USACO MAR08 Problem 'ctravel' Analysis

by Alex Schwendner

The brute-force approach to this problem would be to recursively try all 4^T paths starting at (R1,C1) and count those that end at (R2,C2) without going through a tree. Since 4^15 = 1,073,741,824, this is too slow.

However, there are a lot of things that are being done twice. In particular, if two paths of length T/2 starting at (R1,C1) both end at the same point (R3,C3), then there will be exactly the same number of ways to extend each path to a path of length T. This repetition makes this problem a prime candidate for Dynamic Programming.

The essential aspects of a partial path are its length and its ending point. Thus, we keep a TxNxM table, the (t,i,j)-th entry of which is the number of paths of length t starting at (R1,C1) and ending at (i,j). Call this number f(t,i,j). If we are not next to a tree or to the edge of the board, then

f(t,i,j) = f(t-1,i-1,j) + f(t-1,i+1,j) + f(t-1,i,j-1) + f(t-1,i,j+1).

If there is a tree or if we are adjacent to the edge of the board, than one or more of these terms may be missing. This equation gives us a way to express the t-th slice of our TxNxM table in terms of the (t-1)-st slice. We use it to build up the table, at which point our answer is the entry at (T,R1,C2).

The total time is 4*T*N*M=600,000 which is easily fast enough.

```
#include <assert.h>
#include <stdio.h>

const int MAXN = 256;
const int MAXT = 32;

int h,w;
int t;
int r1,c1,r2,c2;
char grid[MAXN][MAXN+1];
int dp[MAXN][MAXN][MAXT];

int main() {

  FILE *fin = fopen("ctravel.in", "r");
  fscanf(fin, "%d %d %d", &h, &w, &t);
  for(int i = 0; i < h; ++i){
    fscanf(fin, "%s", grid[i]);
  }
  fscanf(fin, "%d %d %d %d", &r1, &c1, &r2, &c2);
  --r1; --c1; --r2; --c2;
  assert(0 <= r1 && r1 <= h);
  assert(0 <= r2 && r2 <= h);
  assert(0 <= c1 && c1 <= w);
```

```
  assert(0 <= c2 && c2 <= w);
  assert(grid[r1][c1] == '.');
  assert(grid[r2][c2] == '.');
  fclose(fin);

  for(int i = 0; i < h; ++i){
    for(int j = 0; j < w; ++j){
      dp[i][j][0] = 0;
    }
  }
  dp[r1][c1][0] = 1;

  for(int i = 1; i <= t; ++i){
    for(int y = 0; y < h; ++y){
      for(int x = 0; x < w; ++x){
        dp[y][x][i] = 0;
        if(grid[y][x] == '*') continue;
        const int dy[4] = {0,0,+1,-1};
        const int dx[4] = {+1,-1,0,0};
        for(int d = 0; d < 4; ++d){
          int y2 = y + dy[d];
          int x2 = x + dx[d];
          if(0 <= x2 && x2 < w &&
             0 <= y2 && y2 < h){
            if(grid[y2][x2] != '*'){
              dp[y][x][i] += dp[y2][x2][i-1];
            }
          }
        }
      }
    }
  }

  FILE *fout = fopen("ctravel.out", "w");
  fprintf(fout, "%d\n", dp[r2][c2][t]);
  fclose(fout);

  return(0);
}
```