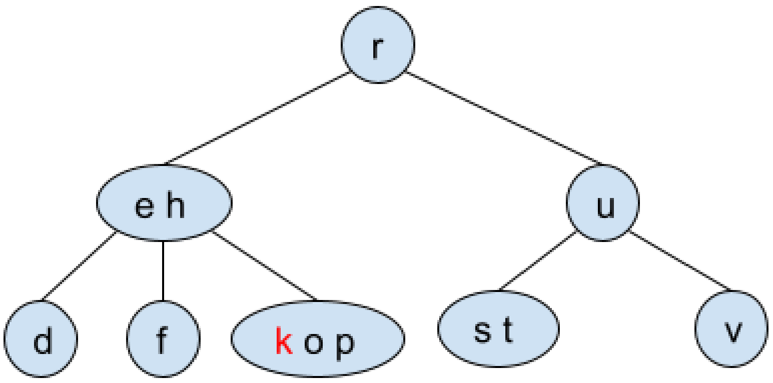
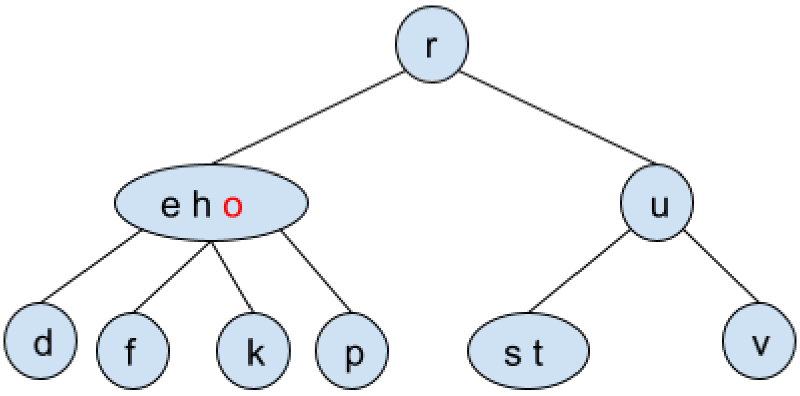
1.
BST在最好的情況下搜尋時間複雜度為O(logn), 但如果插入節點是有序地會就會退化成O(n), 想減少比較次數就需要降低樹的高度。
2-3 Tree允許一個節點存放兩個元素, 最多可以擁有三個子樹, 大大的降低樹高, 並且所有葉節點都會在同一層, 能夠建立出平衡的樹。

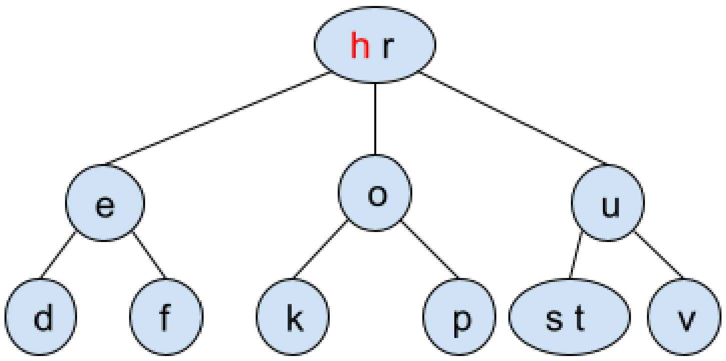
2.
```
/**
策略:
1. 先找到最接近且大於等於 "from" value 在哪
2. 找到後, 設 had found "from" value 為 true, 並根據情況對 norder traverse node
3. inorder traverse 直到找到最接近且小於等於 "to" value 後, 設 done 為 true, 結束 inorder traverse
**/

void traverseValueBetween(int from, int to, Node *r)
{
        if (r is leaf) return;

        if (traversion had finish) return;

        if (r has two values)
        {
                if (had found "from" value && traversion not finish) inorder until "to" value;

                if (r.small > from)
                {
                        traverseValueBetween(from, to, r->left);
                        if (had found "from" value && traversion not finish)
                        {
                                mark "r" node had been traversed;
                                inorder until "to" value;
                        }
                }
                else if (r.small == from)
                {
                        found from, and mark "found" to true;
                        mark "r" node had been traversed;
                        if (traversion not finish)  inorder until "to" value;
                }
                else if (r.small < from && r.large > from)
                {
                        traverseValueBetween(from, to, r->middle);
                if (had found "from" value && traversion not finish)
                {
                    mark "r" node had been traversed;
                    inorder until "to" value;
                }
            }
            else if (r.large == from)
            {
                found from, and mark "found" to true;
                mark "r" node had been traversed;
                if (traversion not finish) inorder until "to" value;
            }
            else if (r.large < from)
            {
                traverseValueBetween(from, to, r->right);
            }
        }
        else if (r has one value)
        {
            if (had found "from" value && traversion not finish) inorder until "to" value;

            if (r.small > from)
            {
                traverseValueBetween(from, to, r->left);
                if (had found "from" value && traversion not finish)
                {
                    mark "r" node had been traversed;
                    inorder until "to" value;
                }
            }
            else if (r.small == from)
            {
                found from, and mark "found" to true;
                mark "r" node had been traversed;
                if (traversion not finish) inorder until "to" value;
            }
            else if (r.small < from)
            {
                traverseValueBetween(from, to, r->right);
            }
        }
        return;
    }
```

