भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर
Indian Institute of Information Technology
Bhagalpur

AI-PROJECT

CS303-Artificial Intelligence Lab
Submission Date – 20/09/19

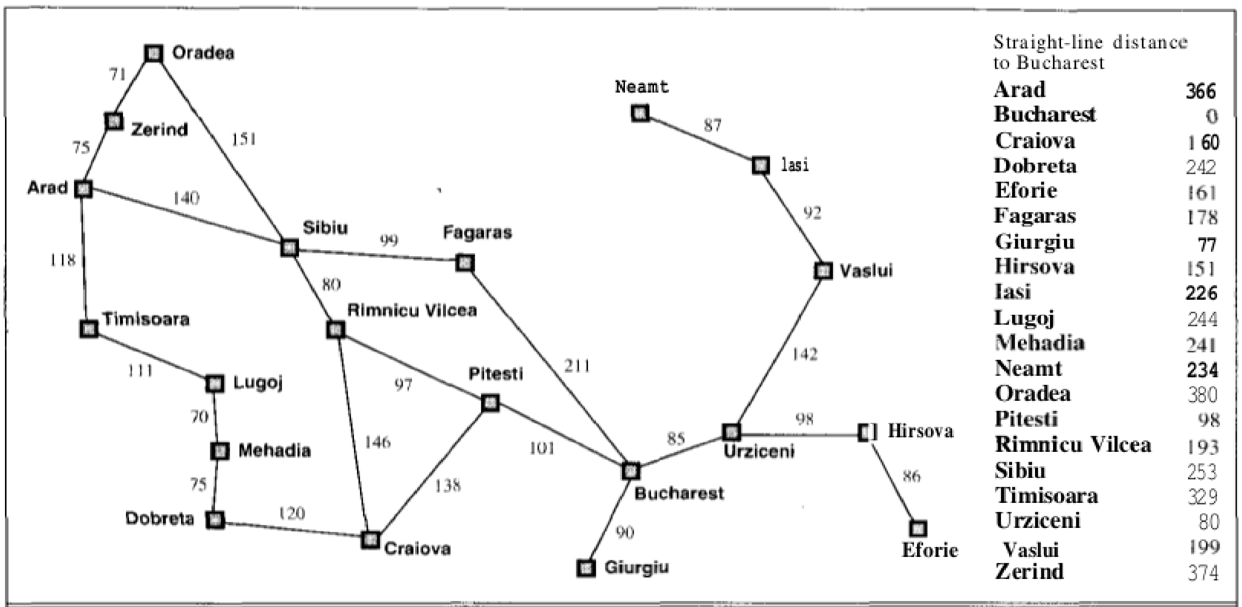
P1: Shortest Path Using A* Algorithm

Team Leader:
Roushan Kumar (170101045)
Team Members:
Vipul Kumar (170101054)
Rishabh Singh (170101044)
Mukul Sharma (170101028)

Course Instructor:
Dr.Rupam Bhattacharyya
Department of Computer Science Engineering
IIIT BHAGALPUR,BIHAR 813210,INDIA
July-Dec 2019

Problem Description

Figure 1 Shows The Map Of Romania And The Starting State Of Your Agent Is <u>Arad</u> . Your Agent Must Reach <u>Bucharest</u> .



Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figure 4.2 Map of Romania with road distances in km, and straight-line distances to Bucharest.

The <u>Evalution Function</u> f(n) be used in the above problem set-up is given below

f(n) = g(n) + h(n) ;

Where g(n) = Path cost from starting node to n
h(n) = Path cost from node n to goal .

Your agent should have <u>The above problem</u> using <u>A* Algorithm</u> .Utilize C/C++ programming language to implement your agent .

Expected output :

1. ***The number of steps required to solve the puzzle.***
2. ***At each step, your program must print the state information which is being currently selected for expension until you reach the goal .***

Statement of Problem :

A $\ast$ depends on the heuristic.A $\ast$ is faster and give good results if we have a good heuristic.In this project I am going to path finding problems, that is, planning routes from a start node to some goal nodes in a graph.Such problems arise in many fields of technology, for example, production planning, message routing in large networks, resource allocation and vehicle navigation systems.I concentrate mostly on planning a minimum cost path using the A $\ast$ algorithm.

