Assignment #3: A* Pathfinding

Due October 25, 2016 by midnight

CS 335 Introduction to Artificial Intelligence

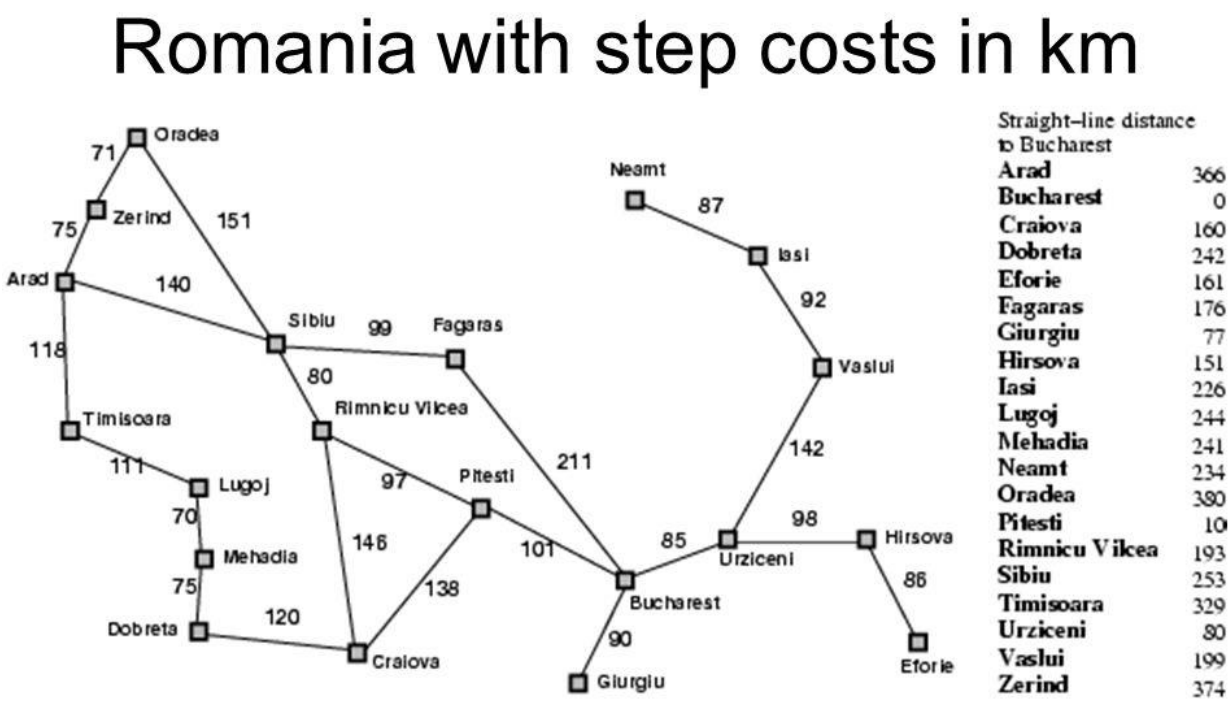
Fall 2016

Northeastern Illinois University

Instructor: Oguzcan Adabuk

1- Objective

The main goal of this assignment is to create a program that finds the shortest path from a city that is selected by the user to the city of Bucharest on the map of Romania.

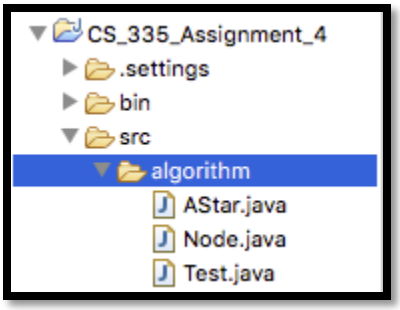


2- Files

The assignment has three files. These files are AStar.java, Node.java and Test.java.

- AStar.java contains the implementation of the A* search algorithm.

- Node.java contains the implementation of nodes that are used by the A* search algorithm. **DO NOT** modify the Node.Java.
- Test.java has the code for a GUI (Graphical User Interface) application that allows the A* search to be visualized with graphical components of SWING library.



