第五次实验报告

Q1(POJ2182)

Description

Description

N (2 <= N <= 8,000) cows have unique brands in the range 1..N. In a spectacular display of poor judgment, they visited the neighborhood 'watering hole' and drank a few too many beers before dinner. When it was time to line up for their evening meal, they did not line up in the required ascending numerical order of their brands.

Regrettably, FJ does not have a way to sort them. Furthermore, he's not very good at observing problems. Instead of writing down each cow's brand, he determined a rather silly statistic: For each cow in line, he knows the number of cows that precede that cow in line that do, in fact, have smaller brands than that cow.

Given this data, tell FJ the exact ordering of the cows.

Input

* Line 1: A single integer, N

* Lines 2..N: These N-1 lines describe the number of cows that precede a given cow in line and have brands smaller than that cow. Of course, no cows precede the first cow in line, so she is not listed. Line 2 of the input describes the number of preceding cows whose brands are smaller than the cow in slot #2; line 3 describes the number of preceding cows whose brands are smaller than the cow in slot #3; and so on.

Output

* Lines 1..N: Each of the N lines of output tells the brand of a cow in line. Line #1 of the output tells the brand of the first cow in line; line 2 tells the brand of the second cow; and so on.

Sample Input

```
5
1
2
1
0
```

Sample Output

```
2
4
5
3
1
```

Solution

从后往前倒着做，我们发现可以唯一确定每只牛的号牌，如样例：

input：5 1 2 1 0

我们用**b**数组表示当前牛的编号

则现在有五个编号，置一数组表示

a：1 1 1 1 1（1表示还未选过此号牌）

读入0，则选取第一个号牌，则

b：x x x x 1

同时，1号牌被选中，a1置零

a：0 1 1 1 1

读入1，则选取还未被选取的第二个号牌

b：x x x 3 1

a：0 1 0 1 1

依此类推

最终：

b：2 5 4 3 1

a：0 0 0 0 0

我们选用树状数组去维护**a**数组，同时用二分查找查找当前未被选取的第**i**个号牌

Code

main.cpp

```
#include <iostream>
using namespace std;
const int MAXN = 1e5 + 10;
int c[MAXN], a[MAXN], b[MAXN], N;

void add(int x, int val)
{
    while (x <= N)
    {
        c[x] += val;
        x += x & (-x);
    }
}

int getsum(int x)
{
    int sum = 0;
```

```
    while (x > 0)
    {
        sum += c[x];
        x -= x & (-x);
    }
    return sum;
}

int find(int x)
{
    int l = 1, r = N;
    int p = 0;
    int mid;
    while (l <= r)
    {
        mid = (l + r) / 2;
        if (getsum(mid) >= x)
        {
            p = mid;
            r = mid - 1;
        }
        else
            l = mid + 1;
    }
    return p;
}

int main()
{
    cin >> N;
    int k;
    a[0] = N;
    for (int i = 1; i < N; i++)
    {
        cin >> k;
        a[i] = k;
    }
    for (int i = 1; i <= N; i++)
    {
        add(i, 1);
    }

    int pos;
    for (int i = N - 1; i > 0; i--)
    {
        pos = find(a[i] + 1);
        b[i] = pos;
        add(pos, -1);
    }
    pos = find(1);
    b[0] = pos;

    for (int i = 0; i < N; i++)
        cout << b[i] << endl;
}
```

Q2(POJ2828)

Description

Description

Railway tickets were difficult to buy around the Lunar New Year in China, so we must get up early and join a long queue…

The Lunar New Year was approaching, but unluckily the Little Cat still had schedules going here and there. Now, he had to travel by train to Mianyang, Sichuan Province for the winter camp selection of the national team of Olympiad in Informatics.

It was one o'clock a.m. and dark outside. Chill wind from the northwest did not scare off the people in the queue. The cold night gave the Little Cat a shiver. Why not find a problem to think about? That was none the less better than freezing to death!

People kept jumping the queue. Since it was too dark around, such moves would not be discovered even by the people adjacent to the queue-jumpers. "If every person in the queue is assigned an integral value and all the information about those who have jumped the queue and where they stand after queue-jumping is given, can I find out the final order of people in the queue?" Thought the Little Cat.

Input

There will be several test cases in the input. Each test case consists of N + 1 lines where N ($1 \le N \le 200{,}000$) is given in the first line of the test case. The next N lines contain the pairs of values *Posi* and *Vali* in the increasing order of i ($1 \le i \le N$). For each i, the ranges and meanings of *Posi* and *Vali* are as follows:

- *Posi* $\in [0, i-1]$ — The i-th person came to the queue and stood right behind the *Posi*-th person in the queue. The booking office was considered the 0th person and the person at the front of the queue was considered the first person in the queue.
- *Vali* $\in [0, 32767]$ — The i-th person was assigned the value *Vali*.