3.

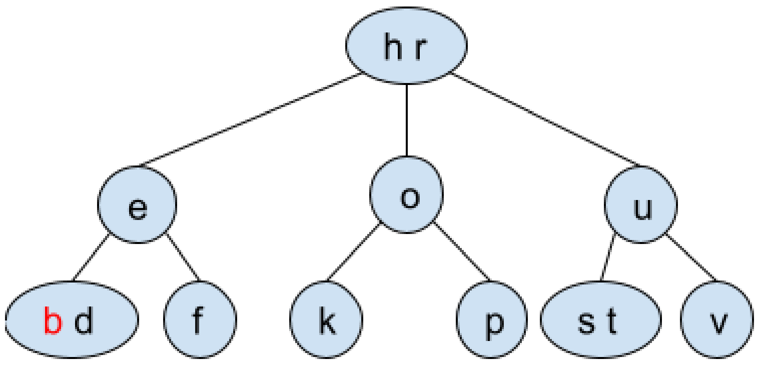
Insert k



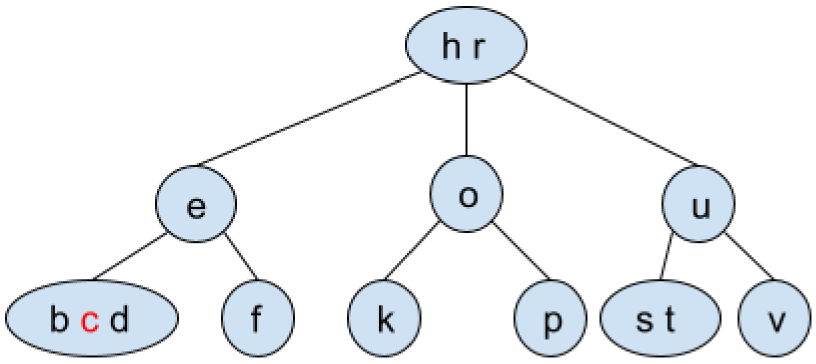


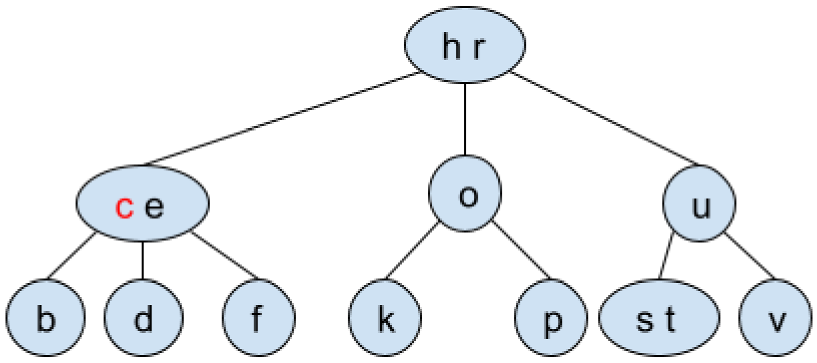


Insert b

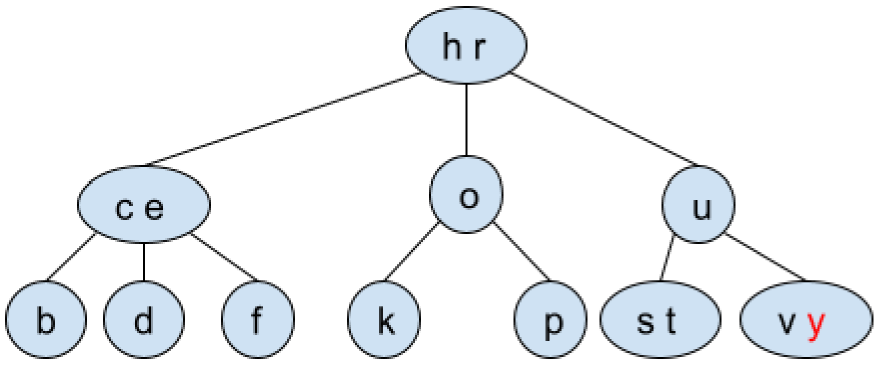


Insert c

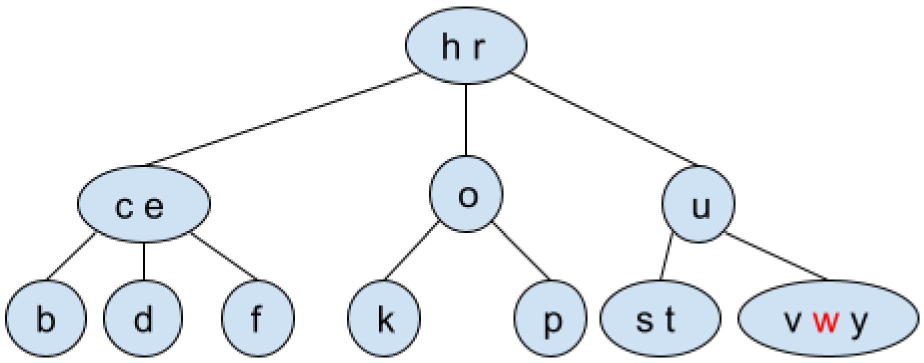


