HACKERRANK DAA ASSIGNMENT

P YOSHITHA

2211CS020607

AIML EPSILON

Qs. Queen's Attack II

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

using namespace std;

string ltrim(const string &); string rtrim(const string &); vector split(const string &);

/*

- Complete the 'queensAttack' function below.
-
- The function is expected to return an INTEGER.
- The function accepts following parameters:
-
    - INTEGER n
-
    - INTEGER k
-
    - INTEGER r_q
-
    - INTEGER c_q
-
    - 2D_INTEGER_ARRAY obstacles

*/

int queensAttack(int n, int k, int r_q, int c_q, vector<vector> obstacles) { vector<pair<int, int>> directions = { {-1, 0}, {1, 0}, {0, -1}, {0, 1}, // Vertical and Horizontal {-1, -1}, {-1, 1}, {1, -1}, {1, 1} // Diagonal directions };
```

```
// Set to store the obstacles for quick lookup
set<pair<int, int>> obstacleSet;
for (const auto& obstacle : obstacles) {
    obstacleSet.insert({obstacle[0], obstacle[1]});
}

int attackCount = 0;

// Loop over all 8 directions
for (auto dir : directions) {
    int r = r_q, c = c_q;

    // Move step-by-step in the direction until we hit an edge or obstacle
    while (true) {
        r += dir.first;
        c += dir.second;

        // Check if the new position is out of bounds
        if (r < 1 || r > n || c < 1 || c > n) {
            break; // Out of bounds, stop moving in this direction
        }

        // Check if the position is an obstacle
        if (obstacleSet.find({r, c}) != obstacleSet.end()) {
            break; // Obstacle found, stop moving in this direction
        }

        // If not out of bounds and no obstacle, this is a valid attack square
        attackCount++;
    }
}

return attackCount;


}

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;


}

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;


}

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;


}

vector split(const string &str) { vector 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;

}
```