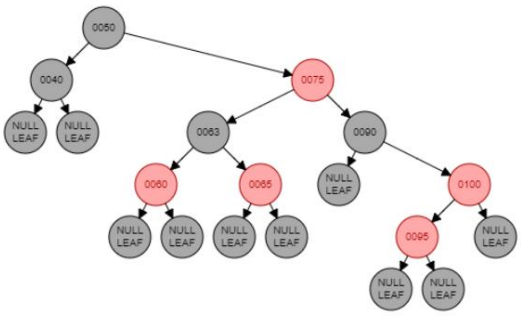
Insert 100

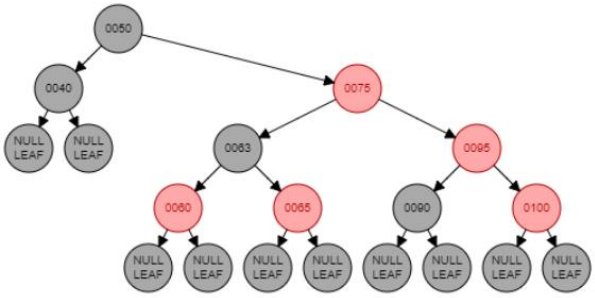
100 > 50 , ke kanan, 100 > 75 ke kanan , 100 > 90, insert di kanan 90,set merah, karena parentnya hitam, maka tidak perlu diapa-apakan

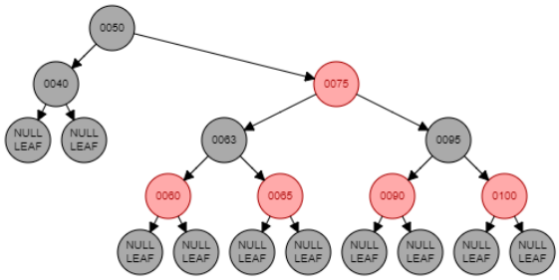


Insert 95

95 > 50 ke kanan, 95 > 75 ke kanan, 95 > 90 ke kanan, 95 < 100 insert di kiri 100, set ke red, karena dia merah dan parent merah cek uncle, unclenya hitam, maka karena 95 anak kiri, dan 100 anak kanan lakukan right left rotation, recolor



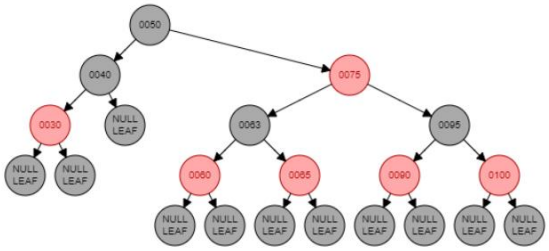




Kevin Bryan – Final Project

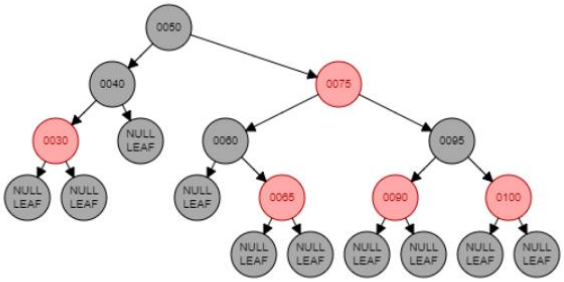
Insert 30

30 < 50 ke kiri, 30 < 40 insert di kiri 40, set ke merah, karena dia parent hitam tidak perlu diapa-apakan



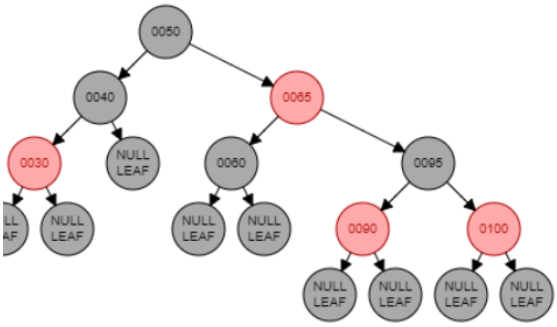
Remove 63

Search 63, 64 memiliki 2 anak, cari predecessor ketemu 60, timpah 63 dengan 60, hapus node predecessor, karena merah tidak perlu rotation