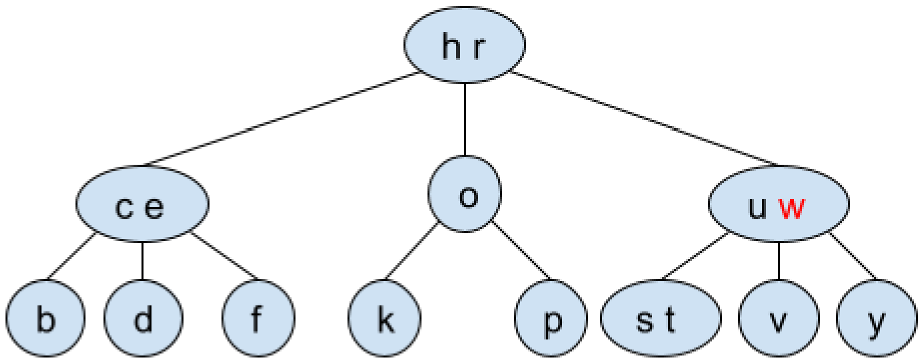


Insert y

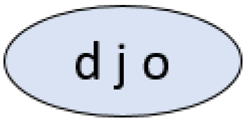


Insert w

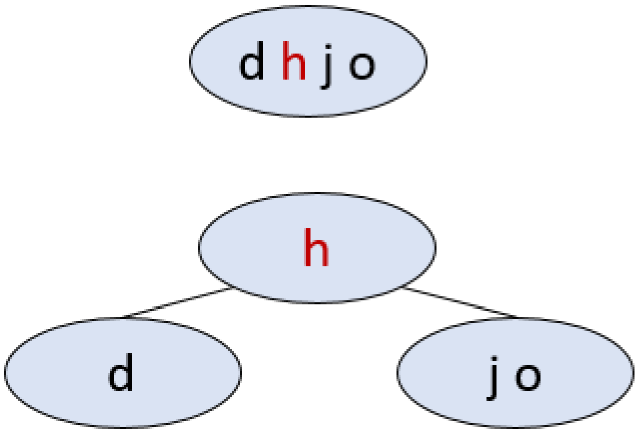

h r
c e
o
u
b
d
f
k
p
s t
v w y


h r
c e
o
u w
b
d
f
k
p
s t
v
y

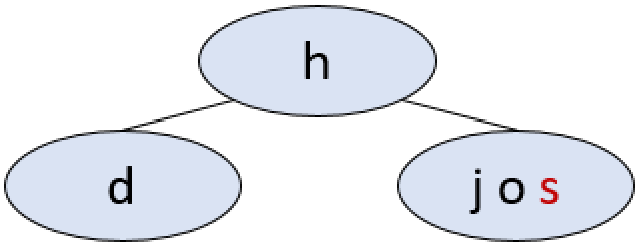
4.
insert o, d, j



