

# 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/JAVA 语言 成绩评定           

实验项目名称 实现一个线段树泛型模板 (JAVA 实现)

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：JAVA

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：

# 暨南大学本科实验报告专用纸(附页)

---

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 `sis7\segTree.java` 实现了一个 `segTree` 类

```
package sis7;

import java.util.ArrayList;

public class segTree<N> {
    private ArrayList<N> d;
    private ArrayList<N> a;
    private ArrayList<N> b;
    private int n;
    private N INI;

    private void subbuild(int s, int t, int p)
    {
        if (s == t)
        {
            d.set(p, a.get(s));
            return;
        }
        int m = s + ((t - s) >> 1); // (s+t)/2
        subbuild(s, m, p * 2);
        subbuild(m + 1, t, p * 2 + 1);
        d.set(p, merge(d.get(p * 2), d.get(p * 2 + 1)));
        // 合并运算符 ↑
    }

    private N subGetSum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d.get(p);
        int m = s + ((t - s) >> 1);
        if (b.get(p) != null)
        {
            d.set(p * 2, merge(d.get(p * 2), b.get(p))); // 合并运算符的高阶
            // 此处运算为应用懒惰标记
        }
    }
}
```

# 暨南大学本科实验报告专用纸(附页)

```
        d.set(p * 2 + 1, merge(d.get(p * 2 + 1), b.get(p))); // 合并运算符的高阶运算 此处运算为应用懒惰标记
        b.set(p * 2, merge(b.get(p * 2), b.get(p))); // 下传标记, 无需修改
        b.set(p * 2 + 1, merge(b.get(p * 2 + 1), b.get(p))); // 下传标记, 无需修改
        b.set(p, null);
    }
    N ans1 = INI;
    N ansr = INI;
    if (l <= m)
        ans1 = subGetSum(l, r, s, m, p * 2);
    if (r > m)
        ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);
    return merge(ans1, ansr);
    // 合并运算符↑
}

private void subUpdate(int l, int r, N c, int s, int t, int p)
{
    if (l <= s && t <= r)
    {
        d.set(p, merge(d.get(p), merge(c, (t - s + 1)))); // 合并运算符的高阶运算 此处运算为修改整匹配区间值
        b.set(p, merge(b.get(p), c)); // 记录懒惰标记, 无需修改
        return;
    }
    int m = s + ((t - s) >> 1);
    if (b.get(p) != null && s != t)
    {
        d.set(p * 2, merge(d.get(p * 2), merge(b.get(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.set(p * 2 + 1, merge(d.get(p * 2 + 1), merge(b.get(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.set(p * 2, merge(b.get(p * 2), b.get(p))); // 下传标记, 无需修改
        b.set(p * 2 + 1, merge(b.get(p * 2 + 1), b.get(p))); // 下传标记, 无需修改
        b.set(p, null);
    }
    if (l <= m)
        subUpdate(l, r, c, s, m, p * 2);
```

# 暨南大学本科实验报告专用纸(附页)

---

```
        if (r > m)
            subUpdate(1, r, c, m + 1, t, p * 2 + 1);
        d.set(p, merge(d.get(p * 2), d.get(p * 2 + 1)));
        //    合并运算符 ↑
    }

    private N merge(N a, N b)
    {
        return a;
    }

    private N merge(N a, int b)
    {
        return a;
    }

    public segTree(int _n, N _INI)
    {
        n = _n;
        INI = _INI;
        d = new ArrayList<N>(4 * n + 5);
        a = new ArrayList<N>(4 * n + 5);
        b = new ArrayList<N>(4 * n + 5);
    }

    public void build(ArrayList<N> _a)
    {
        a = _a;
        subbuild(1, n, 1);
    }

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

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

    public void update(int idx, N tar)
    { // 修改单点, 未测试
```

# 暨南大学本科实验报告专用纸(附页)

```
    N tmp = getsum(idx, idx);
    tar = merge(tar, tmp);
    subUpdate(idx, idx, tar, 1, n, 1);
}

}
```

## 五、 出现的问题、原因与解决方法

在将 C++ 代码写成 JAVA 的过程中，遇到的问题是 JAVA 中并没有运算符重载，因此选择用函数 merge 实现区间的合并。

这样在保证代码可以接受的类型下不会出错。

## 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 x y k: 将区间  $[x, y]$  内每个数加上  $k$
- 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

# 暨南大学本科实验报告专用纸(附页)

---

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。