AStar.java

- This is the class that contains the actual implementation of the A* search algorithm.
- You are required to implement the ***public void*** *AStarSearch(**Node** startNode, **Node** endNode)* method.
- If works properly, this method will take a start node (a node that the user chooses) and a goal node (Bucharest) and it will find the shortest path between them.

```
public void AStarSearch(Node startNode, Node endNode){

    ArrayList<Node> OpenList = new ArrayList<Node>();
    ArrayList<Node> ClosedList = new ArrayList<Node>();

    OpenList.add(startNode);

    // Your code here...


}
```

- A* search algorithm expands nodes by evaluating their costs. Your program should expand the nodes with the lowest cost.
- When you expand a node, loop through its adjacents and calculate their costs (G+H). Each node that is evaluated should be painted yellow by using the *PaintNodeYellow(Node node)* method.
- If an adjacent doesn't exist in neither *OpenList* nor *ClosedList*, add it to the *OpenList*.
- The node with the least cost should be expanded. Nodes that are expanded should be painted green using the *PaintNodeGreen(Node node)* method.
- In each iteration, apply a goal test. If you reached the goal node, terminate the search. When you reach the goal state, paint the goal node blue by using the *PaintNodeBlue(Node node)* method.

Other Methods:

Important: You do **NOT** need to implement these methods. They are already implemented. You should use them in correct places.

- *void PrintNodeCost(Node node)* : Prints the cost of the node on the node. Pass in the node object that you want to print cost of.
- *void PaintNodeGreen(Node node)* : This method takes a node object as a parameter and paints its GUI component green. Use it by passing in the node that you want to paint green.
- *void PaintNodeYellow(Node node)* : Works similar to PaintNodeGreen(Node node) method.
- void *PaintNodeBlue(Node node) :* Works similar to other paint methods.

```
void PrintNodeCost(Node node){ // This metod prints the cost of each node on the node
    node.nodeBtn.setText(node.name + " F: " + (node.G + node.H));
}

void PaintNodeGreen(Node node){ // Paint expanded nodes green
    node.nodeBtn.setBackground(Color.GREEN);
    node.nodeBtn.setOpaque(true);
}

void PaintNodeYellow(Node node){ // Paint evaluated nodes yellow
    node.nodeBtn.setBackground(Color.YELLOW);
    node.nodeBtn.setOpaque(true);
}

void PaintNodeBlue(Node node){ // Paint the goal node blue
    node.nodeBtn.setBackground(Color.BLUE);
    node.nodeBtn.setOpaque(true);
}
```

Node.java:

- Implements nodes in the graph. Do **NOT** modify anything inside the Node.java.
- It stores the name of the node, its adjacent nodes and distances to them, button object that represents the node visually. Node.java also contains the nodes F, G and H values.
- The H (heuristic) function is calculated by the straight distance to Bucharest from the node plus the obstacle score (H = straightDist + obstacles).
- For example, if there is a road construction close to a city, then the obstacle score will be increased, as a result a path that does not include this node will be more tempting for the A* search.
- Nodes are created in the Test.java using public constructor.

