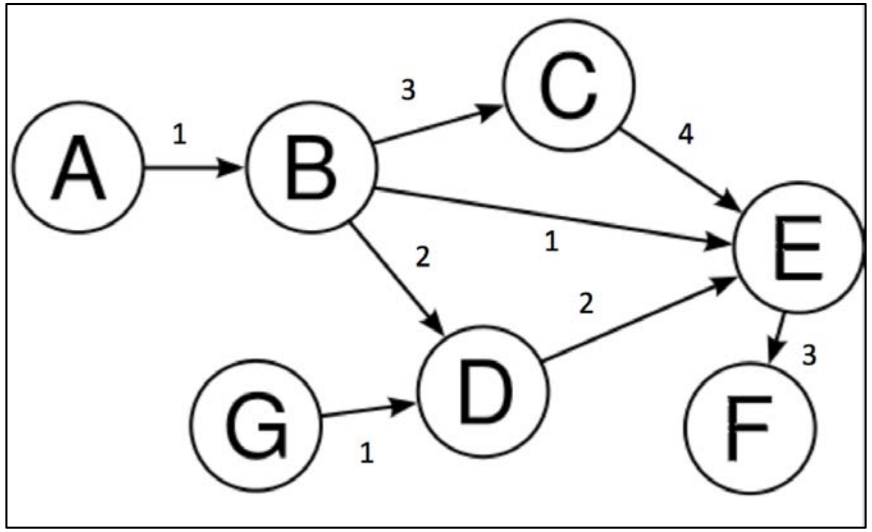
Searching Algorithms

Computational Perception and Artificial Intelligence

Marist School

Graph Theory and Definition

- A Graph G consists of Vertices V and Edges E
- Written G = (V, E)
- Vertices are “Points” on the Graph
- Edges connect two Vertices
- Edges can have a “weight”
- Types of Graphs:
  - Cycles
  - Trees
  - Cyclical
  - Directed
  - . . .

