Game AI

Where is the AI

- Route planning / Search
- Movement
- Group behavior
- Decision making

General Search Algorithm Design

- Keep a pair of set of states:
  - One, the set of states to explore, called the ***open set*** or the *frontier*.
  - The second, the set of states you have seen, called the ***closed set***.
- Initially just put the start node in the open set
- While the open set is not empty
  - Take a node from the open set. Add it to the closed set.
  - If the node is a goal node you're done!
  - if next is in closed then continue.
  - Necessary? Not if we check if an item is in Open before adding
  - Otherwise ***expand*** the node, ***generating*** all of the node's **successors** and add the ones we have "not seen" to the open list
- If finish the loop because the open set becomes empty, then failure.
- What if one of the successors was a goal node?
- Could we have just declared success right away?
- What order should we remove items to the open set?
- Is our algorithm **complete**? Is it **optimal**?

Basic Search Algorithms

- Depth First (DFS)
  - Organize open set as a stack (LIFO)
- Breadth First (BFS)
  - Organize open set as a queue (FIFO)
- Data structure for closed set?
  - What are the operations? Add and check membership.
- Advantages?
- Complete?
- Optimal?

Breadth First Search (modified)

- Breadth first search can be made a little more efficient:
  - If start is a goal then success!
  - Add start to open
  - While open set is not empty
    - Take the next node off of open.
    - If in closed set continue
      - Generate its successors and for each successor
      - If it is a goal state then done.
      - Otherwise if not seen add to open.
- Where is the gain? Seeing if a state is the goal when we *generate* it.

Saving space

- BFS has a large frontier / open set.
  - It grows exponentially.
- Can we reduce it?
- Bi-directional
- Bounded DFS
- Iterative DFS

Bidirectional

- Apply a breadth first search from both start and from goal.
- When their “frontiers” intersect we have a solution.
- Benefit?
- Space reduced from O(bd) to O(bd/2)

Depth Limited

- If you know the solution can not be any deeper than depth k
- Then use DFS and cut off your search at depth k.
- May greatly speed up DFS.
- Ensures completeness.
- Optimal?

Iterative Deepening

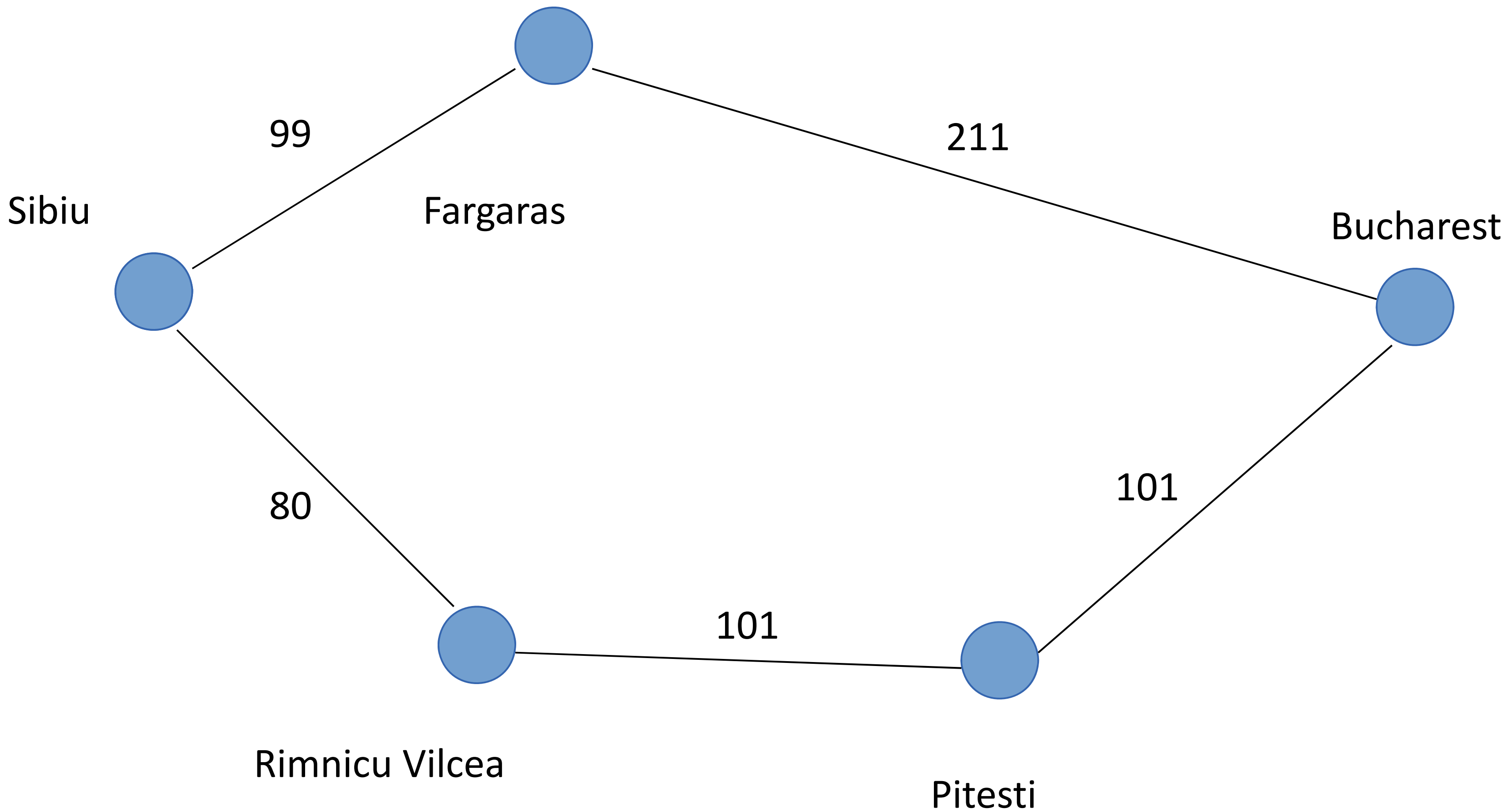
- for (int level = 0; found?; ++level)
  depthLimited(level)
- Huh? We will be searching the beginning levels over and over! All our efforts will be thrown out!
- Yes, compared to BFS we are trading time for space.
- But it is optimal!

