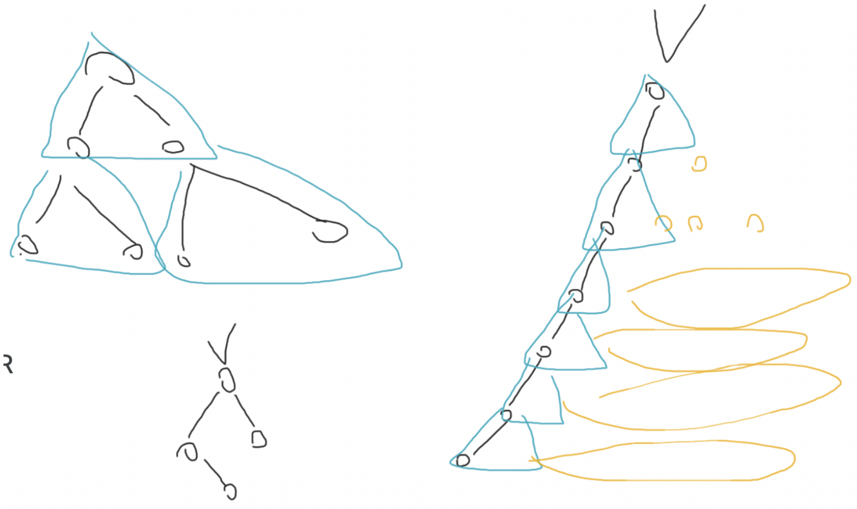
Phill-DS-0320

AVL tree - Adelson V L

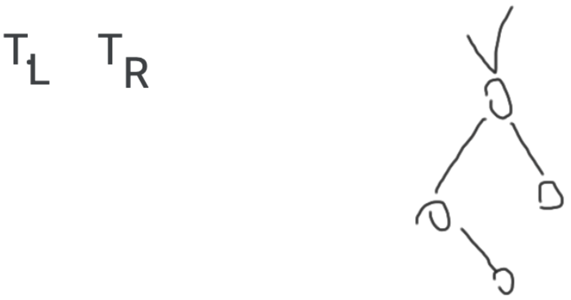
高度平衡的二元搜尋樹 → efficiency (不能有空隙, 計算量會增加)