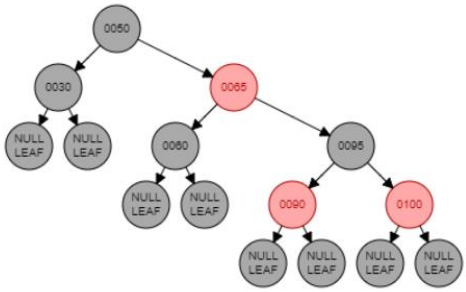
Remove 75

Search 75, 75 memiliki 2 anak, cari predecessor ketemu 65, timpah 75 dengan 65, delete node 65 yg lama, karena merah tidak perlu rotation



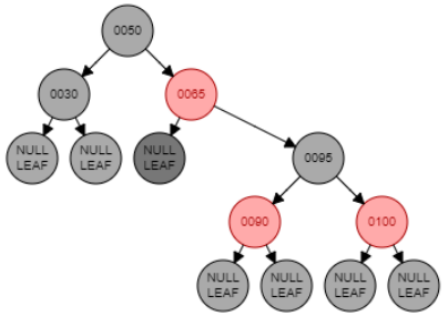
Remove 40

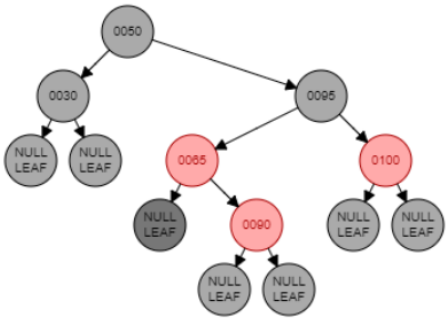
Search 40, 40 memiliki 1 anak saja maka jadikan timpah dia dengan anaknya yaitu 30, dan delete node anak, node anak warna merah jadi langsung hapus

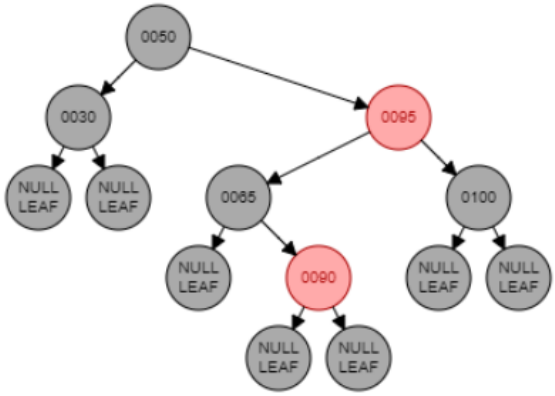


Remove 60

Search 60, 60 merupakan leaf, maka delete karena dia hitam maka warnai null leafnya double black, karena double blacknya memilki sibling hitam dan double black merupakan anak kiri , dan nephew kanannya merah, maka left rotate dan recolor

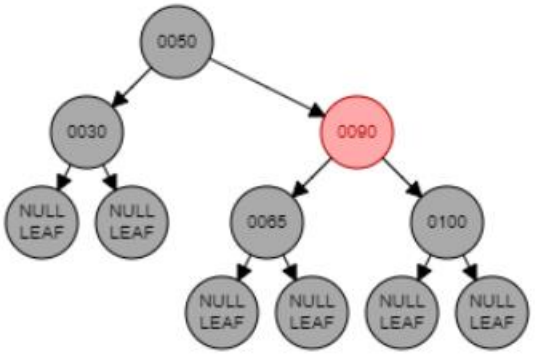






Remove 95

Search 95, ketemu 95 memiliki 2 anak, maka cari predecessornya ketemu 90, timpah 95 dengan 95, karena 90 merah maka delete langsung