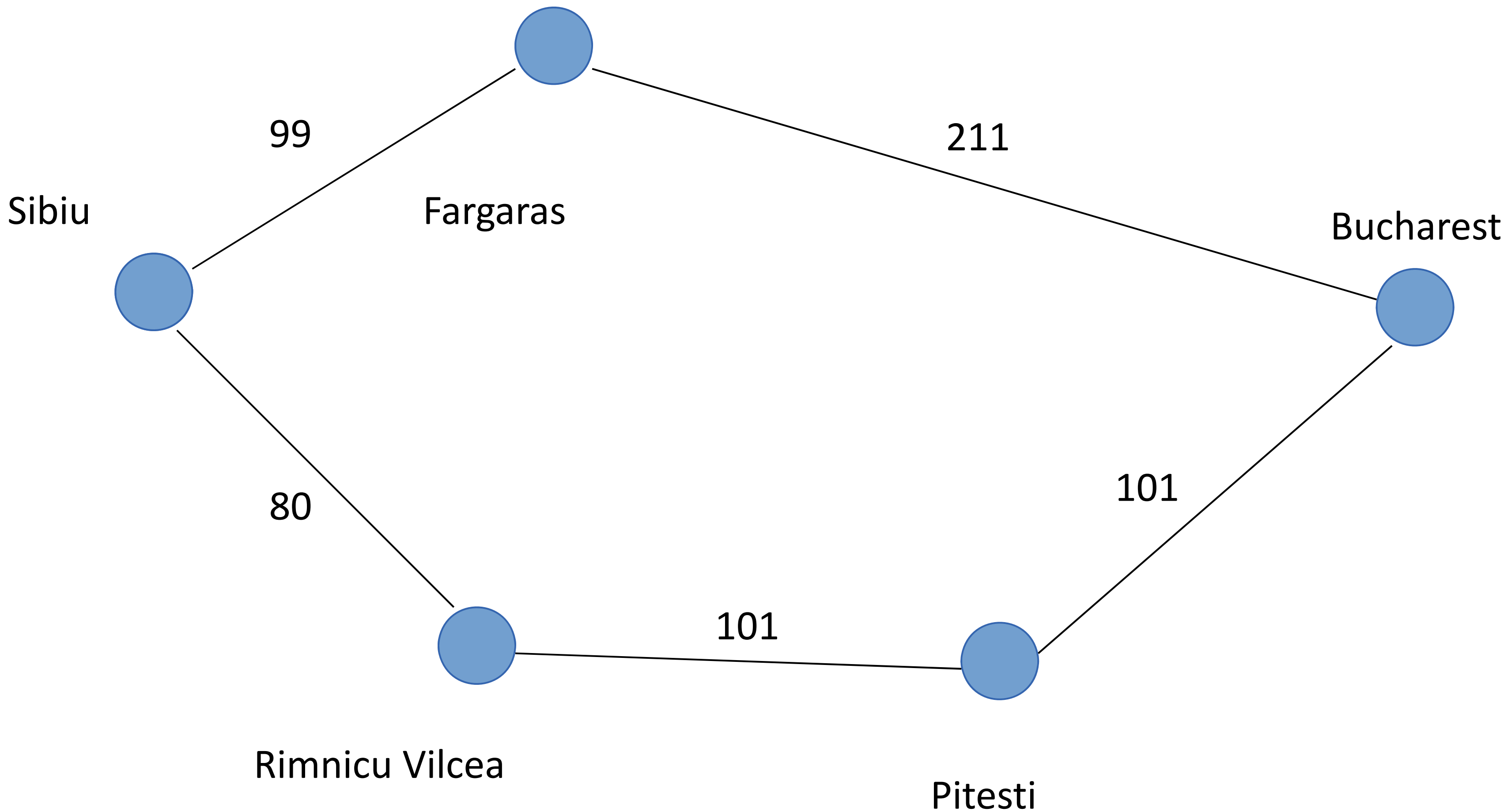
Uniform Cost

- What if the effort to go from one state to another is not always the same?
- E.g. Traveling on a diagonal in a grid might cost 1.4 times the cost of left/right/up/down.
- Or our “steps” might involve plane trips of different distances.
- “Uniform cost” means we want to expand our search so that we explore nodes *uniformly* in how much it costs to get to them.
- Same as Dijkstra’s shortest path, but that is to all nodes, not just to a goal.
- Critical point in the algorithm:
  - a state on the open set may change!
  - Cannot use optimization that we used in BFS

