Assignment 1

Assignment Report

This report has been designed to demonstrate the work done in this assignment and discover the graph-based path finding algorithms, and how grids are represented.

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

In this scenario, an agent in the 2nd cell requests to move to the 25th cell.

Rules of this scenario are

- Agent can only move along rows or columns
- In one action, the agent can move only to an adjacent cell

I made up some assumptions, such as

- There are no any obstacles in the paths

First we should convert this grid pattern into a graph representation because searching algorithms can be easily done in graphs. When working with a square grid, we need to make a list of nodes. Here's the code for a 5 X 5 rectangle.

```
//Lets create nodes as given as an example in the article
      Node[][] grid= new Node[5][5];
          int k=1;
           for(int i=0;i<5;i++){
               for(int j=0;j<5;j++){
                   String temp=Integer.toString(k++);
                   grid[i][j]=new Node(temp);
               }
           }
```

(Graph.java given on page 9)

A graph is a common data structure consisting of a set of nodes containing data and edges connecting the nodes to each other. Path finding allows us to determine if and how we can reach one node from another. Additionally we can consider not only if two nodes are connected but also assign some form of cost to traveling between them. We can then search for not only the shortest path but also the cheapest, fastest, or safest.

Breadth First Search

Breadth-First Search (BFS), is a way to search a graph and it just uses a queue to enqueue and dequeue nodes to/from. Also here we have an unweighted graph.

The procedures of a BFS algorithm are given below

- Visit start cell and put into a FIFO queue.
- Repeatedly remove a cell from the queue, visit its unvisited adjacent cells, and put newly visited cells into the queue.
- Each visited cell is put on (and so removed from) the queue exactly once.
- When a vertex is removed from the queue, we examine its adjacent vertices.

(BFS.java given on page 10) and bfsSearch() [Lines (25-47)] method in BFS class do these steps given above.

Depth first search

Depth first search is another way of traversing graphs, which is closely related to preorder traversal of a tree. Here DFS uses a stack to put and pop nodes to/from. The strategy of the DFS is to search “deeper” in the graph whenever possible.

DFS.java given on page 11 and dfsSearch() [Lines 24-55] method contains the algorithm of depth search algorithm.

In the contrast, DFS will run to each leaf nodes and find out the path when traverse node along that path. .BFS, uses more memory, traverse all nodes. DFS, uses less memory, might be slightly faster (Don't necessarily have to traverse all nodes). Both of them are using exhaust path finding search.

A Star Algorithm

We have a graph represented as a grid. Each cell is connected to the four cells immediately surrounding it, diagonal movement is not allowed. We want to use A* to find a path between two cells on this grid. For our heuristic we can compute the exact distance we need to travel, assuming we do not hit an obstacle.

A* works by maintaining a pair of lists, one containing locations on the grid map which may be a next step in the path (called the ‘**open**’ list) and one containing locations that have already been analyzed (called the ‘**closed**’ list).

Pseudo code:

```
OPEN = priority queue containing START
CLOSED = empty set
       while lowest rank in OPEN is not the GOAL:
              current = remove lowest rank item from OPEN
              add current to CLOSED
              for neighbors of current:
                 cost = g(current) + movementcost(current, neighbor)
                 if neighbor in OPEN and cost less than g(neighbor):
                   remove neighbor from OPEN, because new path is better
                 if neighbor in CLOSED and cost less than g(neighbor): **
                   remove neighbor from CLOSED
                 if neighbor not in OPEN and neighbor not in CLOSED:
                     set g(neighbor) to cost
                     add neighbor to OPEN
                     set priority queue rank to g(neighbor) + h(neighbor)
                     set neighbor's parent to current

reconstruct reverse path from goal to start
by following parent pointers
```

This Pseudo code taken from this site [1]

The A* algorithm works like this: [2]

1. At initialization add the starting location to the open list and empty the closed list
2. While there are still more possible next steps in the open list and we haven't found the target:
    1. Select the most likely next step (based on both the heuristic and path costs)
    2. Remove it from the open list and add it to the closed
    3. Consider each neighbor of the step. For each neighbor:
        1. Calculate the path cost of reaching the neighbor
        2. If the cost is less than the cost known for this location then remove it from the open or closed lists (since we've now found a better route)
        3. If the location isn't in either the open or closed list then record the costs for the location and add it to the open list (this means it'll be considered in the next search). Record how we got to this location

The loop ends when we either find a route to the destination or we run out of steps. If a route is found we backtrack up the records of how we reached each location to determine the path.