$T \in AVL$

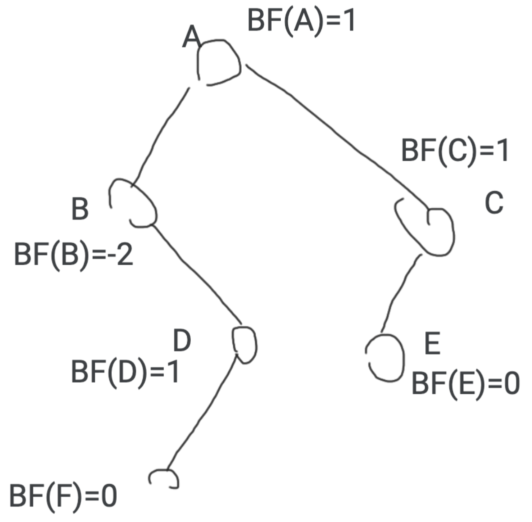
$T_L, T_R \in AVL$

$|h_L - h_R| <= 1$ → balanced



balance factor 平衡因子 BF → 針對某一個節點的高度計算結果 → BF(n) , n→ node

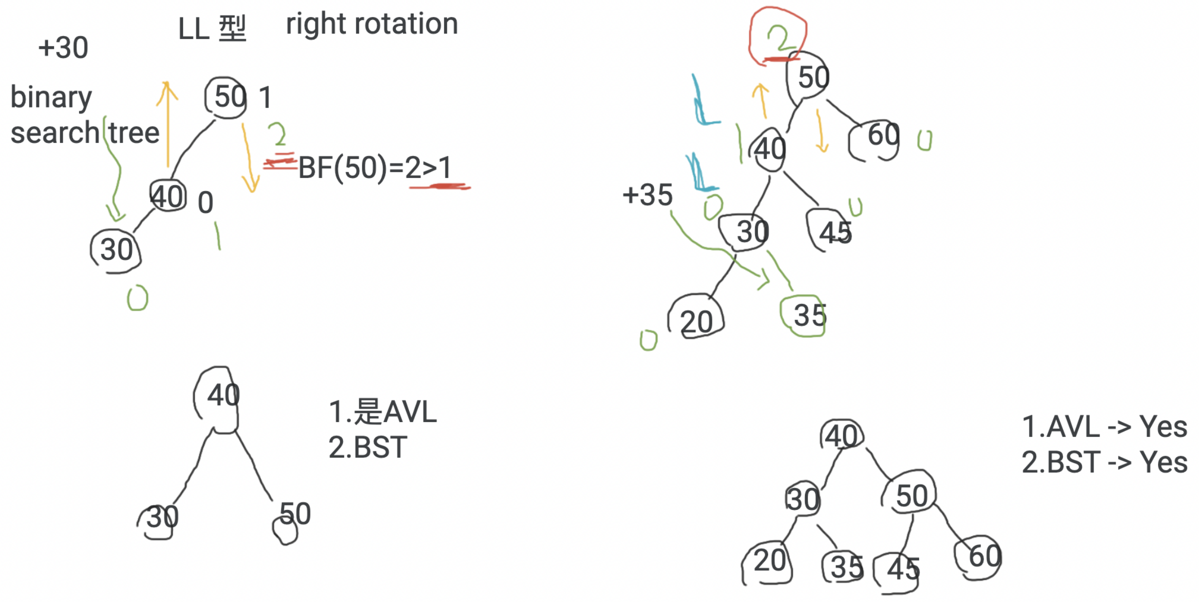
|BF(n)| ≤ 1 → -1, 0, 1



AVL tree 加入 node

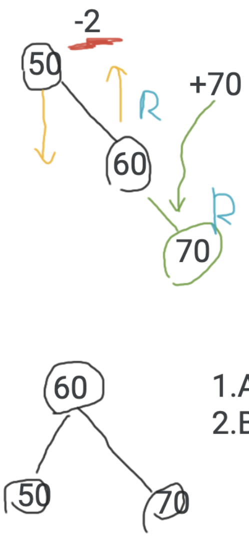
- 針對四種不同的 types → 作 re-balanced