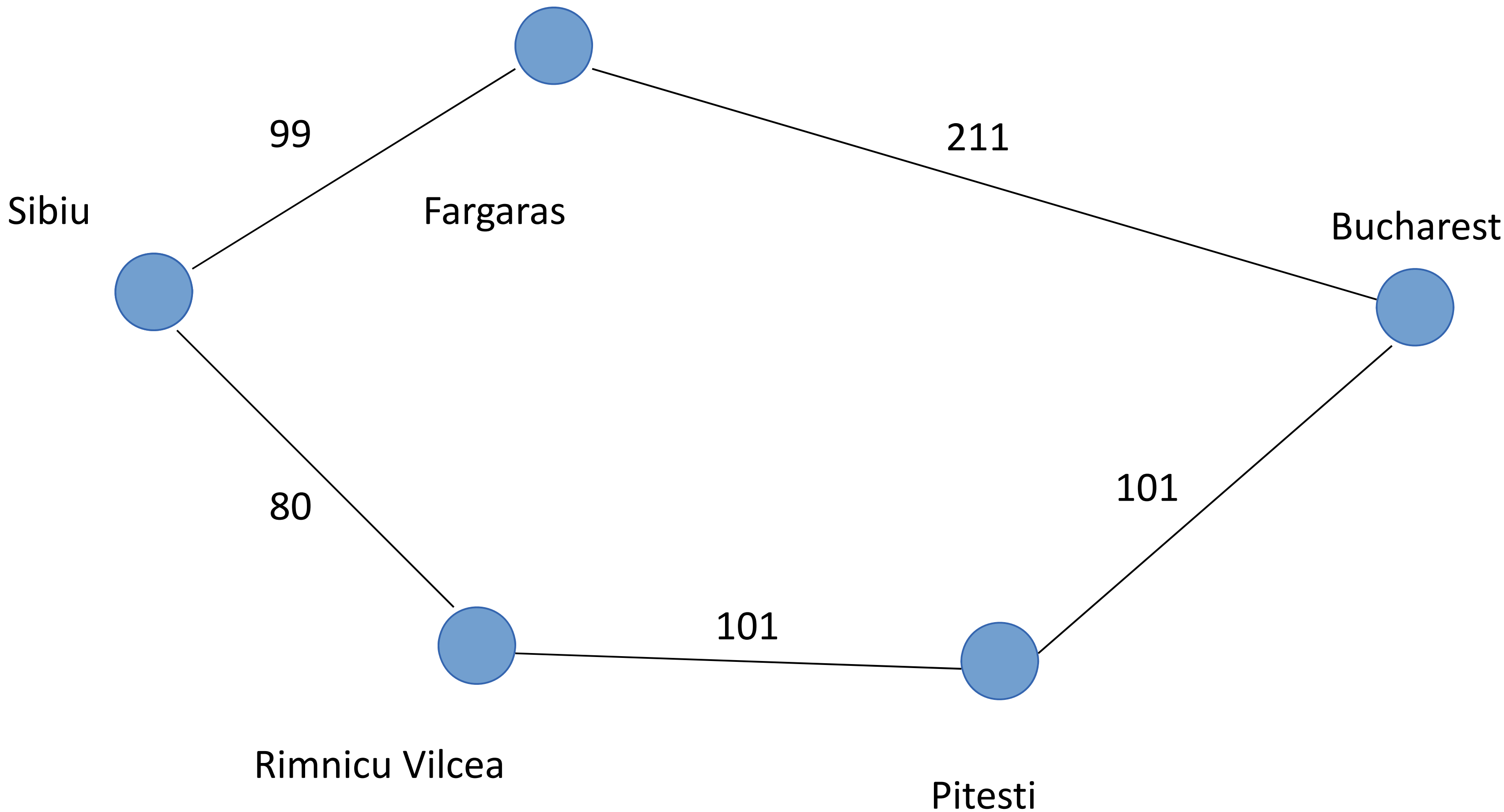
From Sibiu to Bucharest

Sibiu	Fagaras	99
Sibiu	Rimnicu Vilcea	80
Fargaras	Bucharest	211
Rimnicu Vilcea	Pitesti	101
Pitesti	Bucharest	101

