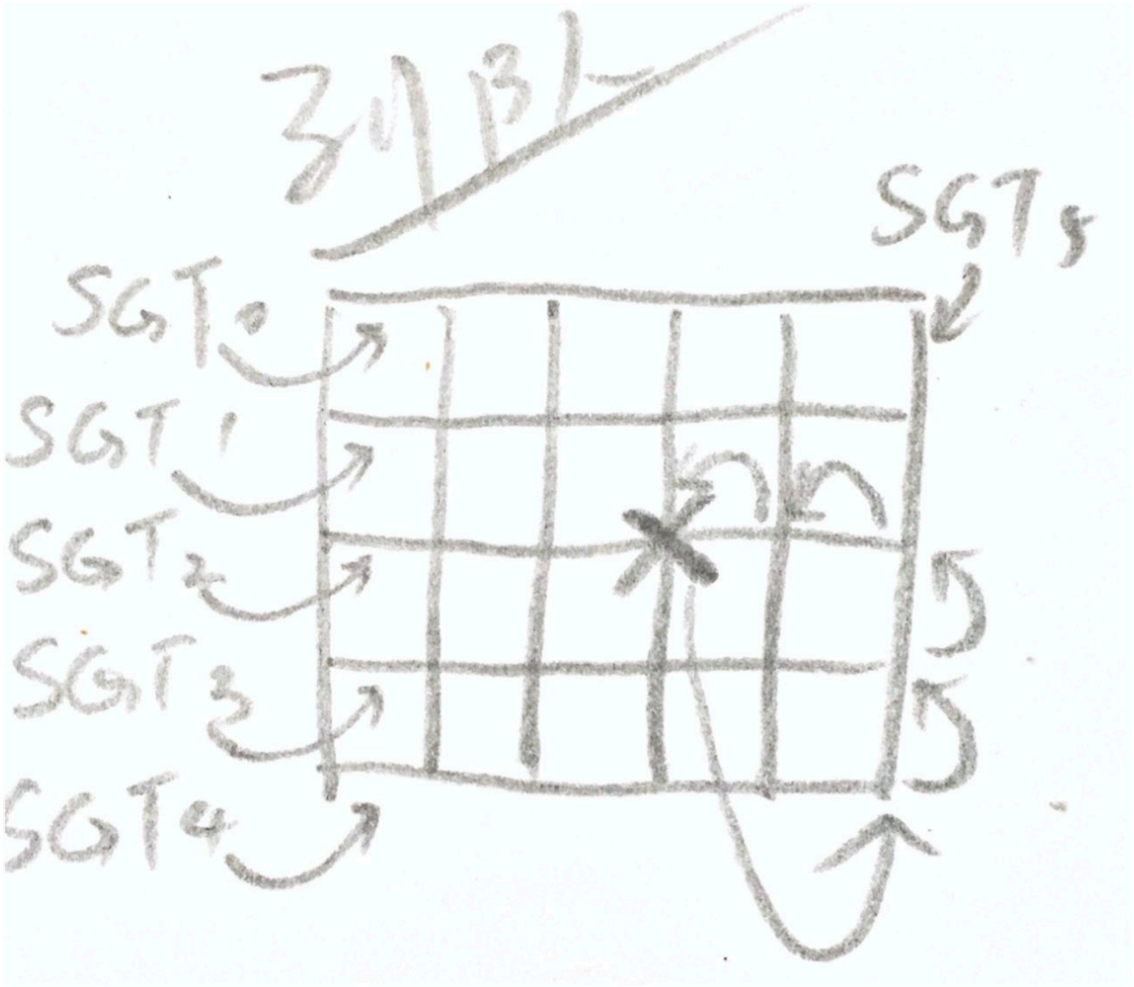
# 列队

## 解题思路




先模拟一下出队程序。上图打❌的地方当作出队人员，右边的人填过去，之后最后一列下面的人填上去。

如果用数组结构，一个个移动这样模拟的话，自然会TLE掉。

这里要借助**值域线段树**，用k-th最大的思想解决这个问题。 k-th最大的持久化线段树做法点这里。

我们对每一行的人员进行建线段树。假设有4个人员，一开始是 0 0 0 0，如果要找第2个数，那么就通过一下方程式递归查找。 左边是 m+1-l, 右边是 r-m个数。

```
int query(int root, int l, int r, int k) {
    if (l == r) {
        return l; // return the real position
    }
    int m = (l+r) >> 1;
    int num = m-l+1-sum[lson[root]]; // adjust element's position
    if (k > num) return query(rson[root], m+1, r, k-num);
    else return query(lson[root], l, m, k);
}
```