AStar.java given on page 12

Full code of this program is available in my github repository

(https://github.com/Linganesan/PathFinder)

- ❖ Main class of this program (PathFinder.java) given on page 6
- ❖ Screenshots given on the next page

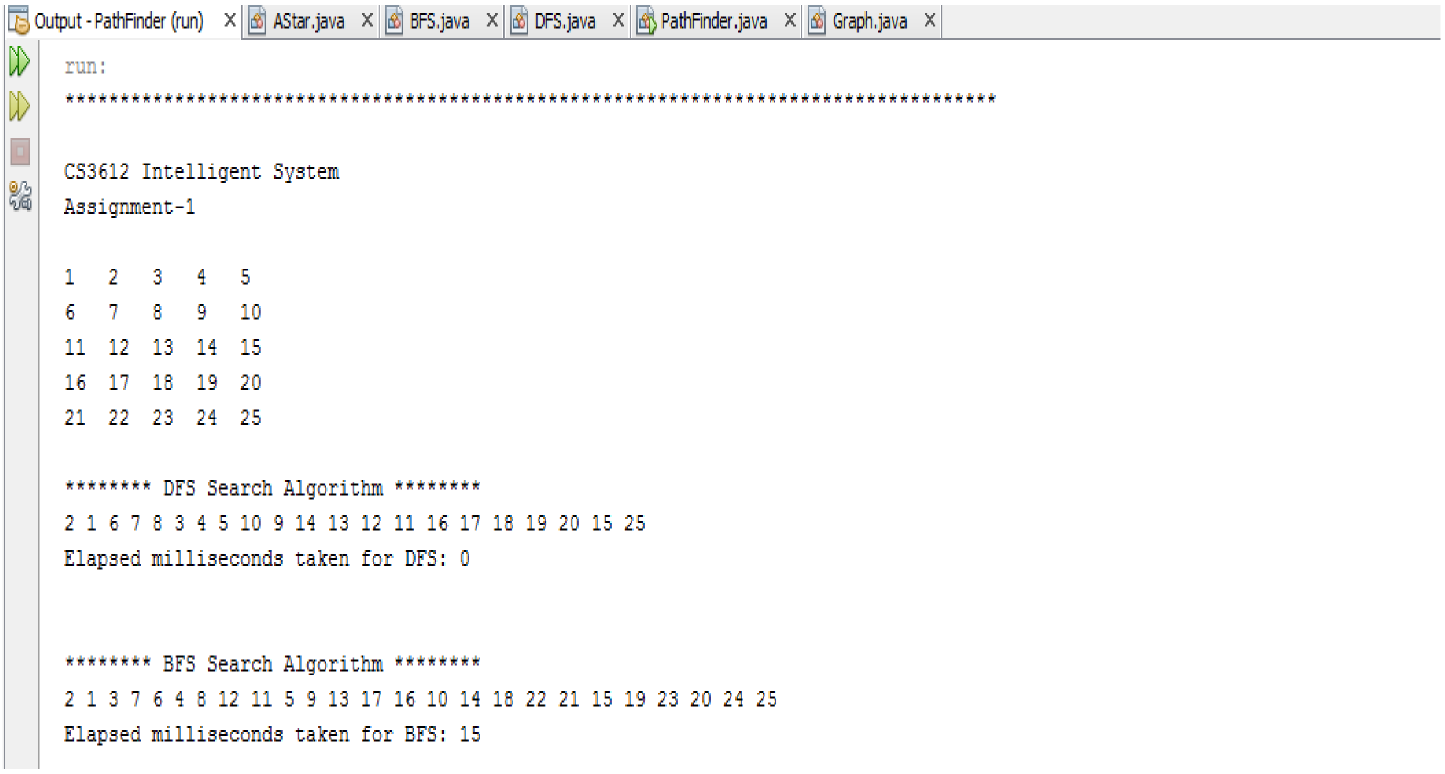
References:

[1] http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html

[2] http://wiki.gamegardens.com/Path_Finding_Tutorial

[3] Here is a nice tutorial on A* algorithm (https://www.youtube.com/watch?v=-L-WgKMFuhE)

Screenshots



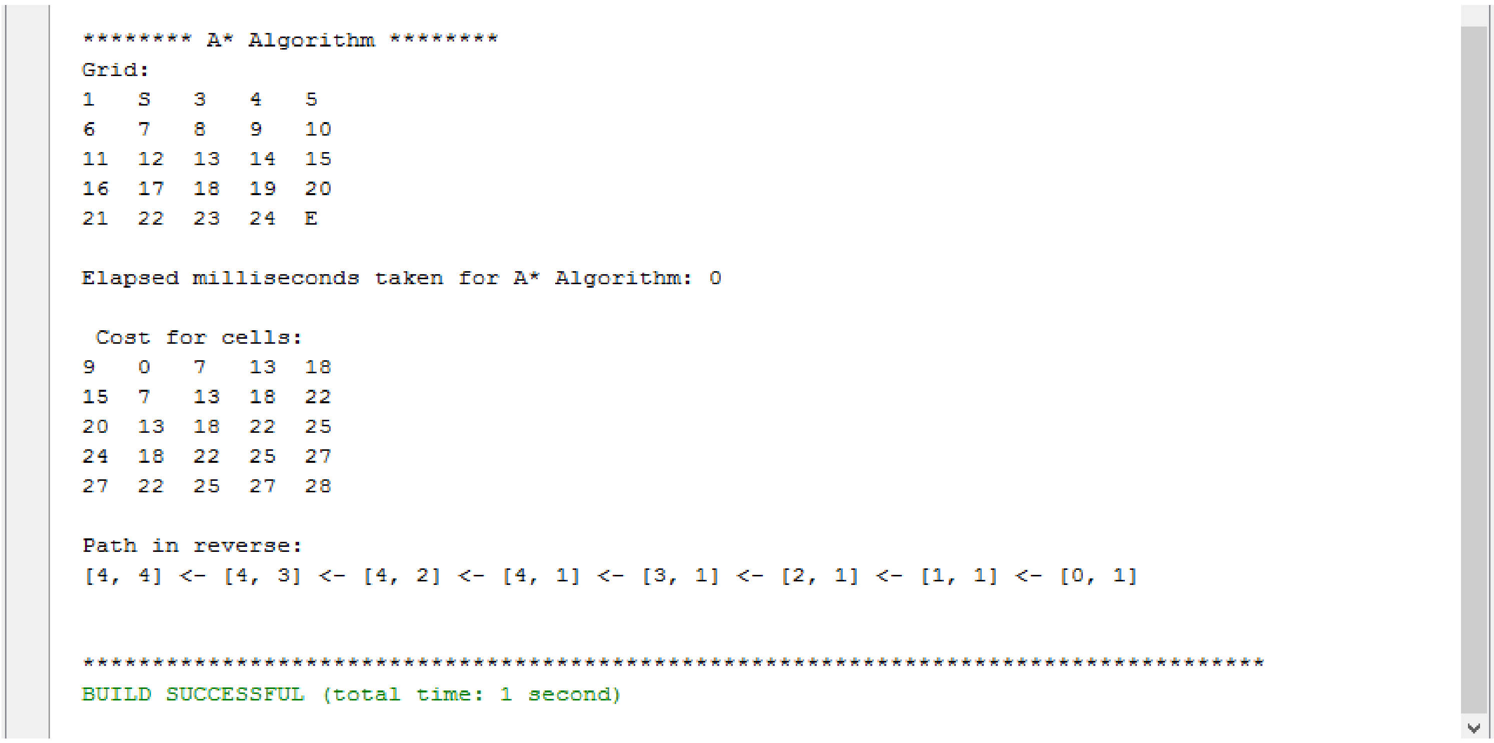
```
run:
*************************************************************************************

CS3612 Intelligent System
Assignment-1

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

******** DFS Search Algorithm ********
2 1 6 7 8 3 4 5 10 9 14 13 12 11 16 17 18 19 20 15 25
Elapsed milliseconds taken for DFS: 0


******** BFS Search Algorithm ********
2 1 3 7 6 4 8 12 11 5 9 13 17 16 10 14 18 22 21 15 19 23 20 24 25
Elapsed milliseconds taken for BFS: 15
```



```
******** A* Algorithm ********
Grid:
1    S    3    4    5
6    7    8    9    10
11   12   13   14   15
16   17   18   19   20
21   22   23   24   E

Elapsed milliseconds taken for A* Algorithm: 0

 Cost for cells:
9    0    7    13   18
15   7    13   18   22
20   13   18   22   25
24   18   22   25   27
27   22   25   27   28

Path in reverse:
[4, 4] <- [4, 3] <- [4, 2] <- [4, 1] <- [3, 1] <- [2, 1] <- [1, 1] <- [0, 1]


*************************************************************************************
BUILD SUCCESSFUL (total time: 1 second)
```

```
public class PathFinder {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args){

  //Lets create nodes as given as an example in the article
        Node[][] grid= new Node[5][5];
                int k=1;
                 for(int i=0;i<5;i++){
                     for(int j=0;j<5;j++){
                         String temp=Integer.toString(k++);
                         grid[i][j]=new Node(temp);
                     }
                 }


  //Create the graph, add nodes, create edges between nodes
        Graph g=new Graph();
        for(int i=0;i<5;i++){
                     for(int j=0;j<5;j++){
                          g.addNode(grid[i][j]);
                     }
                }

  //add 2 as the starting point of the graph
        g.setRootNode(grid[0][1]);

                for(int i=0;i<5;i++){
                     for(int j=0;j<5;j++){

                        if(i-1>=0)
                        g.connectNode(grid[i][j],grid[i-1][j]);

                        if(i+1<5)