LL type

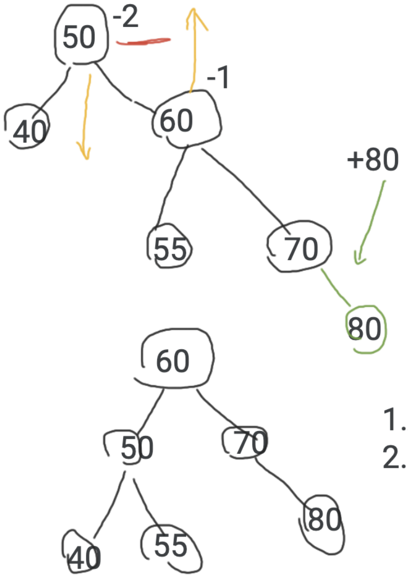


RR type → left rotation

RR 型



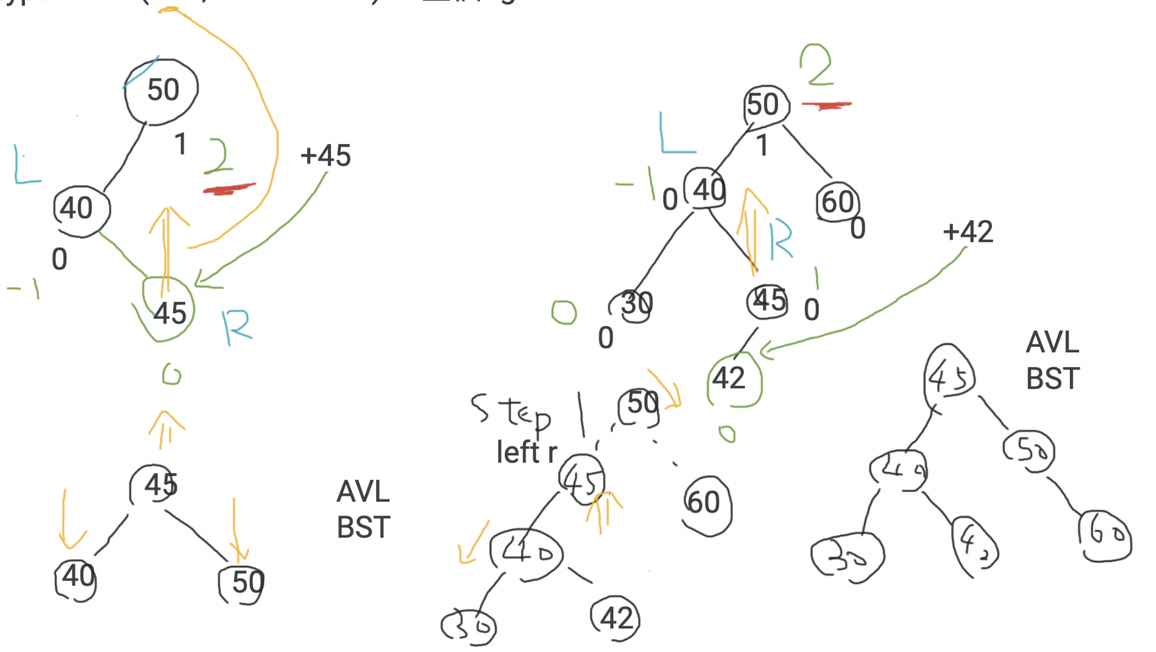
1.AVL -> Yes
2.BST -> Yes



1.AVL-> Yes
2.BST -> Yes

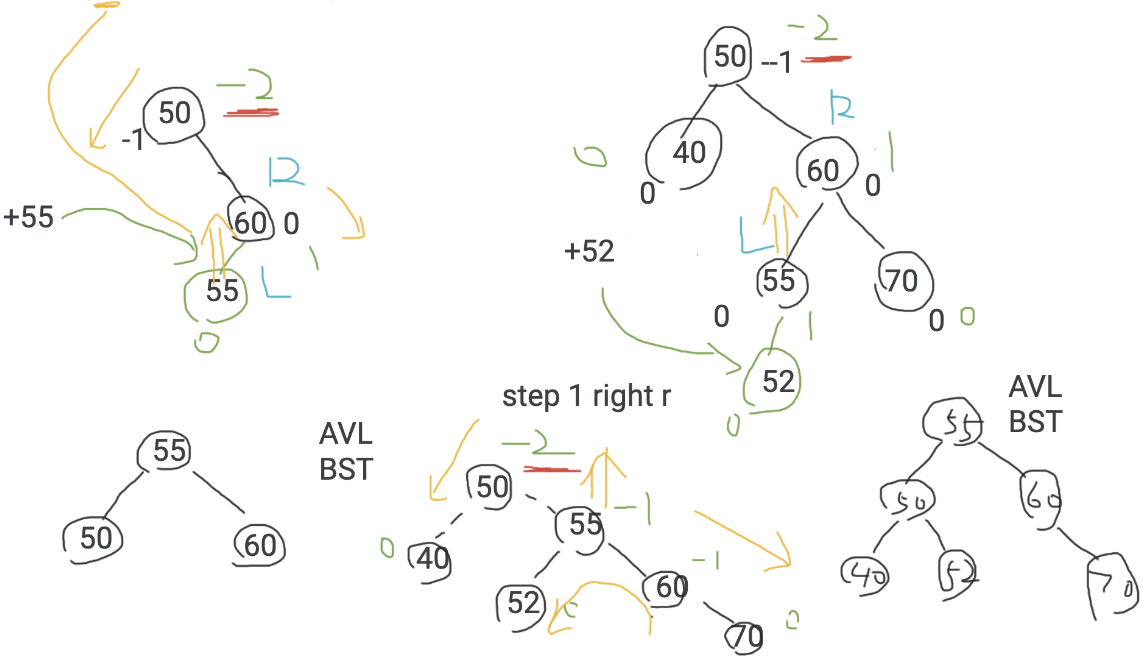
LR type

LR type -> (R45, left rotation) -> 整個right rotation



