AVL树

20123211 高雨晴

1. 实验设计

在AVL树中，任一节点对应的两棵子树的最大高度差为1，因此它也被称为高度平衡树。查找、插入和删除在平均和最坏情况下的时间复杂度都是$O(\log n)$。增加和删除元素的操作则可能需要借由一次或多次树旋转，以实现树的重新平衡。

1.1 算法实现原理

1.2 算法设计与分析

创建一个类，储存avl树可以实现的功能。

```
// 建立avl树
struct avl {
   int d;
   struct avl *l;   // 左子树
   struct avl *r;   // 右子树
}*r;

class avl_tree {
   public:
      int height(avl *);         // 获取树的高度
      int difference(avl *);     // 计算给定树的子树的高度之差
      avl *rr_rotat(avl *);      // 右双旋
      avl *ll_rotat(avl *);      // 左双旋
      avl *lr_rotat(avl*);       // 左右双旋
      avl *rl_rotat(avl *);      // 右左双旋
      avl * balance(avl *);      // 平衡树
      avl * insert(avl*, int);   // 插入操作
      void show(avl*, int);      // 展示AVL树
      avl_tree() {
         r = NULL;
      }
};
```

获取树的高度

```
int avl_tree::height(avl *t) {
   int h = 0;
    // 返回树的高度，采用递归的方式
   if (t != NULL) {
      int l_height = height(t->l);
      int r_height = height(t->r);
      int max_height = max(l_height, r_height);
      h = max_height + 1;
   }
   return h;
}
```

计算给定树的子树的高度之差

```
int avl_tree::difference(avl *t) {
   int l_height = height(t->l);
   int r_height = height(t->r);
   int b_factor = l_height - r_height;   // b_factor为平衡因子
   return b_factor;
}
```

右双旋

```
avl *avl_tree::rr_rotat(avl *parent) {
   avl *t;
   t = parent->r;
   parent->r = t->l;
   t->l = parent;
   cout<<"Right-Right Rotation";
   return t;
}
```

左双旋

```
avl *avl_tree::ll_rotat(avl *parent) {
   avl *t;
   t = parent->l;
   parent->l = t->r;
   t->r = parent;
   cout<<"Left-Left Rotation";
   return t;
}
```

左右双旋

```
avl *avl_tree::lr_rotat(avl *parent) {
   avl *t;
   t = parent->l;
   parent->l = rr_rotat(t);
   cout<<"Left-Right Rotation";
   return ll_rotat(parent);
}
```

右左双旋

```
avl *avl_tree::rl_rotat(avl *parent) {
   avl *t;
   t = parent->r;
   parent->r = ll_rotat(t);
   cout<<"Right-Left Rotation";
   return rr_rotat(parent);
}
```

平衡

```
avl *avl_tree::balance(avl *t) {
    // 通过平衡因子来判断应该采取哪种旋转方式
   int bal_factor = difference(t);
   if (bal_factor > 1) {
      if (difference(t->l) > 0)
         t = ll_rotat(t);
      else
         t = lr_rotat(t);
   } else if (bal_factor < -1) {
      if (difference(t->r) > 0)
         t = rl_rotat(t);
      else
         t = rr_rotat(t);
   }
   return t;
}
```

插入

```
avl *avl_tree::insert(avl *r, int v) {
   if (r == NULL) {
      r = new avl;
      r->d = v;
      r->l = NULL;
      r->r = NULL;
      return r;
   } else if (v< r->d) {
      r->l = insert(r->l, v);
      r = balance(r);
   } else if (v >= r->d) {
      r->r = insert(r->r, v);
      r = balance(r);
   } return r;
}
```

展示树

```
void avl_tree::show(avl *p, int l) {
   int i;
   if (p != NULL) {
      show(p->r, l+ 1);
      cout<<" ";
      if (p == r)
         cout << "Root -> ";
      for (i = 0; i < l&& p != r; i++)
         cout << " ";
         cout << p->d;
         show(p->l, l + 1);
   }
}
```

