Assignment 4: Maze Solving using a Queue

Jake Gendreau

April 9, 2024

Contents

1 Program Design

1.1 Time Estimate

I estimate that this assignment will take me 4 - 6 hours to complete. The problem seems simple enough, and the logic is all provided. I estimate that adding a better pathfinding algorithm will take an additonal 2 - 4 hours.

1.2 Data Structures

1.2.1 Priority Queue

The queue will be implemented in a separate file through the use of `priorityQADT.h` and its corresponding implementation, `priorityQADT.cpp`. It uses a linked list under the hood. The queue will consist of structs called `cell`.

Cell struct

- `int x`
- `int y`
- `int h`

x and y can store the position of cells in the queue, but h is used to store the heuristic for the Greedy BeFS algorithm.

Queue functions:

- `enqueue()` - Adds an item to the queue based on it's h value. Lower ones are towards the front and higher ones are at the back.
- `dequeue()` - Returns and deletes the first item from the queue.
- `print()` - Prints the queue, front to back.
- `size()` - Returns the size of the queue.

1.3 Program

1.3.1 Maze Input

I will be using an agent model to solve this maze. The agent will be a cell.

- Take in maze through command line args.
- Store the width and height provided in the maze header.
- Create a 2D array using the width and height, then read in each character from the text file to the 2D array.
- Make a queue to add cells to.
- Find the start node, and set the agent to point to it.

- Repeat the following process recursively:
  - If the current cell is adjacent to the exit, stop.
  - Mark the current cell as visited.
  - Add unvisited neighbors to the north, east, south, and west to a queue based on their Manhattan distance to the goal.
  - Remove the next element from the queue and make it the current cell.
- Print the solved maze.

2 Program Log

2.1 Time Requirements

This program took me about 5 hours to complete. I was getting a bit confused on the dynamic arrays, but once I figured those out, it was pretty simple. Implementing the Greedy BeFS algorithm was also way easier than I thought it would be. I only had to rework the enqueue function and add a heuristic function to the program. It was essentially a drop-in replacement.

2.2 Things I encountered

- I forgot to properly place the bounds when searching for available unvisited cells, so I ran into some memory errors there that I found using GDB.
- I'm very happy that we were taught to use DATA_TYPE in header files instead of defined data types, because it made converting the queue from characters to cells.
- With dynamic arrays, my initial implementation was using C syntax because that's what I understood. The C++ version is much easier to read though. The C version does teach it better, I feel.

3 Source Code Files

3.1 BeFSSolver.cpp

Listing 1: BeFSSolver.cpp

```
/*
BeFSSolver.cpp
A program to solve mazes using a
Greedy Best First Search implementation
Jake Gendreau
April 9, 2024
*/

//boilerplate
#include <iostream>
#include <fstream>
#include "priorityQADT.cpp"

using namespace std;

//prototypes
void solveMaze(cell, char**, int, Queue&);
void printMaze(char**, int);
void addUnvisited(cell, char**, int, Queue&);

bool checkGoal(cell, char**, int);

int getDimension(string);
int getManhattanDist(int, int, char**, int);

char** getMaze(string, int);
cell findStart(char**, int);
cell findGoal(char**, int);

/*
start - S
goal - G
wall - #
blank - .
*/

int main(int argc, char* argv[])
{
    //check proper usage
    if(argc != 2)
    {
        cout << "USAGE: ./a.out <map.txt>. Exiting program..." << endl;
        exit(-1);
    }
```

```
45
46      string fileName = argv[1];
47
48      //init array
49      int dimension = getDimension(fileName);
50      char** maze = getMaze(fileName, dimension);
51
52      //make the agent
53      cell agent = findStart(maze, dimension);
54
55      //make the queue
56      Queue queue = Queue();
57
58      //solve and print the maze
59      solveMaze(agent, maze, dimension, queue);
60      printMaze(maze, dimension);
61  }
62
63  void solveMaze(cell agent, char** maze, int dimension, Queue &queue)
64  {
65      //check for goal cell
66      if(checkGoal(agent, maze, dimension))
67      {
68          cout << "Found solution" << endl;
69          return;
70      }
71
72      //add all unvisited neighbors
73      addUnvisited(agent, maze, dimension, queue);
74
75      //remove next element and make it current cell
76      cell newAgent = queue.dequeue();
77
78      solveMaze(newAgent, maze, dimension, queue);
79  }
80
81  void printMaze(char** maze, int dimension)
82  {
83      //print the maze
84      for(int i = 0; i < dimension; i++)
85      {
86          for(int j = 0; j < dimension; j++)
87          {
88              cout << maze[i][j];
89          }
90          cout << endl;
91      }
92  }
93
94  void addUnvisited(cell agent, char** maze, int dimension, Queue &queue)
```

```
{
    int x = agent.x;
    int y = agent.y;

    //check south
    if(y + 1 < dimension && maze[y + 1][x] == '.' && maze[y + 1][x] == '.')
    {
        maze[y + 1][x] = 'v';
        queue.enqueue(y + 1, x, getManhattanDist(x, y + 1, maze, dimension));
    }

    //check east
    if(x + 1 < dimension && maze[y][x + 1] == '.' && maze[y][x + 1] == '.')
    {
        maze[y][x + 1] = '>';
        queue.enqueue(y, x + 1, getManhattanDist(x + 1, y, maze, dimension));
    }

    //check north
    if(y - 1 >= 0 && maze[y - 1][x] == '.' && maze[y - 1][x] == '.')
    {
        maze[y - 1][x] = '^';
        queue.enqueue(y - 1, x, getManhattanDist(x, y - 1, maze, dimension));
    }

    //check west
    if(x - 1 >= 0 && maze[y][x - 1] == '.' && maze[y][x - 1] == '.')
    {
        maze[y][x - 1] = '<';
        queue.enqueue(y, x - 1, getManhattanDist(x - 1, y, maze, dimension));
    }
}

bool checkGoal(cell agent, char** maze, int dimension)
{
    int x = agent.x;
    int y = agent.y;

    //check north
    if(y - 1 >= 0 && maze[y - 1][x] == 'G')
        return true;

    //check south
    if(y + 1 < dimension && maze[y + 1][x] == 'G')
        return true;

    //check west
    if(x - 1 >= 0 && maze[y][x - 1] == 'G')
        return true;