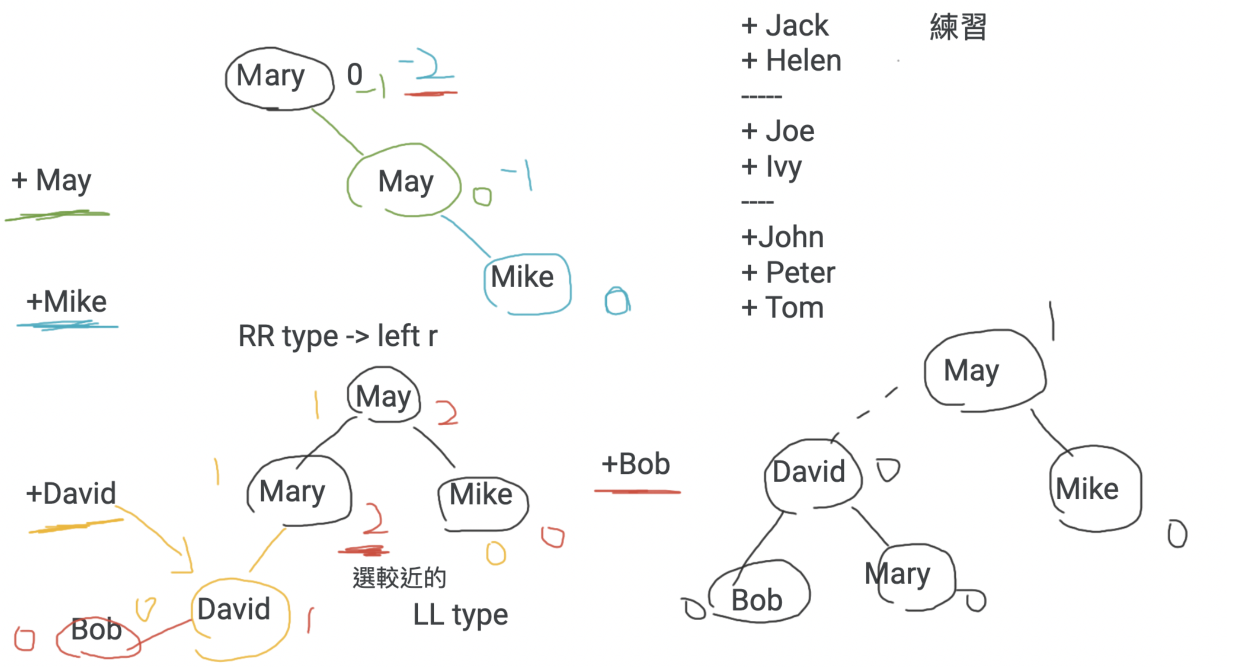
RL type

RL type -> (L55 Right rotation), 整個left rotation



例子

字串 AVL → BST 原則大小 alphabetlic order 大小



tldraw

A free and instant collaborative diagramming tool.

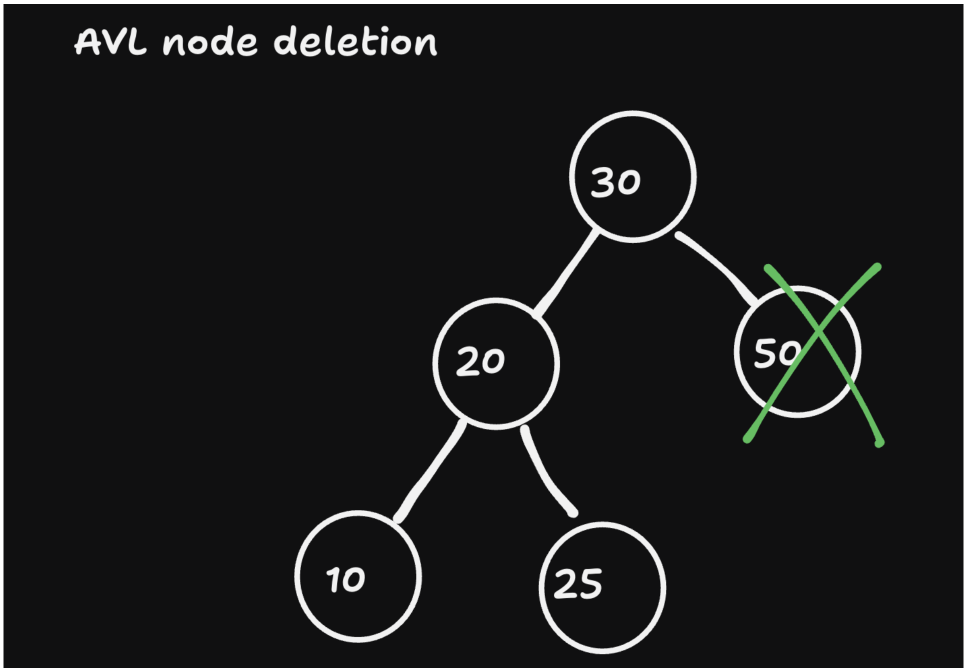
https://www.tldraw.com/r/V5WtU61rVHub3ZHeUXEg8?v=11,4492,1920,1004&p=page

