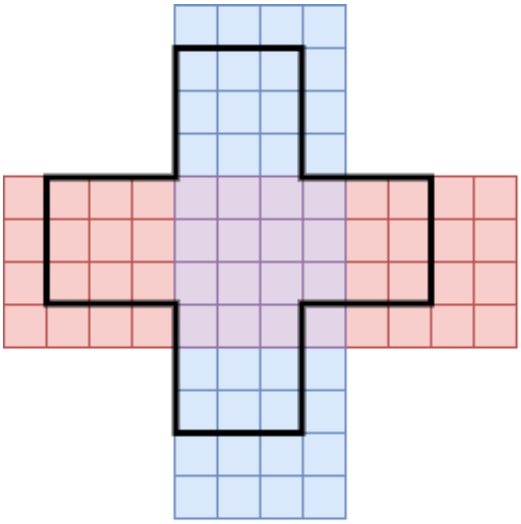
Crossed Block Again

> 출제자 : 서태수
> 정해 : 이분탐색, 슬라이딩 윈도우, 세그먼트 트리

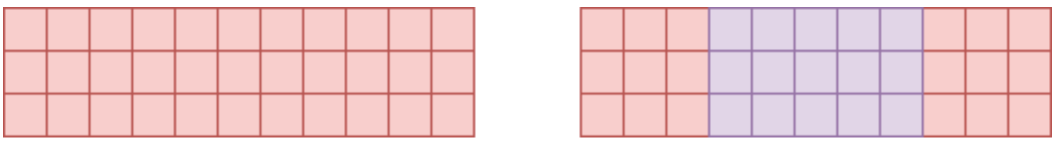
풀이



위의 그림에서 알 수 있듯이 K-십자 블럭이 존재한다면 그 안에 $(K-1)$-십자 블럭도 존재합니다. 따라서 최대 K를 이분 탐색으로 찾을 수 있습니다. 이제 어떤 K에 대해서 K-십자 블럭이 존재하는 지 확인하는 방식으로 문제의 정답을 찾아내면 됩니다.

문제를 잘 파악해보면 사실 십자 형태의 블럭을 찾는 것은 별 의미가 없고 가운데 $K \times K$ 보라색 정사각형 블럭의 존재 유무만 확인하면 됩니다. (물론 상하좌우 $K \times K$ 정사각형 블럭이 존재해야하지만, 이는 후술할 방식으로 처리한다면 보라색 블럭만 남길 수 있습니다.)

K가 3이라고 가정해봅시다. 그렇다면 아래 그림에서 왼쪽 빨간색 직사각형 블럭에서 가운데 보라색 블럭이 될 수 있는 후보는 오른쪽 그림처럼 될 것입니다.



이 보라색 직사각형의 범위를 잘 설정해봅시다.

각 블럭이 차지하고 있는 구역의 범위를 $[l_i, r_i]$라고 하면, 보라색 블럭이 될 수 있는 범위는 $[max(l_i)+K, min(r_i)-K]$가 될 것입니다. 해당 행에 페인트 칠을 안했다면 $[\infty, -\infty]$로 설정하면 됩니다. 이는 deque를 이용한 슬라이딩 윈도우 기법을 통해서 구할 수 있습니다. 나중에 좀 더 계산하기 편하게 보라색 블럭이 시작할 수 있는 곳으로 구간을 잡아주면 $[max(l_i)+K, min(r_i)-2K+1]$ 꼴로 정해줄 수 있습니다. 행 $[i, i+K-1]$ 블럭에 대해서 위에서 구한 구간 $[a, b]$를 vector 같은 곳에 아래와 같이 저장해줍니다.

> a열에서 시작할 수 있는 행은 i행이고, b열까지 가능하다.

이제 열에 대해서 확인해보면서 앞에서 저장했던 정보들과 비교해가면서 답을 구할 수 있습니다.

