CS 91 USACO

Bronze Division

Unit 4: Basic Tree and Graphs

LECTURE 19: BREADTH FIRST SEARCH

DR. ERIC CHOU

IEEE SENIOR MEMBER

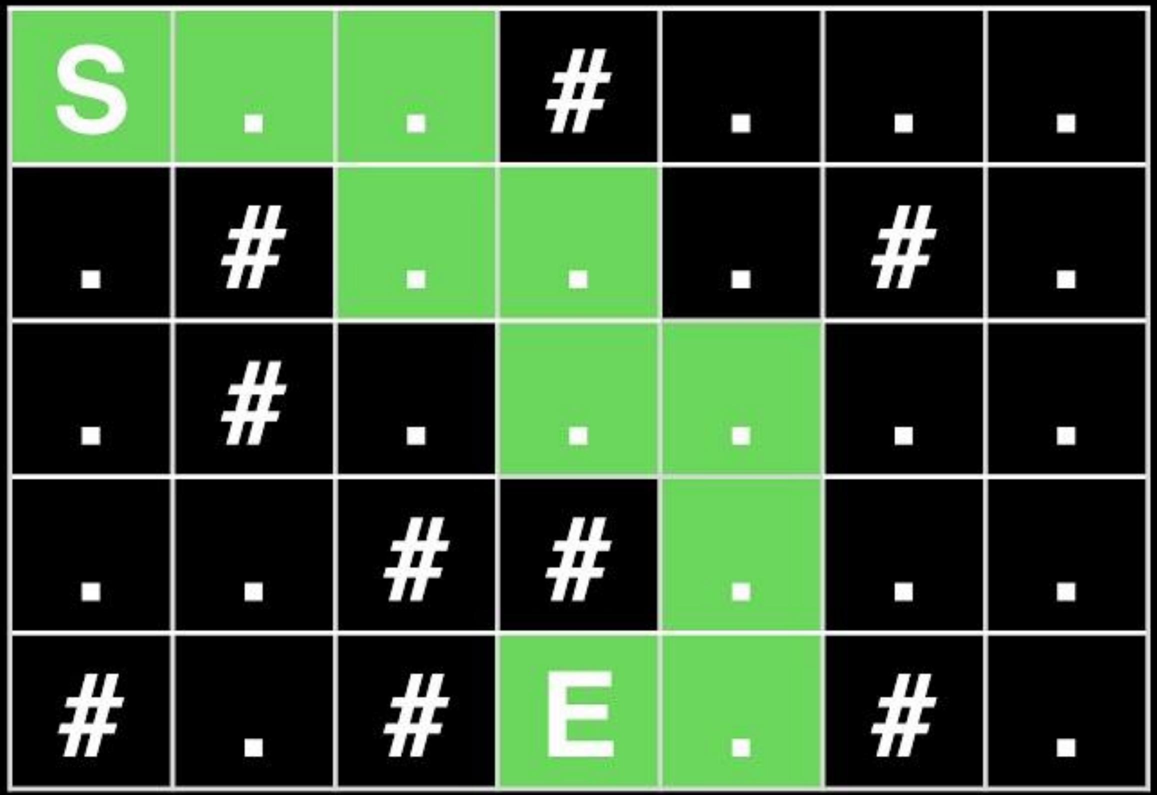
Objectives

- Breadth-first searching on 2D Map
- Shortest Path Detection

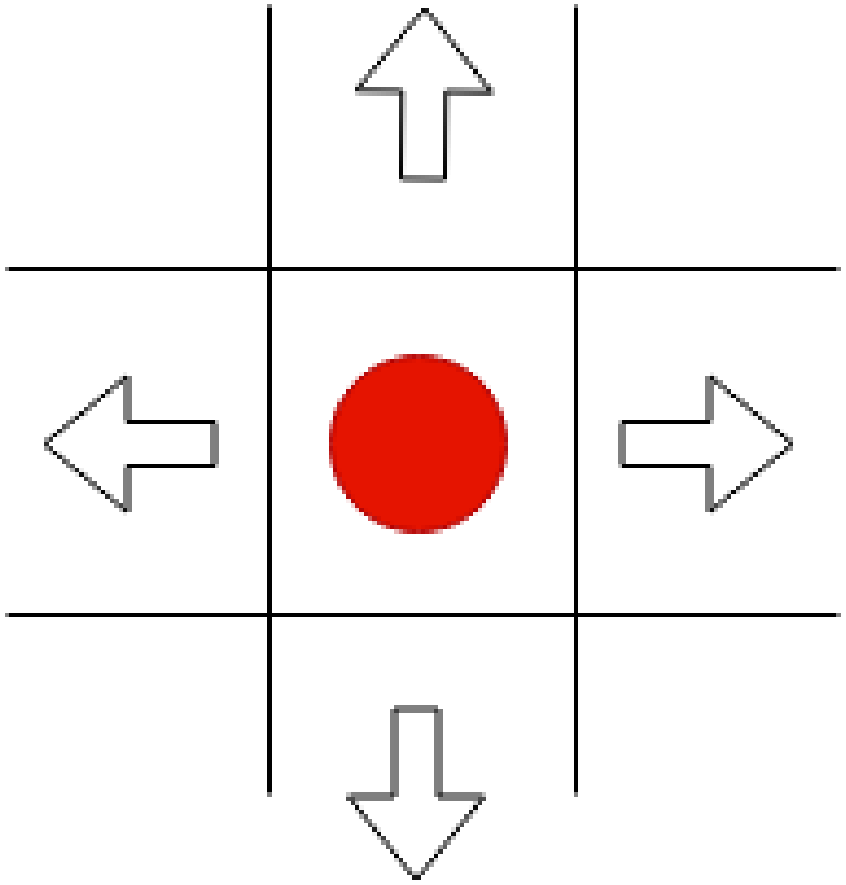
Shortest Path on 2D Map

