暨南大学本科实验报告专用纸(附页)

线段树`segTree`

课程名称_____数据结构_____成绩评定_____
实验项目名称__线段树`segTree`__指导老师___干晓聪___
实验项目编号__13__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_2024 年 6 月 13 日上午_~__2024 年 7 月 13 日中午___

1. 实验目的

实现一个泛型线段树`segTree`库

2. 实验环境

计算机：PC X64

操作系统：Windows + Ubuntu20.0LTS

编程语言：C++：GCC std20

IDE：Visual Studio Code

3. 程序原理

线段树可以在 $\mathbb{O}(\log n)$ 的时间复杂度内实现单点修改、区间修改、区间查询等操作。

定义合并运算符 “$\oplus$” 及其高阶运算 “$\otimes$”

对于一个连续的序列a递归地对连续两个区域进行合并，形成新序列t

$$s.t. \forall i \in t, s_i = s_{i*2} \oplus s_{i*2+1} \tag{1}$$

其中

$$\forall i \in a, \exists f, s.t. s_{f+i} = a_i \tag{2}$$

则整个 s 序列形成二叉搜索树形结构。

定义运算符a_i^T为递归地向上访问所经过的所有节点集合， s_i^L, s_i^R分别为:

$$s_i^L = \mathrm{Val}^{a_i} \min(i \in \mathbb{N} s.t. s_i \in a_i^T) \tag{3}$$

$$s_i^R = \mathrm{Val}^{a_i} \max(i \in \mathbb{N} s.t. s_i \in a_i^T) \tag{4}$$

则可以描述:

对于区间求合并值：给定区间$[l,r]$

$$\sum_{i\in[l,r]}^{\oplus} a_i \Leftrightarrow \sum_{k\in\{i \mid i\in a_i^T\}}^{\oplus} \left(s_k^R - s_k^L\right) \otimes s_k \tag{5}$$

易得上式k的规模为$\mathbb{O}(\log_2 \mathbb{N})$

对于区间合并修改，保留懒惰标记L_i

对于修改：区间$[l,r]$均$\oplus\, c$

$$\begin{aligned} &\forall i \in [l,r], s_i \oplus c \\ \Leftrightarrow\ &\forall k \in \left\{a_i^T, i \in [l,r]\right\} s.t. \left[s_k^R, s_k^L\right], L_k \oplus \left(s_k^R - s_k^L\right) \otimes c \end{aligned} \tag{6}$$

则每次查询改为

$$\sum_{i\in[l,r]}^{\oplus} a_i \Leftrightarrow \sum_{k\in\{i \mid i\in a_i^T\}}^{\oplus} \left(\left(s_k^R - s_k^L\right) \otimes s_k + \sum_{i\in a_k^T} L_i \right) \tag{7}$$

易得两者规模均为$\mathbb{O}(\log_2 \mathbb{N})$

在每次搜索时按$L_{i*2} \otimes \left(s_{i*2}^R - s_{i*2}^L\right) \oplus L_{i*2+1} \otimes \left(s_{i*2+1}^R - s_{i*2+1}^L\right) = \left(s_i^R - s_i^L\right)$,
则可以证明，维护懒惰标记的均摊复杂度为$\mathbb{O}(1)$

具体实现参考代码，没这么复杂（只是用数学语言描述比较麻烦而已）

4. 程序代码

4.1. segTree.h

```cpp
#include <template_overAll.h>

// AC 带懒惰标记线段树
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:
    vector<TYPE_NAME> d;
    vector<TYPE_NAME> a;
    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);
```

```
89          a.resize(4 * n + 5);
90          b.resize(4 * n + 5);
91      }
92
93      void build(vector<TYPE_NAME> _a)
94      {
95          a = _a;
96          subbuild(1, n, 1);
97      }
98
99      TYPE_NAME getsum(int l, int r)
100     {
101         return subGetSum(l, r, 1, n, 1);
102     }
103
104     void update(int l, int r, TYPE_NAME c) // 修改区间
105     {
106         subUpdate(l, r, c, 1, n, 1);
107     }
108
109     void update(int idx, TYPE_NAME tar)
110     { // 修改单点, 未测试
111         TYPE_NAME tmp = getsum(idx, idx);
112         tar -= tmp;
113         subUpdate(idx, idx, tar, 1, n, 1);
114     }
115 };
```

4.2. `template_overAll.h`

```
1  #include <vector>
2  #include <map>
3  #include <string>
4  #include <string.h>
5  #include <math.h>
6  #include <set>
7  #include <algorithm>
8  #include <iostream>
9  #include <queue>
10
11 using namespace std;
12
13 #define ll long long
14 #define pb push_back
15 #define ld long double
16 const ll int maxn = 1E5+10;
17 const ll int mod1 = 998244353;
18 const ll int mod2 = 1E9+7;
19
20 #define _IN_TEMPLATE_
```

```
ll int str2int(string s)
{
    ll int rec = 0;
    ll int pw = 1;
    for(int i = s.length()-1;i >= 0;i --)
    {
        int gt = s[i] - '0';
        if(gt < 0 || gt > 9) return INT64_MAX;
        rec += gt * pw;
        pw *= 10;
    }
    return rec;
}

vector<ll int> testReadLine()
{
    string s;
    getline(cin,s);
    s.push_back(' ');
    vector<ll int> rearr;
    vector<string> substring;
    string ts;
    for(int i = 0;i < s.size();i ++)
    {
        if(s[i] == ' '){
            substring.push_back(ts);
            ts.clear();
        } else ts.push_back(s[i]);
    }
    for(int i = 0;i < substring.size();i ++)rearr.push_back(str2int(substring[i]));
    return rearr;
}
```

5. 测试数据与运行结果

代码通过在线平台`LUOGU.org`正确性测试