#### Jonathan Liao

#### CS 305 HW 4 Report

1a.  
Meets all specifications

1b. (.5 pt) Copy and paste your terminal window contents here from running solve on the four mazes given in the starter code.

MAZE 1

Solving using breadth-first search.

\*\*\*\*\*\*\*\*\*\*\*\*\*

Soo\*\*\*\*\* \*\* \*

\*\*oo\*\* \*\* \*

\*\*\*oo\*\*\*oo\* \*

\*\*oo\*\*o\*\*ooo\*

\*\*o\*\*ooooo\*\*\*

\*\*ooooo\*\*oo\*\*

\*\*\*\*\*\*\*\*\*\*F\*\*

Solving using depth-first search:

\*\*\*\*\*\*\*\*\*\*\*\*\*

Soo\*\*\*\*\* \*\* \*

\*\*oo\*\* \*\* \*

\*\*\*o \*\*\* \* \*

\*\*oo\*\* \*\* \*

\*\*o\*\*ooooo\*\*\*

\*\*ooooo\*\*oo\*\*

\*\*\*\*\*\*\*\*\*\*F\*\*

MAZE 2

(couldn't solve)

MAZE 3   
(couldn't solve)

MAZE 4

Solving using breadth-first search.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*o\* \*o\* \*ooooo\*

\*o\*\*\*o\*\*\*o\*\*\*o\*

So\*\*\*ooooo\*ooo\*

\*oo\*oo\*\*o\*\*o\*\*\*

\*\*o\*\*o\*\*\*\*\*o\*o\*

\*\*oooo\*\*oo\*o\*o\*

\*\*\*\*\*\*\*\*\*o\*o\*o\*

\* ooooooo\*

\*\*\*\*\*\*\*F\*\*\*\*\*\*\*

Solving using depth-first search:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \* \* \* \*ooooo\*

\* \*\*\* \*\*\*o\*\*\*o\*

So\*\*\*ooooo\*ooo\*

\*oo\*oo\*\*o\*\*o\*\*\*

\*\*o\*\*o\*\*\*\*\*o\* \*

\*\*oooo\*\* \*o\* \*

\*\*\*\*\*\*\*\*\* \*o\* \*

\*ooooooooooo \*

\*\*\*\*\*\*\*F\*\*\*\*\*\*\*

2a. Maze5.txt in package

2b. Maze6.txt in package

3. Suppose the following small maze is used for running solve:

\*\* \*\*

\* \*

\*\* \*

\*\*\* \*

with (0, 2) as the start and (2, 3) as the exit.

a. (1 pt) What is the order of coordinates explored using breadth-first search (you may ignore the coordinates that have walls)? *Hint, you can use the print\_coord function to see the order of exploration. Explore enqueing in the order of top, right, down, left (as written in the code).*

Start: (1,2) (1,3) (2,2) (1,1) (2,3)

b. (1 pt) What is the order of coordinates explored using depth-first search (you may ignore the coordinates that have walls)? *Explore pushing in the order of top, right, down, left.*

Start: (1,2) (1,1) (2,2) (1,3) (2,3)

c. (.5 pt) Which coords are left in the stack when depth-first search returns 1 for the maze in Q3? *Hint: you can do this by hand or print the stack before freeing it.*

(2,3) is all that’s left—that was queued by (1,3)

4a.

About 3 hours in total

4b.   
Understanding how queues and stacks can be used in these powerful implementations. They certainly stand out in use from the in class examples.

**Appendix A:** I verify that the code and this write-up were authored by me. I have documented the help I have received in comments in the code files. I have not distributed my code to anyone else except via this homework submission.

