**POJ-2528**

<https://blog.csdn.net/qq_41765114/article/details/90179868>

我们知道，在实际应用中使用线段树的时候结点 Node 数组 tree 一般都要开 4 倍的空间大小（4\*maxn），但是对于一些情况下 maxn 可能特别大，但用到的点却很少，如果强行开 4 倍空间的话很可能直接爆栈。

因此我们需要使用到 离散 化，离散化的使用不仅仅局限于线段树中，还可以应用于其他的场景中，不过在这里只讨论线段树+离散化。

**那么什么是离散化呢？ 离散化即把无限空间中有限的个体映射到有限的空间中去，以此提高算法的时空效率。**

假设我们需要在 [1, 10^10] 范围中选取 1000 个点进行操作，那么肯定是没有办法 build(1, 1, 10^10) 的，故我们创建一种映射关系，若我们取了 5 2000 300 1e4 2e5 …

用数组 arr 来保存，arr[] = {5, 2000, 300, 1e4, 2e5 … };

**一般离散化都要三个步骤：1. sort 排序 2. unique 去重 3. lower-bound 二分查找**

对 arr 进行 sort 以及 unique 后， arr[] = {5, 300, 2000, 1e4, 2e5 … };

处理过后数组的长度为 len，映射关系为：

1——5， 2——300， 3——2000， 4——1e4， 5——2e5 …

这样我们只要 build(1, 1, len) ，创建大小为 4\*len 的线段树就可以了。

这样要对区间 [x, y] 进行操作的时候，使用lower-bound 在arr 中找到 x 和 y 对应的索引（离散化后），之后就是一般的线段树做法了。比如区间 [300, 2000] 对应离散化后的 [2, 3]。

**使用离散化不仅大大减少了算法的空间复杂度，并且因为减少了结点而提高了时间复杂度。**

下面给出了线段树离散化的简单应用题：

POJ-2528 Mayor’s posters

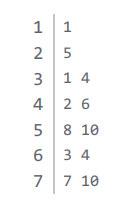
题目大意：在长度为 1e7 的数轴上用不超过 10000 条的不同线段进行覆盖，求最后数轴上可见的不同线段条数。

input：第一行为 case 数量 T，第二行为线段的条数 n (1 <= n <= 10000)，接下来的 n 行每行代表一条线段，其中每行都包含了两个数字，分别为线段

点。

output：对于每个 case 输出最后数轴上可见的不同线段条数。

Sample Input：



Sample Output：

image.png

**思路：** 明显的离散化应用，大数据范围小数据点，直接开 1e7 的线段树应该爆掉了。首先得到所有线段的左右端点，对所有端点进行排序去重，这样离散化后的端点数组。之后再遍历每条线段，使用 lower-bound 找到左右端点对应的离散索引，进行区间修改 change\_interval，最后调用一次区间查ask\_interval 即可。

注意结点color要初始化为-1，这样才不会跟0发生冲突。还要注意在ask\_interval的时候要对扫描过的线段使用vis数组进行记录，保证同一条线段不出现多次计数。

参考代码：

// 离散化

#include<cmath>

#include<cstring>

#include<cstdio>

#include<algorithm>

#include<iostream>

using namespace std;

const int maxn = 10000+5;

int n, ans, cnt;

struct Node {

int left, right;

int color;

int f;

} tree[8\*maxn]; // 最多会有2e4个叶子节点

int vis[8\*maxn];

int li[2\*maxn],ri[2\*maxn],points[2\*maxn];

void build(int k, int l, int r){

tree[k].left = l;

tree[k].right = r;

tree[k].color = -1;

tree[k].f = -1;

if(l == r){

return;

}

int mid = (l+r)/2;

build(2\*k, l, mid);

build(2\*k+1, mid+1, r);

}

void down(int k) {

if(tree[k].left == tree[k].right) return;

tree[2\*k].color = tree[k].color;

tree[2\*k+1].color = tree[k].color;

tree[k].f = -1;

}

void pushUp(int k){

if(tree[k].left == tree[k].right) return;

if(tree[2\*k].color == tree[2\*k+1].color){

tree[k].color = tree[2\*k].color;