열 $[i, i+K-1]$ 블럭에 대해서도 위에서와 같은 방식으로 구간 $[a, b]$를 구할 수 있습니다. 이제 1열~i열에서 시작할 수 있는 블럭 중에서 시작 행이 $[a, b]$에 속하고, 최대 가능한 열이 i열 이상인 것이 존재하면 됩니다. 이는 세그트리를 통해서 업데이트/최대값 쿼리를 처리할 수 있습니다.

총 시간복잡도는 $O(Xlog^2X), X = 2 \times 10^5$ 입니다.

```
#include <bits/stdc++.h>
#define mp make_pair
#define X first
#define Y second
using namespace std;
typedef pair<int, int> Pi;

Pi r[200001],c[200001];
int tree[524288];

void upd(int node, int S, int E, int k, int dif)
{
  if (S == E){
    tree[node]=dif;
    return;
  }
  if (k <= (S + E) / 2)upd(node * 2, S, (S + E) / 2, k, dif);
  else upd(node * 2 + 1, (S + E) / 2 + 1, E, k, dif);
  tree[node]=max(tree[node*2],tree[node*2+1]);
}

int find(int node, int S, int E, int i, int j)
{
  if (i > E || j < S)return 0;
  if (i <= S && j >= E)return tree[node];
  return max(find(node * 2, S, (S + E) / 2, i, j), find(node * 2 + 1, (S + E) /
2 + 1, E, i, j));
}
vector<Pi> v[200001];
bool ok(int k){
  memset((tree),0,sizeof(tree));
  for(int i=1;i<=200000;i++)v[i].clear();
  deque<Pi> L,R;
  for(int i=1;i<k;i++){
    while(!L.empty() && L.back().X<=c[i].X)L.pop_back();
    L.push_back(mp(c[i].X,i));
    while(!R.empty() && R.back().X>=c[i].Y)R.pop_back();
    R.push_back(mp(c[i].Y,i));
  }
  for(int i=k;i<=200000;i++){
```

```
    while(!L.empty() && L.back().X<=c[i].X)L.pop_back();
    while(!L.empty() && L.front().Y<=i-k)L.pop_front();
    L.push_back(mp(c[i].X,i));
    while(!R.empty() && R.back().X>=c[i].Y)R.pop_back();
    while(!R.empty() && R.front().Y<=i-k)R.pop_front();
    R.push_back(mp(c[i].Y,i));
    int a=L.front().X,b=R.front().X;
    a+=k,b+=1-2*k;
    if(a<=b)v[a].push_back(mp(i-k+1,b));
  }
  L.clear(),R.clear();
  for(int i=1;i<k;i++){
    while(!L.empty() && L.back().X<=r[i].X)L.pop_back();
    L.push_back(mp(r[i].X,i));
    while(!R.empty() && R.back().X>=r[i].Y)R.pop_back();
    R.push_back(mp(r[i].Y,i));
  }
  for(int i=k;i<=200000;i++){
    for(Pi p:v[i-k+1])upd(1,1,200000,p.X,p.Y);
    while(!L.empty() && L.back().X<=r[i].X)L.pop_back();
    while(!L.empty() && L.front().Y<=i-k)L.pop_front();
    L.push_back(mp(r[i].X,i));
    while(!R.empty() && R.back().X>=r[i].Y)R.pop_back();
    while(!R.empty() && R.front().Y<=i-k)R.pop_front();
    R.push_back(mp(r[i].Y,i));
    int a=L.front().X,b=R.front().X;
    a+=k,b+=1-2*k;
    if(a<=b && find(1,1,200000,a,b)>=i-k+1)return 1;
  }
  return 0;
}
int main() {
  int n;
  scanf("%d",&n);
  for(int i=1;i<=200000;i++)r[i]=c[i]=mp(200001,0);
  for(int i=0;i<n;i++){
    int d,k,x,y;
    scanf("%d%d%d%d",&d,&k,&x,&y);
    if(d==0){
      r[k]=mp(x,y);
    }else{
      c[k]=mp(x,y);
    }
  }
  int l_=1,r_=66666;
  while(l_<=r_){
    int k=l_+r_>>1;
    if(ok(k))l_=k+1;
    else r_=k-1;
```

```
    }
    printf("%d\n",r_);
}
```