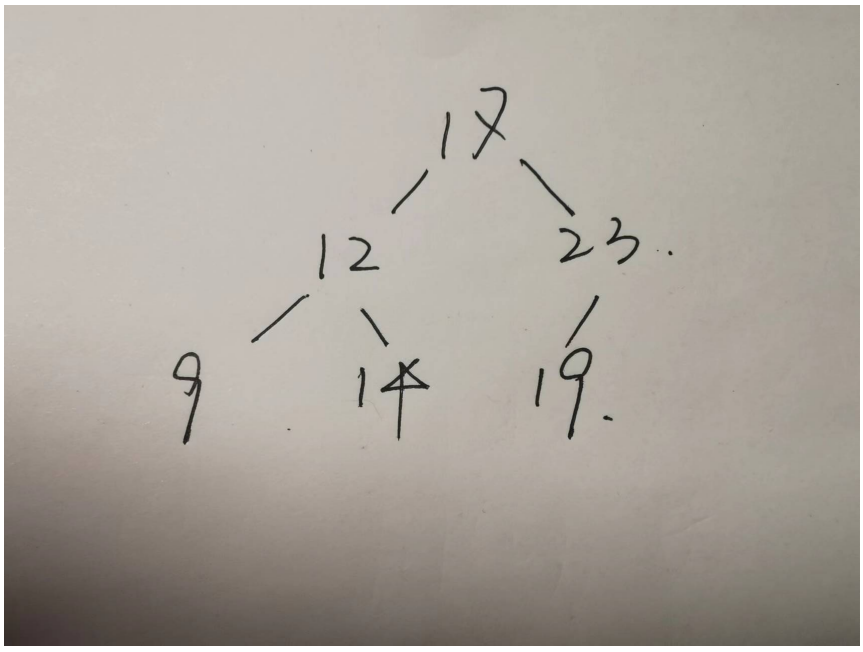
2. 实验结果说明与分析

2.1 实验设置

```
#include<iostream>
#include<cstdio>
#include<sstream>
#include<algorithm>
#define pow2(n) (1 << (n))
using namespace std;

struct avl {...
}*r;
class avl_tree {...

int avl_tree::height(avl *t) {...
int avl_tree::difference(avl *t) {...
avl *avl_tree::rr_rotat(avl *parent) {...
avl *avl_tree::ll_rotat(avl *parent) {...
avl *avl_tree::lr_rotat(avl *parent) {...
avl *avl_tree::rl_rotat(avl *parent) {...
avl *avl_tree::balance(avl *t) {...
avl *avl_tree::insert(avl *r, int v) {...
void avl_tree::show(avl *p, int l) {...

int main() {
   int c, i;
   avl_tree avl;
   while (1) {
      cout << "1.向AVL树中插入元素" << endl;
      cout << "2.展示AVL树" << endl;
      cout << "3.退出" << endl;
      cout << "输入您的选择: ";
      cin >> c;
      switch (c) {
         case 1:
            cout << "输入一个要插入的值: ";
            cin >> i;
            r = avl.insert(r, i);
         break;
```

```
            case 2:
                if (r == NULL) {
                    cout << "树还是空的" << endl;
                    continue;
                }
                cout << "AVL树:" << endl;
                avl.show(r, 1);
                cout<<endl;
            break;
            case 3:
                exit(1);
            break;
            default:
                cout << "没有这个选项哟" << endl;
        }
    }
    return 0;
}
```

用于测试的树：



2.2 实验结果以及相应的分析

将测试树的元素依次输入AVL树。

输出结果：

```
AVL 树:
   23    19 Root -> 17    14   12    9
```

可以看出找到了根为17. 我们再插入一个先序遍历的功能来验证AVL树是否建好。

```
void avl_tree::preorder(avl *t) {
   if (t == NULL)
      return;
      cout << t->d << " ";
      preorder(t->l);
      preorder(t->r);
}
```

输出结果为：

```
Preorder Traversal:
17 12 9 14 23 19
```

和我们对手画的树进行先序遍历结果是一样的。