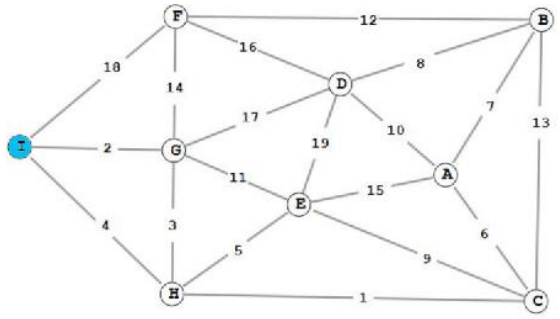


Kevin Bryan – Final Project

Disjoint Set & Graphs

MST using Prim

Start from I,



MIN HEAP

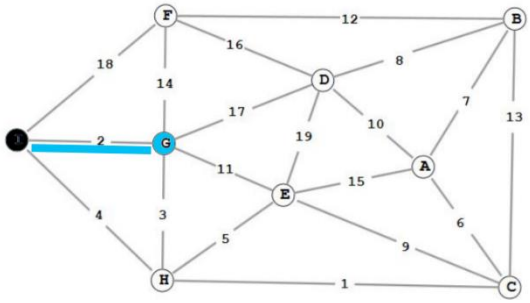
I to G = 2

I to H = 4

I to F = 18

shortest is to G, Unify G&I, pop 1 to G = 2

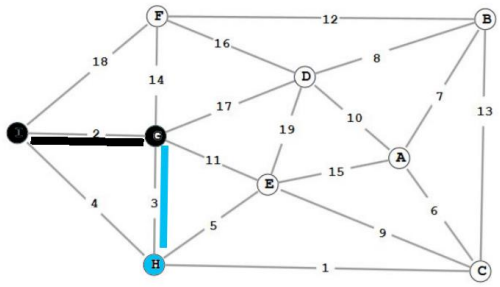
Visit G



MIN HEAP
G to H = 3
I to H = 4
G to E = 11
G to F = 14
G to D = 17
I to F = 18

Shortest is to H, Unify I&H, pop G to H = 3

Visit H

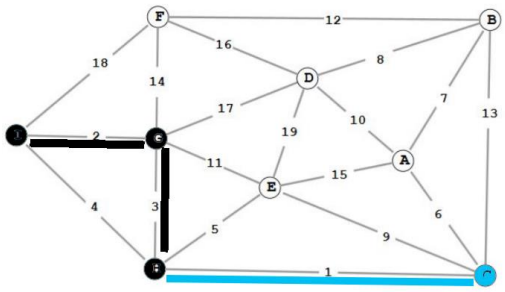


MIN HEAP
H to C = 1
I to H = 4
H to E = 5
G to E = 11
G to F = 14
G to D = 17
I to F = 18

Shortest is to C, Unify H&C, pop H to C = 1

Kevin Bryan – Final Project

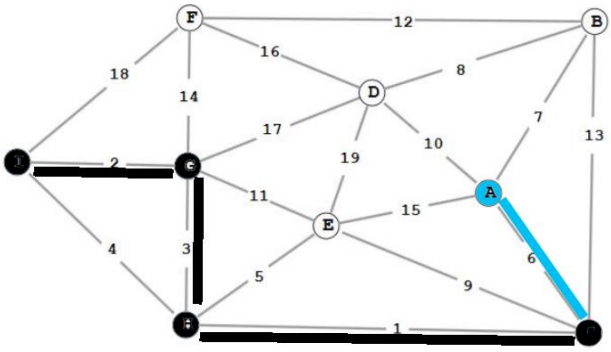
Visit C



MIN HEAP
I to H = 4
H to E = 5
C to A = 6
C to E = 9
G to E = 11
C to B = 13
G to F = 14
G to D = 17
I to F = 18

Shortest is to A, Unify C & A, pop C to A = 6

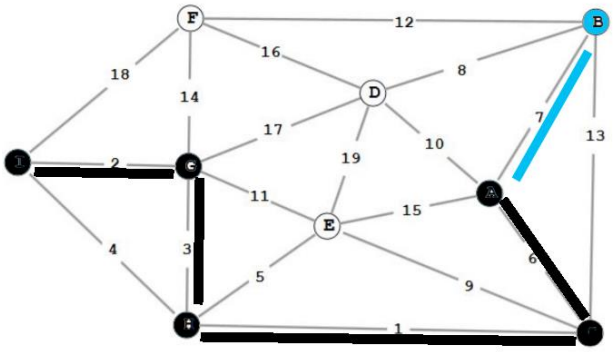
Visit A



MIN HEAP
I to H = 4
H to E = 5
C to A = 6
A to B = 7
A to D = 10
G to E = 11
C to B = 13
G to F = 14
A to E = 15
G to D = 17
I to F = 18

Shortest is to B, unify AB, pop A to B = 7

Visit B

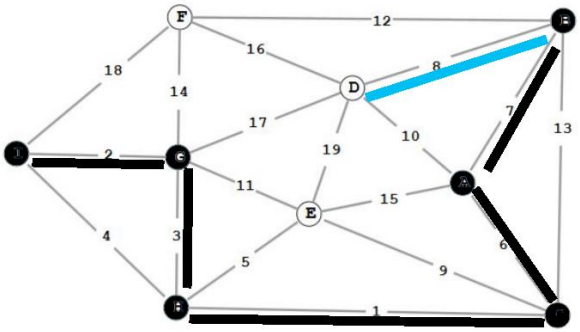


MIN HEAP
I to H = 4
H to E = 5
C to A = 6
B to D = 8
A to D = 10
G to E = 11
B to F = 12
C to B = 13
G to F = 14
A to E = 15
G to D = 17
I to F = 18

