Practice 7 (2016/11/1)

Shortest Path & Knight Tour on a Chessboard

1. Conditional Compilation

A. *#define, #ifdef, #ifndef, #else, #endif*

B. Example

☞ Version 1

```c
#include <stdio.h>
int main()  {
    int i, j, ret_no;

    printf("Please input two numbers:");
    ret_no = scanf("%d %d",&i, &j);
    printf("\nThe legal number of inputs are %d\n",ret_no);
    /* printf("\nThe legal number of inputs are %d\n",scanf("%d %d",&i,&j)); */
    return 1;
}
```

☞ Version 2

```c
#include <stdio.h>
int main()  {
    int i, j, ret_no;

    printf("Please input two numbers:");
    /*
    ret_no = scanf("%d %d",&i, &j);
    printf("\nThe legal number of inputs are %d\n",ret_no);
    */
    printf("\nThe legal number of inputs are %d\n",scanf("%d %d",&i,&j));
    return 1;
}
```

☞ Version 3

```c
#include <stdio.h>
#define _V1
int main()  {
    int i, j;

#ifdef _V1
    int ret_no;
#endif

    printf("Please input two numbers:");
#ifdef _V1
    ret_no = scanf("%d %d",&i, &j);
    printf("\nThe legal number of inputs are %d\n",ret_no);
#else
    printf("\nThe legal number of inputs are %d\n",scanf("%d %d",&i,&j));
#endif

    return 1;
}
```

2. Exhaustive Searching or Brute-Force

Brute-force method is commonly used by programming beginners and engineers. You can also call it exhaustive-searching method. Brute-force method tries to

explore every possible solution and reports the first solution it finds or the best solution after it scans all combinations in the solution space. Due to the significantly increasing computation power of computers, brute-force becomes a feasible way to solve a problem if it is affordable to scan all solution space by current computing power. With the simple techniques we have learned so far we also can solve some interesting problems with brute-force method. For instance, the problem of finding a shortest path from source to target on a chessboard also can be solved by brute-force.

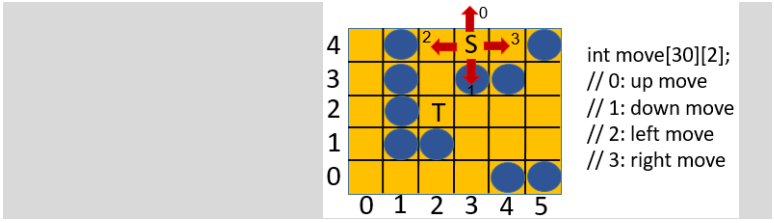


Figure 1. First routing example

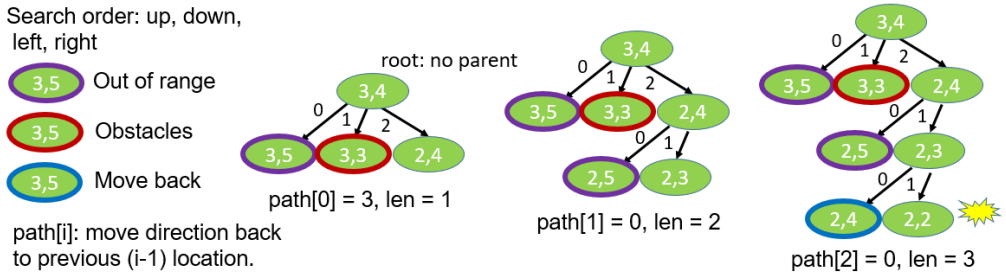


Figure 2. Search for the first path.

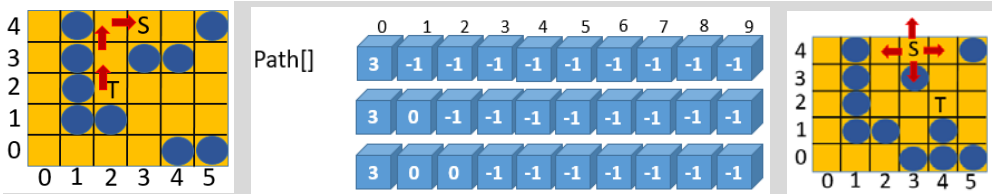


Figure 3. First path on the map; changes on the path array that stores the path of moves with back-tracing method; second routing example.