SECTION 1

Breadth First Search
Shortest Path on a Grid



S	.	.	#	.	.	.
.	#	.	.	.	#	.
.	#	.	.	.	.	.
.	.	#	#	.	.	.
#	.	#	E	.	#	.



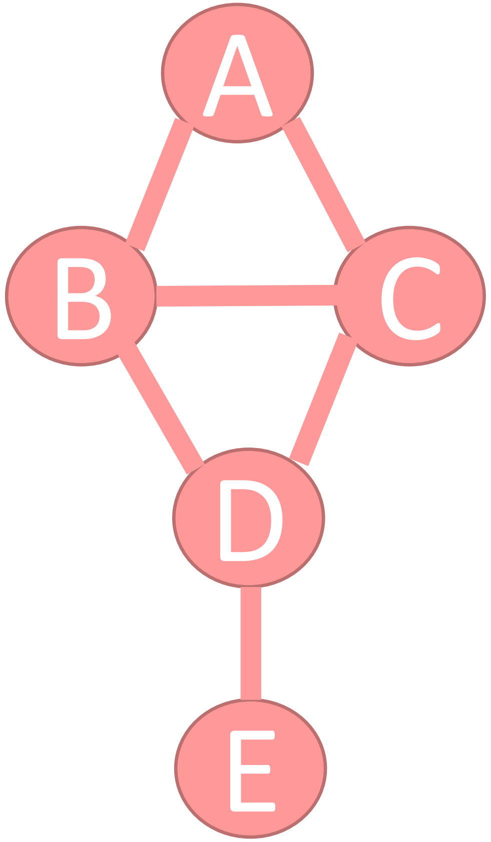
Breadth First Search for Shortest Path

1. 4 neighboring nodes
2. Find the shortest path from point A to point B
3. Count the number of steps