Shortest is to D, Unify BD, pop B to D = 8

Kevin Bryan – Final Project

Visit D

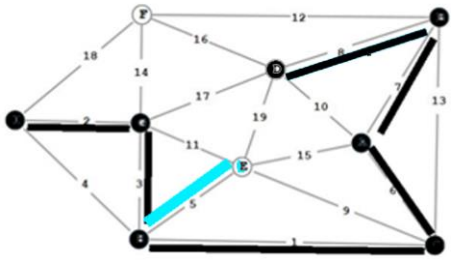


MIN HEAP
I to H = 4
H to E = 5
C to A = 6
B to D = 8
A to D = 10
G to E = 11
B to F = 12
C to B = 13
G to F = 14
A to E = 15
D to F = 16
G to D = 17
I to F = 18
D to E = 19

Shortest to F with cost of 16, but is below B to F with cost of 12, so check MIN HEAP

In mean heap we found H to E not visited yet so we go use that, because the rest above will create loop

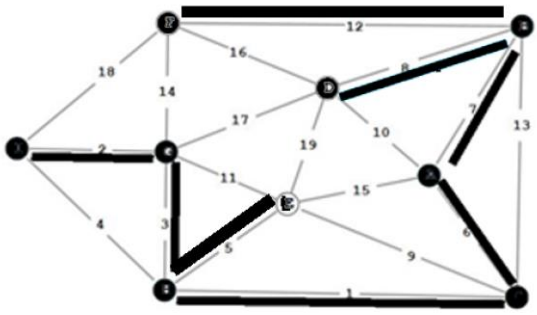
Visit E



MIN HEAP
C to A = 6
B to D = 8
A to D = 10
G to E = 11
B to F = 12
C to B = 13
G to F = 14
A to E = 15
G to D = 17
I to F = 18
D to E = 19

When we check E, all destination will form cycle, so check back to MIN HEAP,

we find B to F won't form cycle so we go to F



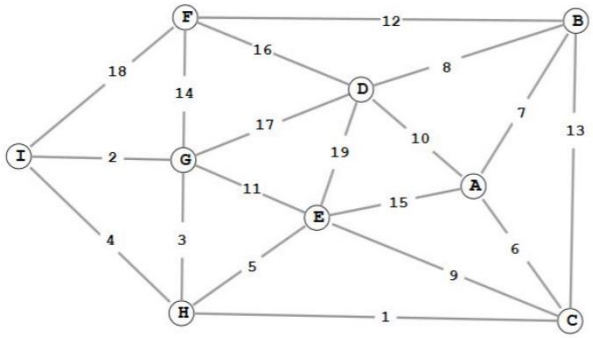
All node is now visited, MST is found with minimum cost of **44**

Kevin Bryan – Final Project

MST using Kruskal

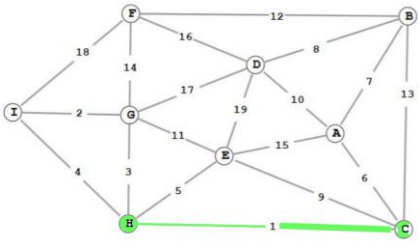
Insert all edges to MIN HEAP

Cost	From/To	To/From
1	H	C
2	I	G
3	G	H
4	I	H
5	H	E
6	A	C
7	A	B
8	D	B
9	E	C
10	D	A
11	G	E
12	F	B
13	B	C
14	F	G
15	E	A
16	F	D
17	G	D
18	F	I
19	E	D



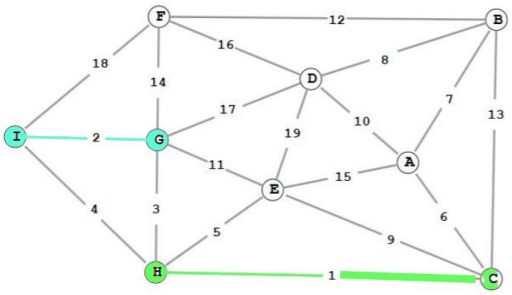
Pick smallest edge, if not cycle include this edge

1 HC, include 1



Check next queue,

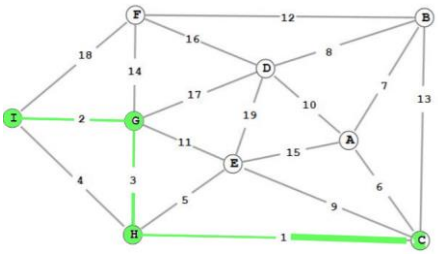
2 IG, not cycle, include 2, total cost = 3



