oddKnap.cpp

```
/* FILE: oddKnap.cpp   last change: 30-Sep-2014   author: Romeo Rizzi
 * a DP solver for problem "oddKnap"
 */

//#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_B = 100000; int B;
const int MAX_N = 1000; int n;
int p[MAX_N];  // i pesi e gli oggetti sono indicizzati partendo da zero

const int DOWN_TO_HELL = -MAX_B -1;

int opt[2][MAX_N][MAX_B+1];
int num[2][MAX_N][MAX_B+1];
/* opt[0][i][b]  stores the maximum sum non exceeding b, for an even-sized subsequence of p[0],p[1]
,...,p[i].
   num[0][i][b]  gives the number (mod 100.000) of even-sized subsequence of p[0],p[1],...,p[i] wit
h sum opt[0][i][b].
   opt[1][i][b]  stores the maximum sum non exceeding b, for all odd-sized subsequences of p[0],p[1
],...,p[i].
   num[1][i][b]  gives the number (mod 100.000) of odd-sized subsequence of p[0],p[1],...,p[i] with
 sum opt[0][i][b].
 */


#ifndef NDEBUG
void displayVect(int vect[], int n) {
  for(int i = 0; i < n; i++)  cout << vect[i] << " ";
  cout << endl;
}
#endif

int myMax( int a, int b ) { return (a>b)? a : b; }

ofstream fout("output.txt");


int main() {
  ifstream fin("input.txt"); assert( fin );
  fin >> n >> B; assert( n <= MAX_N ); assert( B <= MAX_B );
  for(int i = 0; i < n; i++)  fin >> p[i];
  fin.close(); //displayVect(p, n);

  for(int b = 0; b <= B; b++) {
    opt[0][0][b] = 0;
    num[0][0][b] = 1;  // la sola soluzione che non prende nulla
    opt[1][0][b] = ( p[0] <= b ) ? p[0] : DOWN_TO_HELL ;
    num[1][0][b] = ( p[0] <= b ) ? 1 : 0 ;
  }
  // displayVect(opt[0][0], B +1); displayVect(opt[1][0], B +1);

  for(int i = 1; i < n; i++) {
    for(int b = 0; b <= B; b++)
      for(int k = 0; k < 2; k++)
        if( p[i] > b ) {
          opt[k][i][b] = opt[k][i-1][b] ;
          num[k][i][b] = num[k][i-1][b] ;
        }
        else {
          if( opt[k][i-1][b] > p[i] + opt[1-k][i-1][b - p[i]] ) {
            opt[k][i][b] = opt[k][i-1][b] ;
            num[k][i][b] = num[k][i-1][b] ;
          }
          else if( opt[k][i-1][b] == p[i] + opt[1-k][i-1][b - p[i]] ) {
            opt[k][i][b] = opt[k][i-1][b] ;
            num[k][i][b] = ( num[k][i-1][b] + num[1-k][i-1][b - p[i]] ) % 100000 ;
          }
          else {
            opt[k][i][b] = p[i] + opt[1-k][i-1][b - p[i]] ;
            num[k][i][b] = num[1-k][i-1][b - p[i]] ;
```

oddKnap.cpp

```
          }
        }
    // displayVect(opt[0][i], B +1); displayVect(opt[1][i], B +1);
  }
  fout << opt[1][n-1][B] << endl;
  fout << num[1][n-1][B] << endl;
  fout.close();
  return 0;
}
```