Breadth-First Search on Graph with Adjacency List

SECTION 1

Graph of Study



AdjacencyMatrix

- Adjacency List with Edge List
- Using visited array

```
import java.util.*;
public class AjacencyList
{
    static class EdgeList extends ArrayList<Integer>{}
    static String[] n = {"A", "B", "C", "D", "E"};
    static boolean[] visited = new boolean[n.length];
    static EdgeList[] elists = new EdgeList[n.length];

    public static void reset(EdgeList[] elists){
        for (int i=0; i<elists.length; i++){
          elists[i] = new EdgeList(); // no neighbors
        }
    }

    public static void reset(boolean[] visited){
      for (int i=0; i<visited.length; i++){
          visited[i] = false;
        }
    }

    static ArrayList<Integer> toBeVisited = new ArrayList<Integer>();
```

AjacencyList.java

```java
public static void bfs(int root){
  reset(visited);
  bfsHelper(root);
  System.out.println("\n\n");
}

public static void bfsHelper(int root){
  if (visited[root]) return;
  visited[root] = true;
  System.out.println(n[root]);

  for (int i=0; i<elists[root].size(); i++){
      int nodeID = elists[root].get(i);
      if (!visited[nodeID]&&!toBeVisited.contains(nodeID)) toBeVisited.add(nodeID);
    }
  while (toBeVisited.size()>0){
       bfsHelper(toBeVisited.remove(0));
    }
}
```

AjacencyList.java

```
    public static void main(String[] args){
        System.out.print("\f");
        reset(elists);
        elists[0].add(1); elists[0].add(2);
        elists[1].add(0); elists[1].add(2); elists[1].add(3);
        elists[2].add(0); elists[2].add(1); elists[2].add(3);
        elists[3].add(1); elists[3].add(2); elists[3].add(4);
        elists[4].add(3);

        for (EdgeList elist: elists){
            System.out.println(elist);
         }

        System.out.println();
        System.out.println("Part 1: from A");
        bfs(0);
        System.out.println("Part 2: from C");
        bfs(2);
        System.out.println("Part 3: from E");
        bfs(4);
    }
}
```

AjacencyList.java

[1, 2]
[0, 2, 3]
[0, 1, 3]
[1, 2, 4]
[3]