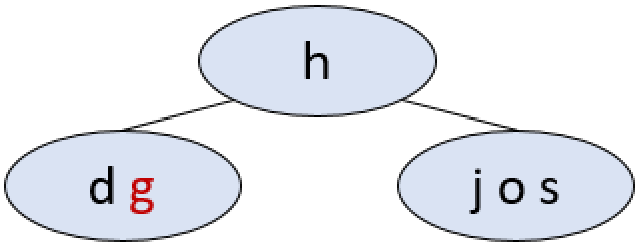
insert h



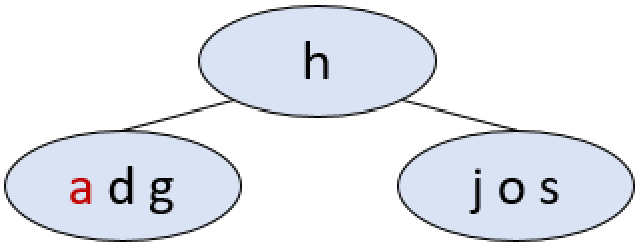
insert s



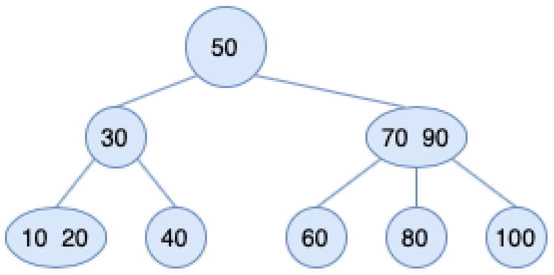
insert g



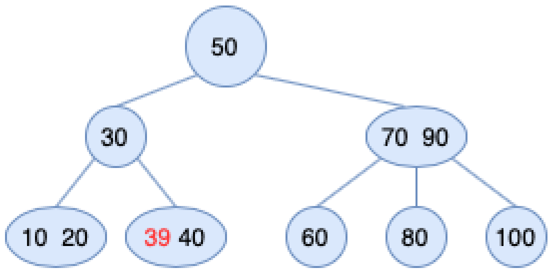
insert a



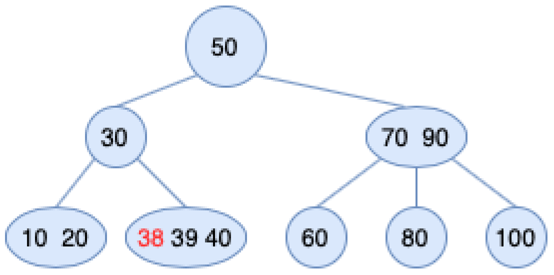
5.