In some cases, A $\ast$ is an optimal method in a large class of algorithms.This means, roughly speaking that A $\ast$ explores a smaller region of the search space than the other algorithms in the given class.

A heuristic controls the search of A $\ast$ so that unnecessary branches of the tree of nodes that A $\ast$ visits are pruned.The new method also finds an optimal path to any node it visits for the first time so that every node will be visited only once.The later is an important property considering the efficiency of the search.

In some cases, the A $\ast$ is an optimal resource allocation method, which means that the number of the nodes the path finding algorithms together visit is min-imized.

A $\ast$ gives the same results as Dijkstra, but faster when we use a good heuristic.A $\ast$ Algorithm has some conditions for to work correctly such as the estimated distance between current node and the final node should be lower than the real distance.A $\ast$ is guaranteed to give the shortest path when the heuristic is admissible.

A* Search Algorithm

Motivation:

To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

What is A* Search Algorithm?

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Why A* Search Algorithm?

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has "brains". What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation) .

Explanation:

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.

What A* Search Algorithm does is that at each step it picks the node according to a value-'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell.

We define 'g' and 'h' as simply as possible below

g = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.
h = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don't know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this 'h' which are discussed in the later sections.

Algorithm

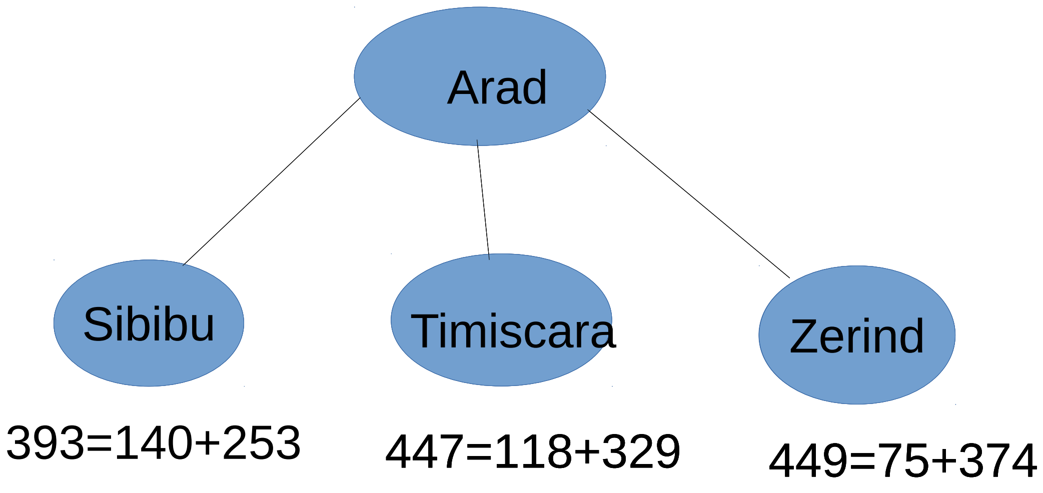
1. Put the start node bs in to OPEN

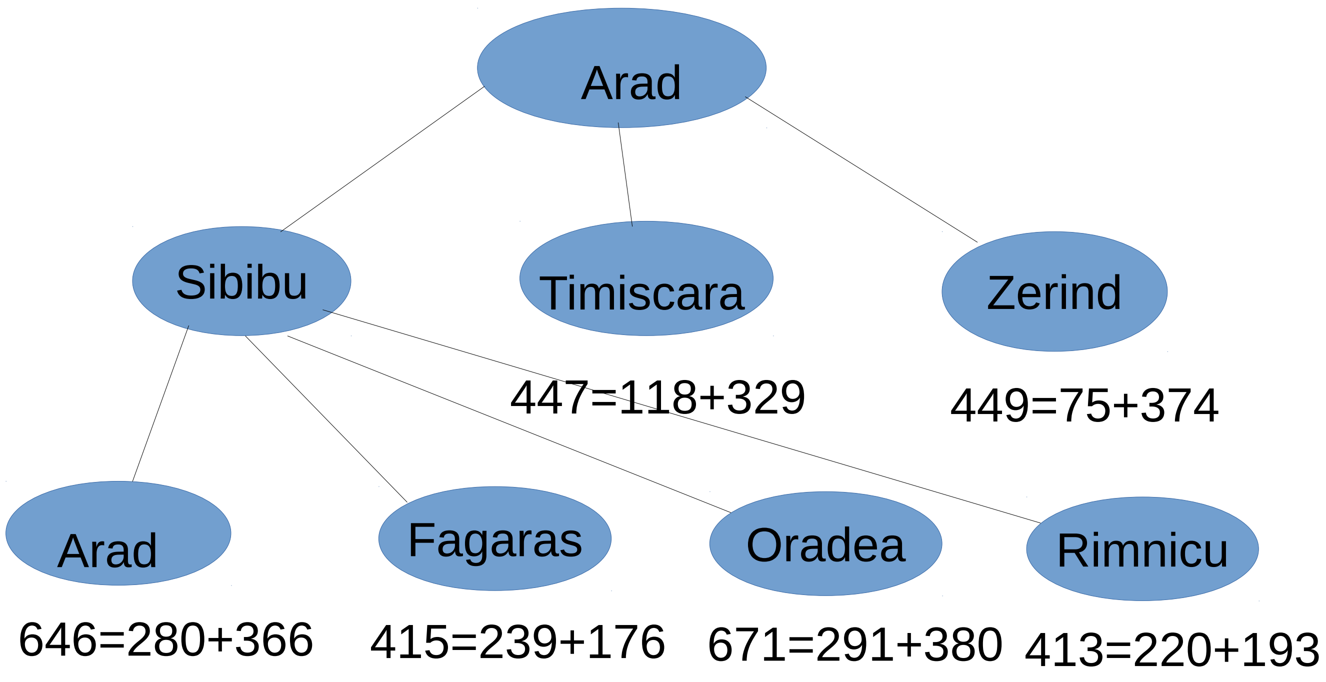
2. IF OPEN is empty THEN exit with failure.