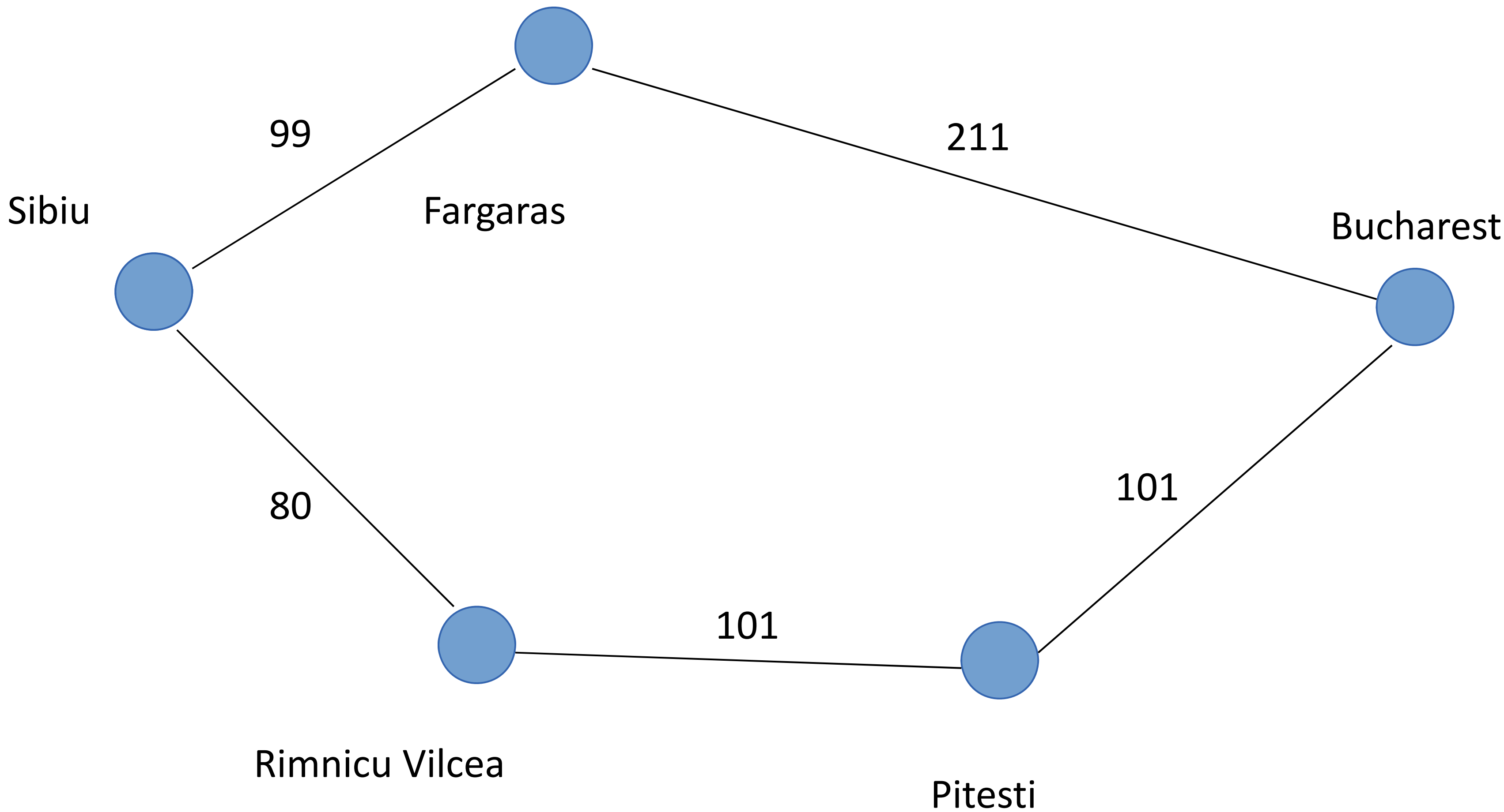

99
211
Sibiu
Fargaras
Bucharest
80
101
101
Rimnicu Vilcea
Pitesti


99
211