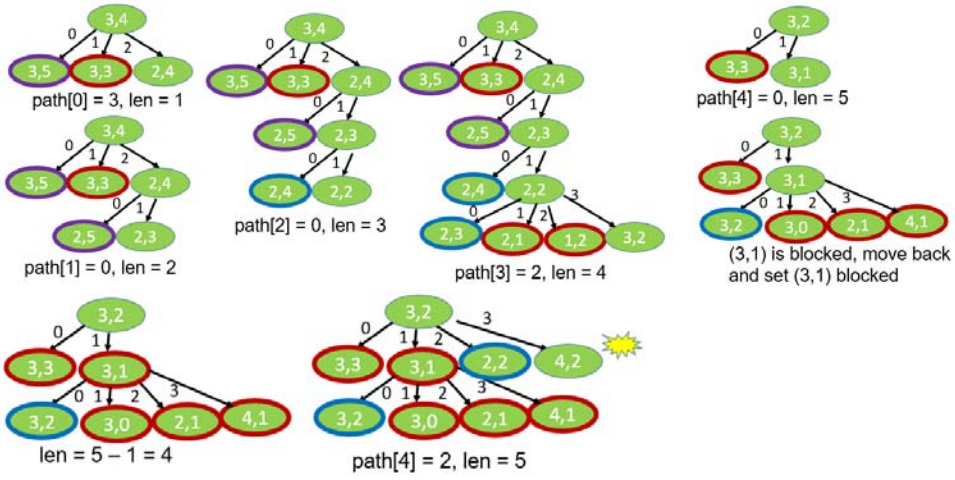


Figure 4. First path for the second routing example.

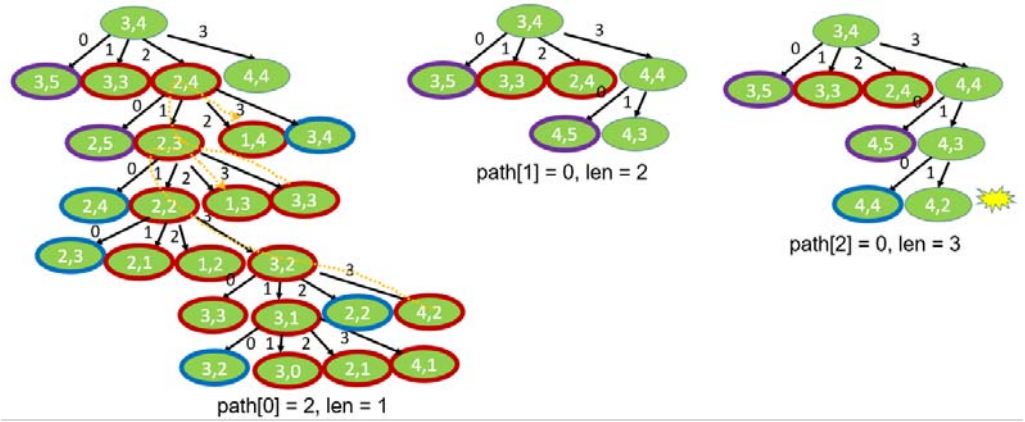


Figure 5. Move back to identify second path.

Policies to move back:
1. If current node has tried all moves, move back again.
2. If current node has not tried all moves, try the new move.

Ex. Move back from (4,2) to (3,2), current node (3,2) has tried all moves, so we move back again and thus back to (2,2). The same situation happens for current node (2,2), so we move back to (2,3). We find current node (2,3) only tried two moves, so we have to try the other two moves – left and right moves.

```
grid[MAX_R][MAX_C], path[MAX_PL];
len = 0; initialize path[] to -1;
while (len >= 0 )       {
    while (1)     {
        next_step = GetNextMove();
        if (next_step is feasible) {
            Move to next node by one step and store back-tracing direction to
            path array;
            If reach target report path and exit program;
        } else
            Exit inner loop to stop the operation of moving to next node;
    }
    Move backwards to previous node by one step
}
```

3. Problem 1

Routing problem is one famous optimization problem that has been applied in many practical areas such as network routing, robotic walking planning and integrated circuit (IC) routing problems. One fundamental problem is to identify a legal path connecting two points on a grid map that may contain existing obstacles on it. The legal path cannot overlap any obstacle.