```

```
145     //check east
146     if(x + 1 < dimension && maze[y][x + 1] == 'G')
147         return true;
148 
149     return false;
150 }
151 
152 int getManhattanDist(int x, int y, char** maze, int dimension)
153 {
154     //get manhattan distance to goal
155     cell start = findGoal(maze, dimension);
156     return(abs(x - start.x) + abs(y - start.y));
157 }
158 
159 int getDimension(string fileName)
160 {
161     //open file
162     string word;
163     fstream file;
164 
165     file.open(fileName);
166 
167     //check that file is valid and open
168     if(!file.is_open())
169     {
170         cout << "ERROR OPENING FILE: " << fileName << ". Exiting program..." << endl;
171         exit(-1);
172     }
173 
174     //extract just the first word
175     file >> word;
176 
177     //close the file
178     file.close();
179 
180     //return the dimension
181     return stoi(word);
182 }
183 
184 char** getMaze(string fileName, int dimension)
185 {
186     //open file
187     string line;
188     fstream file;
189 
190     file.open(fileName);
191 
192     //check that file is valid and open
193     if(!file.is_open())
194     {
```

```
195         cout << "ERROR OPENING FILE: " << fileName << ". Exiting program..." << endl;
196         exit(-1);
197     }
198 
199     //adjust file pointer to be start of the maze
200     getline(file, line);
201 
202     //start 2D array
203     char** maze = new char*[dimension];
204 
205     //read in the map
206     for(int i = 0; i < dimension; i++)
207     {
208         //define row of 2D array
209         maze[i] = new char[dimension];
210         //get current line
211         getline(file, line);
212         for(int j = 0; j < dimension; j++)
213         {
214             //write the chars of the current line into the array
215             maze[i][j] = line[j];
216         }
217     }
218 
219     //close the file
220     file.close();
221 
222     return maze;
223 }
224 
225 cell findStart(char** maze, int dimension)
226 {
227     //go through whole maze, finding start cell
228     for(int i = 0; i < dimension; i++)
229     {
230         for(int j = 0; j < dimension; j++)
231         {
232             if(maze[i][j] == 'S')
233             {
234                 cell c = cell();
235 
236                 c.x = j;
237                 c.y = i;
238 
239                 return c;
240             }
241         }
242     }
243 
244     cout << "ERROR: COULDN'T FIND START" << endl;
```

```
245     exit(-1);
246 }
247
248 cell findGoal(char** maze, int dimension)
249 {
250     //go through whole maze, finding start cell
251     for(int i = 0; i < dimension; i++)
252     {
253         for(int j = 0; j < dimension; j++)
254         {
255             if(maze[i][j] == 'G')
256             {
257                 cell c = cell();
258
259                 c.x = j;
260                 c.y = i;
261
262                 return c;
263             }
264         }
265     }
266
267     cout << "ERROR: COULDN'T FIND GOAL" << endl;
268     exit(-1);
269 }
```