Sibiu
Fargaras
Bucharest
80
101
101
Rimnicu Vilcea
Pitesti

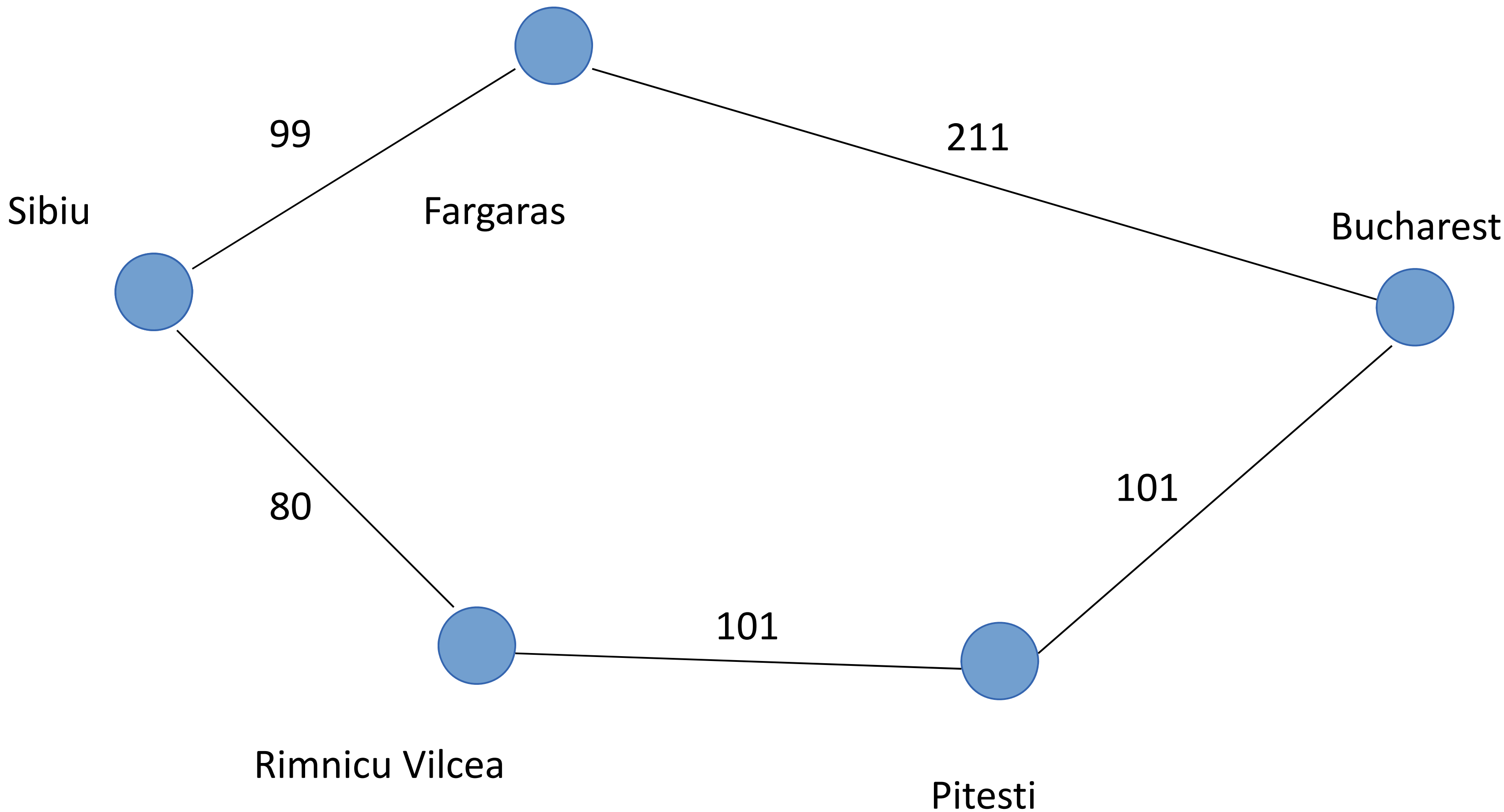
Open: [S(0)]