}

else{

tree[k].color = -1;

}

}

void change\_interval(int k, int C, int l, int r) {

if(tree[k].left >= l && tree[k].right <= r) {

tree[k].color = tree[k].f = C;

return ;

}

if(tree[k].color != -1) down(k);

int mid = (tree[k].left+tree[k].right)/2;

if(mid >= l) change\_interval(2\*k, C, l, r);

if(mid < r) change\_interval(2\*k+1, C, l, r);

pushUp(k);

}

void ask\_interval(int k, int l, int r) {

if(!vis[tree[k].color] && tree[k].color != -1) {

ans++;

vis[tree[k].color] = 1;

return;

}

if(tree[k].left == tree[k].right) return;

if(tree[k].color!=-1) down(k);

ask\_interval(2\*k, l, r);

ask\_interval(2\*k+1, l,r);

}

int main() {

int T;

scanf("%d", &T);

while(T--) {

scanf("%d", &n);

memset(vis, 0, sizeof(vis));

int total = 0;

for(int i = 0; i < n; i++) {

scanf("%d%d", &li[i], &ri[i]);

points[total++] = li[i];

points[total++] = ri[i];

}

sort(points, points+total);

int newn = unique(points, points+total) - points;

cnt = 0;

build(1, 0, newn-1);

for(int i = 0; i < n; i++) {

int x = lower\_bound(points, points+newn, li[i]) - points;

int y = lower\_bound(points, points+newn, ri[i]) - points;

change\_interval(1, i, x, y);

}

ans = 0;

ask\_interval(1, 0, newn-1);

printf("%d\n", ans);

}

}

**Java version (convert by chatGPT)**

This Java code should function similarly to the provided C++ code. Note that I added a **Scanner** for input and modified the array utility functions accordingly.

import java.util.Arrays;

import java.util.Scanner;

public class Main {

static final int maxn = 10005;

static int n, ans, cnt;

static Node[] tree = new Node[8 \* maxn];

static int[] vis = new int[8 \* maxn];

static int[] li = new int[2 \* maxn];

static int[] ri = new int[2 \* maxn];

static int[] points = new int[2 \* maxn];

static class Node {

int left, right;

int color;

int f;

Node(int left, int right) {

this.left = left;

this.right = right;

this.color = -1;

this.f = -1;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int T = scanner.nextInt();

while (T-- > 0) {

n = scanner.nextInt();

Arrays.fill(vis, 0);

int total = 0;

for (int i = 0; i < n; i++) {

li[i] = scanner.nextInt();

ri[i] = scanner.nextInt();

points[total++] = li[i];

points[total++] = ri[i];

}

Arrays.sort(points, 0, total);

int newn = unique(points, 0, total) - points;

cnt = 0;

build(1, 0, newn - 1);

for (int i = 0; i < n; i++) {

int x = lower\_bound(points, 0, newn, li[i]);

int y = lower\_bound(points, 0, newn, ri[i]);

changeInterval(1, i, x, y);

}

ans = 0;

askInterval(1, 0, newn - 1);

System.out.println(ans);

}

}

static void build(int k, int l, int r) {

tree[k] = new Node(l, r);

if (l == r) {

return;

}

int mid = (l + r) / 2;

build(2 \* k, l, mid);

build(2 \* k + 1, mid + 1, r);

}

static void down(int k) {

if (tree[k].left == tree[k].right) return;

tree[2 \* k].color = tree[k].color;

tree[2 \* k + 1].color = tree[k].color;

tree[k].f = -1;

}

static void pushUp(int k) {

if (tree[k].left == tree[k].right) return;

if (tree[2 \* k].color == tree[2 \* k + 1].color) {

tree[k].color = tree[2 \* k].color;

} else {

tree[k].color = -1;

}

}

static void changeInterval(int k, int C, int l, int r) {

if (tree[k].left >= l && tree[k].right <= r) {

tree[k].color = tree[k].f = C;

return;

