```
/* FILE: 15game16.cpp   last change: 23-July-2012   author: Romeo Rizzi
 * This program is a solver for problem 3 (15game16.cpp) in 23-07-2012 exam in A
lgorithms
 */

// #include <iostream>  // NOTA: ho commentato questa riga per essere sicuro di
non aver lasciato operazioni di input/output di debug nella fretta di consegnare
, se compila cosi' non ci sono ed io non perdo stupidamente i miei punti-esame.
#include <cstdlib>
#include <fstream>
#include <cassert>

using namespace std;

const int N = 4;  // ma funziona anche per N piu' piccolo (utile per congettura
su cosa sia risolvibile per N generale ... ). Per N piu' grande abbiamo rapida e
splosione del numero di configurazioni (che vanno in memoria).
int table[N+1][N+1];
const int MAX_CODE = 522255;
int dist[MAX_CODE +1];
int FIFOqueue[MAX_CODE +1], w_pos, r_pos;

int encode(int table[N+1][N+1]) {
  int code = 0;
  int pos_empty;
  for(int i = 1, cell = 0; i <= N; i++)
    for(int j = 1; j <= N; j++, cell++) {
      if( table[i][j] == -1 ) pos_empty = cell;
      else {
        code *= 2;
        code += table[i][j];
      }
    } code *= 16;
  return code += pos_empty;
}

/* DECODING:
  int pos_empty = code % 16; code /= 16;
  for(int i = N, cell = N*N-1; i >= 1; i--)
    for(int j = N; j >= 1; j--, cell--)
      if(cell == pos_empty) table[i][j] = -1;
      else {
        table[i][j] = code % 2;
        code /= 2;
      }
*/

/*  nota: sono stato forzato a commentarla per riuscire a compilare (non avendo
<iostream>)
          col vantaggio di essere stato costretto a lasciare in evidenza solo ci
o' che e' necessario
void display(int table[N+1][N+1]) {
  for(int i = 1; i <= N; i++) {
    for(int j = 1; j <= N; j++)
      if( table[i][j] == -1) cout << "X ";
      else cout << table[i][j] << " ";
    cout << endl;
  }  cout << endl;
}
*/
void swap (int &a, int &b) { int tmp = a; a = b; b = tmp; }

int main() {
  for(int i = 1, cell = 0; i <= N; i++)
    for(int j = 1; j <= N; j++, cell++)
      if( cell < (N*N)/2)  table[i][j] = 1;
      else  table[i][j] = 0;
  table[N][N] = -1;
  //display(table);
```



```
  int target_code = encode(table);
  /* fact: the target code is also the maximum code for a configuration with pre
cisely (N*N)/2 cells
           set to 1. We used this fact to size the bool vector reached:

      cout << target << endl;   returned: 522255  which we used to define MAX_CO
DE
  */

  for(int code = 0; code <= MAX_CODE; code++)
    dist[code] = -1;

  int numOnes = 0;
  ifstream fin("input.txt"); assert(fin); char spoon;
  for(int i = 1; i <= N; i++)
    for(int j = 1; j <= N; j++) {
      fin >> spoon;  while( spoon == ' ' ) fin >> spoon;
      if( spoon == 'B' ) { table[i][j] = 1; numOnes++; }
      if( spoon == 'N' ) table[i][j] = 0;
      if( spoon == '0' ) table[i][j] = -1;
    }
  fin.close();
  // display(table);
  w_pos = r_pos = 0;
  if( numOnes != (N*N)/2 ) ;  //in questo caso la raggiungibilita' e' impossibil
e, quindi non avvio la BFS
  else { dist[encode(table)] = 0; FIFOqueue[w_pos++] = encode(table); }
  while( w_pos > r_pos) {
    int table_code = FIFOqueue[r_pos++];
    // decoding the table code of the current configuration (or node in the BFS
tree) ...
    int code = table_code; // non vogliamo sporcare table_code che resta quello
del nodo corrente da esplorare
    int pos_empty = code % 16; code /= 16;
    int X_row, X_col;
    for(int i = N, cell = N*N-1; i >= 1; i--)
      for(int j = N; j >= 1; j--, cell--)
        if(cell == pos_empty)  { X_row = i; X_col = j; table[i][j] = -1; }
        else {
          table[i][j] = code % 2;
          code /= 2;
        }
    // decoding done. Now exploring the neighborhood of the node in the configur
ation graph ...
    // cout << "explore node: " << table_code << endl; display(table);
    if( X_row > 1 ) {
      swap( table[X_row][X_col], table[X_row -1][X_col]);
      code = encode(table); // display(table);
      swap( table[X_row][X_col], table[X_row -1][X_col]);
      if( dist[code] == -1 ) {
        dist[code] = 1 + dist[table_code];
        FIFOqueue[w_pos++] = code;
      }
    }
    if( X_row < N ) {
      swap( table[X_row][X_col], table[X_row +1][X_col]);
      code = encode(table); // display(table);
      swap( table[X_row][X_col], table[X_row +1][X_col]);
      if( dist[code] == -1 ) {
        dist[code] = 1 + dist[table_code];
        FIFOqueue[w_pos++] = code;
      }
    }
    if( X_col > 1 ) {
      swap( table[X_row][X_col], table[X_row][X_col -1]);
      code = encode(table); // display(table);
      swap( table[X_row][X_col], table[X_row][X_col -1]);
      if( dist[code] == -1 ) {
```

```
        dist[code] = 1 + dist[table_code];
        FIFOqueue[w_pos++] = code;
      }
    }
    if( X_col < N ) {
      swap( table[X_row][X_col], table[X_row][X_col +1]);
      code = encode(table); // display(table);
      swap( table[X_row][X_col], table[X_row][X_col +1]);
      if( dist[code] == -1 ) {
        dist[code] = 1 + dist[table_code];
        FIFOqueue[w_pos++] = code;
      }
    }
    //getchar();
  }

  ofstream fout("output.txt"); assert(fout);
  fout << dist[target_code];
  fout.close();
  return 0;
}
```