3.2 priorityQADT.cpp

Listing 2: priorityQADT.cpp

```
/*
priorityQADT.cpp
Implementation of a priority queue using a linked list
Jake Gendreau
April 9, 2024
*/

//dependencies
#include <iostream>
#include "priorityQADT.h"

using namespace std;

//enqueue() - adds an item to the queue
void Queue::enqueue(int y, int x, int h)
{
    cell data = cell();

    data.x = x;
    data.y = y;
    data.h = h;

    nodePtr p = new node();

    //error check
    if (p == NULL)
    {
        cout << "ERROR: FAILED TO ALLOCATE NEW NODE" << endl;
        exit(-1);
    }

    p -> data = data;
    p -> next = NULL;

    //handle empty list
    if(head == NULL)
    {
        head = p;
        count++;
        return;
    }

    //insertion at the beginning of the list if h < head -> data.h
    if(h <= head -> data.h)
    {
        p -> next = head;
```

```
47          head = p;
48          count++;
49          return;
50      }
51
52      //find correct insert spot
53      nodePtr n = head;
54      while(n -> next != NULL && h > n -> next -> data.h)
55      {
56          n = n->next;
57      }
58
59      //insert the new node after n
60      p -> next = n -> next;
61      n -> next = p;
62
63      //increment count
64      count++;
65  }
66
67
68  //dequeue() - removes and returns the first item from the queue
69  DATA_TYPE Queue::dequeue()
70  {
71      //error check
72      if(size() <= 0)
73      {
74          cout << "ERROR: DEQUEING EMPTY QUEUE" << endl;
75          exit(-1);
76      }
77
78      DATA_TYPE returnVal = head -> data;
79
80      //move head and delete old head
81      nodePtr n = head;
82      head = head -> next;
83
84      n -> next = NULL;
85      delete n;
86
87      //decrement counter
88      count--;
89
90      return returnVal;
91  }
92
93  //print() - prints the queue
94  void Queue::print()
95  {
96      nodePtr n = head;
```

```
97
98      //print data of each node
99      while(n != NULL)
100     {
101         cout << n -> data.h << endl;
102         n = n -> next;
103     }
104 }
105
106 //size() - returns the size of the queue
107 int Queue::size()
108 {
109     return(count);
110 }
```

3.3 priorityQADT.h

Listing 3: priotityQADT.h

```
/*
priorityQADT.h
priority queue header using a linked list
Jake Gendreau
April 9, 2024
*/

#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>

struct cell
{
    int x;
    int y;
    int h;
};

typedef cell DATA_TYPE;

class Queue
{
private:
    struct node
    {
        DATA_TYPE data;
        node* next;
    };
    typedef node* nodePtr;

    nodePtr head;

    int count;

public:
    Queue()
    {
        head = NULL;
        count = 0;
    }

    ~Queue()
    {
        nodePtr p = head;
        nodePtr n;
```

```
47
48          while(p != NULL)
49          {
50              n = p;
51              p = p -> next;
52              delete n;
53          }
54      }
55
56      void enqueue(int y, int x, int h);
57
58      DATA_TYPE dequeue();
59
60      void print();
61
62      int size();
63  };
64
65  #endif // QUEUE_H
```

4 Input Files

4.1 maze1.txt

```
10 10
S##.##.##.
..........
#...#.##.#
.#.#...#.#
.....#....
.#.###.##.
.#.#...#.#
.#.####...
........#.
##..##.##G
```

4.2 maze2.txt

```
20 20
S...###..#####..#..#
.##...#....#......#.
....#...#.#..#.###..
##.....##...##.####.
#..#..#...#.....#...
..##.......#.##....#
..###.##..#.##..#..#
#.......#.##...#....
#...###....#.#.###..
..#...#.#.....#...#.
.##..#..#..##.#.#.#.
.#...#.#.#..#......#
..##.#..#####.##.##.
.#.......####..###..
....####.#....#...#.
##.#.#....##...###..
.#.#...#.#.#.#.....#
.#.##..#.....#.###.#
....#..##.###...##.#
#..###...G#####....#
```