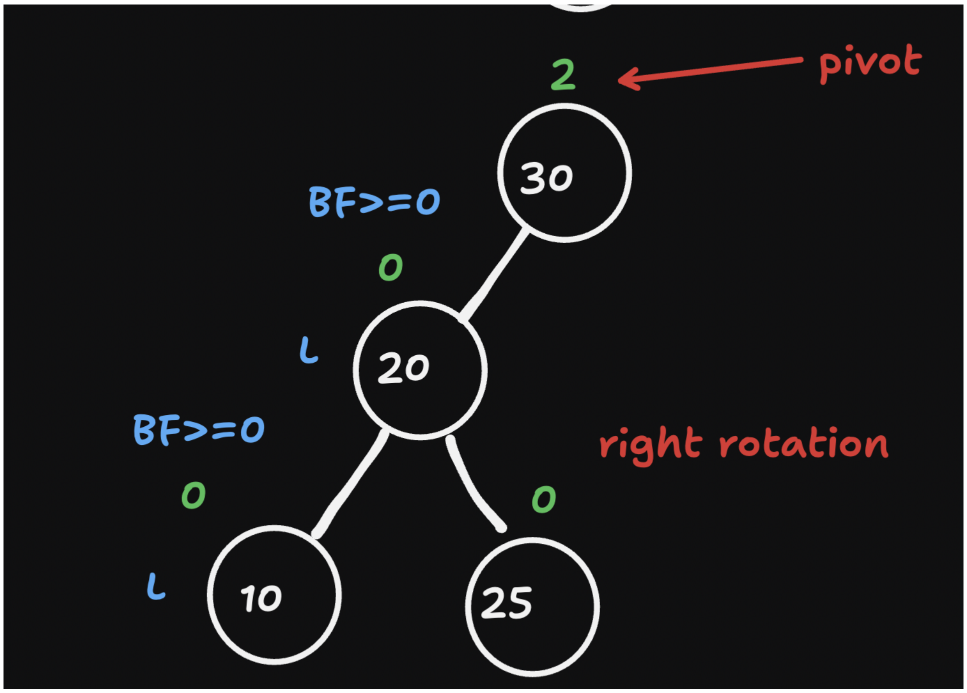
實作

```
#include <iostream>
using namespace std;

class Node{
  public:
    int key;
    Node* left;
    Node* right;
    int height;
    Node(int k):key(k), left(nullptr), right(nullptr), height(1
};

class AVLTree{
  public:
    Node* root;

    AVLTree():root(nullptr){}

    int height(Node* n){
      if(n==nullptr) return 0;
      return n->height;
    }

    int getBalance(Node* n){
      if(n==nullptr) return 0;
      return height(n->left) - height(n->right);
    }

    void updateHeight(Node* n){
      if(n!=nullptr){
        n->height = max(height(n->left), height(n->right))+1;
      }
    }

    Node* rightRotate(Node* y){
      Node* x=y->left;
      Node* T2 = x->right;

      x->right = y;
      y->left = T2;

      updateHeight(y);
      updateHeight(x);

      return x;
    }

    Node* leftRotate(Node* x){
        Node* y= x->right;
        Node* T2 = y->left;

        y->left = x;
        x->right = T2;

        updateHeight(x);
        updateHeight(y);

        return y;
    }

    Node* insert(Node* node, int key){ //內部
        //BST
        if(node==nullptr) return (new Node(key)); //樹為空 或在末
        if(key< node->key){ //
               node->left = insert(node->left, key);
        }
        else if(key> node->key){
               node->right = insert(node->right, key);
        }
        else{
            return node;
        }

        updateHeight(node);

        int balance = getBalance(node);

        if(balance >=2 && key < node->left->key){//LL
          return rightRotate(node);
        }
        if(balance >=2 && key > node->left->key){//LR
```