99
211
Sibiu
Fargaras
Bucharest
80
101
101
Rimnicu Vilcea
Pitesti

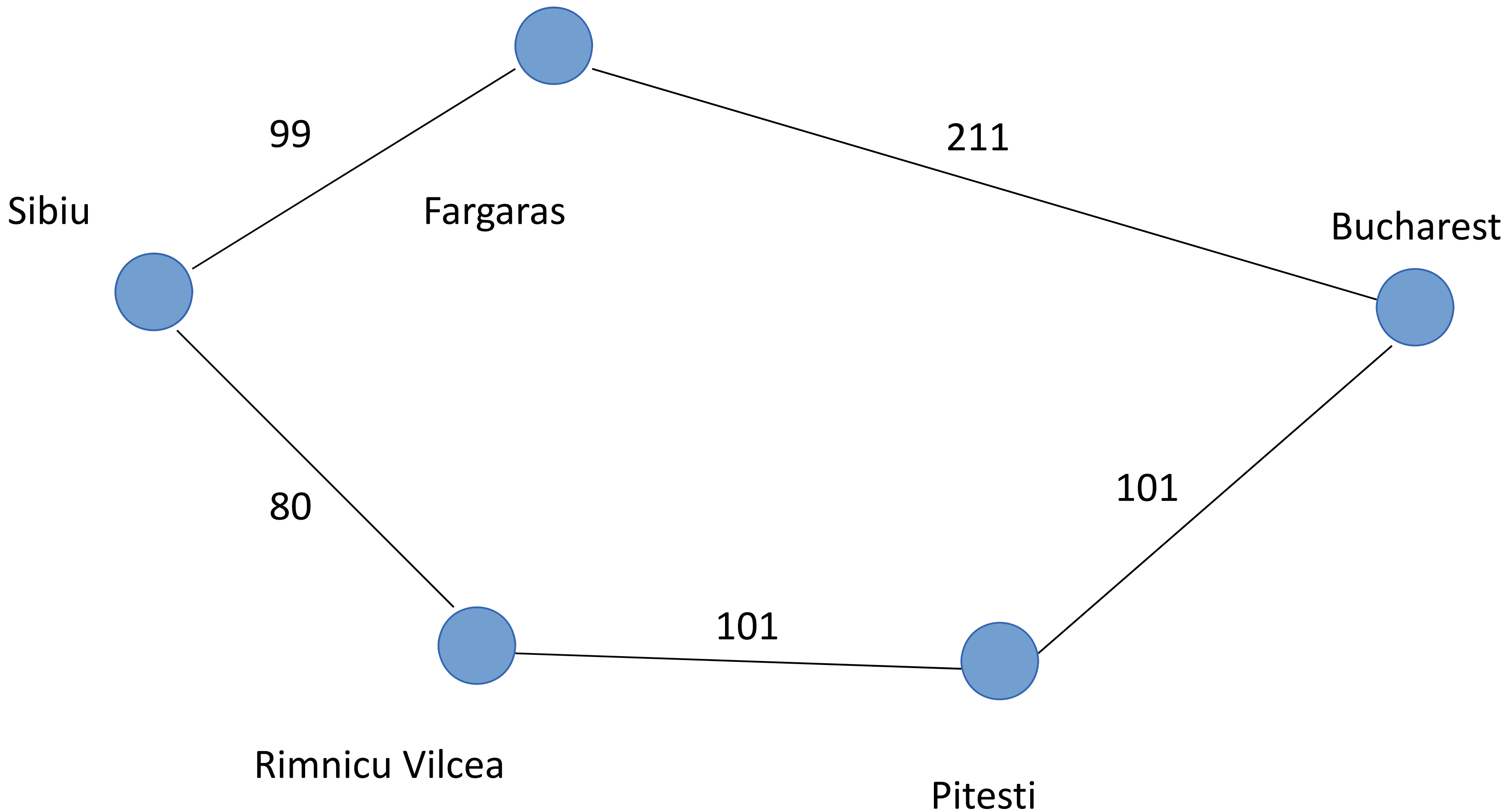
Open: [R(80), F(99)]