}

if (tree[k].color != -1) down(k);

int mid = (tree[k].left + tree[k].right) / 2;

if (mid >= l) changeInterval(2 \* k, C, l, r);

if (mid < r) changeInterval(2 \* k + 1, C, l, r);

pushUp(k);

}

static void askInterval(int k, int l, int r) {

if (vis[tree[k].color] == 0 && tree[k].color != -1) {

ans++;

vis[tree[k].color] = 1;

return;

}

if (tree[k].left == tree[k].right) return;

if (tree[k].color != -1) down(k);

askInterval(2 \* k, l, r);

askInterval(2 \* k + 1, l, r);

}

static int unique(int[] arr, int start, int end) {

int index = start;

for (int i = start + 1; i < end; i++) {

if (arr[i] != arr[index]) {

arr[++index] = arr[i];

}

}

return index + 1;

}

static int lower\_bound(int[] arr, int start, int end, int key) {

int lo = start, hi = end - 1;

while (lo <= hi) {

int mid = lo + (hi - lo) / 2;

if (arr[mid] < key) {

lo = mid + 1;

} else {

hi = mid - 1;

}

}

return lo;

}

}

**HDU-1199**

<https://cloud.tencent.com/developer/article/1380648>

**给球上色（线段树+离散化) - HDU-1199**

离散化是一中元素映射方法，它可以有效的降低时间复杂度。其基本思想就是在众多可能的情况中，只考虑需要用的值。离散化可以改进一个低效的算法，甚至实现根本不可能实现的算法。要掌握这个思想，必须从大量的题目中理解此方法的特点。例如，在建造线段树空间不够的情况下，可以考虑离散化。

使用STL算法离散化：

思路是：先排序，再删除重复元素，最后就是索引元素离散化后对应的值。

假定待离散化的序列为a[n]，b[n]是序列a[n]的一个副本，则对应以上三步为：

sort(sub\_a,sub\_a+n);

//size为离散化后元素个数

int size=unique(sub\_a,sub\_a+n)-sub\_a;

for(i=0;i<n;i++)

a[i]=lower\_bound(sub\_a,sub\_a+size,a[i])-sub\_a + 1;//k为b[i]经离散化后对应的值

对于第3步，若离散化后序列为0,1,2,...,size - 1则用lower\_bound，从1,2,3,...,size则用upper\_bound。其中lower\_bound返回第1个不小于b[i]的值的指针，而upper\_bound返回第1个大于b[i]的值的指针，当然在这个题中也可以用lower\_bound然后再加1得到与upper\_bound相同结果，两者都是针对以排好序列。使用STL离散化大大减少了代码量且结构相当清晰。

**本题涉及离散化和线段树的应用，线段树采用数组实现，新手看起来较为吃力，可以多多琢磨实现思路。**

Problem Description

There are infinite balls in a line (numbered 1 2 3 ....), and initially all of them are paint black. Now Jim use a brush paint the balls, every time give two integers a b and follow by a char 'w' or 'b', 'w' denotes the ball from a to b are painted white, 'b' denotes that be painted black. You are ask to find the longest white ball sequence.

有无限的球，排成一行，初始颜色为黑色。现在吉姆用一个刷子给球上色，整数a、b，w为白色，b为黑色。找出最长的白色序列。

Input

First line is an integer N (<=2000), the times Jim paint, next N line contain a b c, c can be 'w' and 'b'.

第一行是整数N，表示吉姆的绘制次数，后面N行包含2个数字和一个字母c，c表示w或者b。

There are multiple cases, process to the end of file.

多组测试用例

Output

Two integers the left end of the longest white ball sequence and the right end of longest white ball sequence (If more than one output the small number one). All the input are less than 2^31-1. If no such sequence exists, output "Oh, my god".

