USACO FEB09 Problem 'coggle' Analysis

by Rob Kolstad

The winning paradigm for this task is to read the dictionary to see what words can beformed (not, e.g., to try to form all the words and see if they are in the dictionary).

Program elements include:

- Proper input that, in this case, retains the '\n' character on the end of each dictionary word.
- A recursive 'walker' that meanders among the letters, trying to form a word. It uses a set of integers chosen carefully from {-1, 0, 1} to name all eight possible ways one can move from any given die. The seen[5][5] array keeps track of which letters have been used so that no die's letter is used twice. Note that the recursive routine marks and unmarks seen[][] and thus simplifies bookkeeping.
- Boundary checking to make sure no character outside the 5 x 5 matrix is accessed.
- A pair loops that ensure testing of all possible starting locations.
- A 'goto' to exit the inner of two loops.

All in all a classic recursive program; see below for my realization.

```
#include<stdio.h>

struct dir_f {
    int dx;
    int dy;
} dirs[] = { {-1,0}, {-1,-1}, {0,-1}, {1,-1}, {1,0}, {1,1}, {0,1},
                {-1,1}};

char letters[5][5];
char line[100];
int count;              /* # solutions */
char seen[5][5];

search (i, j, k) {
if (debug)printf("%d:search i=%d j=%d k=%d\n", k, i, j, k);
    if (i < 0 || i >= 5 || j < 0 || j >= 5) return 0;
    if (seen[i][j]) return 0;
    if (line[k] == '\n') {              /* match! */
        if (debug) printf("MATCH -- %s", line);
        count++;
        return 1;
    }
    if (letters[i][j] != line[k]) return 0;
    seen[i][j] = 1;
    int l;
    for (l = 0; l < 8; l++) {
        if (search (i + dirs[l].dx, j + dirs[l].dy, k+1)) {
            seen[i][j] = 0;
            return 1;
```

```
        }
    }
    seen[i][j] = 0;
    return 0;
}

int main() {
    FILE *fin = fopen ("coggle.in", "r");
    FILE *fout = fopen ("coggle.out", "w");
    FILE *fdict = fopen ("dict.txt", "r");
    int i, j;

    for (i = 0; i < 5; i++) {
        fgets (line, 100, fin);
        for (j = 0; j < 5; j++)
            letters[i][j] = line[2*j];
    }
    count = 0;
    while (fgets (line, 100, fdict)) {
        for (i = 0; i < 5; i++)
            for (j = 0; j < 5; j++)
                if (letters[i][j] == line[0])
                    if (search(i, j, 0))
                        goto DONE;
        DONE:;
    }
    fprintf (fout, "%d\n", count);
    exit (0);
}
```