```
/* FILE: lessie.cpp   last change: 20-Aug-2012   author: Romeo Rizzi
 * a solver for problem "lessie" in the 03-Sept-2012 exam in Algorithms.
 */

#define NDEBUG // NDEBUG disabilita tutte le assert se definita prima di include
re <cassert>
#include <cassert>

#ifndef NDEBUG
#  include <iostream>  // non voglio includerla quando Non compilo in modalita'
DEBUG
#endif
#include <cstdlib>
#include <fstream>

using namespace std;

double round(long double p) { return int(p*10000 +0.0001)/10000.0; }

const int MAX_M = 1000;
const int MAX_N = 1000;
int m, n, qL, qH;
bool wall[MAX_M+2][MAX_N+2];                                    // +2 perche' inten
do bordare tutta la griglia in input con una cornice di celle muro
long double prob_getting_to_m_n_from[MAX_M+2][MAX_N+2];      // per evitare di d
over trattare distintamente le celle ai margini della griglia
long double prob_lessie_from_1_1_reaches[MAX_M+2][MAX_N+2];  // (puo' essere vis
to come un esempio di uso di sentinelle in programmazione)

/* template <class genType>
void displayMatrix(genType mat[MAX_M+2][MAX_N+2]) {
  for(int i = 1; i <= m; i++) {
    for(int j = 1; j <= n; j++)
      cout << mat[i][j] << " ";
    cout << endl;
  }  cout << endl;
}

void checkPrecision() {
  long double one_half = 1.0/2; long double one_half_to_the_n = 1;
  for(int n = 0; n <= 100; n++, one_half_to_the_n *= one_half)
    cout << " n = " << n << ", 1/2 = " << one_half << ", 1/2^n = " << one_half_t
o_the_n << ", 1/2+1/2^n = " << one_half+one_half_to_the_n << ", round(1/2) = " <
< round(one_half) << ", round(1/2^n) = " << round(one_half_to_the_n) << ", round
(1/2+1/2^n) = " << round(one_half+one_half_to_the_n) << endl;
} */

int main() {
  //checkPrecision();
  ifstream fin("input.txt"); assert(fin);   ofstream fout("output.txt"); assert(fout);
  fin >> m >> n >> qL >> qH;
  for(int i = 0; i <= m+1; i++)
    for(int j = 0; j <= n+1; j++) {
      wall[i][j] = true;    // mettiamo ovunque muro, in particolare sul contorn
o (dove servira' come sentinella)
      prob_getting_to_m_n_from[i][j] = 0;
      prob_lessie_from_1_1_reaches[i][j] = 0;
    }
  for(int i = 1; i <= m; i++)
    for(int j = 1; j <= n; j++)
      fin >> wall[i][j];
  //displayMatrix(wall);

  prob_getting_to_m_n_from[m][n] = 1;
  for(int i = m; i >= 1; i--)
    for(int j = n; j >= 1; j--)
      if(!wall[i][j]) {
        if( (wall[i][j+1]) && (!wall[i+1][j]) )  prob_getting_to_m_n_from[i][j]
= prob_getting_to_m_n_from[i+1][j];
```



```
        if( (wall[i+1][j]) && (!wall[i][j+1]) )  prob_getting_to_m_n_from[i][j]
= prob_getting_to_m_n_from[i][j+1];
        if( (!wall[i+1][j]) && (!wall[i][j+1]) )
          prob_getting_to_m_n_from[i][j] = ( prob_getting_to_m_n_from[i+1][j] +
prob_getting_to_m_n_from[i][j+1] ) /2;
      }

  prob_lessie_from_1_1_reaches[1][1] = 1;
  for(int i = 1; i <= m; i++)
    for(int j = 1; j <= n; j++)
      if(!wall[i][j]) {
        if( wall[i-1][j+1] ) prob_lessie_from_1_1_reaches[i][j] += prob_lessie_f
rom_1_1_reaches[i-1][j];
        else prob_lessie_from_1_1_reaches[i][j] += prob_lessie_from_1_1_reaches[
i-1][j] / 2;
        if( wall[i+1][j-1] ) prob_lessie_from_1_1_reaches[i][j] += prob_lessie_f
rom_1_1_reaches[i][j-1];
        else prob_lessie_from_1_1_reaches[i][j] += prob_lessie_from_1_1_reaches[
i][j-1] / 2;
      }
  //displayMatrix(prob_getting_to_m_n_from); displayMatrix(prob_lessie_from_1_1_
reaches);


  fout << round( prob_getting_to_m_n_from[1][1] ) << endl;

  int i, j;
  for(int count = 1; count <= qL; count++) {
    fin >> i>> j;
    fout << round( prob_getting_to_m_n_from[i][j] ) << endl;
  }
  for(int count = 1; count <= qH; count++) {
    fin >> i >> j;
    fout << round( prob_lessie_from_1_1_reaches[i][j] ) << endl;
  }
  fout.close();
  fin.close();
  return 0;
}
```