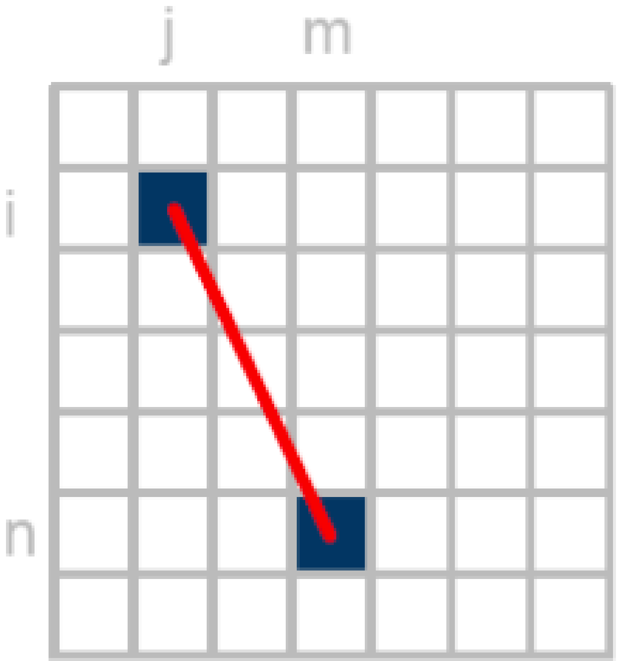
Part 1: from A
A
B
C
D
E

Part 2: from C
C
A
B
D
E

Part 3: from E
E
D
B
C
A

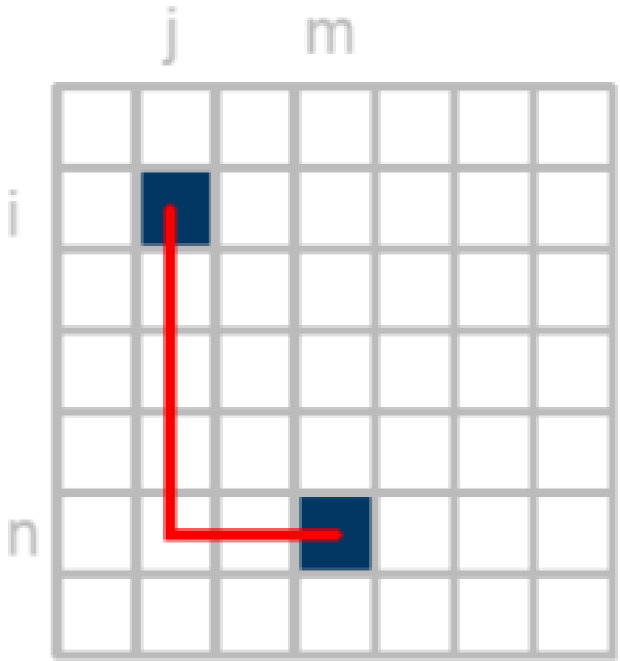
Distance Calculation

SECTION 1



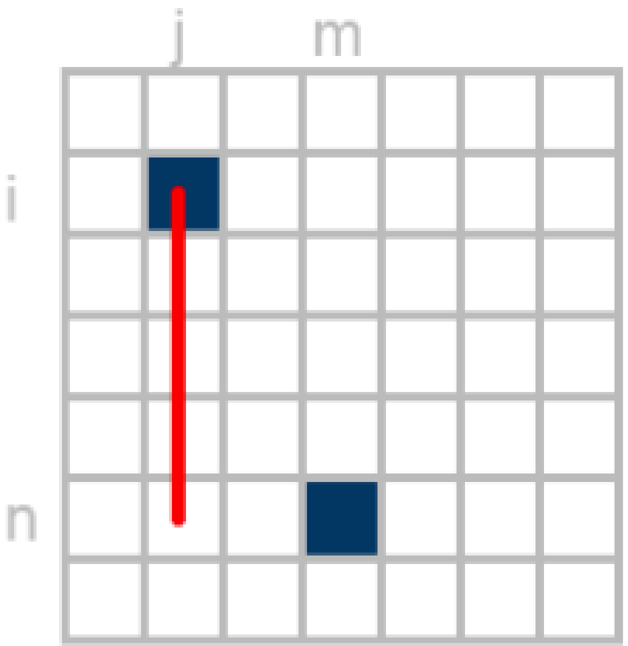
Euclidean Distance

$$=\sqrt{(i-n)^{2} + (j-m)^{2}}$$



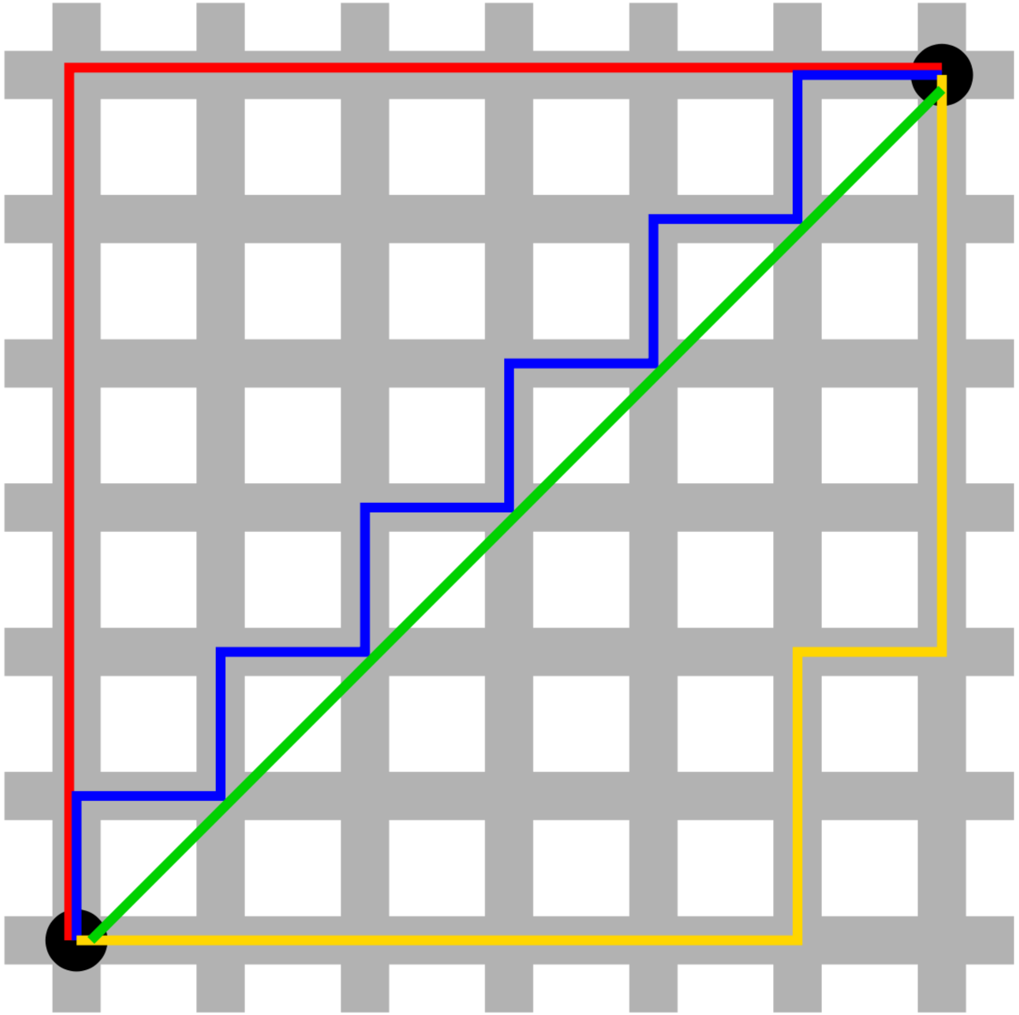
City Block Distance

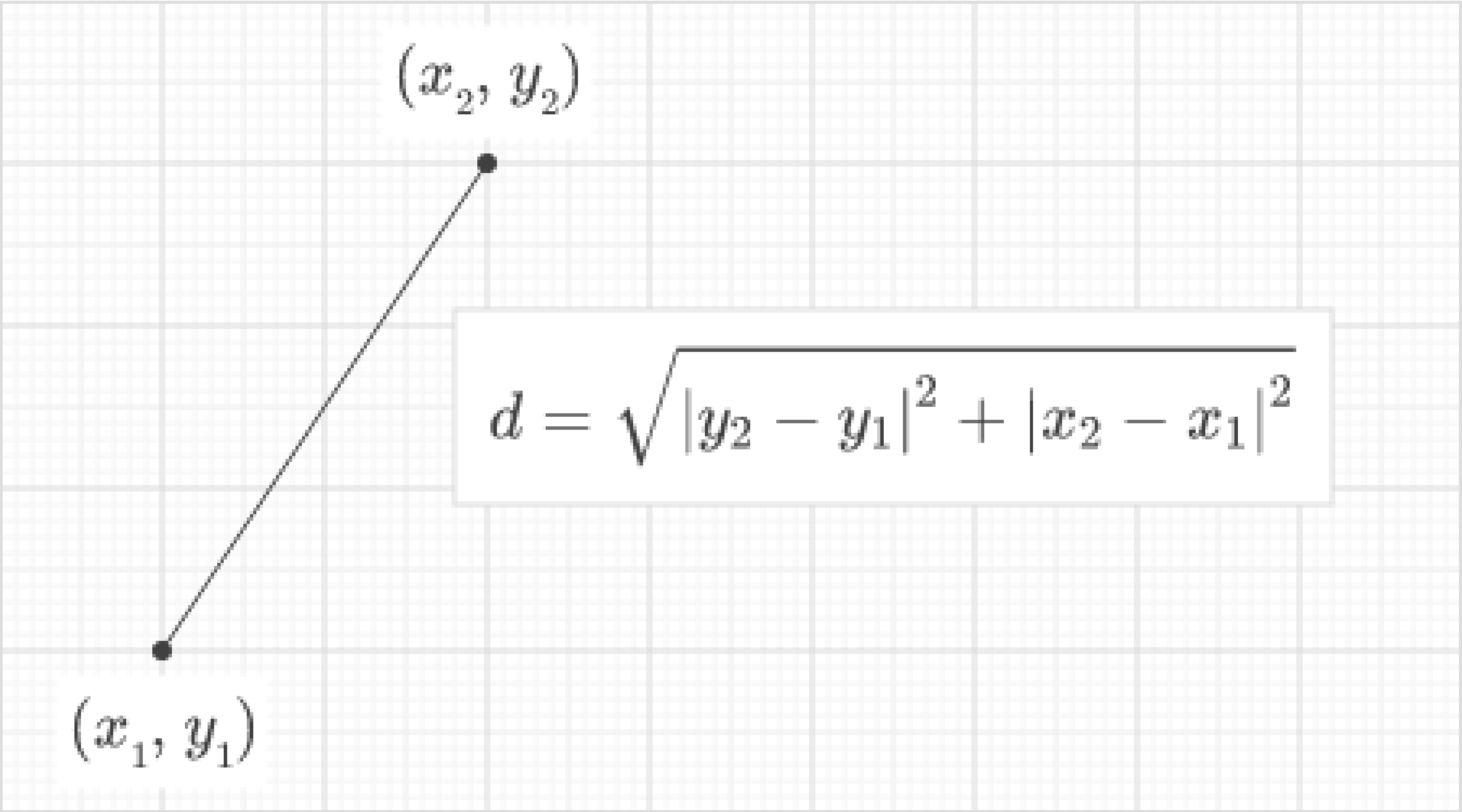
$$= |i-n| + |j-m|$$



Chessboard Distance

$$= \max[\, |i-n|, |j-m| \,]$$





$$d = \sqrt{|y_2 - y_1|^2 + |x_2 - x_1|^2}$$

Distance transform
using city-block (or 4) distance

4	3	2	1	2	3	4
3	2	1	0	1	2	3
2	1	0	1	0	1	2
2	1	0	1	1	0	1
1	0	1	2	2	1	0
1	0	1	2	3	2	1
0	1	2	3	4	3	2

	1	
1	0	1
	1	

Breadth-First Search for Shortest City-Block Distance Path

