```
/*
 * Solution to problem "raisins"
 *
 * This solution employs memoization in order to compute the the best
 * way of cutting any sub-rectangle of the chocolate. To do this
 * sufficiently quickly, we also precompute the number of raisins in
 * each sub-rectangle subtended from the top-left corner of the chocolate.
 *
 * This allows us to compute the number of raisins in any rectangular
 * sub-region of the chocolate -- consider the diagram below:
 *
 * (0,0)-------+-------(c,0)
 *   |    A    |    B    |
 *   +-------(a,b)-------+
 *   |    C    |    D    |
 * (0,d)-------+-------(c,d)
 *
 * Suppose we wish to compute the number of raisins in the region
 * (a,b) -> (c,d), D. Now D = (A + B + C + D) - (A + B) - (A + C) + A
 * Each of these four quantities is given by the number of raisins
 * in a rectangular sub-region whose top-left corner is (0,0).
 *
 * Carl Hultquist, chultquist@gmail.com
 */

#include <iostream>
#include <cassert>
#include <climits>
#include <cstring>

using namespace std;

#define MAX_SIZE 50
#define MAX_VAL 1000

// The dimensions of the chocolate
int r, c;

// The number of raisins on each piece of chocolate
short chocolate[MAX_SIZE][MAX_SIZE];

// The number of raisins in a rectangular sub-region subtended from the
// top-left corner of the chocolate.
int raisins[MAX_SIZE][MAX_SIZE];
```

```
// The minimum payment that Bonny must make for Peter to cut any
// rectangular sub-region of the chocolate.
int best[MAX_SIZE][MAX_SIZE][MAX_SIZE][MAX_SIZE];

/**
 * Computes the smallest cost of cutting up the rectangular portion of
 * the chocolate from (r1,c1) to (r2,c2).
 */
int solve(int r1, int c1, int r2, int c2)
{
   if (best[r1][c1][r2][c2] == -1)
   {
      // We haven't computed this previously, so compute it now.

      if (r1 == r2 && c1 == c2) {
         // If we have a single cell left, then we're done:
         // there's no additional cost.
         best[r1][c1][r2][c2] = 0;
      }
      else {
         // We try all the possible cuts, and see which results in
         // the smallest total payment.

         int best_payment = INT_MAX;

         // First try the rows
         for (int r = r1 + 1; r <= r2; r++)
         {
            int payment = solve(r1, c1, r - 1, c2) + solve(r, c1, r2, c2);
            if (payment < best_payment)
               best_payment = payment;
         }

         // Now try the columns
         for (int c = c1 + 1; c <= c2; c++)
         {
            int payment = solve(r1, c1, r2, c - 1) + solve(r1, c, r2, c2);
            if (payment < best_payment)
               best_payment = payment;
         }

         // Determine how many raisins
         int base_raisins = raisins[r2][c2];
```

```
            if (r1 > 0)
               base_raisins -= raisins[r1 - 1][c2];
            if (c1 > 0)
               base_raisins -= raisins[r2][c1 - 1];
            if (r1 > 0 && c1 > 0)
               base_raisins += raisins[r1 - 1][c1 - 1];

            best[r1][c1][r2][c2] = best_payment + base_raisins;
         }
      }

      return best[r1][c1][r2][c2];
}

int main()
{
      cin >> r >> c;
      assert(1 <= r && r <= MAX_SIZE);
      assert(1 <= c && c <= MAX_SIZE);

      // Read in the number of raisins on each piece of chocolate
      for (int i = 0; i < r; i++)
         for (int j = 0; j < c; j++)
         {
            cin >> chocolate[i][j];
            assert(0 <= chocolate[i][j] && chocolate[i][j] <= MAX_VAL);
         }

      memset(raisins, -1, sizeof(raisins));
      memset(best, -1, sizeof(best));

      // First precompute the number of raisins on each rectangular
      // sub-region subtended from the top-left corner of the chocolate.
      raisins[0][0] = chocolate[0][0];

      for (int i = 1; i < r; i++)
         raisins[i][0] = chocolate[i][0] + raisins[i - 1][0];

      for (int j = 1; j < c; j++)
      {
         raisins[0][j] = chocolate[0][j] + raisins[0][j - 1];

         for (int i = 1; i < r; i++)
            raisins[i][j] = chocolate[i][j] + raisins[i - 1][j] + raisins[i][j - 1] - raisins[i - 1][j - 1];
```

```
    }

    // Find the minimum payment for cutting up the whole slab of
    // chocolate.
    cout << solve(0, 0, r - 1, c - 1) << endl;

    return 0;
}
```