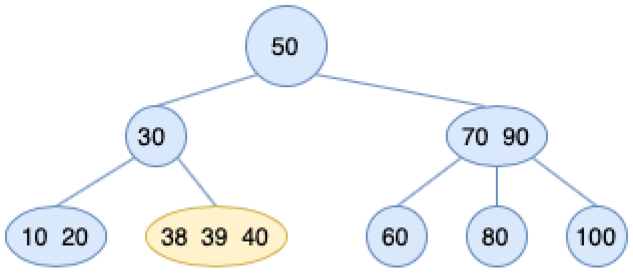
Insert 39

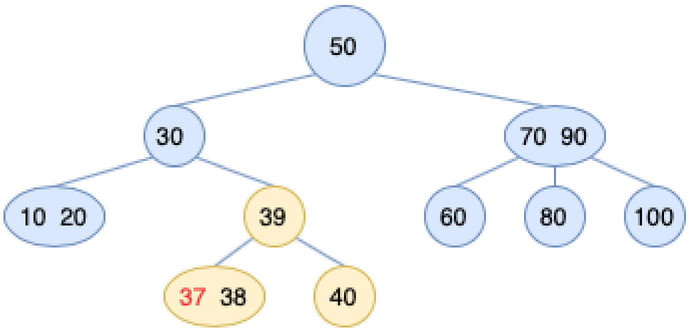


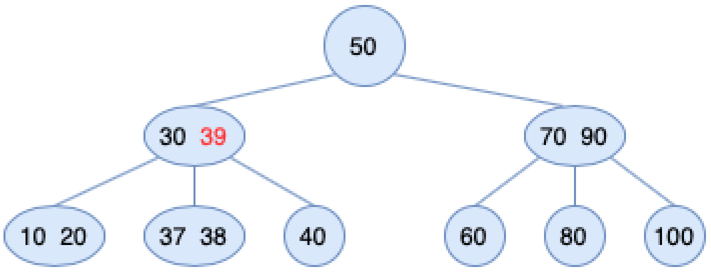
Insert 38



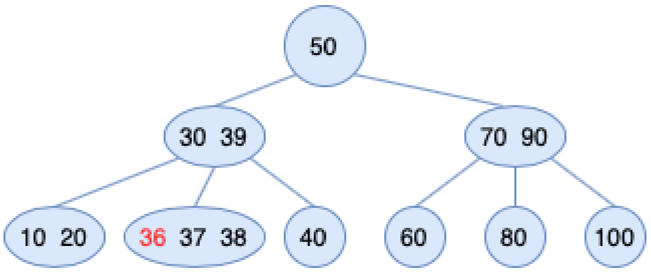
Insert 37



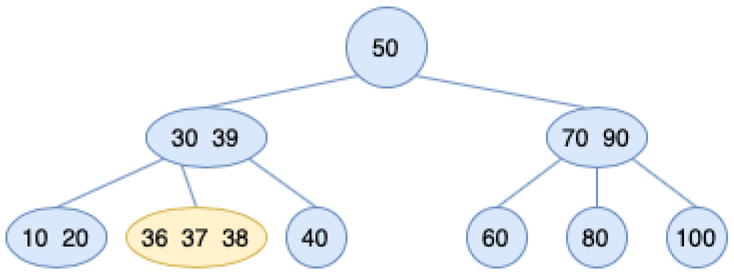


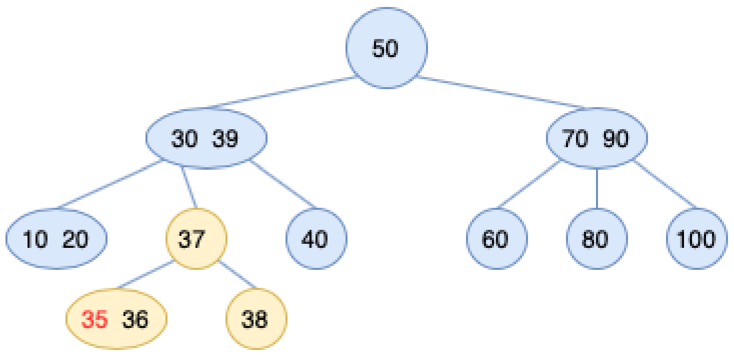


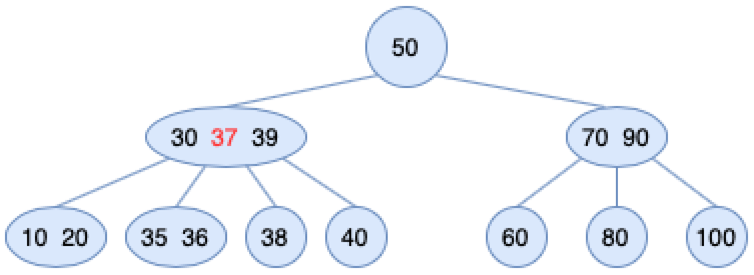