SECTION 1

```java
import java.util.*;
import java.io.*;
public class ShortestDistanceInCity{
   static int[] dx= {1, 0, 0, -1};
   static int[] dy= {0, 1, -1, 0};

   public static void printMap(char[][] m){
      for (int r=0; r<m.length; r++){
           for (int c=0; c<m[r].length; c++){
               System.out.printf("%3c", m[r][c]);
            }
           System.out.println();
        }
    }
   public static void printDMap(int[][] d){
      for (int r=0; r<d.length; r++){
           for (int c=0; c<d[r].length; c++){
               System.out.printf("%3d", d[r][c]);
            }
           System.out.println();
        }
    }
```

ShortestDistanceInCity.java

```
   public static void resetDMap(int[][] d){
   for (int r=0; r<d.length; r++){
        for (int c=0; c<d[r].length; c++){
            d[r][c] = -1;
         }
     }
 }
public static int getR(int x, int N){ return x/N; }
public static int getC(int x, int N){ return x%N; }

public static ArrayList<Integer> toBeVisited = new ArrayList<Integer>();
public static ArrayList<Integer> level = new ArrayList<Integer>();
public static boolean bfs(int root, int B, char[][] map, int[][] d,  int distance){
   int M = d.length, N = d[0].length;
   int rA = getR(root, N), cA=getC(root, N);
   //System.out.printf("N(%d, %d)\n", rA, cA);
   if (d[rA][cA]>=0) return false;
   d[rA][cA] = distance;
   if (root == B) { return true; }
```