There no blank lines between test cases. Proceed to the end of input.

Output

For each test cases, output a single line of space-separated integers which are the values of people in the order they stand in the queue.

Sample Input

```
4
0 77
1 51
1 33
2 69
4
0 20523
1 19243
1 3890
0 31492
```

Sample Output

```
77 33 69 51
31492 20523 3890 19243
```

Solution

解题思路同第一题，倒着做，用树状数组维护，并用二分查询

Code

main.cpp

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int MAXN = 2e5 + 10;
int c[MAXN], ran[MAXN], N;

typedef struct person{
    int jq;
    int key;
} PER;

PER ps[MAXN];

void add(int x, int val)
{
    while (x <= N)
    {
        c[x] += val;
        x += x & (-x);
    }
}

int getsum(int x)
{
    int s = 0;
    while (x > 0)
    {
        s += c[x];
        x -= x & (-x);
    }
    return s;
}

int find(int x)
{
    int l = 1, r = N;
    int p = 0, mid;
    while (l <= r)
    {
        mid = (l + r) / 2;
        if (mid - getsum(mid) >= x)
        {
            p = mid;
```

```
            r = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }
    return p;
}

int main()
{
    while (scanf("%d", &N) != EOF)
    {
        memset(c, 0, sizeof(c));

        for (int i = 0; i < N; i++)
            scanf("%d%d", ps[i].jq, ps[i].key);

        int pos;
        for (int i = N - 1; i >= 0; i--)
        {
            pos = find(ps[i].jq + 1);
            ran[pos - 1] = ps[i].key;
            add(pos, 1);
        }

        for (int i = 0; i < N; i++)
            cout << ran[i] << " ";
        cout << endl;
    }
}
```

Q3(POJ2528)

Description

Description

The citizens of Bytetown, AB, could not stand that the candidates in the mayoral election campaign have been placing their electoral posters at all places at their whim. The city council has finally decided to build an electoral wall for placing the posters and introduce the following rules:

- Every candidate can place exactly one poster on the wall.
- All posters are of the same height equal to the height of the wall; the width of a poster can be any integer number of bytes (byte is the unit of length in Bytetown).
- The wall is divided into segments and the width of each segment is one byte.
- Each poster must completely cover a contiguous number of wall segments.

They have built a wall 10000000 bytes long (such that there is enough place for all candidates). When the electoral campaign was restarted, the candidates were placing their posters on the wall and their posters differed widely in width. Moreover, the candidates started placing their posters on wall segments already occupied by other posters. Everyone in Bytetown was curious whose posters will be visible (entirely or in part) on the last day before elections.
Your task is to find the number of visible posters when all the posters are placed given the information about posters' size, their place and order of placement on the electoral wall.

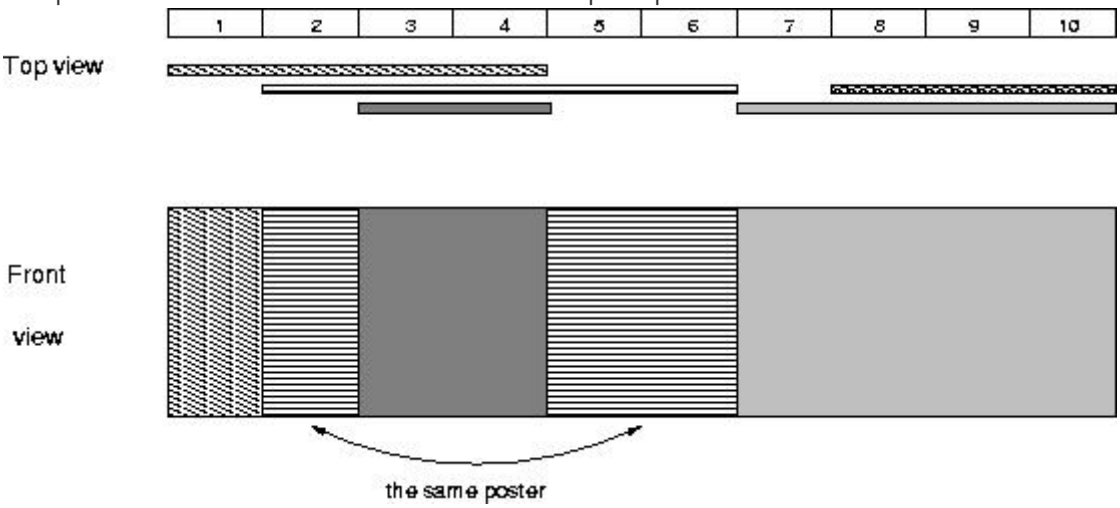
Input

The first line of input contains a number c giving the number of cases that follow. The first line of data for a single case contains number 1 <= n <= 10000. The subsequent n lines describe the posters in the order in which they were placed. The i-th line among the n lines contains two integer numbers li and ri which are the number of the wall segment occupied by the left end and the right end of the i-th poster, respectively. We know that for each 1 <= i <= n, 1 <= li <= ri <= 10000000. After the i-th poster is placed, it entirely covers all wall segments numbered li, li+1 ,... , ri.

Output

For each input data set print the number of visible posters after all the posters are placed.

The picture below illustrates the case of the sample input.



Sample Input