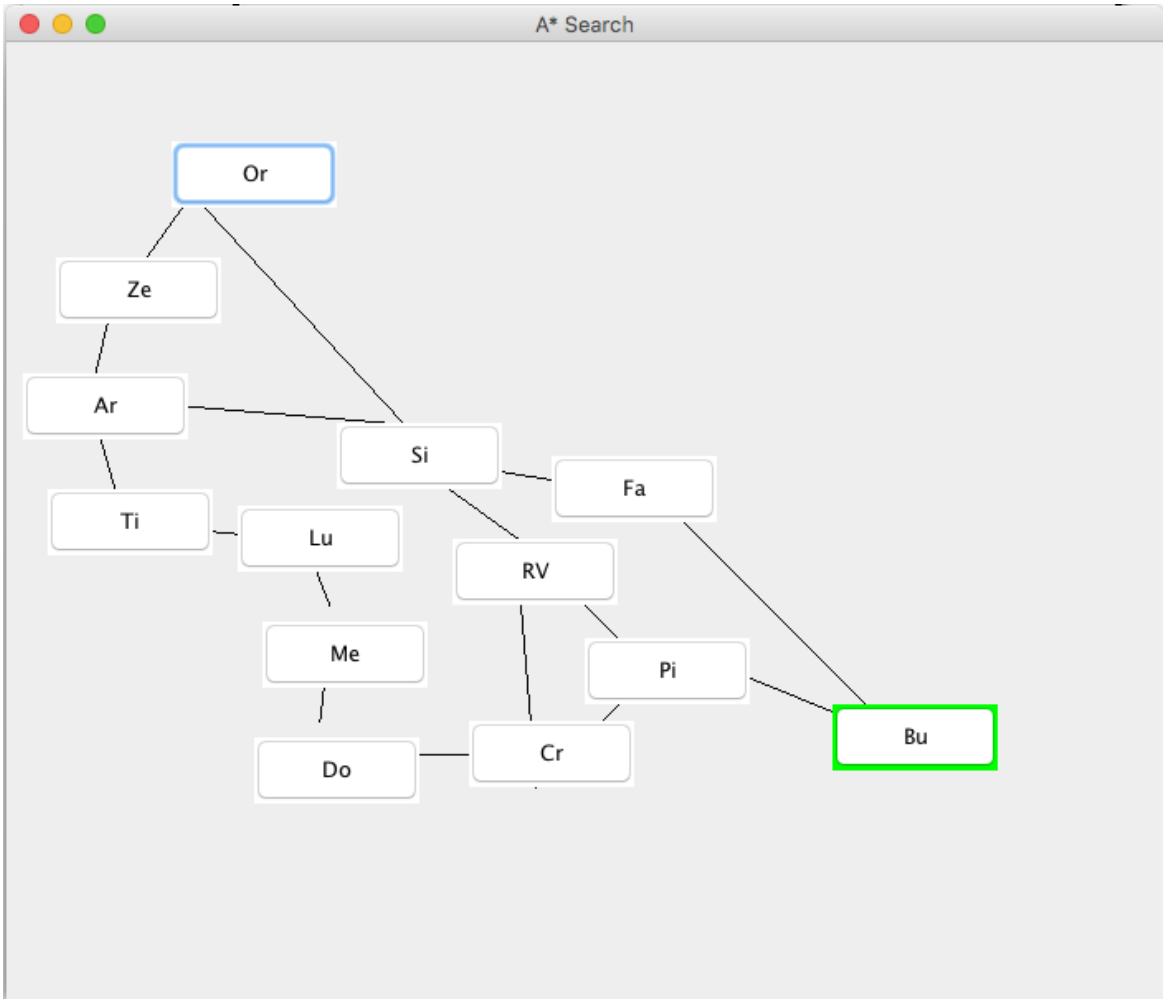
```java
public class Node {

    public String name; // Name of the node

    public int G = 0;
    public int H = 0; // Heuristic cost
    public int F = 0;

    public int straightDist = 0; // Straight distance to Bucharest from this no
    public int obstacles = 0; // If there are obstacles around this node, this

    public Node parentNode;

    public JButton nodeBtn; // Reference to the button object that visually rep

    public ArrayList<Node> adjacents = new ArrayList<Node>(); // List of adjace
    public ArrayList<Integer> distancesToAdjacents = new ArrayList<Integer>();

    public Node(String n, int sdt, int obs){ // Node initialization
        this.name = n;
        this.H = sdt + obs; // Calculating the heuristic cost
    }

    public void SetAdjacentNode(Node adj, int dist){ // Set adjacent nodes and
        adjacents.add(adj);
        distancesToAdjacents.add(dist);
    }
}
```

Test.java

- Test.java contais the code that visualizes the map of Romania, using JButton objects and a JFrame.
- Test.java also initializes each city as a node.

```java
/* INITIALIZE NODES */
// Constructor parameters: (Node name, straight distance to Bucharest, obstacles)
final Node Arad = new Node("Ar", 366, 0);
final Node Bucharest = new Node("Bu", 0, 0);
final Node Craoiva = new Node("Cr", 160, 0);
final Node Dobreta = new Node("Db", 242, 0);
final Node Fagaras = new Node("Fg", 178, 0);
final Node Lugoj = new Node("Lu", 244, 0);
final Node Mehadia = new Node("Me", 241, 0);
final Node Oradea = new Node("Or", 380, 0);
final Node Pitesti = new Node("Pi", 98, 0);
final Node RV = new Node("RV", 193, 0);
final Node Sibiu = new Node("Si", 253, 0);
final Node Timisoara = new Node("Ti", 329, 0);
final Node Zerind = new Node("Ze", 374, 0);
```