ShortestDistanceInCity.java

```
  for (int i=0; i<dx.length; i++){
       int nR = rA + dy[i];
       int nC = cA + dx[i];
       if (nR<0 || nR >= M) continue;
       if (nC<0 || nC >= N) continue;
       if (d[nR][nC]>=0) continue;
       if (map[nR][nC]=='#') continue;
       if (toBeVisited.contains(nR*N+nC)) continue;
       toBeVisited.add(nR*N+nC);
       level.add(distance+1);
    }
  //System.out.println(toBeVisited.size());
  while (toBeVisited.size()>0){
       //System.out.println("I am here.");
       //System.out.println(toBeVisited);
       if (bfs(toBeVisited.remove(0), B, map, d, level.remove(0))) return true;
    }
  return false;
}
```

ShortestDistanceInCity.java

```java
public static void main(String[] args)throws Exception{
     Scanner input = new Scanner(new File("maze2.txt"));
     int M = input.nextInt(); // number of row
     int N = input.nextInt(); // number of column
     input.nextLine();
     char[][] map = new char[M][N];
     int[][] d = new int[M][N]; /* works as visited map as well */
     resetDMap(d);
     int A=0;
     int B=0;
     for (int i=0; i<M; i++){
        String line = input.nextLine().trim();
        map[i] = line.toCharArray();
        for (int j=0; j<map[i].length; j++){
            if (map[i][j] == 'A') A = i*N+j;
            if (map[i][j] == 'B') B = i*N+j;
         }
     }
     System.out.printf("A=(%d, %d)\n", getR(A, N), getC(A, N));
     System.out.printf("B=(%d, %d)\n", getR(B, N), getC(B, N));
     System.out.println();
     printMap(map);
```

ShortestDistanceInCity.java

```java
        int rA = getR(A, N), cA= getC(A, N);
        int rB = getR(B, N), cB= getC(B, N);
        bfs(A, B, map, d, 0);

        System.out.println();
        printDMap(d);
        System.out.printf("A->B: %d\n", d[rB][cB]);
    }
}
```

```
10 15
_______________
_______________
____#__________
____#__________
__B_#__________
__###__________
_______________
_____#_________
_____#_____A___
_____#_________
```

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ # _ _ _ _ _ _ _ _ _ _
_ _ _ _ # _ _ _ _ _ _ _ _ _ _
_ _ B _ # _ _ _ _ _ _ _ _ _ _
_ _ # # # _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ # _ _ _ _ _ _ _ _ _
_ _ _ _ _ # _ _ _ _ _ A _ _ _
_ _ _ _ _ # _ _ _ _ _ _ _ _ _
```

```
 -1 -1 -1 -1 15 14 13 12 11 10  9  8  9 10 11
 -1 -1 -1 15 14 13 12 11 10  9  8  7  8  9 10
 -1 -1 -1 -1 -1 12 11 10  9  8  7  6  7  8  9
 -1 -1 -1 -1 -1 11 10  9  8  7  6  5  6  7  8
 -1 14 15 -1 -1 10  9  8  7  6  5  4  5  6  7
 14 13 -1 -1 -1  9  8  7  6  5  4  3  4  5  6
 13 12 11 10  9  8  7  6  5  4  3  2  3  4  5
 14 13 12 11 10 -1  6  5  4  3  2  1  2  3  4
 15 14 13 12 11 -1  5  4  3  2  1  0  1  2  3
 -1 15 14 13 12 -1  6  5  4  3  2  1  2  3  4

A->B: 15
```

```
10 10
__________
__A_______
__________
__________
__________
__________
________B_
__________
__________
__________
```