```
1
5
1 4
2 6
8 10
3 4
7 10
```

Sample Output

```
4
```

Solution

data struct

用线段树，

每个结点增加color, lazy两个值；

color =

-1：当前区间有多张海报

0：当前区间没有海报

n：当前区间为第n张海报（n>0)

离散化

对于海报长度的处理，由于海报只有10000张，故可以考虑对海报长度进行离散化处理，

但简单的离散化处理会出错：

例如：

1 10

1 4

6 10

结果为3

但简单离散化后：

1 4

1 2

3 4

结果为2

故而离散化时，对于不相邻的数字(即两个数字差值大于1时，需要在两个数字中间再插入一个数字进行离散化)

例如：

1 10

1 4

6 10

此离散化后(a[1] = 1, a[2] = 3, a[3] = 4, a[4] = 5, a[5] = 6, a[6] = 9, a[7] = 10)(a的下标值即为离散化后的数值，a的值即为离散化前的数值)

1 7

1 3

5 7

details

建立线段树s

用segms保存原长度区间

用x数组保存离散化后的结果

利用二分快速在x数组中查找segms对应的离散化区间

然后利用线段树快速修改区间color

最后利用数组a记录线段树中出现的color的值的个数，即有几张海报可见

例如：

若某区间color = 2， 则a[2]=1，即第二张海报可见

最后对a数组求和即为最终结果

Code

main.cpp

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAXN = 1e4 + 10;

struct segTree{
    int l, r;
    int color;
    int lazy;
}s[MAXN << 4];

void push_up(int rt)
{
    int cl = s[rt << 1].color;
    int cr = s[rt << 1 | 1].color;

    if (cl == -1 || cr == -1)
        s[rt].color = -1;
    else if (cl != cr)
        s[rt].color = -1;
    else
        s[rt].color = cl;
}

void build(int l, int r, int rt)
{
    s[rt].l = l;
    s[rt].r = r;
    s[rt].color = s[rt].lazy = 0;

    if (l == r)
        return;

    int m = (l + r) >> 1;
    build(l, m, rt << 1);
    build(m + 1, r, rt << 1 | 1);
}

void push_down(int rt)
{
    if (s[rt].lazy != 0)
    {
        s[rt << 1].lazy = s[rt << 1 | 1].lazy = s[rt].lazy;
        s[rt << 1].color = s[rt << 1 | 1].color = s[rt].lazy;
        s[rt].lazy = 0;
    }
}

void update(int L, int R, int c, int rt)
{
    if (s[rt].l >= L && s[rt].r <= R)
```

```
    {
        s[rt].color = c;
        s[rt].lazy = c;
        return;
    }
    if (s[rt].l > R || s[rt].r < L)
        return;

    push_down(rt);
    update(L, R, c, rt << 1);
    update(L, R, c, rt << 1 | 1);

    push_up(rt);
}


void query(int L, int R, int rt, int *a)
{
    if (s[rt].color == 0)
        return;
    else if (s[rt].color > 0)
    {
        a[s[rt].color] = 1;
        return;
    }
    push_down(rt);
    query(L, R, rt << 1, a);
    query(L, R, rt << 1 | 1, a);
}


struct segmax{
    int left, right;
} segms[MAXN << 1];

int x[MAXN << 2];

int bsearch(int l, int r, int key)
{
    int ll = l, rr = r;
    int mid = 0;
    while (ll <= rr)
    {
        mid = (ll + rr) >> 1;
        if (x[mid] == key)
            return mid;
        else if (x[mid] > key)
        {
            rr = mid - 1;
        }
        else
            ll = mid + 1;
    }
    return mid;
}


int main()
```

```
{
    int T, N;
    cin >> T;
    while (T--)
    {
        cin >> N;
        for (int i = 0; i < N; i++)
        {
            cin >> segms[i].left >> segms[i].right;
            x[i * 2 + 1] = segms[i].left;
            x[i * 2 + 2] = segms[i].right;
        }

        sort(x + 1, x + 2 * N);

        int m = 0;
        x[m++] = x[1];
        for (int i = 2; i <= 2 * N; i++)
        {
            if (x[i] != x[i-1])
                x[m++] = x[i];
        }

        // for (int i = 0; i < m; i++)
        //     cout << x[i] << " ";
        // cout << endl;

        for (int i = m-1; i >= 1; i--)
        {
            if (x[i] - x[i-1] > 1)
                x[m++] = x[i] - 1;
        }

        sort(x, x + m);

        // for (int i = 0; i < m; i++)
        //     cout << x[i] << " ";
        // cout << endl;

        build(0, m-1, 1);

        int *a = new int[N];
        for (int i = 0; i <= N; i++)
            a[i] = 0;

        for (int i = 0; i < N; i++)
        {
            int l = bsearch(0, m - 1, segms[i].left);
            int r = bsearch(0, m - 1, segms[i].right);

            //cout << l << " " << r << endl;
            update(l, r, i + 1, 1);
        }

        query(0, m - 1, 1, a);

        int ans = 0;
        for (int i = 1; i <= N; i++)
```

```
        {
            ans += a[i];
        }

        cout << ans << endl;

        delete[] a;
    }
}
```