Insert 36



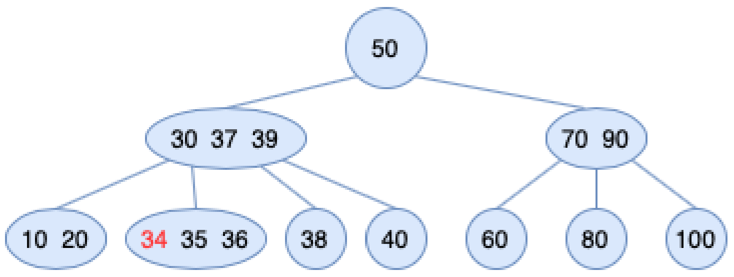
Insert 35



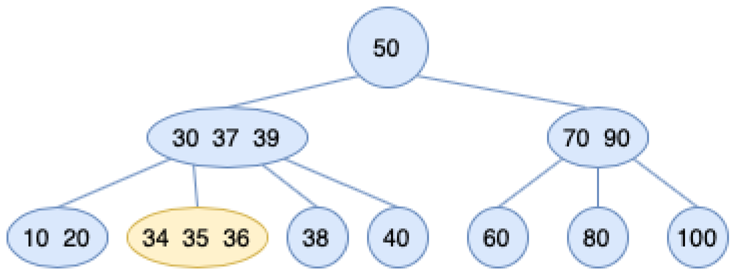




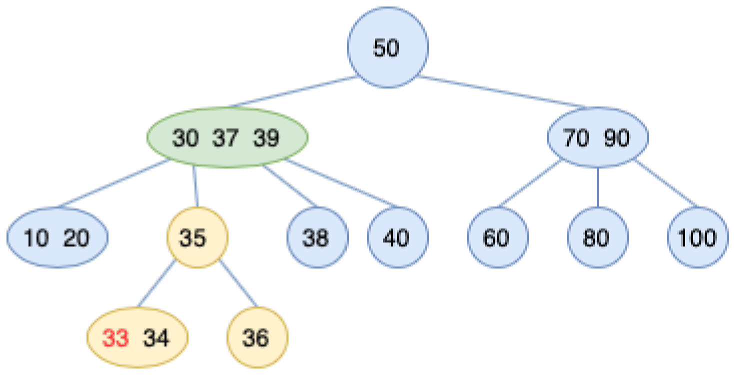
Insert 34



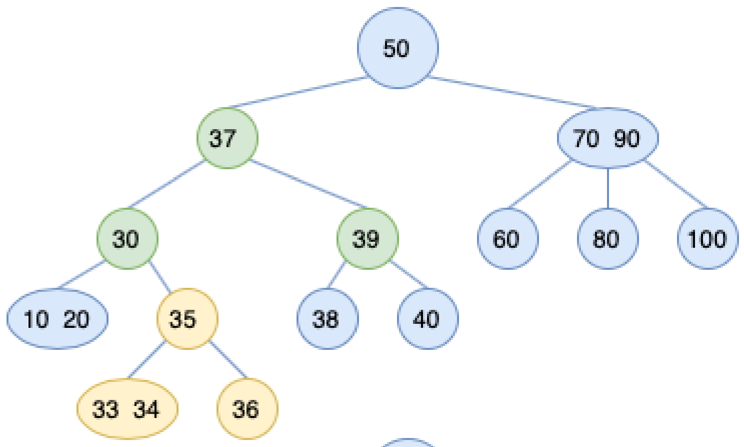
Insert 33



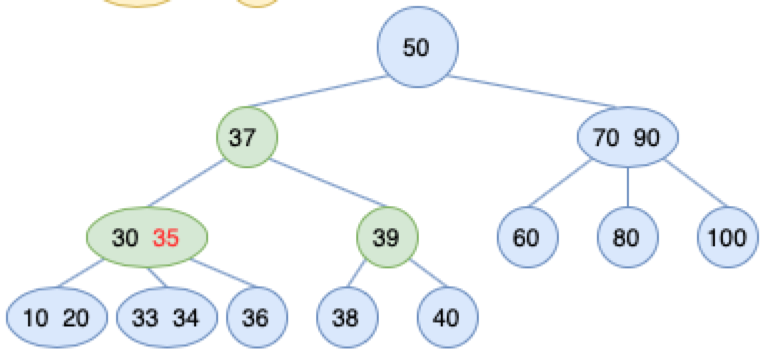
50
30 37 39
70 90
10 20
34 35 36
38
40
60
80
100



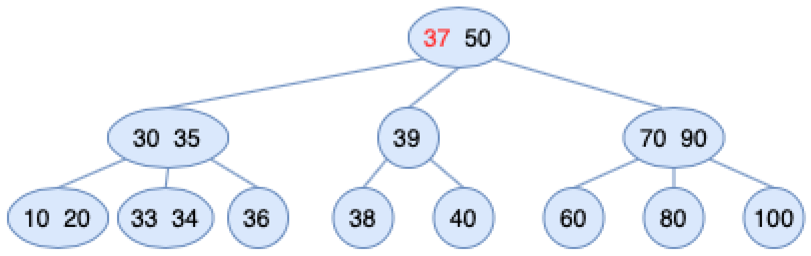
50
30 37 39
70 90
10 20
35
38
40
60
80
100
33 34
36



50
37
70 90
30
39
60
80
100
10 20
35
38
40
33 34
36

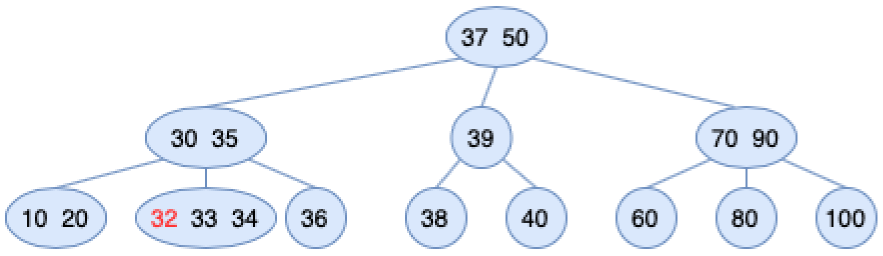


50
37
70 90
30 35
39
60
80
100
10 20
33 34
36
38
40



37 50
30 35
39
70 90
10 20
33 34
36
38
40
60
80
100

Insert 32


37 50
30 35
39
70 90
10 20
32 33 34
36
38
40
60
80
100