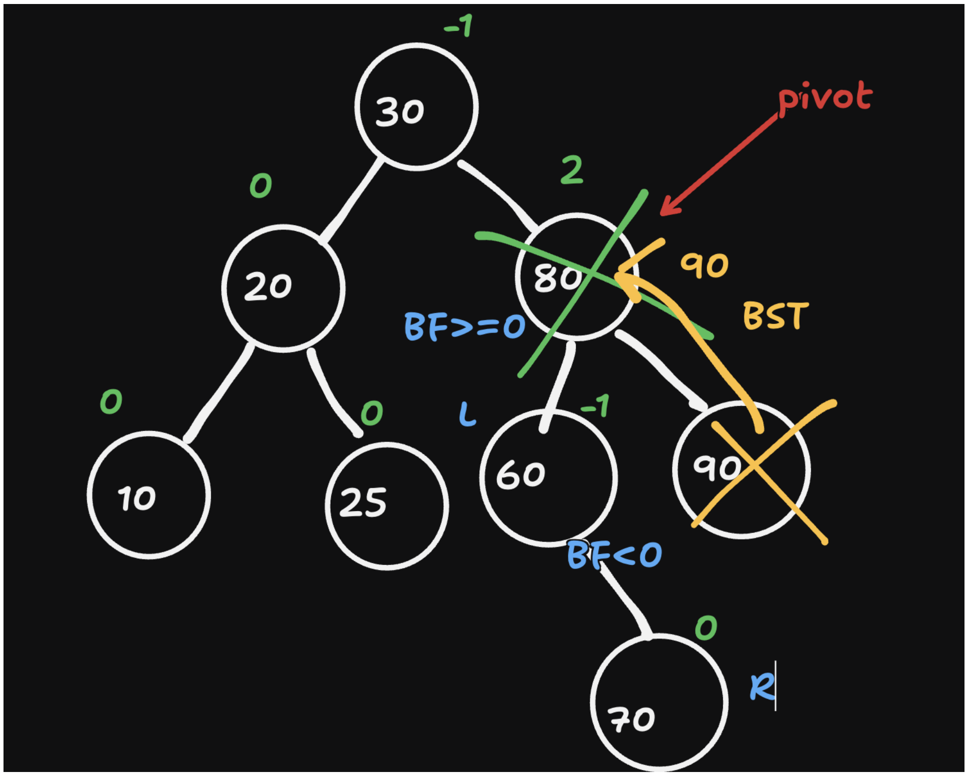
```
          node->left = leftRotate(node->left);
          return rightRotate(node);
        }
        if(balance <=-2 && key>node->right->key){ //RR
          return leftRotate(node);
        }
        if(balance <=-2 && key<node->right->key){ //RL
          node->right = rightRotate(node->right);
          return leftRotate(node);
        }

        return node;
    }

    void insert(int key){ //對外的單純method
          root = insert(root, key);
    }
};

int main()
{
    cout << "Hello, World!";
    return 0;
}
```

