1.

Insertion

```
struct RedBlackNode{
    int key;
    enum { red, black } color;
    RedBlackNOde *left, *right, *parent;
};

RB-INSERT(T, z)
        y ← null
        x ← T->root
        while x ≠ null
                do y ← x
                        if z->key < x->key
                                then x ← x->left
                                else x ← x->right
        z->p ← y
        if y = null
                then T->root ← z
                else if z->key < y->key
                        then y->left ← z
                        else y->right ← z
        z->left ← null
        z->right ← null
        z->color ← RED
        RB-INSERT-FIXUP(T, z)
```

```
RB-INSERT-FIXUP(T, z)
while  z->p->color  = RED
        do if z->p =  z->p->p->left
                then y ← z->p->p->right
                        if y->color = RED
                                then    z->p->color ← BLACK
                                        y->color ← BLACK
                                        z->p->p->color ← RED
                                        z ← z->p->p
                                else if z =  z->p->right
                                        then    z ← z->p
                                                LEFT-ROTATE(T, z)
                                                z->p->color ← BLACK
                                                z->p->p->color ← RED
                                                RIGHT-ROTATE(T, z->p->p)
                        else (same as then clause with "right" and "left" exchanged)
 T->root->color ← BLACK
```

```
LEFT-ROTATE(T, x)
        y ← x->right
        x->right← y->left
        y->left->p ← x
        y->p ← x->p
        if      x->p = Null
                then    T->root ← y
                else if x = x->p->left
                        then    x->p->left ← y
                        else    x->p->right ← y
         y->left ← x
         x->p ← y
```

```
RIGHT-ROTATE(T, x)
        y ← x->left
        x->left← y->right
        y->right->p ← x
        y->p ← x->p
        if      x->p = Null
                then    T->root ← y
                else if x = x->p->right
                        then    x->p->right ← y
                        else    x->p->left ← y
         y->right ← x
         x->p ← y
```

removal

RB-Delete(*T*, *z*)

```
1.  if left[z] = nil[T] or right[z] = nil[T]
2.     then y ← z
3.     else y ← TREE-SUCCESSOR(z)
4.  if left[y] ≠nil[T ]
5.      then x ← left[y]
6.      else x ← right[y]
7.  p[x] ← p[y]   // Do this, even if x is nil[T]
8.  if p[y] = nil[T ]
9.    then root[T ] ← x
10.   else if y = left[p[y]]  (*if y is a left child.*)
11.         then left[p[y]] ← x
12.         else right[p[y]] ← x  (*if y is a right
13. if y ≠z                              child.*)
14.   then key[z] ← key[y]
15.   copy y's satellite data into z
16. if color[y] = BLACK
17.    then RB-Delete-Fixup(T, x)
18. return y
```

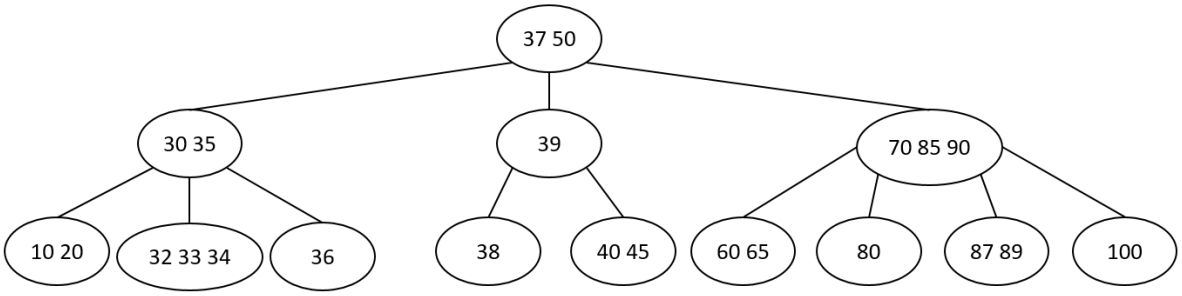
RB-Delete-Fixup(T, x)

```
1.  while x ≠ root[T] and color[x] = BLACK
2.    do if x = left[p[x]]
3.       then w ← right[p[x]]
4.            if color[w] = RED
5.              then color[w] ← BLACK
6.                   color[p[x]] ← RED
7.                   LEFT-ROTATE(T, p[x])
8.                 w ← right[p[x]]
9.       if color[left[w]] = BLACK and color[right[w]] = BLACK
10.        then color[w] ← RED
11.             x ← p[x]
12.             p[x] ← p[p[z]]
13.        else if color[right[w]] = BLACK
14.                then color[left[w]] ← BLACK
15.                     color[w] ← RED
16.                     RIGHT-ROTATE(T,w)
17.                     w ← right[p[x]]
18.             color[w] ← color[p[x]]
19.             color[p[x]] ← BLACK
20.             color[right[w]] ← BLACK
21.             LEFT-ROTATE(T, p[x])
22.             x ← root[T]
23.    else  right and left exchanged
24. color[x] ← BLACK
```

