USACO Traingate 'camelot' Analysis

by Hal Burch

This is a modification of the shortest path algorithm. If there was no king, then the shortest path algorithm can determine the distance that each knight must travel to get to each square. Thus, the cost of gathering in a particular square is simply the sum of the distance that each knight must travel, which is fairly simple to calculate.

In order to consider the king, consider a knight which 'picks-up' the king in some square and then travels to the gathering spot. This costs some number of extra moves than just traveling to the gathering spot. In particular, the king must move to the pick-up square, and the knight must travel to this square and then to the final gathering point. Consider the number of extra moves to be the `cost' for that knight to pick-up the king. It is simple to alter the shortest path algorithm to consider picking-up the king by augmenting the state with a boolean flag stating whether the knight has the king or not.

In this case, the cost for gathering at a particular location is the sum of the distance that each knight must travel to get to that square plus the minimum cost for a knight picking up the king on the way.

Thus, for each square, we keep two numbers, the sum of the distance that all the knights that we have seen thus far would have to travel to get to this square and the minimum cost for one of those knights picking up the king on the way (note that one way to 'pick-up' the king is to have the king travel all by itself to the gathering spot). Then, when we get a new knight, we run the shortest path algorithm and add the cost of getting that knight (without picking up the king) to each square to the cost of gathering at that location. Additionally, for each square, we check if the new knight can pick-up the king in fewer moves than any previous knight, and update that value if it can.

After all the knights have been processed, we determine the minimum over all squares of the cost to get to that square plus the additional cost for a knight to pick-up the king on its way to that square.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* "infinity"... > maximum distance possible (for one knight) */
#define MAXN 10400

/* maximum number of rows */
#define MAXR 40

/* maximum number of columns */
#define MAXC 26

/* cost of collecting all knights here */
int cost[MAXC][MAXR];

/* cost of getting a knight to collect the king */
int kingcost[MAXC][MAXR];

/* distance the king must travel to get to this position */
int kdist[MAXC][MAXR];

/* distance to get for current knight to get to this square */
/* third index: 0 => without king, 1 => with king */
int dist[MAXC][MAXR][2];
```

```
/* number of rows and columns */
int nrow, ncol;

int do_step(int x, int y, int kflag) {
    int f = 0; /* maximum distance added */
    int d = dist[x][y][kflag]; /* distance of current move */

  /* go through all possible moves that a knight can make */
    if (y > 0) {
        if (x > 1)
             if (dist[x-2][y-1][kflag] > d+1) {
                 dist[x-2][y-1][kflag] = d+1;
                 f = 1;
             }
            if (x < ncol-2) {
                if (dist[x+2][y-1][kflag] > d+1) {
                    dist[x+2][y-1][kflag] = d+1;
                    f = 1;
                }
            }
            if (y > 1) {
                if (x > 0)
                    if (dist[x-1][y-2][kflag] > d+1) {
                        dist[x-1][y-2][kflag] = d+1;
                        f = 1;
                    }
                if (x < ncol-1)
                    if (dist[x+1][y-2][kflag] > d+1) {
                        dist[x+1][y-2][kflag] = d+1;
                        f = 1;
                    }
            }
    }
    if (y < nrow-1) {
        if (x > 1)
            if (dist[x-2][y+1][kflag] > d+1) {
                dist[x-2][y+1][kflag] = d+1;
                f = 1;
            }
            if (x < ncol-2) {
                if (dist[x+2][y+1][kflag] > d+1) {
                    dist[x+2][y+1][kflag] = d+1;
                    f = 1;
                }
            }
        if (y < nrow-2) {
            if (x > 0)
                if (dist[x-1][y+2][kflag] > d+1) {
                    dist[x-1][y+2][kflag] = d+1;
                    f = 1;
                }
            if (x < ncol-1)
                if (dist[x+1][y+2][kflag] > d+1) {
                    dist[x+1][y+2][kflag] = d+1;
                    f = 1;
                }
        }
    }