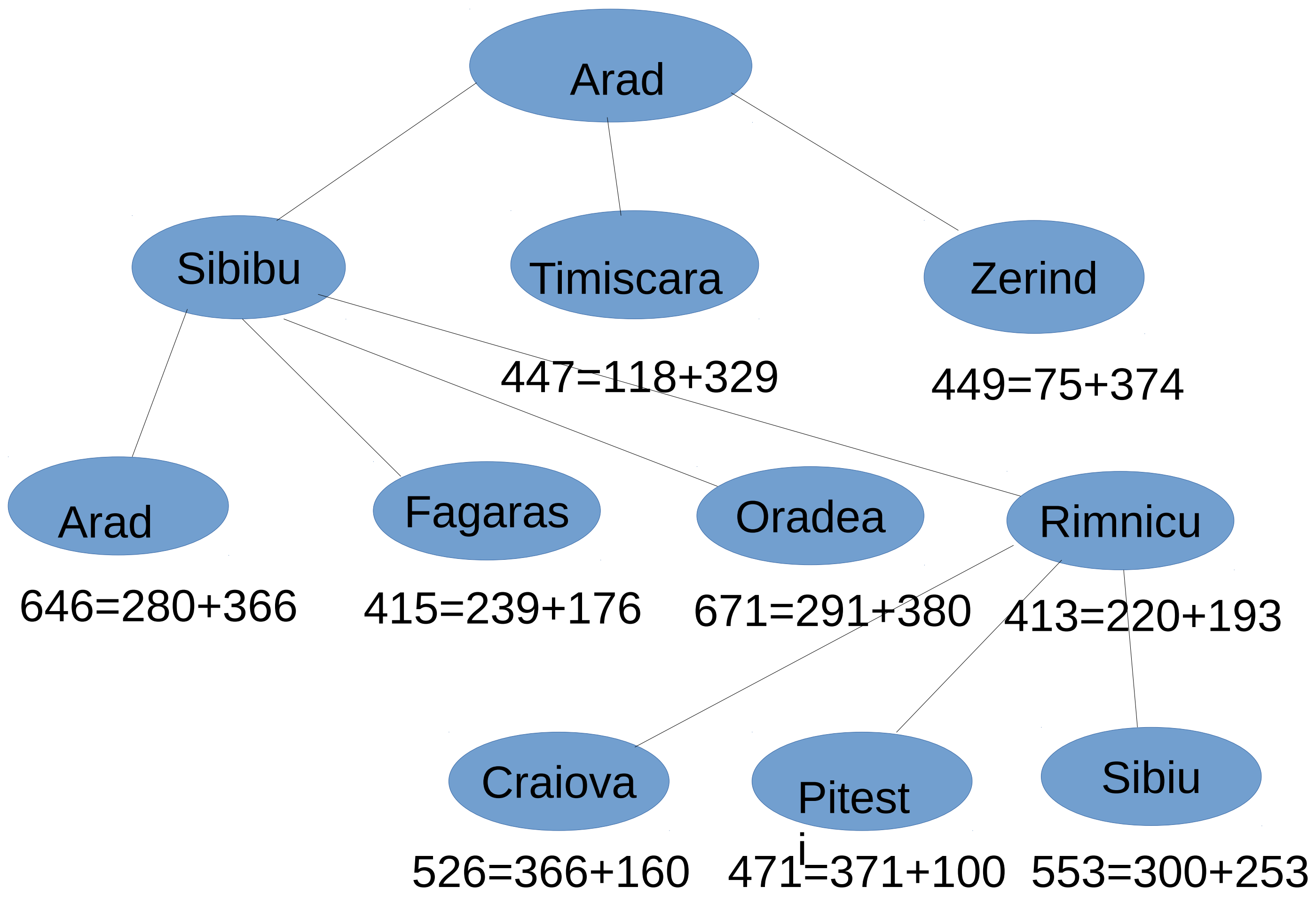
3. Remove from OPEN and place in CLOSED a node n for which f is mimimum.