**Appendix B**:   
/\* author: Jonathan Liao

CS 305 HW 4

solve.c -- code to solve maze

\*/

#include <stdio.h>

#include <stdlib.h>

#include "maze.h"

#include "stack.h"

#include "queue.h"

int solve\_bfs(maze \* the\_maze) {

Queue q = initQueue();

enqueue(q, the\_maze->entry);

while(q != NULL) {

coord to\_explore = dequeue(q);

if(to\_explore.row == the\_maze->exit.row && to\_explore.col == the\_maze->exit.col) { //found exit case

print\_maze(the\_maze);

return 1;

}

else { //exit not found yet; look for adjacent routes

the\_maze->data[to\_explore.row][to\_explore.col] = 'o'; //mark that this part is explored

if(the\_maze->data[to\_explore.row - 1][to\_explore.col] == ' ' && (to\_explore.row - 1) >= 0 ) {

enqueue(q, \*make\_coord(to\_explore.row - 1, to\_explore.col) );

}

if(the\_maze->data[to\_explore.row][to\_explore.col + 1] == ' ' && (to\_explore.col + 1) < the\_maze->width ) {

enqueue(q, \*make\_coord(to\_explore.row, to\_explore.col + 1) );

}

if(the\_maze->data[to\_explore.row + 1][to\_explore.col] == ' ' && (to\_explore.row) < the\_maze->height ) {

enqueue(q, \*make\_coord(to\_explore.row + 1, to\_explore.col) );

}

if(the\_maze->data[to\_explore.row][to\_explore.col - 1] == ' ' && (to\_explore.col - 1) >= 0 ) {

enqueue(q, \*make\_coord(to\_explore.row, to\_explore.col - 1) );

}

}

}

//expeled from while loop. Means queue is empty now

print\_maze(the\_maze);

free(q);

return 0;

}

int solve\_dfs(maze \* the\_maze) {

Stack s = initStack();

push(s, the\_maze->entry);

while(s != NULL) {

coord to\_explore = pop(s);

if(to\_explore.row == the\_maze->exit.row && to\_explore.col == the\_maze->exit.col) { //found exit case

print\_maze(the\_maze);

return 1;

}

else { //exit not found yet; look for adjacent routes

the\_maze->data[to\_explore.row][to\_explore.col] = 'o'; //mark that this part is explored

if(the\_maze->data[to\_explore.row - 1][to\_explore.col] == ' ' && (to\_explore.row - 1) >= 0 ) {

push(s, \*make\_coord(to\_explore.row - 1, to\_explore.col) );

}

if(the\_maze->data[to\_explore.row][to\_explore.col + 1] == ' ' && (to\_explore.col + 1) < the\_maze->width ) {

push(s, \*make\_coord(to\_explore.row, to\_explore.col + 1) );

}

if(the\_maze->data[to\_explore.row + 1][to\_explore.col] == ' ' && (to\_explore.row) < the\_maze->height ) {

push(s, \*make\_coord(to\_explore.row + 1, to\_explore.col) );

}

if(the\_maze->data[to\_explore.row][to\_explore.col - 1] == ' ' && (to\_explore.col - 1) >= 0 ) {

push(s, \*make\_coord(to\_explore.row, to\_explore.col - 1) );

}

}

}

//expeled from while loop. Means stack is empty now

print\_maze(the\_maze);

free(s);

return 0;

}

void print\_maze(maze \* the\_maze) {

the\_maze->data[the\_maze->entry.row][the\_maze->entry.col] = 'S';

the\_maze->data[the\_maze->exit.row][the\_maze->exit.col] = 'F';

int i;

int j;

for(i = 0; i < the\_maze->height; i++) {

for(j = 0; j < the\_maze->width; j++) {

printf("%c", the\_maze->data[i][j]);

}

printf("\n"); //for some strange reason, the newline from the\_maze->data won't print. Had to add this line

}

return;

}

coord \* make\_coord(int r, int c) {

coord \* new\_coord = (coord \*) malloc(sizeof(coord));

new\_coord->row = r;

new\_coord->col = c;

return new\_coord;

}

void print\_coord(coord c) {

printf("(%d),(%d)\n", c.row, c.col);

return;

}