

实验三

1. 实验要求

- 1) 实现红黑树左旋操作 LEFT-ROTATE(T, x)，实现右旋操作 RIGHTROTATE(T, x)；
- 2) 实现红黑树插入节点的操作 RB-INSERT(T, z)以及插入之后修正为红黑树的算法 RB-INSERT-FIXUP(T, x)；
- 3) 实现红黑树删除节点的操作 RB-DELETE(T, z)以及插入之后修正为红黑书的算法 RB-DELETE-FIXUP(T, x)；
- 4) 实现按要求数据构建顺序统计树的操作；
- 5) 实现遍历输出构建好的红黑树的操作；
- 6) 实现查找顺序统计树的第 i 小关键字的操作 OS-SELECT($T.root, i$)；
- 7) 为了验证第二个实验的正确性，要求编写一个检测程序，使用中位数一章的线性时间的选择算法 Select(a, p, r, i) 在输入数据找到找到第 $n/3$ 小的节点和第 $n/4$ 小节点，与 OS-SELECT($T.root, i$) 的结果 delete_data 进行对比检查

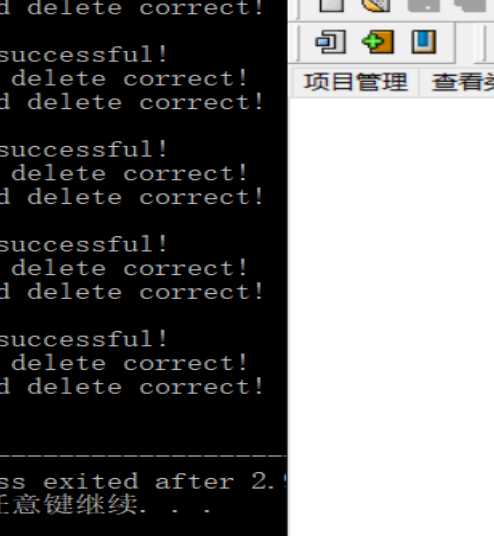
2. 实验环境

编译环境: Dev C++ 5.11

机器内存：16G

时钟主频: 2.20GHz

3. 实验过程



The screenshot shows a Windows desktop with two windows. The command prompt window on the left displays the output of a program, showing five successful reads and deletions. The file explorer window on the right shows the 'C:\Users\95850\Desktop' directory, with a toolbar containing icons for file operations and a search bar.

Command Prompt Output:

```
read successful!  
first delete correct!  
second delete correct!  
  
read successful!  
first delete correct!  
second delete correct!  
  
read successful!  
first delete correct!  
second delete correct!  
  
read successful!  
first delete correct!  
second delete correct!  
  
read successful!  
first delete correct!  
second delete correct!
```

File Explorer Window:

- Address bar: C:\Users\95850\Desktop
- Toolbar: 文件[F], 编辑[E], 搜索[S], (global), 项目管理, 查看类, 调试
- Icons: File Explorer icons including a folder, a document, a trash can, and a search icon.

4. 实验关键代码截图（结合文字说明）

1. 重要的全局变量：dt[1...6]为每插入十个结点的运行时间，dt[7]为插入总时间。

```
LARGE_INTEGER nFreq; //cpu frequency
LARGE_INTEGER t1[63]; //begin    //time1: t[1...60]  time2: t[61],t[62]
LARGE_INTEGER t2[63]; //end
double dt[8]; //time
```

```
int arr[5] = {12,24,36,48,60};
int count;
```

```
FILE *fp;
```

2. 顺序统计树结构：

```
enum color
{
    RED,
    BLACK
};

typedef struct
{
    int key;
    int size;
    int color;
}TreeData;

typedef struct OSTNode
{
    TreeData data;
    struct OSTNode *parent;
    struct OSTNode *left;
    struct OSTNode *right;
}OSTNode, *OSTree;

const OSTree NIL = new OSTNode;
```

3. 随机生成函数

```

srand((unsigned)time(NULL));
for(i = 0; i < m; i++)
{
    num[i] = rand()%1000;
    for(j = 0; j < i; j++)
    {
        if(num[j] == num[i])?
        {
            num[i] = rand()%20;
            j = 0;
        }
    }
}

```

4.左旋，注意维护 size 即可。

```

void LEFT_ROTATE(OSTree &T, OSTree x)
{
    OSTree y;
    y = x->right;
    x->right = y->left;
    if(x->left != NIL)
        y->left->parent = x;
    y->parent = x->parent;
    if(x->parent == NIL)
        T = y;
    else if(x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;

    y->data.size = x->data.size;
    x->data.size = x->left->data.size + x->right->data.size + 1;
}