Check next queue,

Kevin Bryan – Final Project

3 GH, not cycle, include 3, total cost = 6

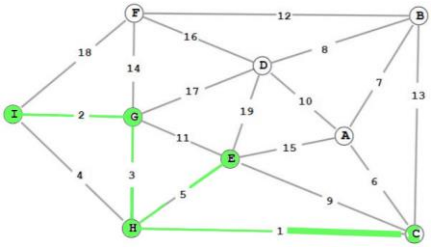


Check next queue,

4 IH, it forms cycle, don't include, continue to next queue

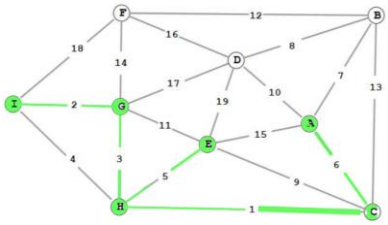
Check next queue,

5 HE, not cycle, include 5, total cost = 11



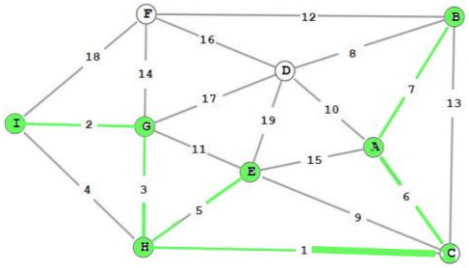
Check next queue,

6 AC, not cycle, include 6, total cost = 17



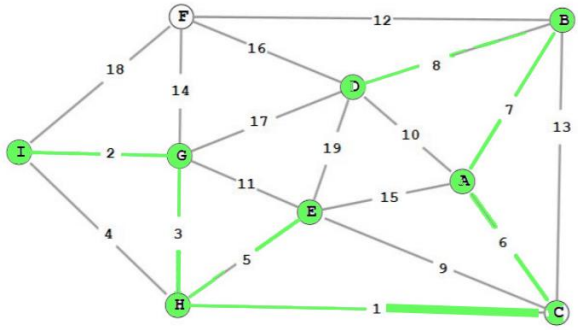
Check next queue,

7 AB, not cycle, include 7, total cost = 24



Check next queue,

8 DB, not cycle, include in 8, total cost = 32



Check next queue,

9 EC, forms cycle, don't include, continue to next queue

Check next queue,

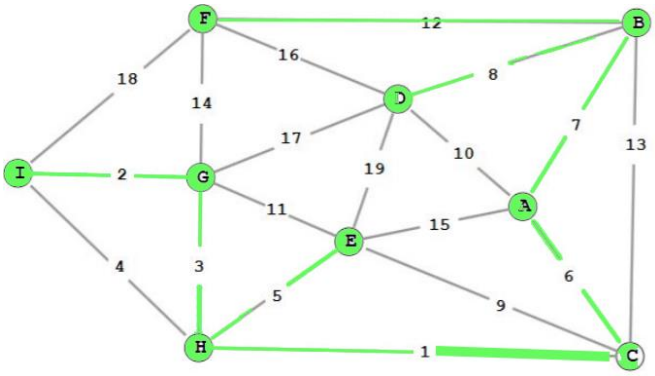
10 DA, forms cycle, don't include, continue to next queue

Check next queue,

11 G E, forms cycle, don't include, continue to next queue

Check next queue,

12 BF, not cycle, include 12, total cost = 44

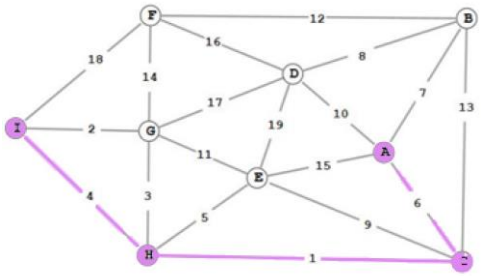


All node visited, MST is done with total cost of **44**

Shortest Path w/ Dijkstra

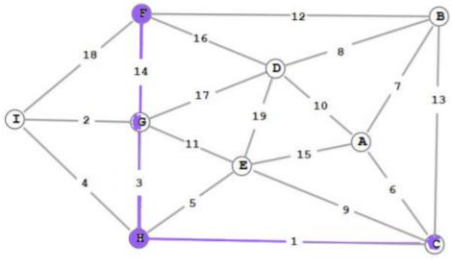
Kevin Bryan – Final Project

I to A



I to H to C to A with cost of 11

F to C



F to G to H to C with cost of 18