4. IF n is a goal node THEN exit sucessfully with the solution obtained by tracking back the pointers from n to s.

5. ElSE expand n, generating all its succesors,and attach them pointers back to n.







6. FOR every succesor n' of n DO IF n is not already in OPEN or CLOSED
THEN estimate

$$h(n'),$$

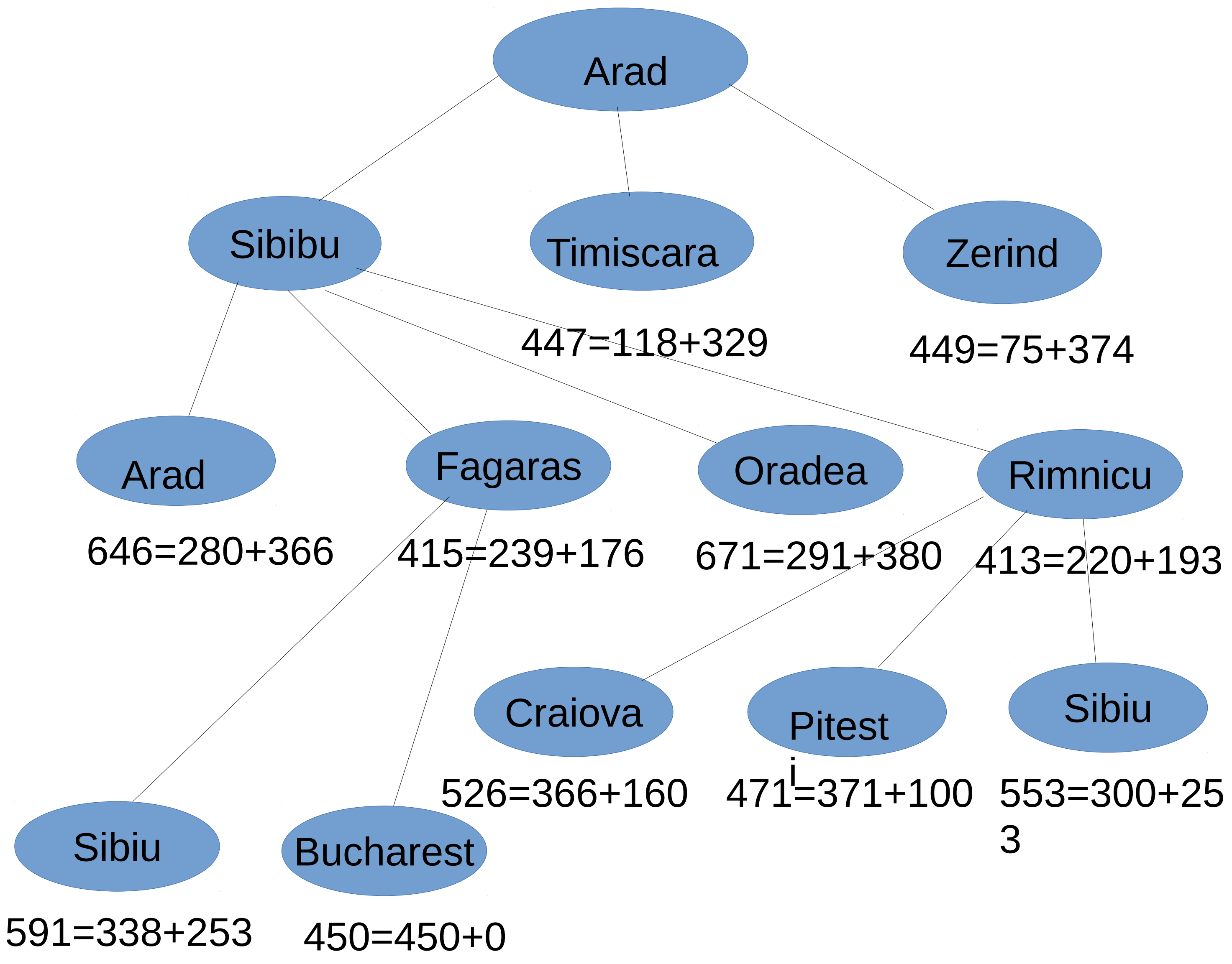
and calculate

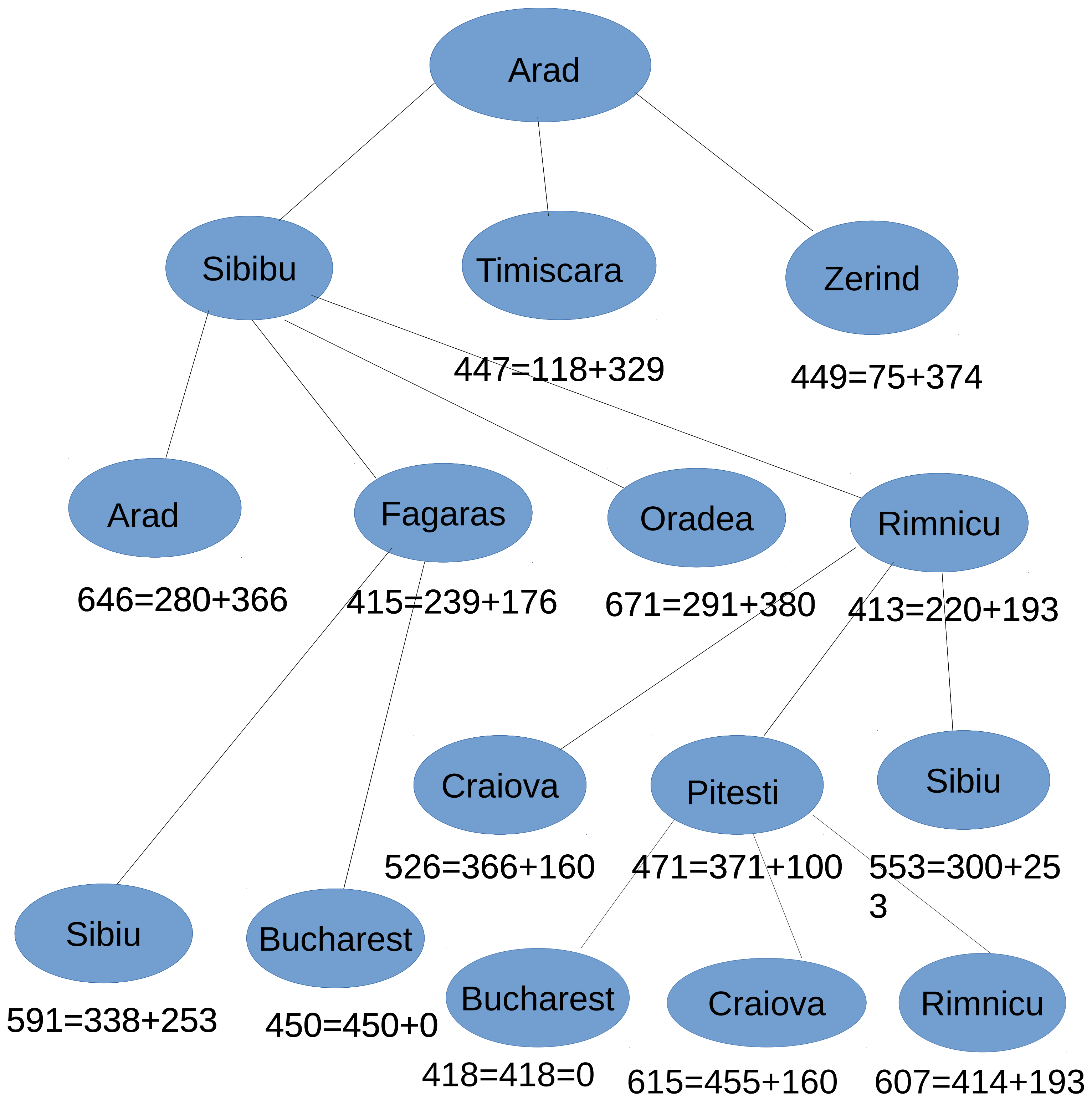
$$f(n') = g(n') + h(n')$$

where

$$g(n') = g(n) + c(n, n')$$

and put n' in to OPEN


Arad
Sibibu
Timiscara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu
646=280+366
415=239+176
671=291+380
413=220+193
Craiova
Pitest
i
Sibiu
526=366+160
471=371+100
553=300+25
3
Sibiu
Bucharest
591=338+253
450=450+0



Arad
Sibibu
Timiscara
Zerind
447=118+329
449=75+374
Arad
Fagaras
Oradea
Rimnicu
646=280+366
415=239+176
671=291+380
413=220+193
Craiova
Pitesti
Sibiu
526=366+160
471=371+100
553=300+25
3
Sibiu
Bucharest
591=338+253
450=450+0
Bucharest
Craiova
Rimnicu
418=418=0
615=455+160
607=414+193

```
7. IF n is already in OPEN or CLOSED THEN direct its pointer along
   the path yielding the lowest
                          g(n)
END FOR

8. GO TO step 2
```

Conclusion and Observation:

1.The time complexity of A ∗ depends on the heuristic.In the worst case, the number of nodes expanded is exponential in the length of the solution.

$$|h(x) - h*(x)| = O(\log(h)*(x))$$

where h ∗ is the optimal heuristic and it is also defined as true cost of getting from n to the goal.For almost all heuristics in practical use, the error is at least proportional to the path cost, and the resulting exponential growth eventually.

2. It is guranteed to provide underestimate of the true cost of the goal, it is guranteed that solution will be found and solution will be best.

3. A* provides optimal solution.