找出最长的白色序列。

数字小于2^31-1。

Sample Input

3

1 4 w

8 11 w

3 5 b

Sample Output

8 11

解题思路：

1、输入区间，离散化区间

2、建立线段树，然后通过线段树对每个节点着色

3、依次计算连续区间的颜色

源代码：g++

#include <algorithm>

#include <stdio.h>

#include <string.h>

#define MAKE\_ZERO(array) memset(array, 0, sizeof(array))

#define MAX\_N 4020

int a[MAX\_N], b[MAX\_N], colors[MAX\_N];

int tmp[2 \* MAX\_N], vis[MAX\_N];

int cnt;

struct node {

int l, r, col;

} segtree[MAX\_N << 2];

//创建线段树

//数组存储（性能点）

void build(int rt, int l, int r) {

if (l >= r)

return ;

//设置好区间值

segtree[rt].l = l;

segtree[rt].r = r;

//创建阶段为0，黑色

segtree[rt].col = 0;

if (l == r - 1)

return ;

int mid = (l + r) / 2;

//建立左子树，比如rt=1，左子树下标为2，右子树下标为3，依次类推

build(rt << 1, l, mid);

//创建右子树

build(rt << 1 | 1, mid, r);

}

//为线段树的每个节点设置颜色

void update(int rt, int l, int r, int color) {

if (l >= r) return ;

int L = segtree[rt].l, R = segtree[rt].r;

//说明这个位置是处于l和r之间的

if (L <= l &&R <= r)

{

segtree[rt].col = color;

return ;

}

//如果之前着色了，现在回产生分裂，不能通过这个父节点获取颜色了

if (segtree[rt].col >= 0 && segtree[rt].col != color) {

segtree[rt << 1].col = segtree[rt << 1 | 1].col = segtree[rt].col;

segtree[rt].col = -1;

}

//继续遍历子节点

int mid = (L + R) / 2;

if ( l >= mid )

update(rt << 1 | 1, l, r, color);

else if ( r <= mid )

update(rt << 1, l, r, color);

else

{

update(rt << 1, l, mid, color);

update(rt << 1 | 1, mid, r, color);

}

}

//将线段树转为数组存储关键位置和颜色

void query(int rt) {

int L = segtree[rt].l, R = segtree[rt].r;

//只有大于0才能够决定这个区域的颜色，否则一直遍历到叶子节点

if (segtree[rt].col >= 0) {

for (int i = L; i < R; i++)

vis[i] = segtree[rt].col;

return ;

}

if (L == R) return ;

//二维数组的树形结构

query(rt << 1);

query(rt << 1 | 1);

}

int main() {

int n = 0;

char color;

//freopen("test.txt", "r", stdin);

//n为区间数

while (~scanf("%d", &n)) {

cnt = 0;

MAKE\_ZERO(a);

MAKE\_ZERO(b);

MAKE\_ZERO(colors);

MAKE\_ZERO(tmp);

for (int i = 0; i < n; i++) {

//输入区间和颜色

scanf("%d %d %c", &a[i], &b[i], &color);

// 2

// 1 3 w

// 4 5 w 更新的是点，使右边加一，就变成连续的区间了。否则就更新不上去1~5这整块区间了

b[i]++;

//离散化位置

tmp[cnt++] = a[i];

tmp[cnt++] = b[i];

//white对应1，black对应0

if (color == 'w') {

colors[i] = 1;

}

}

//排序

std::sort(tmp, tmp + cnt);

//去掉重复

cnt = std::unique(tmp, tmp + cnt) - tmp;

//线段树

build(1, 0, cnt);

//更新每一个区间的颜色

for (int i = 0; i < n; i++) {

int p1, p2;

p1 = std::lower\_bound(tmp, tmp + cnt, a[i]) - tmp;

p2 = std::lower\_bound(tmp, tmp + cnt, b[i]) - tmp;

update(1, p1, p2, colors[i]);

}

MAKE\_ZERO(vis);

//设置区域颜色

query(1);

tmp[cnt] = tmp[cnt - 1];

int start = 0, end = 0, ts, te;

for (int i = 0; i < cnt; i++) {

//如果是黑色，不处理

if ( vis[i] != 1 )

continue;

ts = tmp[i];

while ( vis[i] == 1 )

i++;

if ( i > cnt )

break;

te = tmp[i];

//比较区域大小

if (te - ts > end - start)