                        g.connectNode(grid[i][j],grid[i+1][j]);

                        if(j-1>=0)
                        g.connectNode(grid[i][j],grid[i][j-1]);

                        if(j+1<5)
                        g.connectNode(grid[i][j],grid[i][j+1]);
                     }
                  }
  //Run the program
System.out.println("*****************************************************");

                System.out.println();

                System.out.println("CS3612 Intelligent System ");
                System.out.println("Assignment-1");

                System.out.println();
  //Print the Grid
                for(int i=0;i<5;++i){
                    for(int j=0;j<5;++j){
                        if(grid[i][j]!=null)System.out.printf("%-3d ",
Integer.parseInt(grid[i][j].label));

                     }
                    System.out.println();
                }
                System.out.println();



               //Run DFS algorithm

                long lStartTime = System.currentTimeMillis();
        System.out.println("******** DFS Search Algorithm ********");
        DFS dfs = new DFS(g);
                dfs.dfsSearch();
        System.out.println();
                long lEndTime = System.currentTimeMillis();
                long difference = lEndTime - lStartTime;
                System.out.println("Elapsed milliseconds taken for DFS: " +
difference);
                System.out.println();


                //Run BFS algorithm

                lStartTime = System.currentTimeMillis();
        System.out.println("\n******** BFS Search Algorithm ********");
        BFS bfs = new BFS(g);
                bfs.bfsSearch();
                lEndTime = System.currentTimeMillis();
                difference = lEndTime - lStartTime;
                System.out.println();
                System.out.println("Elapsed milliseconds taken for BFS: " +
difference);

                //Run A Star Algorithm

                System.out.println("\n******** A* Algorithm ******** ");
                AStar astar =new AStar();
                astar.test(5, 5, 0, 1, 4, 4);
                System.out.println();

                System.out.println();

System.out.println("*****************************************************");

    }

}
```

```
public class Graph{
    public Node rootNode;
