```
#include <iostream>
#include <vector>
#include <cmath>

#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/push_relabel_max_flow.hpp>
#include <boost/tuple/tuple.hpp>

using namespace std;
using namespace boost;

typedef adjacency_list_traits<vecS, vecS, directedS> traits_t;
typedef adjacency_list<vecS, vecS, directedS, no_property,
        property<edge_capacity_t, long,
            property<edge_residual_capacity_t, long,
                property<edge_reverse_t, traits_t::edge_descriptor> > > > gra
typedef property_map<graph_t, edge_capacity_t>::type edge_capacity_map_t;
typedef property_map<graph_t, edge_residual_capacity_t>::type residual_capacit
typedef property_map<graph_t, edge_reverse_t>::type reverse_edge_map_t;
typedef graph_traits<graph_t>::vertex_descriptor vertex_t;
typedef graph_traits<graph_t>::edge_descriptor edge_t;

void mf_add_edge(int u, int v, long c, edge_capacity_map_t &capacity,
        reverse_edge_map_t &rev_edge, graph_t &g) {
    edge_t e, reverse_edge;
    tie(e, tuples::ignore) = add_edge(u, v, g);
    tie(reverse_edge, tuples::ignore) = add_edge(v, u, g);
    capacity[e] = c;
    capacity[reverse_edge] = 0;
    rev_edge[e] = reverse_edge;
    rev_edge[reverse_edge] = e;
}

int main() {
    ios_base::sync_with_stdio(false);
    int t; cin >> t;

    for(int i=0; i<t; ++i) {
        int m, n, k; cin >> m >> n >> k;

        vector<pair<int, int> > knights;
        for(int j=0; j<k; ++j) {
            int x, y; cin >> x >> y; // column and row
            knights.push_back(make_pair(x, y));
        }
```

```
graph_t g(2 * n * m);
edge_capacity_map_t capacity = get(edge_capacity, g);
reverse_edge_map_t rev_edge = get(edge_reverse, g);
int source = add_vertex(g);
int sink = add_vertex(g);

// connect source to knight location intersections
for(int j=0; j<k; ++j)
    mf_add_edge(source, (m * knights[j].second) + knights[j].first, 1

// connect the intersections with their primes
// as only one knight can pass through an intersection
for(int j=0; j<(m * n); ++j)
    mf_add_edge(j, (m * n) + j, 1, capacity, rev_edge, g);


// connect corridors
for(int j=0; j<(m * n); ++j) {
    int column = j % m;
    int row = floor(j / m);

    if((column == 0 && row != 0) && (column == 0 && row != n - 1)) {
        mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
        mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
        mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
    } else if(column == 0 && row == 0) { // top left corner
        mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
        mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
    } else if(column == 0 && row == n - 1) { // bottom left corner
        mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
        mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
    } else if((column == m - 1 && row != 0) && (column == m && row !=
        mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
        mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
        mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
    } else if(column == m - 1 && row == 0) { // top right corner
        mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
        mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
    } else if(column == m - 1 && row == n - 1) { // bottom right corn
        mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
        mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
    } else if((row == 0 && column != 0) && (row == 0 && column != m -
        mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
        mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
        mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
    } else if((row == n - 1 && column != 0) && (row == n - 1 && colum
```

```
                mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
                mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
                mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
            } else { // inside connect everything
                mf_add_edge((m * n) + j, (row * m) + (column + 1), 1, capacity
                mf_add_edge((m * n) + j, (row * m) + (column - 1), 1, capacity
                mf_add_edge((m * n) + j, ((row + 1) * m) + column, 1, capacity
                mf_add_edge((m * n) + j, ((row - 1) * m) + column, 1, capacity
            }
        }

        // connect the intersections that can lead outside
        for(int j=0; j<(m * n); ++j) {
            int column = j % m;
            int row = floor(j / m);

            if((column == 0 && row != 0) && (column == 0 && row != n - 1))
                mf_add_edge((m * n) + j, sink, 1, capacity, rev_edge, g); //
            else if((column == 0 && row == 0) || (column == 0 && row == n - 1
                mf_add_edge((m * n) + j, sink, 2, capacity, rev_edge, g); //
            else if((column == m - 1 && row != 0) && (column == m - 1 && row
                mf_add_edge((m * n) + j, sink, 1, capacity, rev_edge, g); //
            else if((column == m - 1 && row == 0) || (column == m - 1 && row =
                mf_add_edge((m * n) + j, sink, 2, capacity, rev_edge, g); //
            else if((row == 0 && column != 0) && (row == 0 && column != m - 1
                mf_add_edge((m * n) + j, sink, 1, capacity, rev_edge, g); //
            else if((row == n - 1 && column != 0) && (row == n - 1 && column
                mf_add_edge((m * n) + j, sink, 1, capacity, rev_edge, g); //

        }

        long flow = push_relabel_max_flow(g, source, sink);

        cout << flow << endl;
    }
    return 0;
}
```