作业05 Splay和AVL树对比

编写代码：

- 关于AVL的树的代码，我直接使用我第一次作业的内容
- 关于SplayTree的代码，我使用的是 https://github.com/BigWheel92/Splay-Tree/blob/master/SplayTree.h

实验结果：

首先介绍我实验的逻辑，我固定了测试数据集的大小为 $n = 1024$， k/n的比例依次如下， $m = n \times 1024$;:

$$rate[] = \{0.00125, 0.0025, 0.005, 0.01, 0.05, 0.25, 0.45, 0.65, 0.95\}; \tag{1}$$

实验开始之前首先使用随机化生成数据函数，产生长度为 n 的一个数组作为随机数。

```
void generateData(int n){
    map<int,bool> dataExit;
    for (size_t i = 0; i < n; i++)
    {
        int dataRand = rand();
        while(dataExit.count(dataRand) == 1){
            dataRand = rand();
        }
        dataExit[dataRand] = true;
        dataAll.push_back(rand());
    }
}
```

然后，把所有生成的数据插入到两种要测试的数据结构中，最后计时运行测试。

```
void runSingleTest(long int k, long int m){
    AVL<int> avl;
    SplayTree<int,int> spTree;

    // 插入数据
    for (size_t i = 0; i < dataAll.size(); i++)
    {
        avl.insert(dataAll[i]);
        spTree.insert(dataAll[i], dataAll[i]);
    }

    long int target = 0;
    clock_t start,end;
    start = clock();
    for (size_t i = 0; i < m; i++){
        // avl.search(dataAll[target]);
        avl.search(dataAll[dataAll.size() - 1 - target]);
        target = (target + 1) % k;
    }
```

```
        end = clock();
        cout << "k = " << k << " m = " << m << endl;
        cout << "AVL search Time = " << double(end-start) / CLOCKS_PER_SEC << "s"<<endl;

        target = 0;
        start = clock();
        for (size_t i = 0; i < m; i++){
            spTree.search(dataAll[dataAll.size() - 1 - target]);
            // spTree.search(dataAll[target]);
            target = (target + 1) % k;
        }
        end = clock();
        cout << "spTree search Time = " << double(end-start) / CLOCKS_PER_SEC << "s"<<endl;
        cout <<endl;
    }
```

为了能够测试不同比例的情况，还需要做一次封装。

```
    void runTest(){
        double rate[] = {0.00125,0.0025, 0.005, 0.01, 0.05 , 0.25, 0.45, 0.65, 0.95};
        long int n = 1024;
        long int m = n * 1024;

        generateData(n);

        for (size_t i = 0; i < 9; i++)
        {
            long int k = n * rate[i];
            runSingleTest(k , m);
        }
    }
```

测试基于：$n = 1024, m = 1048576$

比值(k/n)	AVL	Splay
0.00125	0.037686s	0.007402s
0.0025	0.034656s	0.029104s
0.005	0.035381s	0.054224s
0.01	0.036187s	0.076795s
0.05	0.036285s	0.228936s
0.25	0.05376s	0.346071s
0.45	0.064352s	0.398169s
0.65	0.059331s	0.429622s
0.95	0.069971s	0.452562s

分析原因

- SplayTree的性能特别依赖于局部性的体现，如果我们搜索的局部性很好（小），恰恰是这个树结构的某个区域的，因为节点提升到了根节点，带来的性能优势就很明显，但是如果局部性不太好，搜索的范围长，或者搜索的没有什么规律，AVL的性能就大大折扣。
- 综上而说，课堂上讲到说实际上用到伸展树的情况很少，我觉得这也是因为在真实搜索的场景中，这种局部性的体现其实并不是很强，反倒是更多情况是无规律的随机搜索，所以伸展树应用也很有限。
- 此外，基于我对这个插入算法、搜索算法的理解，SPlayTree相比较AVL，在搜索的时候也会调整树的结构，但是我认为这种调整并不是一个好的表现。（而且每次搜索都调整，消耗的代价太大。这种调整牺牲了时间，换来了后续搜索的时候，数据局部化前提下的性能优势，但是实际上这种局部化数据搜索在真实情况中并不是那么突出，所以我认为这种代价换的**不划算**）