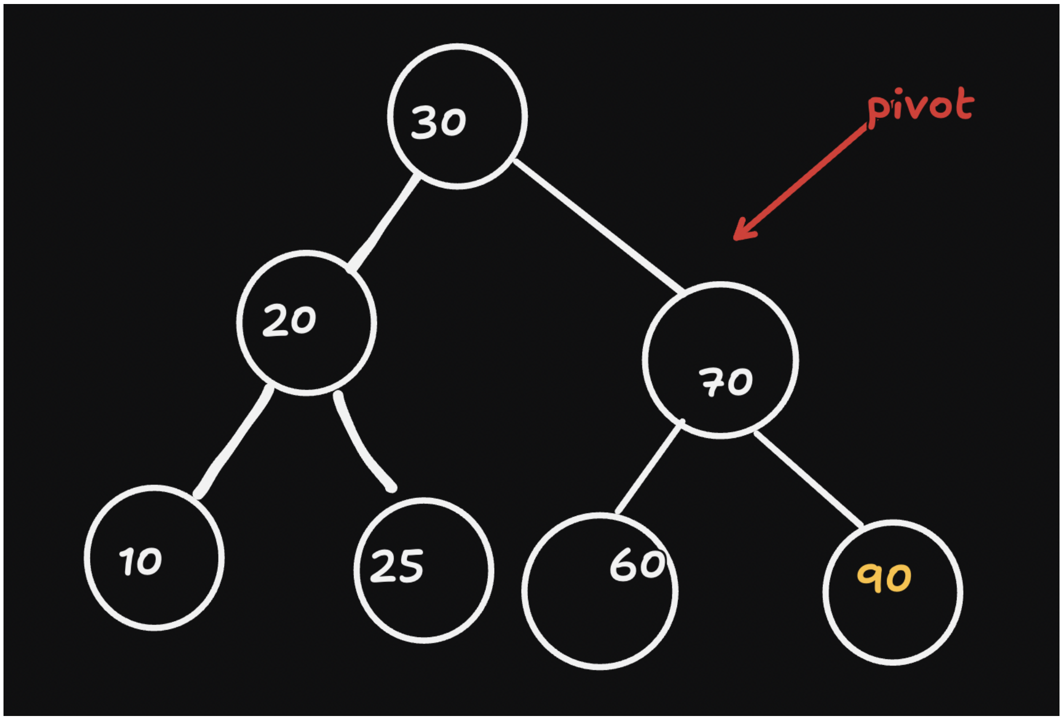
AVL node 刪除


AVL node deletion
30
20
50
10
25


2
pivot
30
BF>=0
0
L
20
BF>=0
0
right rotation
0
L
10
25

Example





刪除動作

- root → 找 pivot → |BF| >1

- pivot BF>0 + pivot →llink BF≥0 → LL
- pivot BF>0 + pivot →rlink BF<0 → LR
- pivot BF<0 + pivot →rlink BF≥0 → RL
- pivot BF<0 + pivot →rlink BF<0 → RR
- 根據類型再做對應的 rotation

```
#include <iostream>
using namespace std;

class Node{
  public:
    int key;
    Node* left;
    Node* right;
    int height;

    Node(int k):key(k), left(nullptr), right(nullptr), height(1
};

class AVLTree{
  public:
    Node* root;

    AVLTree():root(nullptr){}

    int height(Node* n){
      if(n==nullptr) return 0;
      return n->height;
    }

    int getBalance(Node* n){
      if(n==nullptr) return 0;
      return height(n->left) - height(n->right);
    }
```

```
void updateHeight(Node* n){
  if(n!=nullptr){
    n->height = max(height(n->left), height(n->right))+1;
  }
}

Node* rightRotate(Node* y){
  Node* x=y->left;
  Node* T2 = x->right;

  x->right = y;
  y->left = T2;

  updateHeight(y);
  updateHeight(x);

  return x;
}

Node* leftRotate(Node* x){
    Node* y= x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    updateHeight(x);
    updateHeight(y);

    return y;
}

Node* insert(Node* node, int key){ //內部
    //BST
    if(node==nullptr) return (new Node(key)); //樹為空 或在末
```

```
        if(key< node->key){ //
               node->left = insert(node->left, key);
        }
        else if(key> node->key){
               node->right = insert(node->right, key);
        }
        else{
            return node;
        }

        updateHeight(node);

        int balance = getBalance(node);

        if(balance >=2 && key < node->left->key){//LL
          return rightRotate(node);
        }
        if(balance >=2 && key > node->left->key){//LR
          node->left = leftRotate(node->left);
          return rightRotate(node);
        }
        if(balance <=-2 && key>node->right->key){ //RR
          return leftRotate(node);
        }
        if(balance <=-2 && key<node->right->key){ //RL
          node->right = rightRotate(node->right);
          return leftRotate(node);
        }
        return node;
    }

    void insert(int key){ //對外的單純method
          root = insert(root, key);
    }

    Node* remove(Node* node, int key){
};
```

```
      //刪除 -> BST
      if(node==nullptr) return node; //空

      if(key< node->key){
        node->left = remove(node->left);
      }
      else if(key> node->key){
        node->right = remove(node->right);
      }
      else{
        if(node->left==nullptr || node->right==nullptr){ //單子節
          //temp -> 要保留的節點 or 無子節點
          Node* temp = node->left? node->left:node->right;

          if(temp==nullptr){
            temp = node;
            node = nullptr;
            delete temp;
          }
          else{
            *node = *temp;
          }
        }
        else{
          Node* temp = minNode(node->right);
          node->key = temp->key;
          node->right = remove(node->right, temp->key); //把重整
        }
      }
    }

    void remove(int key){

    }
};
```

```
int main()
{
    AVLTree tree;
    tree.insert(94);
    tree.insert(87);
    tree.insert(945);
    return 0;
}
```