4.3 maze3.txt

```
40 40
S##......##.....###...#....#...S....#...
....####...###.##.##.#..##...######...#.
.#....#......##.......#.....##....#.###.
..#......##.....##..#...#..#..##.######.
.#.....##..#..#...####.###.###....####..
..#..#.##.#.#..##.....##.....##.#.####.#
.#...#.....#.##.#####....###.....#.....#
....#.##.#..#.##..#.###...##..##....####
.###..#....#...#.#.#.#..#.####.......###
..###.#.##..#...#.##.##.#####....##.###.
#......###.#..##..##....##.#####..###.#.
.###.#...#.#.##.#..#........##.....#.#..
.###.##.......#..#.###..######.###....#.
..##.#.#...##..####.###.....##.####.#..#
##...###.......###..###.##.##..##..#.##.
..##...#.#.##...#..##......#..#####..##.
.###.##..#..##..#.#.....#....#.#..#.....
...,.#..#.##.##....##.##.#####.###.##..#
####....##..###.#.....###..##......####.
..#....#..#.#...##.##.##.##....###...#..
....##.....#..#..#..#.#..##.###...###.#.
.###...##.#..##..##........#...#..#..##.
..#.#...#.##.##.#..#..#..#...#..#..#.#.#
.###.##...........#..#..####..#.##.####.
..###....#..##.#....##..####...#..###..#
.##..###..#####.#.#...#..##........#..#.
.#.##....#..##...#..##.#.....#...#..###.
..#.###...#...#......###.##.#.#...#.#...
.###.##.#..#.##.#..##..###....#..#...###
.##.##.#.#..#..#...##.#..#...####.##....
.####.#.##...##.##...#.#...#.##..###.##.
#.#..#..#.##...##..##.#...###.###.#.##..
...#..####...#.......#.#...###.#....#...
.##....##.##..#.###.###.#.#...####.#...#
...##...##...#...#.....#..##..##...#..##
##......#.##...###..#....#...######.#...
.....##........##..#.##....#...#.....##.
.##..##.#..#..########.#.############....
.#.#.#....##...##.....##....####.....##.
G...###.###........######........#######
```

5 Program output

'S' represents start, 'G' represents goal, '#' represents a wall, '.' represents an unvisited space, and the arrows indicate the direction of travel in visited spaces. Notice how the solution is found with considerably less wasted traversal when compared to the given algorithm.

Listing 4: out.txt

```
./a.out ../mazes/maze1.txt
Found solution
S##^##^##^
v>>>>>>>>>
#vvv#v##v#
.#^#<<^#v#
.<<<v#<<v>
.#v###v##v
.#v#<<v#^#
.#v####^>>
.<v>>>>>#v
##vv##v##G

./a.out ../mazes/maze2.txt
Found solution
S>>>###^^#####..#..#
v##v>>#^>>>#......#.
...v#v>>#v#..#.###..
##...vv##v>.##.####.
#..#..#.<v#.....#...
..##....<v>#.##....#
..###.##<v#.##..#..#
#......^#v##...#....
#...###<<v>#.#.###..
..#...#v#v>>>.#...#.
.##..#<v#v>##.#.#.#.
.#...#v#.#v>#......#
..##.#v>#####.##.##.
.#....vv>####..###..
....####v#....#...#.
##.#.#.<v>##...###..
.#.#...#v#.#.#.....#
.#.##..#v>>..#.###.#
....#..##v###...##.#
#..###...G#####....#

./a.out ../mazes/maze3.txt
Found solution
S##......##.....###...#....#...S....#...
v>..####...###.##.##.#..##...######...#.
v#....#......##.......#.....##....#.###.
```

```
v>#......##.....##..#...#..#..##.######.
v#.....##..#..#...####.###.###....####..
v>#..#.##.#.#..##.....##.....##.#.####.#
v#...#.....#.##.#####....###.....#.....#
v>..#.##.#..#.##..#.###...##..##....####
v###..#....#...#.#.#..#.####.......###
v>###.#.##..#...#.##.##.#####....##.###.
#v>>>>.###.#..##..##....##.#####..###.#.
 .###v#...#.#.##.#..#........##.....#.#..
 .###v##.......#..#.###..######.###....#.
 ..##v#.#...##..####.###.....##.####.#..#
##<<v###.......###..###.##.##..##..#.##.
 ..##v>.#.#.##...#..##......#..#####..##.
 .###v##..#..##..#.#.....#....#.#..#.....
 ...,v#..#.##.##....##.##.#####.###.##..#
####v>^.##..###.#.....###..##......####.
^^#<v>>#..#.#...##.##.##.##....###...#..
<<<v##v>...#..#..#..#.#..##.###...###.#.
v###<<v##.#..##..##........#...#..#..##.
v>#.#vv>#.##.##.#..#..#..#...#..#..#.#.#
v###.##v>.........#..#..####..#.##.####.
v>###<<v>#..##.#....##..####...#..###..#
v##..###v>#####.#.#...#..##........#..#.
v#.##<<<v#..##...#..##.#.....#...#..###.
v>#.###vv>#...#......###.##.#.#...#.#...
v###.##v#v>#.##.#..##..###....#..#...###
v##.##.#.#v>#..#...##.#..#...####.##....
v####.#.##v>>##.##...#.#...#.##..###.##.
#.#..#..#.##v>.##..##.#...###.###.#.##..
 ...#..####<<v#.......#.#...###.#....#...
 .##....##.##v>#.###.###.#.#...####.#...#
 ..^##^^^##<<v#...#.....#..##..##...#..##
##<<<<<^#^##v>.###..#....#...######.#...
<<vvv##<<<<<v>.##..#.##....#...#.....##.
v##..##v#vv#v.########.#.###########....
v#.#.#<v>>##...##.....##....####.....##.
G...###v###........######........#######
```