Q4(POJ2155)

Description

Description

Given an N*N matrix A, whose elements are either 0 or 1. A[i, j] means the number in the i-th row and j-th column. Initially we have A[i, j] = 0 (1 <= i, j <= N).

We can change the matrix in the following way. Given a rectangle whose upper-left corner is (x1, y1) and lower-right corner is (x2, y2), we change all the elements in the rectangle by using "not" operation (if it is a '0' then change it into '1' otherwise change it into '0'). To maintain the information of the matrix, you are asked to write a program to receive and execute two kinds of instructions.

\1. C x1 y1 x2 y2 (1 <= x1 <= x2 <= n, 1 <= y1 <= y2 <= n) changes the matrix by using the rectangle whose upper-left corner is (x1, y1) and lower-right corner is (x2, y2).
\2. Q x y (1 <= x, y <= n) querys A[x, y].

Input

The first line of the input is an integer X (X <= 10) representing the number of test cases. The following X blocks each represents a test case.

The first line of each block contains two numbers N and T (2 <= N <= 1000, 1 <= T <= 50000) representing the size of the matrix and the number of the instructions. The following T lines each represents an instruction having the format "Q x y" or "C x1 y1 x2 y2", which has been described above.

Output

For each querying output one line, which has an integer representing A[x, y].

There is a blank line between every two continuous test cases.

Sample Input

```
1
2 10
C 2 1 2 2
Q 2 2
C 2 1 2 1
Q 1 1
C 1 1 2 1
C 1 2 1 2
C 1 1 2 2
Q 1 1
C 1 1 2 1
Q 2 1
```

Sample Output

```
1
0
0
1
```

Solution

利用二维树状数组，

用A表示原元素数组

利用奇偶性表示当前点为0还是1

差分

同时利用差分的思想，将区间修改转化为点修改，

则原区间 x1, y1 x2, y2 修改转化为

A[x1] [y1] += 1

A[x2+1] [y2+1] += 1

A[x1] [y2+1] += 1

A[x2+1] [y1] += 1

同时点查询转换为求和

求和即为当前元素值

S[x] [y] = Σ Σ A[i] [j]

Code

main.cpp

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int MAXN = 1e3+10;
int c[MAXN][MAXN];
```

```
int N;

void update(int x, int y, int val)
{
    for (int i = x; i <= N; i += i & (-i))
        for (int j = y; j <= N; j += j & (-j))
            c[i][j] += val;
}

int getsum(int x, int y)
{
    int s = 0;
    for (int i = x; i > 0; i -= i & (-i))
        for (int j = y; j > 0; j -= j & (-j))
            s += c[i][j];
    return s;
}

int main()
{
    int T;
    scanf("%d", &T);
    int opt;
    char op[10];
    int x1, y1, x2, y2,ans;
    while (T--)
    {
        scanf("%d %d", &N, &opt);
        memset(c, 0, sizeof(c));

        for (int i = 0; i < opt; i++)
        {
            scanf("%s", op);
            if (op[0] == 'C')
            {
                scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
                update(x1, y1, 1);
                update(x1, y2 + 1, 1);
                update(x2 + 1, y1, 1);
                update(x2 + 1, y2 + 1, 1);
            }
            else if (op[0] == 'Q')
            {
                scanf("%d %d", &x1, &y1);
                ans = getsum(x1, y1);
                printf("%d\n", ans % 2);
            }
            // for (int j = 1; j <= N; j++)
            // {
            //     for (int k = 1; k <= N; k++)
            //     {
            //         ans = getsum(j, k);
            //         cout << (ans & 1) << " ";
            //     }
            //     cout << endl;
            // }
        }
        if (T)
```

```
            printf("\n");
    }
}
```