    public ArrayList<Node> nodes=new ArrayList<>();
    public int[][] adjMatrix;//Edges will be represented as adjacency Matrix
    int size;
    public void setRootNode(Node n){
        this.rootNode=n;
    }
    public Node getRootNode(){
        return this.rootNode;
    }
    public void addNode(Node n){
        nodes.add(n);
    }
   //This method will be called to make connect two nodes
    public void connectNode(Node start,Node end){
        if(adjMatrix==null)
        {
            size=nodes.size();
            adjMatrix=new int[size][size];
        }

        int startIndex=nodes.indexOf(start);
        int endIndex=nodes.indexOf(end);
        adjMatrix[startIndex][endIndex]=1;
        adjMatrix[endIndex][startIndex]=1;
    }
    public Node getUnvisitedChildNode(Node n){

        int index=nodes.indexOf(n);
        int j=0;
        while(j<size)
        {
            if(adjMatrix[index][j]==1 && ((Node)nodes.get(j)).visited==false)
            {
                return (Node)nodes.get(j);
            }
            j++;
        }
        return null;
    }
    //Utility methods for clearing visited property of node
    public void clearNodes(){
        int i=0;
        while(i<size)
        {
            Node n=(Node)nodes.get(i);
            n.visited=false;
            i++;
        }
    }
    //Utility methods for printing the node's label
    public void printNode(Node n){
        System.out.print(n.label+" ");
    }
}
```

```java
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package pathfinder;

import java.util.LinkedList;
import java.util.Queue;

/**
 *
 * @author linganesan
 */
public class BFS {

     Graph g;


     BFS(Graph g){
        this.g=(Graph)g;
     }

    public void bfsSearch()
    {

        //BFS uses Queue data structure

        Queue q=new LinkedList();
        q.add(g.rootNode);
        g.printNode(g.rootNode);
        g.rootNode.visited=true;
        while(!q.isEmpty())
        {
            Node n=(Node)q.remove();
            Node child=null;
            while((child=g.getUnvisitedChildNode(n))!=null)
            {
                child.visited=true;
                g.printNode(child);
                q.add(child);
            }
        }
        //Clear visited property of nodes
        g.clearNodes();
    }
}
```