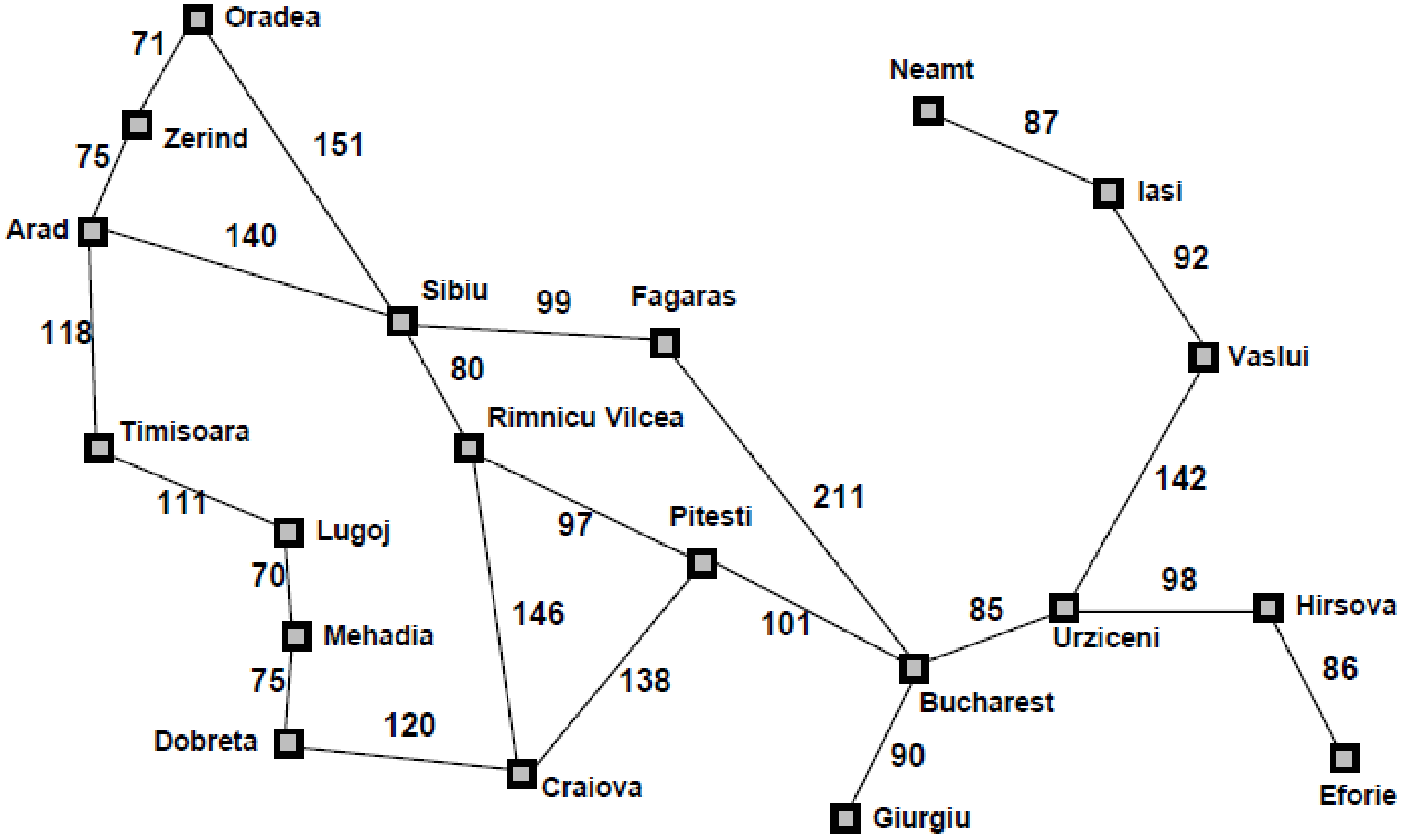


Goal: Given a Weighted non directed Graph G . . .

- Given a start and end vertex on a graph, find the shortest path between start and end in graph G.
- Use three types of Searches
  - Breadth First Search
  - Uniform Cost Search
  - A* Search
- Compare searches and identify strengths for each

Sample Map: Romania

What is shortest path between Arad and Bucharest?