Main features (75 pts):

(a) Read a test case from an input file and complete the function of moving to next node. Report the path you have explored on the screen.

(b) Complete your whole program to find a legal path connecting two points on a grid map.

Input format

The input file format is as follows.

6 5

The first line describes the column (6 for this case) and row numbers of the grid map. The row/column number ranges from 0 to (row_no – 1)/(col_no – 1) (0 to 5 and 0 to 4 for this case).

3 4 4 2

The second line describes two points to be connected. In this case, (3,4) is the first point while (4,2) is the second point.

11

The third line describes the number of existing obstacles.

1 4

1 3

1 2

1 1

2 1

3 3

3 0

4 1

4 0

5 4

5 0

The following lines describe the location of each obstacle in one line. You can use the number of obstacles to control the number of lines to read in your program.

Output format

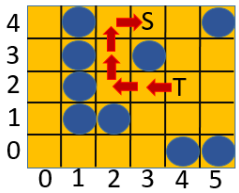
Write your path to a file called “route.txt”. Its format is as follows.

5

The first line describes the path length.

2 2 0 0 3

The second lines describe the moves of back-tracing from the target to the source.



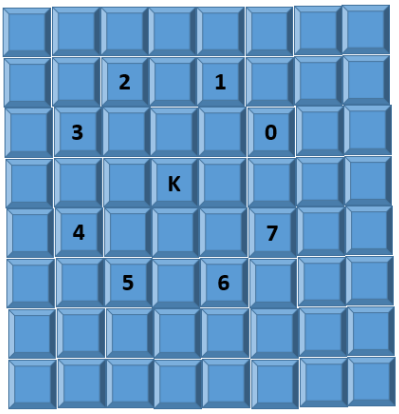
Additional feature

If you report the shortest path but not the first path you found, you can get 5 more points.

4. **Problem 2**

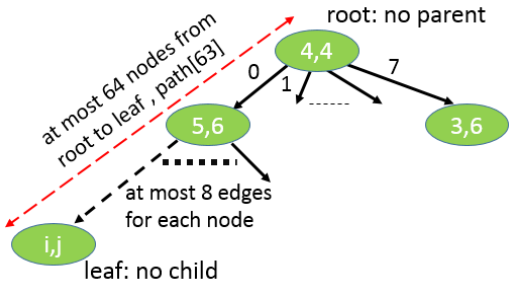
One of the more interesting puzzlers for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. The question is: Can the chess piece called the knight move around an empty chessboard and touch each of the 64 squares once and only once? The knight make L-shape moves (over two in one direction and then over one in a perpendicular direction). Thus from a square in the middle of an empty chessboard, the knight can make eight different moves (numbered 0 through 7) as shown in the following. Write a program to see how many steps your knight can make. You can define 8 types of moves for your knight and define movType for each type of move (in the following figure). Then the location after one move is:

```
curRow += verMove[movType];
curCol += horMove[movType];
```




```
horMove[0] = 2;    horMove[1] = 1;
horMove[2] = -1;   horMove[3] = -2;
horMove[4] = -2;   horMove[5] = -1;
horMove[6] = 1;    horMove[7] = 2;

verMove[0] = 1;    verMove[1] = 2;
verMove[2] = 2;    verMove[3] = 1;
verMove[4] = -1;   verMove[5] = -2;
verMove[6] = -2;   verMove[7] = -1;
```




```
    1   2   3   4   5   6   7   8
1   7  42  27  12   5  10  17  22
2  28  13   6   9  16  21   4  19
3  41   8  15  26  11  18  23  36
4  14  29   0  39  24  35  20   3
5   0  40  25   0   0   0  37  34
6  30   0   0   0  38  33   2   0
7   0   0   0  32   0   0   0   0
8   0  31   0   0   0   0   0   1

Total moves = 41
```

```
    1   2   3   4   5   6   7   8
1  27   6  25  20   9   4  15  12
2  24  21   8   5  14  11  18   3
3   7  26  23  10  19   2  13  16
4  22   0   0   1   0  17   0   0
5   0   0   0   0   0   0   0   0
6   0   0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0
8   0   0   0   0   0   0   0   0

Total moves = 26
```

Main feature (100 pts):

Use brute-force method to move your Knight on an empty chessboard and touch each of the 64 squares once and only once. (due in 11/8)