/* also check the 'pick up king here' move */
    if (kflag == 0 && dist[x][y][1] > d + kdist[x][y]) {
        dist[x][y][1] = d + kdist[x][y];
        if (kdist[x][y] > f) f = kdist[x][y];
```

```
    }
    return f; /* 1 if simple knight move made, 0 if no new move found */
}

void calc_dist(int col, int row) {
    int lv, lv2;       /* loop variables */
    int d;             /* current distance being checked */
    int max;           /* maximum finite distance found so far */
    int f;             /* temporary variable (returned value from do_step */

/* initiate all positions to be infinite distance away */
    for (lv = 0; lv < ncol; lv++)
        for (lv2 = 0; lv2 < nrow; lv2++)
            dist[lv][lv2][0] = dist[lv][lv2][1] = MAXN;

/* starting location is zero w/o king, kdist[col][row] with king */
    dist[col][row][0] = 0;
    max = dist[col][row][1] = kdist[col][row];

    for (d = 0; d <= max; d++) { /* for each distance away */
        for (lv = 0; lv < ncol; lv++)
            for (lv2 = 0; lv2 < nrow; lv2++) {
                            /* for each position that distance away */
                if (dist[lv][lv2][0] == d) {
                             /* update with moves through this square */
                    f = do_step(lv, lv2, 0);
                    if (d + f > max)     /* update max if necessary */
                       max = d + f;
                 }

                 if (dist[lv][lv2][1] == d) {
                       /* same as above, except this time knight has king */
                     f = do_step(lv, lv2, 1);
                     if (d + f > max) max = d + f;
                 }
            }
    }
}

int main(int argc, char **argv) {
    FILE *fout, *fin;
    char t[10];
    int pr, pc;
    int lv, lv2;
    int i, j;

    if ((fin = fopen("camelot.in", "r")) == NULL) {
        perror ("fopen fin");
        exit(1);
    }
    if ((fout = fopen("camelot.out", "w")) == NULL) {
        perror ("fopen fout");
        exit(1);
    }

    fscanf (fin, "%d %d", &nrow, &ncol);
    fscanf (fin, "%s %d", t, &pr);
    pc = t[0] - 'A';
    pr--;

  /* Calculate cost of moving king from starting position to
   * each board position.  This is just the taxi-cab distance */
   for (lv = 0; lv < ncol; lv++)
```

```
      for (lv2 = 0; lv2 < nrow; lv2++) {
          i = abs(pc-lv);
          j = abs(pr-lv2);
          if (i < j) i = j;
          kingcost[lv][lv2] = kdist[lv][lv2] = i;
      }

    while (fscanf (fin, "%s %d", t, &pr) == 2) { /* for all knights */
        pc = t[0] - 'A';
        pr--;

        /* calculate distances */
        calc_dist(pc, pr);

        for (lv = 0; lv < ncol; lv++)
            for (lv2 = 0; lv2 < nrow; lv2++) {
                /* to collect here, we must also move knight here */
                cost[lv][lv2] += dist[lv][lv2][0];

                /* check to see if it's cheaper for the new knight to
                   pick the king up instead of whoever is doing it now */
                if (dist[lv][lv2][1] - dist[lv][lv2][0] < kingcost[lv][lv2]) {
                    kingcost[lv][lv2] = dist[lv][lv2][1] - dist[lv][lv2][0];
                }
            }
    }
    /* find best square to collect in */
    pc = cost[0][0] + kingcost[0][0];

    for (lv = 0; lv < ncol; lv++)
        for (lv2 = 0; lv2 < nrow; lv2++)
            if (cost[lv][lv2] + kingcost[lv][lv2] < pc) /* better square? */
                pc = cost[lv][lv2] + kingcost[lv][lv2];
  fprintf (fout, "%i\n", pc);
  return 0;
}
```