```

5. INSERT 中对 size 的维护

```

while(x != NIL)
{
    y = x;
    if(z->data.key < x->data.key)
    {
        ++x->data.size;
        x = x->left;
    }
    else
    {
        ++x->data.size;
        x = x->right;
    }
}

```

6. DELETE 中对 SIZE 的维护

```
k = y;
while(k != T)
{
    --k->parent->data.size;
    k = k->parent;
}
```

7.建立顺序统计树以及记录插入时间

```
void CreatOSTree(OSTree &T,int num[],int n)
{
    TreeData e;
    e.color = RED;
    int i;
    for(i = 1;i <= n;i++)
    {
        e.key = num[i-1];
        QueryPerformanceCounter(&t1[i]);
        OS_INSERT(T,e);
        QueryPerformanceCounter(&t2[i]);
    }
}
```

8.中序遍历，另外两个遍历此处略

```
void InOrderTraverse(OSTree T)
{
    if(T != NIL)
    {
        InOrderTraverse(T->left);
        // printf("( %d,%d,%d )",T->data.key,T->data.color,T->data.size);
        switch (count)
        {
            case 0:fp = fopen("../output/size12/inorder.txt","a");break;
            case 1:fp = fopen("../output/size24/inorder.txt","a");break;
            case 2:fp = fopen("../output/size36/inorder.txt","a");break;
            case 3:fp = fopen("../output/size48/inorder.txt","a");break;
            case 4:fp = fopen("../output/size60/inorder.txt","a");break;
        }
        fprintf(fp,"( %d,%d,%d )",T->data.key,T->data.color,T->data.size);
        fclose(fp);
        InOrderTraverse(T->right);
    }
}
```

9.顺序统计树中查找第 i 小的结点

```

OSTree OS_SELECT(OSTree x,int i)
{
    int r = 1;
    if(x->left != NIL)
        r = x->left->data.size + 1;
    if(i == r)
        return x;
    else if(i < r)
        return OS_SELECT(x->left,i);
    else
        return OS_SELECT(x->right,i - r);
}

```

10.线性时间查找第 i 小数的随机算法

```

int RANDOM(int p,int q)
{
    int i,number;
    srand((unsigned)time(NULL));
    number = rand()%(q-p+1)+p;
    return number;
}

int PARTITION(int a[],int p,int r)
{
    int x = a[r];
    int i = p-1;
    for(int j = p;j <= r-1;j++)
    {
        if(a[j] <= x)
        {
            i = i + 1;
            swap(&a[j],&a[i]);
        }
    }
    swap(&a[i+1],&a[r]);
    return i+1;
}

int RANDOMIZED_PARTITION(int a[],int p,int r)
{
    int i = RANDOM(p,r);
    swap(&a[r],&a[i]);
    return PARTITION(a,p,r);
}

```

```

int RANDOMIZED_SELECT(int a[],int p,int r,int i)
{
    if(p == r)
        return a[p];
    int q = RANDOMIZED_PARTITION(a,p,r);
    int k = q - p + 1;
    if(i == k)
        return a[q];
    else if(i < k)
        return RANDOMIZED_SELECT(a,p,q-1,i);
    else
        return RANDOMIZED_SELECT(a,q+1,r,i-k);
}

```

11.变量及顺序统计树初始化

```

// Random_Num();
int i,j,m = 62;
int num[m];
memset(num,0,sizeof(num));
QueryPerformanceFrequency(&nFreq);
for(count = 0;count < 5;count++)
{
    NIL->left = NIL->right = NIL->parent = NULL;
    NIL->data.key = INF;
    NIL->data.color = BLACK;
    NIL->data.size = 0;
    OSTree T;
    T = NIL;
    T->parent = NIL;
    memset(num,0,sizeof(num));
    memset(t1,0,sizeof(t1));
    memset(t2,0,sizeof(t2));
}

