Queen's Attack II

```
#include <bits/stdc++.h>

using namespace std;

// Function prototypes for rtrim, ltrim, and split
string ltrim(const string &);
string rtrim(const string &);
vector<string> split(const string &);

// The queensAttack function definition (already provided in the
previous solution)
int queensAttack(int n, int k, int r_q, int c_q,
vector<vector<int>> obstacles) {
    // Directions: (row change, column change)
    vector<pair<int, int>> directions = {
        {-1, 0}, {1, 0}, {0, -1}, {0, 1},  // up, down, left, right
        {-1, -1}, {-1, 1}, {1, -1}, {1, 1}  // 4 diagonals
    };

    // Set for quick lookup of obstacles
    set<pair<int, int>> obstacleSet;
    for (auto& obs : obstacles) {
        obstacleSet.insert({obs[0], obs[1]});
    }

    // Variable to store number of attackable squares
    int attackableSquares = 0;

    // Check all 8 possible directions
    for (auto& dir : directions) {
        int r = r_q, c = c_q;

        // Keep moving in the current direction until we hit the
boundary or an obstacle
        while (true) {
            r += dir.first;
            c += dir.second;

            // If out of bounds, break the loop
```

```
            if (r < 1 || r > n || c < 1 || c > n) break;

            // If there's an obstacle, break the loop
            if (obstacleSet.count({r, c}) > 0) break;

            // Otherwise, this square is attackable
            attackableSquares++;
        }
    }

    return attackableSquares;
}

int main() {
    ofstream fout(getenv("OUTPUT_PATH"));

    string first_multiple_input_temp;
    getline(cin, first_multiple_input_temp);

    vector<string> first_multiple_input =
split(rtrim(first_multiple_input_temp));

    int n = stoi(first_multiple_input[0]);
    int k = stoi(first_multiple_input[1]);

    string second_multiple_input_temp;
    getline(cin, second_multiple_input_temp);

    vector<string> second_multiple_input =
split(rtrim(second_multiple_input_temp));

    int r_q = stoi(second_multiple_input[0]);
    int c_q = stoi(second_multiple_input[1]);

    vector<vector<int>> obstacles(k);

    for (int i = 0; i < k; i++) {
        obstacles[i].resize(2);

        string obstacles_row_temp_temp;
        getline(cin, obstacles_row_temp_temp);
```

```
        vector<string> obstacles_row_temp =
split(rtrim(obstacles_row_temp_temp));

        for (int j = 0; j < 2; j++) {
            int obstacles_row_item = stoi(obstacles_row_temp[j]);

            obstacles[i][j] = obstacles_row_item;
        }
    }

    int result = queensAttack(n, k, r_q, c_q, obstacles);

    fout << result << "\n";

    fout.close();

    return 0;
}

// Function to trim leading spaces
string ltrim(const string &str) {
    string s(str);
    s.erase(s.begin(), find_if(s.begin(), s.end(),
not1(ptr_fun<int, int>(isspace))));
    return s;
}

// Function to trim trailing spaces
string rtrim(const string &str) {
    string s(str);
    s.erase(find_if(s.rbegin(), s.rend(), not1(ptr_fun<int,
int>(isspace))).base(), s.end());
    return s;
}

// Function to split the string by space into a vector
vector<string> split(const string &str) {
    vector<string> tokens;
    string::size_type start = 0;
    string::size_type end = 0;

    while ((end = str.find(" ", start)) != string::npos) {
        tokens.push_back(str.substr(start, end - start));
```

```
        start = end + 1;
    }

    tokens.push_back(str.substr(start));
    return tokens;
}
```