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package pathfinder;

import java.util.Stack;

/**
 *
 * @author linganesan
 */
public class DFS {

     Graph g;

     public DFS(Graph g){
        this.g=(Graph)g;
     }


    public void dfsSearch()
    {
        //DFS uses Stack data structure
        Stack s=new Stack();
        s.push(g.rootNode);
        g.rootNode.visited=true;
        g.printNode(g.rootNode);
        while(!s.isEmpty())
        {

            Node n=(Node)s.peek();
            Node child=g.getUnvisitedChildNode(n);

            if(child!=null)
            {
                                child.visited=true;
                g.printNode(child);
                s.push(child);
                                if(child.label.equals("25")){
                        break;
                        }
            }
            else
            {
                s.pop();
            }


        }
        //Clear visited property of nodes
        g.clearNodes();
    }

}
```

```

package pathfinder;

   import java.util.*;

public class AStar {

    public static final int COST = 1;

    static class Cell{
        int heuristicCost = 0; //Heuristic cost
        int finalCost = 0; //G+H
        int i, j;
        Cell parent;

        Cell(int i, int j){
            this.i = i;
            this.j = j;
        }

        @Override
        public String toString(){
            return "["+this.i+", "+this.j+"]";
        }
    }

    //Blocked cells are just null Cell values in grid
    static Cell [][] grid = new Cell[5][5];

    static PriorityQueue<Cell> open;

    static boolean closed[][];
    static int startI, startJ;
    static int endI, endJ;

    public static void setStartCell(int i, int j){
        startI = i;
        startJ = j;
    }

    public static void setEndCell(int i, int j){
        endI = i;
        endJ = j;
    }



   static void checkAndUpdateCost(Cell current, Cell t, int cost){
        if(t == null || closed[t.i][t.j])return;
        int final_cost = t.heuristicCost+cost;

        boolean inOpen = open.contains(t);
        if(!inOpen || final_cost<t.finalCost){
            t.finalCost = final_cost;
            t.parent = current;
            if(!inOpen)open.add(t);
        }
```

```java
    }

    public static void AStar(){

        //add the start location to open list.
        open.add(grid[startI][startJ]);

        Cell current;

        while(true){
            current = open.poll();
            if(current==null)break;
            closed[current.i][current.j]=true;

            if(current.equals(grid[endI][endJ])){
                return;
            }

            Cell t;
            if(current.i-1>=0){
                t = grid[current.i-1][current.j];
                checkAndUpdateCost(current, t, current.finalCost+COST);
            }

            if(current.j-1>=0){
                t = grid[current.i][current.j-1];
                checkAndUpdateCost(current, t, current.finalCost+COST);
            }

            if(current.j+1<grid[0].length){
                t = grid[current.i][current.j+1];
                checkAndUpdateCost(current, t, current.finalCost+COST);
            }

            if(current.i+1<grid.length){
                t = grid[current.i+1][current.j];
                checkAndUpdateCost(current, t, current.finalCost+COST);
            }
        }

    }

    public static void test(int x, int y, int si, int sj, int ei, int ej){
          //Reset
           grid = new Cell[x][y];
           closed = new boolean[x][y];
           open = new PriorityQueue<>((Object o1, Object o2) -> {
                Cell c1 = (Cell)o1;
                Cell c2 = (Cell)o2;

                return c1.finalCost<c2.finalCost?-1:
                        c1.finalCost>c2.finalCost?1:0;
            });
           //Set start position
           setStartCell(si, sj);  //Setting to 0,0 by default. Will be useful
for the UI part
```

```
            //Set End Location
            setEndCell(ei, ej);

            for(int i=0;i<x;++i){
               for(int j=0;j<y;++j){
                   grid[i][j] = new Cell(i, j);
                   grid[i][j].heuristicCost = Math.abs(i-endI)+Math.abs(j-
endJ);
//                   System.out.print(grid[i][j].heuristicCost+" ");
               }
//               System.out.println();
            }
            grid[si][sj].finalCost = 0;


            //Display initial map
            int k=1;
            System.out.println("Grid: ");
             for(int i=0;i<x;++i){
                 for(int j=0;j<y;++j){
                    if(i==si&&j==sj){
                        k++;
                        System.out.print("S   ");} //Source
                    else if(i==ei && j==ej){
                        k++;
                        System.out.print("E  "); } //Destination
                    else if(grid[i][j]!=null){
                     System.out.printf("%-3d ", k++);}

                 }
                 System.out.println();
             }
             System.out.println();

              long lStartTime = System.currentTimeMillis();

             AStar();

             long lEndTime = System.currentTimeMillis();

             long difference = lEndTime - lStartTime;

             System.out.println("Elapsed milliseconds taken for A* Algorithm:
" + difference);

            System.out.println("\n Cost for cells: ");
            for(int i=0;i<x;++i){
                for(int j=0;j<x;++j){
                    if(grid[i][j]!=null)System.out.printf("%-3d ",
grid[i][j].finalCost);

                }
                System.out.println();
            }
            System.out.println();
            if(closed[endI][endJ]){
                //Trace back the path
                 System.out.println("Path in reverse: ");
                 Cell current = grid[endI][endJ];
                 System.out.print(current);
                 while(current.parent!=null){
                     System.out.print(" <- "+current.parent);
                     current = current.parent;
                 }
                 System.out.println();
            }else System.out.println("No possible path");
    }

  }
```