Open: [F(99), P(181)]



Open: [P(181), B(310)]


99
211
Sibiu
Fargaras
Bucharest
80
101
101
Rimnicu Vilcea
Pitesti

Open: [B(282)]

Uninformed Search

- Uninformed?
- Means we don't know how "far" it is to the goal.
- Depth First
  - Uses a stack to represent open list.
- Breadth First
  - Uses a queue to represent open.
- Uniform Cost / Dijkstra (for a single goal)
  - Add nodes based on current cost of reaching the node.
  - i.e. use a priority queue. Similar to breadth first, but does not assume every step has the same cost.
- Variations
  - Depth Limited
  - Iterative Deepening
  - Bidirectional

Informed

- Suppose we have *some* idea how “far” the goal is away from each node?
- The idea is known as a **heuristic**.
- It is not assumed to be accurate.
- We could always expand a node that promises to get us closest to the goal.
- That’s called being **greedy**.
- Or we could also take into account the <u>actual cost</u> to get us as far as we have come.
- That’s known as **A* algorithm**

A*

- Data Structures:
  - start node
  - target node
  - successor function
  - For each node:
    - Path taken (e.g. pointer to prior)
    - Cost from start: $g(n)$
    - Estimated cost to goal: $h(n)$,
  - open list (**ordered by cost function $f(n) = g(n) + h(n)$**)
  - closed set

A*

- Add start node to open list
- While open list is not empty
  - remove highest priority node (lowest **estimated** cost)
  - if node is goal, then <u>success</u>: return its solution path.
  - Else
    - place node on closed list
    - generate successors. For each successor:
      - Compute cost**: f(n) = g(n) + h(n)**
      - If successor is on open and new g(n) is better than old,
        - update entry on open
      - Else **if successor is on closed and new g(n) is better than old,**
        - **remove from closed and add to open with new f(n)**
      - Else if not previously seen
        - add to open.
- If the queue becomes empty then there was no solution.

Admissibility

- A heuristic is called **admissible** if it is guaranteed to be an underestimate of the actual distance for all cases
- An admissible heuristic will result in A* being optimal.

A* in Problem Solving

- 15-Puzzle
  - Heuristics
    1) How many tiles are out of position?
    2) What is the total *distance* that tiles are out of place?
- Path-finding
  - Heuristics
    1) Crow flies distance
    2) Manhattan distance.

Triangle Inequality

- **Consistent** (aka **monotonic**)
- A heuristic function h is consistent if
  - Given nodes n1 and n2
  - And their heuristic costs h(n1) and h(n2)
  - Together with the actual cost to go from n1 to n2, c(n1, n2)
  - Then: $h(n1) \leq h(n2) + c(n1, n2)$
- If a heuristic function is consistent then when a node is placed on the closed set, it never has to move back to the open set.
- Admissible functions are *almost always* consistent.
- But games like to use inadmissible functions.

Data Structures for Open Set

	Insert	Membership	Get/Remove Best	Adjust
Unsorted Array/List	O(1)	O(F)	O(F)	O(F)
Sorted Array	O(F)	Binary search: O(log F)	Keep at end: O(1)	Find: O(log F) Change: O(F)
Sorted Linked List	Find: O(F)	O(F)	O(1)	O(F) for find. O(1) to adjust
Indexed Array	O(1)	O(1)	O(N)	O(1)
Hash Table	O(1)	O(1)	O(N)	O(1)
Heap	O(log F)	O(F)	O(log F)	O(F) for find O(log F) to adjust
Heap + Indexed Array	O(log F)	O(1)	O(log F)	O(F)