queens.cpp

```
/* FILE: queens.cpp   last change: 26-Feb-2013   author: Romeo Rizzi
 * a solver for problem queens
 */

//#define NDEBUG   // NDEBUG definita nella versione che consegno
#include <cassert>
#ifndef NDEBUG
#  include <iostream>  // uso di cin e cout non consentito in versione finale
#endif
#include <fstream>

using namespace std;

const int MAX_M = 100, MAX_N = 100;  // massimo numero di righe e colonne;
int m, n; // numero di righe e colonne
int qB, qW; // numeri di regine Black & White

int num_covers[MAX_M][MAX_N];  // nella rappresentazione interna le righe e le colonne sono numerat
e partendo da 0.
                               // num_covers[i][j] = quante delle regine gia' collocate insistono s
ulla cella (i,j).
int num_sol[MAX_M + MAX_N +1]; // per evitare di dover prima stabilire il valore della soluzione ot
tima, e solo poi contare il numero delle soluzioni ottime, num_sol[q] = numero di soluzioni massima
li con q regine. (Massimale = non risulta possibile aggiungere alcuna regina. Nota: tutte quelle co
n numero massimo di regine bianche sono massimali).

void displayMatrix(int mat[MAX_M][MAX_N]) {
  for(int i = 1; i <= m; i++) {
    for(int j = 1; j <= n; j++)
      cout << mat[i][j] << " ";
    cout << endl;
  }  cout << endl;
}

void cover(int i, int j, int delta) {
  //displayMatrix(num_covers);
  //cout << "Colloca regina di val " << delta << " in (" << i << "," << j << ")" << endl;
  //displayMatrix(num_covers);
  num_covers[i][j] += delta;
  for(int k = 1; k < n; k++) num_covers[i][ (j+k) % n ]  += delta; // aggiorno la riga
  for(int k = 1; k < m; k++) num_covers[ (i+k) % m ][j]  += delta; // aggiorno la colonna
  int ii, jj;  // ed ora le 4 diagonali ...
  for(ii = i+1, jj = j+1; (ii < m) && (jj < n); ii++, jj++ ) num_covers[ii][jj] += delta;
  for(ii = i+1, jj = j-1; (ii < m) && (jj >= 0); ii++, jj-- ) num_covers[ii][jj] += delta;
  for(ii = i-1, jj = j+1; (ii >= 0) && (jj < n); ii--, jj++ ) num_covers[ii][jj] += delta;
  for(ii = i-1, jj = j-1; (ii >= 0) && (jj >= 0); ii--, jj-- ) num_covers[ii][jj] += delta;
}

void ric(int qW, int n_cells_settled) {
// Si consideri di avere numerato le m*n celle della scacchiera riga per riga.
// Quando questa procedura viene chiamata, la situazione delle prime n_cells_settled caselle della
scacchiera deve essere considerata come gia' determinata, collocando in esse precisamente qW regine
 bianche, e la procedura deve conteggiare le soluzioni compatibili a questo "prefisso" di soluzione
.

  if( n_cells_settled == m*n ) num_sol[qW]++;
  else {
    // calcolo coordinate di prima cella non ancora determinata:
    int i = n_cells_settled / n;
    int j = n_cells_settled % n;
    // ipotesi di non utilizzo di tale cella:
    ric( qW, n_cells_settled + 1 );
    // ipotesi alternativa:
    if( num_covers[i][j] == 0 ) {  // (i,j) e' una casella possibile
      cover(i, j, +1);  // BEGIN: ipotesi colloco in (i,j)
      ric( qW +1, n_cells_settled + 1 );
      cover(i, j, -1);  // END: ipotesi colloco in (i,j)
    }
  }
}

int main() {
  ifstream fin("input.txt"); assert( fin );
  fin >> m >> n >> qB;
  for(int i = 0; i <= m + n; i++)
    num_sol[i] = 0;
  for(int i = 0; i < m; i++)
```

queens.cpp

```
    for(int j = 0; j < n; j++)
        num_covers[i][j] = 0;
  int a, b;
  for(int q = 1; q <= qB; q++) {
    fin >> a >> b;
    cover(a-1, b-1, +1);
  }
  fin.close();

  ric(0, 0); // richiesta conteggio soluzioni di varia cardinalita'
  int i = m + n;
  while( num_sol[i] == 0 )  { assert( i >= 0 ); i--; }
  ofstream fout("output.txt"); assert( fout );
  fout << i << " " << num_sol[i] << endl;
  fout.close();
  return 0;
}
```