如果有一名人员出队，我们在树的那个位置进行+1，之后查找的时候减掉这个多余的（区间大小 - 区间出队次数），这样查询k-th最大的时候就会后移一位了。记录新增的人员编号；如果查询结果大于原队列，在新增人员编号中索引即可

***所以***我们可以用n+1个线段树对出队问题进行维护。 线段树空间复杂度为$O(N*(N<<2)$显然会MLE

***考虑到***问题只有$Q$个操作，每次操作修改$O(logn)$的节点，动态分配就好了。

## 时间复杂度分析

$O(Q*logN)$

## 代码

```
/*
    https://www.acwing.com/problem/content/532/
    多个线段树进行维护，值域线段树，动态分配节点
    Reference:
    https://blog.csdn.net/qq_38678604/article/details/78575672
    https://www.lagou.com/lgeduarticle/5024.html
*/
#include <iostream>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <vector>
#include <queue>
#include <set>
#include <map>
#include <unordered_map>
#include <stack>
#include <deque>

#define ll long long
#define ull unsigned long long

using namespace std;
```

```cpp
const int MOD = 1e9+7;
const int INF = 0x3f3f3f3f;
const int PI = acos(-1.0);

const int N = 3e5+10;

int n, m, q, mx, total = 0;

int sum[N<<5], lson[N<<5], rson[N<<5], rt[N];
vector<ll> G[N]; // N max time of query

int query(int root, int l, int r, int k) {
    if (l == r) {
        return l; // return the real position
    }
    int m = (l+r) >> 1;
    int num = m-l+1-sum[lson[root]]; // adjust element's position
    if (k > num) return query(rson[root], m+1, r, k-num);
    else return query(lson[root], l, m, k);
}

void add(int& root, int l, int r, int pos) {
    if (root == 0) root = ++total; // dynamically assigning new space!
    sum[root]++; // add from top to bottom of this tree if pos is in the
interval
    if (l < r) {
        int m = (l+r) >> 1;
        if (pos <= m) add(lson[root], l, m, pos);
        else add(rson[root], m+1, r, pos);
    }
}

ll verticalSeg(int root, int l, int r, int k, ll v) { // segment tree
formed bythe last column
    int pos = query(rt[root], l, r, k); add(rt[root], 1, mx, pos);
    // cout << "pos" << pos << endl;
    ll ans = pos > n ? G[root][pos-n-1]: 1ll * m * pos;
    G[root].emplace_back(v ? v:ans); // v==0 or v!=0
    return ans;
}

ll horizonalSeg(int root, int l, int r, int k) { // n number of segment
tree from n rows all without the rightmost element
    int pos = query(rt[root], l, r, k); add(rt[root], 1, mx, pos);
    ll ans = pos >= m ? G[root][pos-m]: 1ll*(root-1)*m + pos;
    G[root].emplace_back(verticalSeg(n+1, 1, mx, root, ans));
    return ans;
}
```

```
int main() {
    scanf("%d %d %d", &n, &m, &q);
    mx = max(n, m) + q; // maximal size of segment tree
    memset(sum, 0, sizeof(sum));
    memset(rt, 0, sizeof(rt));
    memset(lson, 0, sizeof(lson));
    memset(rson, 0, sizeof(rson));
    for (int i = 0; i < q; ++i) {
        int x, y;
        scanf("%d %d", &x, &y);
        if (y == m) {
            printf("%lld\n", verticalSeg(n+1, 1, mx, x, 0));
        } else {
            printf("%lld\n", horizonalSeg(x, 1, mx, y));
        }
    }
    return 0;
}
```

## 参考


https://blog.csdn.net/qq_38678604/article/details/78575672

https://www.lagou.com/lgeduarticle/5024.html