```

12.判断删除结点是否正确

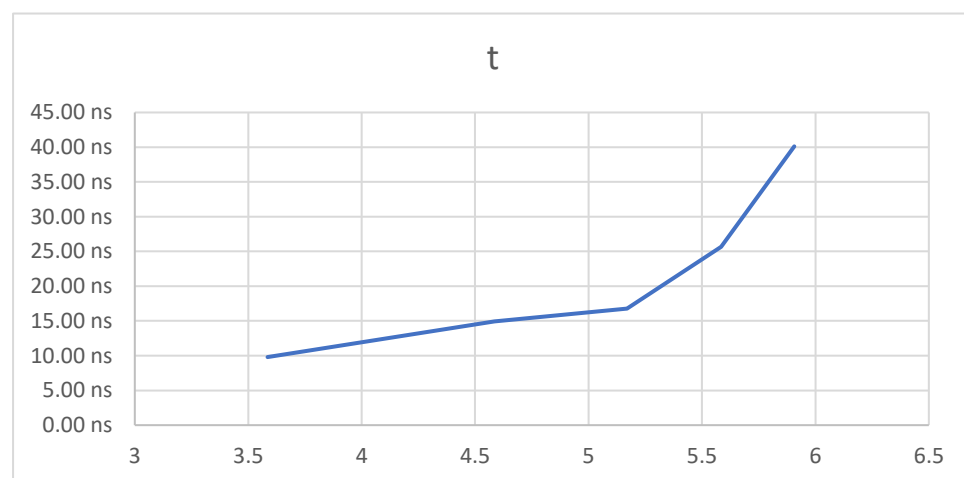
```

int y = RANDOMIZED_SELECT(num,0,m,m/3);
CreatOSTree(T,num,m);
PreOrderTraverse(T);
InOrderTraverse(T);
PostOrderTraverse(T);
OSTree x;
x = OS_SELECT(T,m/3);
// printf("%d\n%d",x->data.key,RANDOMIZED_SELECT(num,0,m,m/3));
if(x->data.key == y)
    printf("first delete correct!\n");
else
    printf("first delete wrong!\n");

```

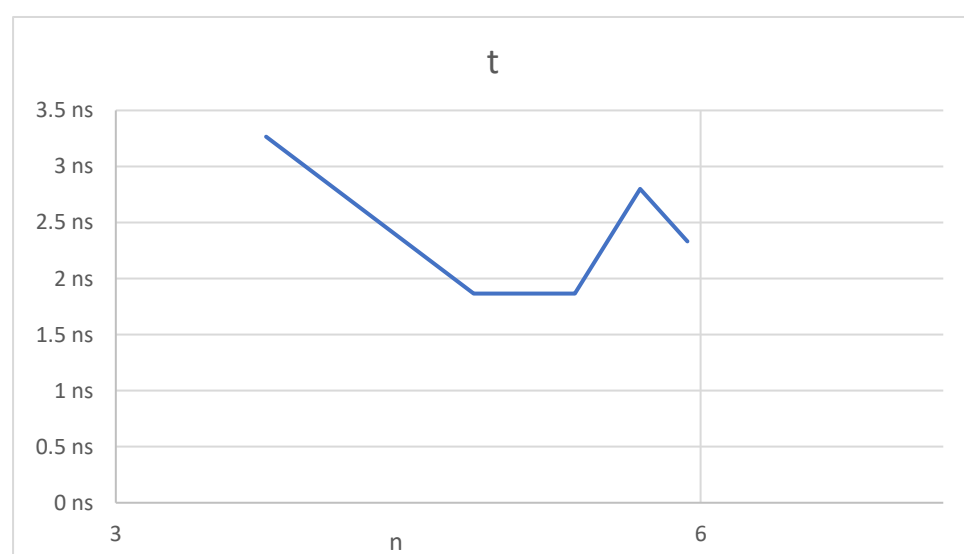
7. 实验结果、分析（结合相关数据图表分析）

插入操作：（ n 已取 \log_2 对数）



对原始数据的观察可以发现，无论 n 的大小，前十个结点插入的运行时间明显高于后面的结点，这应该与空间分配的消耗有关。另外，由于数据规模较小以及测量工具的局限，因此得到的测量数据难以支持算法导论中插入操作 $O(\lg n)$ 的论断。

删除操作：（ n 已取 \log_2 对数）



由于本实验删除操作的数据规模极小，单次删除操作运行时间与待删除结点在树中的位置有较大关系，因此得到的测量数据难以支持算法导论中删除操作 $O(\lg n)$ 的论断。

8. 实验心得

通过对红黑树/顺序统计树各种操作算法的实现，我较清楚地理解了红黑树/顺序统计树的基本原理，了解了红黑树/顺序统计树的优缺点，了解了红黑树的应用方向。进一步学习了怎样测量程序运行时间以及图表的制作。
