Code Jam 2008 - AMER Semifinal

Analysis: King

The problem corresponds to a relatively unknown game named Slither. This game was popularized by Martin Gardner in one of his columns written for Scientific American. It wasn't solved then, but eventually William Anderson developed a solution. Of course Code Jam participants are much smarter, so we thought the problem would easily get solved in two hours.

I'll explain first the solution for a simpler variant of the problem. Suppose we have a *3x3* board with the king being in one of the corner cells. Now let's cover the rest of the board with dominoes of size *2x1*. This tiling of the board shows us a winning strategy for the second player. Each time the first player moves to one cell of a domino, the second player will move to the other cell of that domino. Since the second player can always make a move after the first player moves, the second player has a winning strategy.

The general solution goes along the same lines: the domino tiling corresponds to a maximum matching. We think of the board as a graph where the cells are the nodes, and neighboring cells have edges between them. The dominoes, then, correspond to edges in the maximum matching.

The solution is given by the following theorem: *"The first player has a winning strategy if and only if all maximum matchings contain the king's node"* .

This means that no alternating path starting from the king's node ends in a vertex that is not in the matching, because if it ended in a unmatched vertex we could construct another matching with the same number of edges that doesn't use the king's node. Thus the first player could always move along a matched edge.

If there is a maximum matching that doesn't contain the king's node, then in this graph every alternating path that starts from the kings node ends in a matched node, otherwise we could increase the size of the matching which contradicts the assumption that the matching is maximum. Thus now the second player can always move in the second node of the matching edges. So the second player has a winning strategy.

The graph in our problem is not bipartite so we can't use the standard algorithm for bipartite matching. Instead we use a solution that combines brute force and dynamic programming. Our solution will be exponential rather than polynomial, but the size of the problem is small so we can afford it. We go row by row from left to right, and for each cell *(i, j)* and each subset *S* of *{0, 1, .. n - 1}* we compute the best matching that has the following nodes matched already: nodes on the active line (*[i][0..j], [i-1][j+1 .. n - 1]*) that correspond to the numbers in *S*, and matched nodes from before the active line.

Here's Derek Kisman's code that implements this solution:

```
#include <algorithm>
#include <iostream>
using namespace std;

int gx, gy;
char g[15][15];

char memo[15][15][1<<16];
char doit(int x, int y, int b) {
  if (x == gx) {
```

```
    x = 0;
    if (++y == gy) return 0;
  }
  char& ret = memo[x][y][b];
  if (ret != -1) return ret;
  int b2 = (b<<1) & ((1<<(gx+1))-1);
  if (g[y][x] != '.') {
    ret = doit(x+1, y, b2);
  } else {
    ret = doit(x+1, y, b2+1);
    if (x && (b&1)) ret >?= 1 + doit(x+1, y, b2-2);
    if (x && (b&(1<<gx)))
      ret >?= 1 + doit(x+1, y, b2);
    if (b&(1<<(gx-1)))
      ret >?= 1 + doit(x+1, y, b2-(1<<gx));
    if (x < gx-1 && (b&(1<<(gx-2))))
      ret >?= 1 + doit(x+1, y, b2-(1<<(gx-1)));
  }
  return ret;
}

main() {
  int N, prob=1;
  for (cin >> N; N--;) {
    cin >> gy >> gx;
    int kx, ky;
    for (int y = 0; y < gy; y++)
    for (int x = 0; x < gx; x++) {
      cin >> g[y][x];
      if (g[y][x] == 'K') {kx = x; ky = y;}
    }
    memset(memo, -1, sizeof(memo));
    int m1 = doit(0, 0, 0);
    g[ky][kx] = '.';
    memset(memo, -1, sizeof(memo));
    int m2 = doit(0, 0, 0);
    cout << "Case #" << prob++
         << ": " << ((m2 > m1) ? 'A' : 'B')
         << endl;
  }
}
```