暨南大学本科实验报告专用纸(附页)

实现一个线段树泛型模板

* 实验项目类型：设计性
*此表由学生按顺序填写

课程名称 面向对象程序设计 成绩评定
实验项目名称 实现一个线段树泛型模板
指导老师 干晓聪
实验项目编号 1 实验项目类型 设计性 实验地点 数学系机房
学生姓名 郭彦培 学号 2022101149
学院 信息科学技术学院 系 数学系 专业 信息管理与信息系统
实验时间 2023 年 11 月 1 日上午 ~ 2023 年 11 月 1 日中午

一、 实验目的

熟悉泛型类、泛型方法的使用与泛型思想在编码中的应用。利用 C++语言实现泛型。

二、 实验环境

计算机：PC X64

操作系统：Windows

编程语言：C++

IDE：Visual Studio Code

在线测试平台：leetcode

三、 程序原理

线段树是一种较为复杂的数据结构，旨在快速解决区间数据批量修改与特征统计。

本类实现了一个可以批量地对数据进行线性空间内加和运算的线段树，统计内容为区间内的最大值，对于每个操作：

1. 修改单点：时间复杂度为$O(\log n)$
2. 修改区间：均匀修改与查询后最坏时间复杂度为每点渐进$O(\log(nm))$，n 为内容总数，m 为修改区间长度
3. 查询区间：$O(\log n)$

四、 程序代码

文件`sis6\segTree.`实现了一个`segTree`类

```
#include <vector>
#include <map>
#include <string>
#include <string.h>
#include <math.h>
#include <set>
#include <algorithm>
#include <iostream>
#include <queue>

template <class TYPE_NAME>
class lazyTree
{
    /*
     *  TYPE_NAME 需要支持：+ += != 和自定义的合并运算符
     *  实现了懒惰标记，仅支持区间批量增加
     *  区间批量减需要 TYPE_NAME 支持-，且有-a = 0 - a
     *  额外处理了一个单点修改为对应值的函数，非原生实现，其复杂度为 4logn
     *  不提供在线
     *  不提供持久化
     */
private:
    std::vector<TYPE_NAME> d;
    std::vector<TYPE_NAME> a;
    std::vector<TYPE_NAME> b;
    int n;
    const TYPE_NAME INI = 0; // 不会影响合并运算的初始值，如 max 取 INF，min 取 0，mti 取 1

    void subbuild(int s, int t, int p)
    {
        if (s == t)
        {
            d[p] = a[s];
```

```
            return;
        }
        int m = s + ((t - s) >> 1); //  (s+t)/2
        subbuild(s, m, p * 2);
        subbuild(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
        //    合并运算符 ↑
    }

    TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d[p];
        int m = s + ((t - s) >> 1);
        if (b[p] != 0)
        {
            d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为
应用懒惰标记
            d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为
应用懒惰标记
            b[p * 2] += b[p];               // 下传标记，无需修改
            b[p * 2 + 1] += b[p];           // 下传标记，无需修改
            b[p] = 0;
        }
        TYPE_NAME ansl = INI;
        TYPE_NAME ansr = INI;
        if (l <= m)
            ansl = subGetSum(l, r, s, m, p * 2);
        if (r > m)
            ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);
        return ansl + ansr;
        // 合并运算符↑
    }

    void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
    {
        if (l <= s && t <= r)
        {
            d[p] += (t - s + 1) * c; // 合并运算符的高阶运算 此处运算为修改整
匹配区间值
            b[p] += c;               // 记录懒惰标记，无需修改
            return;
        }
        int m = s + ((t - s) >> 1);
```

```
        if (b[p] != 0 && s != t)
        {
            d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应用懒惰标记
            d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应用懒惰标记
            b[p * 2] += b[p];               // 下传标记，无需修改
            b[p * 2 + 1] += b[p];           // 下传标记，无需修改
            b[p] = 0;
        }
        if (l <= m)
            subUpdate(l, r, c, s, m, p * 2);
        if (r > m)
            subUpdate(l, r, c, m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
        //    合并运算符 ↑
    }

public:
    lazyTree(TYPE_NAME _n)
    {
        n = _n;
        d.resize(4 * n + 5);
        a.resize(4 * n + 5);
        b.resize(4 * n + 5);
    }

    void build(std::vector<TYPE_NAME> _a)
    {
        a = _a;
        subbuild(1, n, 1);
    }

    TYPE_NAME getsum(int l, int r)
    {
        return subGetSum(l, r, 1, n, 1);
    }

    void update(int l, int r, TYPE_NAME c) // 修改区间
    {
        subUpdate(l, r, c, 1, n, 1);
    }

    void update(int idx, TYPE_NAME tar)
```

```
    { // 修改单点，未测试
        TYPE_NAME tmp = getsum(idx, idx);
        tar -= tmp;
        subUpdate(idx, idx, tar, 1, n, 1);
    }
};
```

五、 出现的问题、原因与解决方法

我经常参与算法竞赛，非常熟悉 C++与线段树，因此编码过程非常顺利。

六、 测试数据与运行结果

测试数据实例化泛型为 int_32

输入数据规则：第一行包含两个整数m, n，表示这列数字的个数和操作总数

第二行包含n个用空格分隔的数，其中第i个数字表示数列第i项的初始值

接下来m行每行包含 3-4 个整数，表示一个操作。具体如下：

1. `1 x y k`:将区间`[x,y]`内每个数加上k
2. `2 x y`:输出区间`[x,y]`内每个数的和

样例运行结果如下：

输入	输出	解释
`5 5` `1 5 4 2 3`		初始化数据
`2 2 4`	`11`	求出`[2,4]`内元素和
`1 2 3 2`		将`[2,3]`内所有元素+2
`2 3 4`	`8`	求出`[3,4]`内元素和
`1 1 5 1`		将`[1,5]`内所有元素+1
`2 1 4`	`20`	求出`[1,4]`内元素和

注：测试平台`leetcode`的特性为直接向函数传参，因此不需要实现输入输出。