{

end = te;

start = ts;

}

}

if (start == end)

printf("Oh, my god\n");

else

printf("%d %d\n", start, end - 1);

exit(0);

}

return 0;

}

**Java version (convert by chatGPT)**

import java.util.Arrays;

import java.util.Scanner;

public class Main {

private static final int MAX\_N = 4020;

private static int[] a = new int[MAX\_N];

private static int[] b = new int[MAX\_N];

private static int[] colors = new int[MAX\_N];

private static int[] tmp = new int[2 \* MAX\_N];

private static int[] vis = new int[MAX\_N];

private static int cnt;

private static class Node {

int l, r, col;

}

private static Node[] segtree = new Node[MAX\_N << 2];

private static void build(int rt, int l, int r) {

if (l >= r) return;

segtree[rt] = new Node();

segtree[rt].l = l;

segtree[rt].r = r;

segtree[rt].col = 0;

if (l == r - 1) return;

int mid = (l + r) / 2;

build(rt << 1, l, mid);

build(rt << 1 | 1, mid, r);

}

private static void update(int rt, int l, int r, int color) {

if (l >= r) return;

int L = segtree[rt].l, R = segtree[rt].r;

if (L <= l && R <= r) {

segtree[rt].col = color;

return;

}

if (segtree[rt].col >= 0 && segtree[rt].col != color) {

segtree[rt << 1].col = segtree[rt << 1 | 1].col = segtree[rt].col;

segtree[rt].col = -1;

}

int mid = (L + R) / 2;

if (l >= mid)

update(rt << 1 | 1, l, r, color);

else if (r <= mid)

update(rt << 1, l, r, color);

else {

update(rt << 1, l, mid, color);

update(rt << 1 | 1, mid, r, color);

}

}

private static void query(int rt) {

int L = segtree[rt].l, R = segtree[rt].r;

if (segtree[rt].col >= 0) {

for (int i = L; i < R; i++)

vis[i] = segtree[rt].col;

return;

}

if (L == R) return;

query(rt << 1);

query(rt << 1 | 1);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (scanner.hasNext()) {

int n = scanner.nextInt();

cnt = 0;

Arrays.fill(a, 0);

Arrays.fill(b, 0);

Arrays.fill(colors, 0);

Arrays.fill(tmp, 0);

Arrays.fill(vis, 0);

for (int i = 0; i < n; i++) {

a[i] = scanner.nextInt();

b[i] = scanner.nextInt();

char color = scanner.next().charAt(0);

b[i]++;

tmp[cnt++] = a[i];

tmp[cnt++] = b[i];

if (color == 'w') {

colors[i] = 1;

}

}

Arrays.sort(tmp, 0, cnt);

cnt = unique(tmp, 0, cnt);

build(1, 0, cnt);

for (int i = 0; i < n; i++) {

int p1 = lowerBound(tmp, 0, cnt, a[i]);

int p2 = lowerBound(tmp, 0, cnt, b[i]);

update(1, p1, p2, colors[i]);

}

Arrays.fill(vis, 0);

query(1);

tmp[cnt] = tmp[cnt - 1];

int start = 0, end = 0, ts, te;

for (int i = 0; i < cnt; i++) {

if (vis[i] != 1)

continue;

ts = tmp[i];

while (vis[i] == 1)

i++;

if (i > cnt)

break;

te = tmp[i];

if (te - ts > end - start) {

end = te;

start = ts;

}

}

if (start == end)

System.out.println("Oh, my god");

else

System.out.println(start + " " + (end - 1));

}

}

private static int unique(int[] arr, int start, int end) {

int index = start;

for (int i = start + 1; i < end; i++) {

if (arr[i] != arr[index]) {

arr[++index] = arr[i];

}

}

return index + 1;

}

private static int lowerBound(int[] arr, int start, int end, int key) {

int lo = start, hi = end - 1;

while (lo <= hi) {

int mid = lo + (hi - lo) / 2;

if (arr[mid] < key) {

lo = mid + 1;

} else {

hi = mid - 1;

}

}

return lo;

}

}