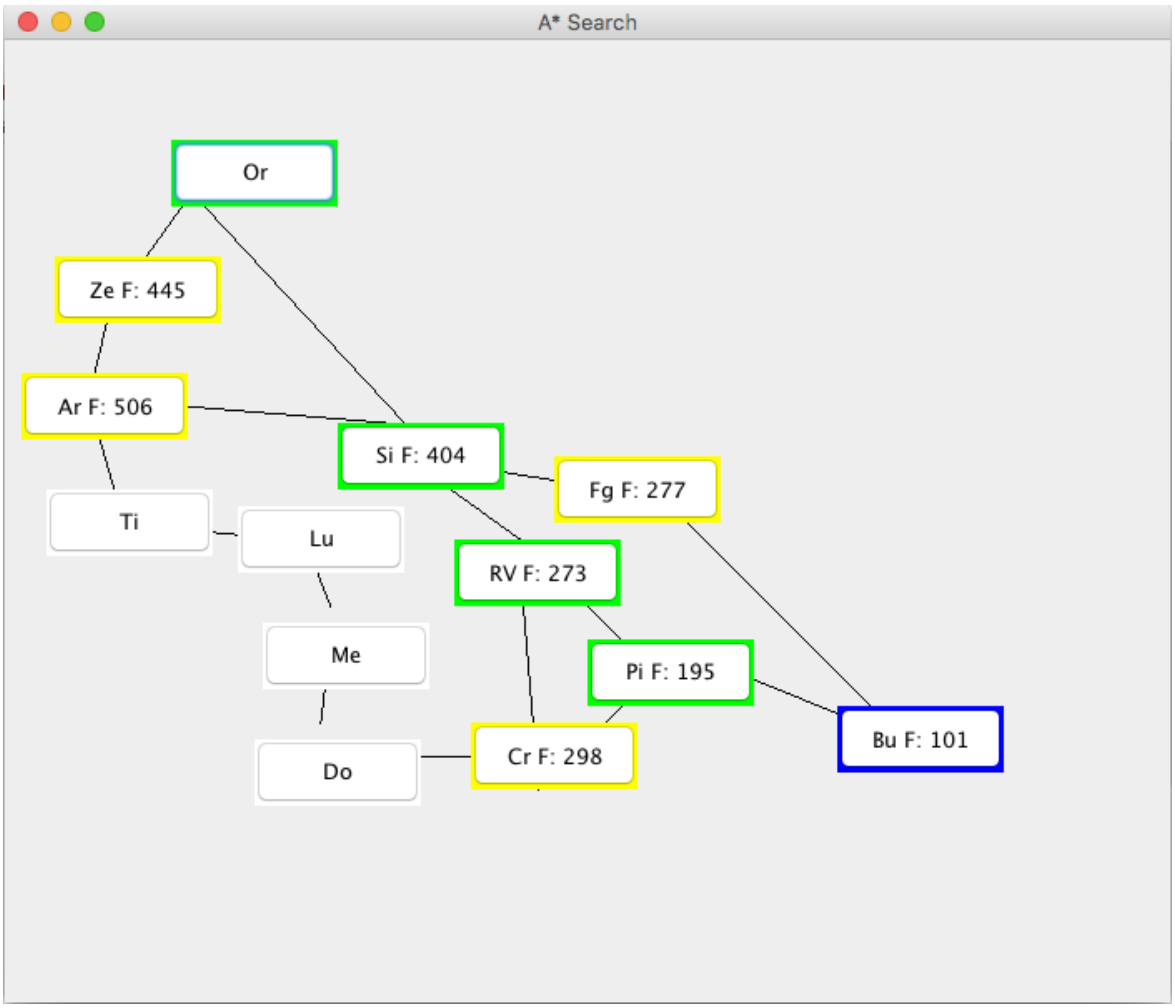
- Each node is initialized with a name, straight distance to Bucharest and score for obstacles.
- Do not modify any code inside the Test.java.
- When the program is run, you should see the map of Romania created in a GUI Window, button objects representing the cities.



3- Running the A* Search:

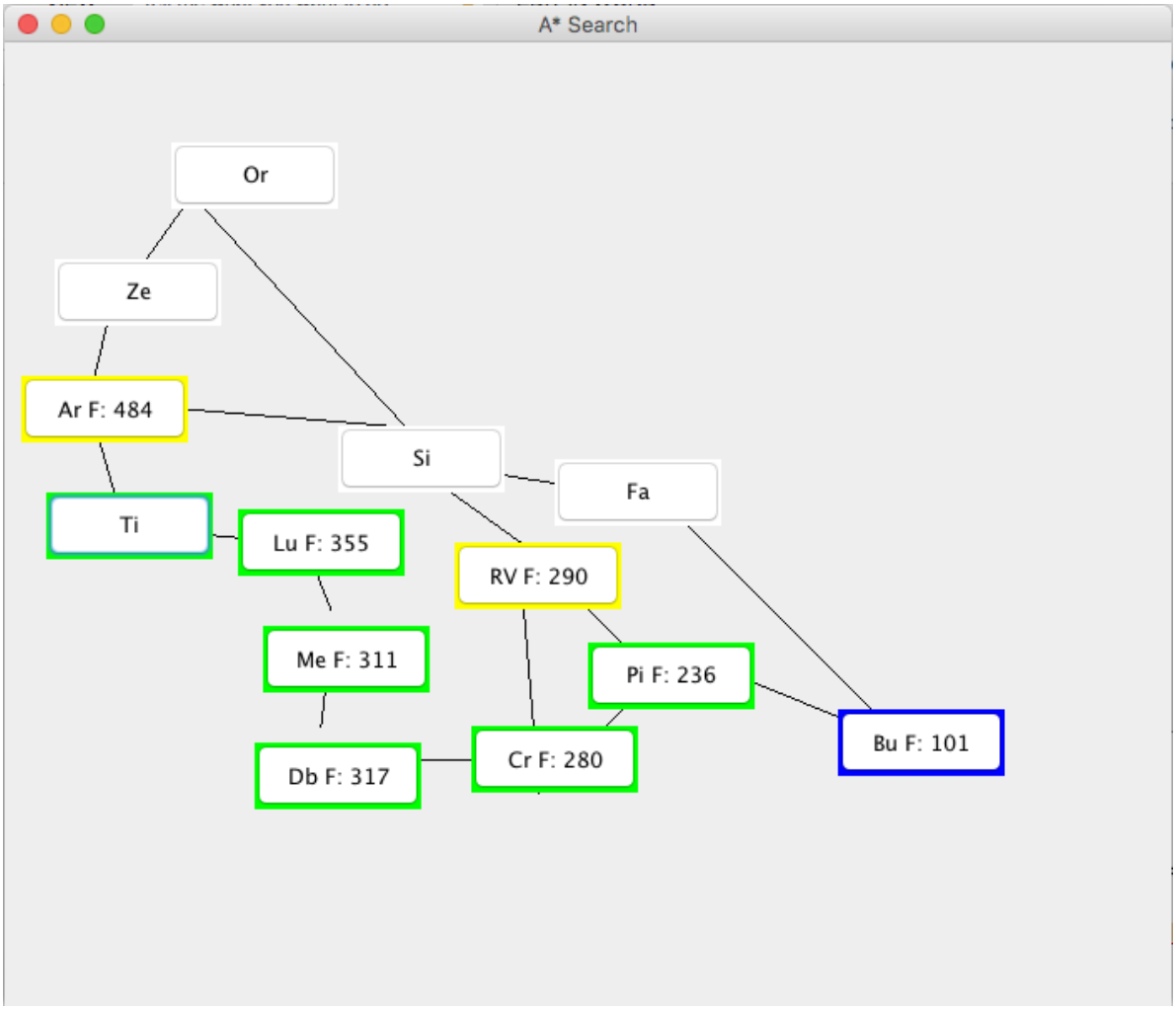
If your implementation of A* search is correct, the program should find the shortest path from a city that is clicked on. All the nodes that make up the path should be painted green and all the evaluated nodes should print their costs (F) so that it is possible to compare the values and see whether the program chose the correct path.

Example 1: This example shows how the program works when Oradea is clicked:



- When Oradea is clicked, the program evaluates all of its adjacents, Zerind and Sibiu, prints both of their F value.
- Evaluated nodes are painted yellow.
- The program decides to expand Sibiu because it has a lower F value (Sibiu F = 404 vs Zerind = 445).
- From Sibiu the program compares its adjacents Arad, Fagaras and Rimnicu Vilcea, and expands Rimniu Vilcea.
- In the next step it compares Pitesti and Craiova and expands pitesti.
- Finally reaches to Bucharest, paints Bucharest blue and terminates the search.

Example 2: This example shows how the completed program works when Timiosara is clicked:



4- Modify Heuristic Values

Experiment by changing the obstacle scores of the nodes between lines 35 and 47 of Test.java.

Obstacle scores are set through the third parameter of Node constructors.

For example, change the obstacle score of Sibiu node to 150 (on line 45) and observe how the path changes.

```
final Node Sibiu = new Node("Si", 253, 0);
```

change to

```
final Node Sibiu = new Node("Si", 253, 150);
```

5- Submitting Your Code

- You must submit your assignment by October 25th by midnight.
- Use Eclipse IDE for this assignment. If you don't have it you can download it from here

https://www.eclipse.org/downloads/

- While It is ok to brainstorm with your classmates, you must keep your code to yourself, assignments that share code will receive F.
- When you complete your assignment, only submit your modified AStar.java file to the dropbox folder for this assignment.
- Do not send me your code via email to check, if you need help, either come to my office hours or if you can't make it to my office hours, send me an email and schedule an appointment at a different time to show your code.