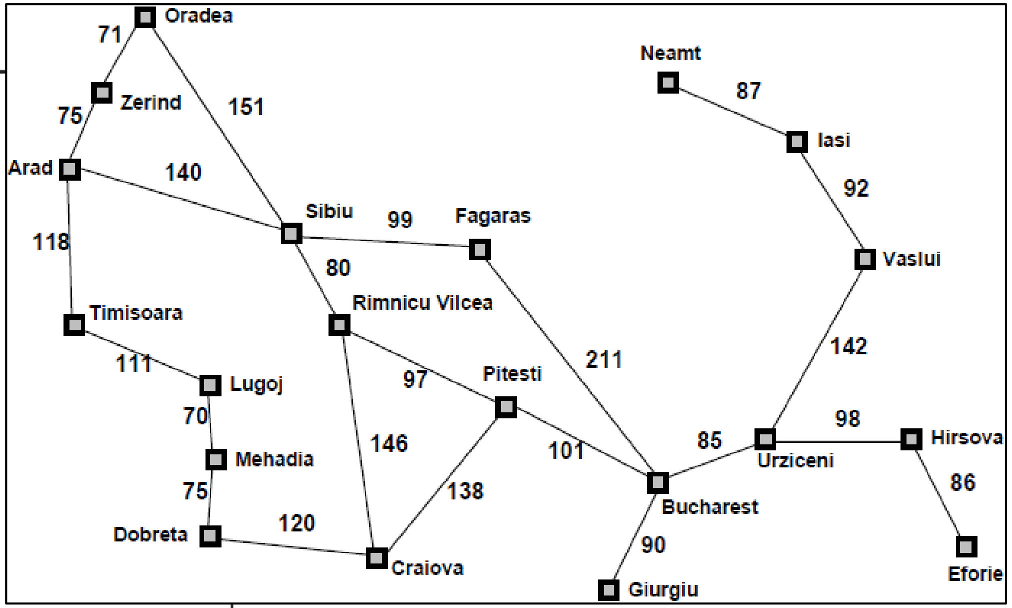


Artificial Intelligence, A Modern Approach. Pg 68

Breadth First Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

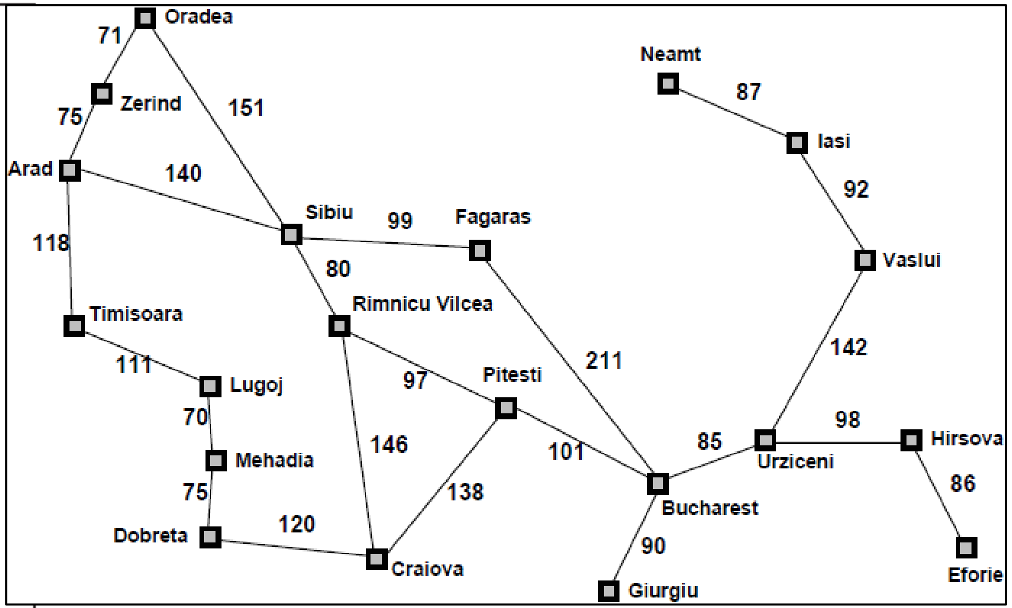


Artificial Intelligence, A Modern Approach. Pg 82

Uniform Cost Search

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
      if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              frontier ← INSERT(child, frontier)
          else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.



Artificial Intelligence, A Modern Approach. Pg 84

A* Heuristic Search

3.5.2 A* search: Minimizing the total estimated solution cost

The most widely known form of best-first search is called **A* search** (pronounced "A-star search"). It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal:

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

Since $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have

$$f(n) = \text{ estimated cost of the cheapest solution through } n .$$

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of $g(n) + h(n)$. It turns out that this strategy is more than just reasonable: provided that the heuristic function $h(n)$ satisfies certain conditions, A* search is both complete and optimal. The algorithm is identical to UNIFORM-COST-SEARCH except that A* uses $g + h$ instead of g.

Artificial Intelligence, A Modern Approach. Pg 93

Problem Set 07

- Download and Extract PS07 from this link:
  - http://www.nebomusic.net/perception/ps07.zip
- Complete the Functions for Breadth First Search, Uniform Cost Search, and A*Search
- Helper functions included in the util.py file. Read and understand the roles of the helper functions.
- Visualization Functions are also included to view maps and paths.
- Random City Function included to generate random maps.
- Use the .pdf from Chapter 3 of Artificial Intelligence to help with algorithms and understanding of search. (Included in ps07 file)

Requirements: PS07

- Complete the Breadth First, Uniform Cost, and A* Search Functions. Run on Tests included in ps07.py file.
- Write a short paragraph in a text file called "analysis.txt". Compare the three search functions. Which one has the shortest running time and why?
- Create at least one random City and run the three search algorithms. Generate a map called "random.png" with a sample route. Place this in the outputs folder.
- Create your own Map using real world examples. Have at least 10 vertices (cities / nodes) and have at least 15 Edges with weights.
- Run the three algorithms and generate a map called "mycity.png" in the output folder with a path.
- Zip ps07 and submit to Google Classroom.