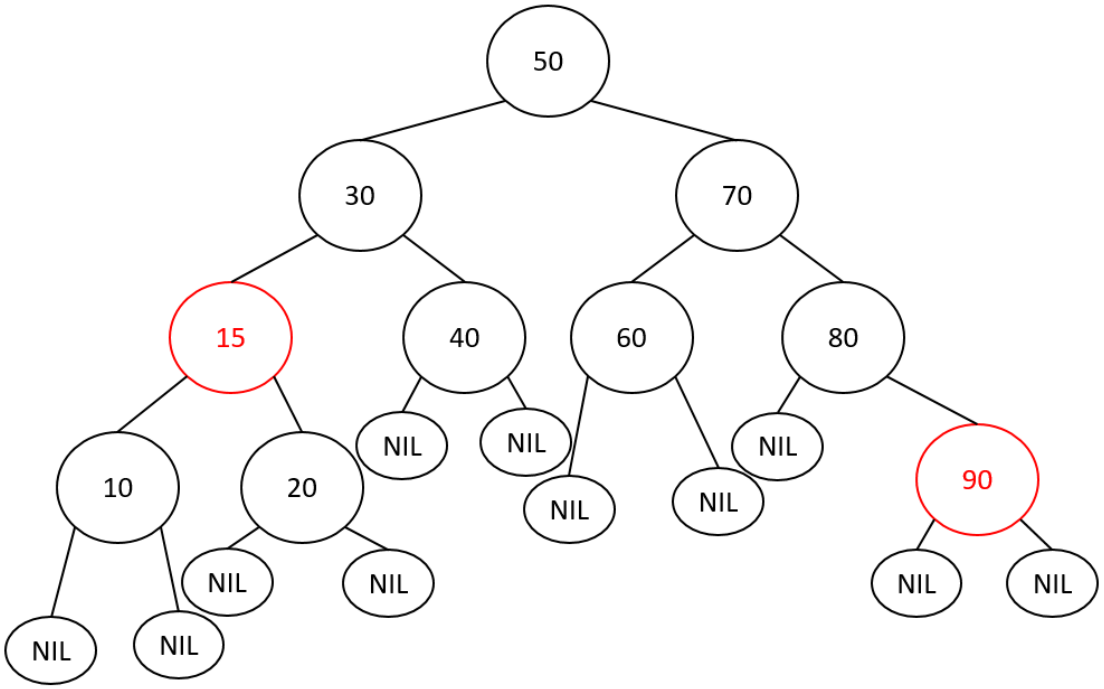
traversal

```
Inorder-traversal(x)
    if x != NIL
        then  Inorder-traversal(left[p])
              print key[x]
              Inorder-traversal(right[p])
```

2.



3.


50
30
70
15
40
60
80
10
20
NIL
NIL
NIL
NIL
90
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL

4.