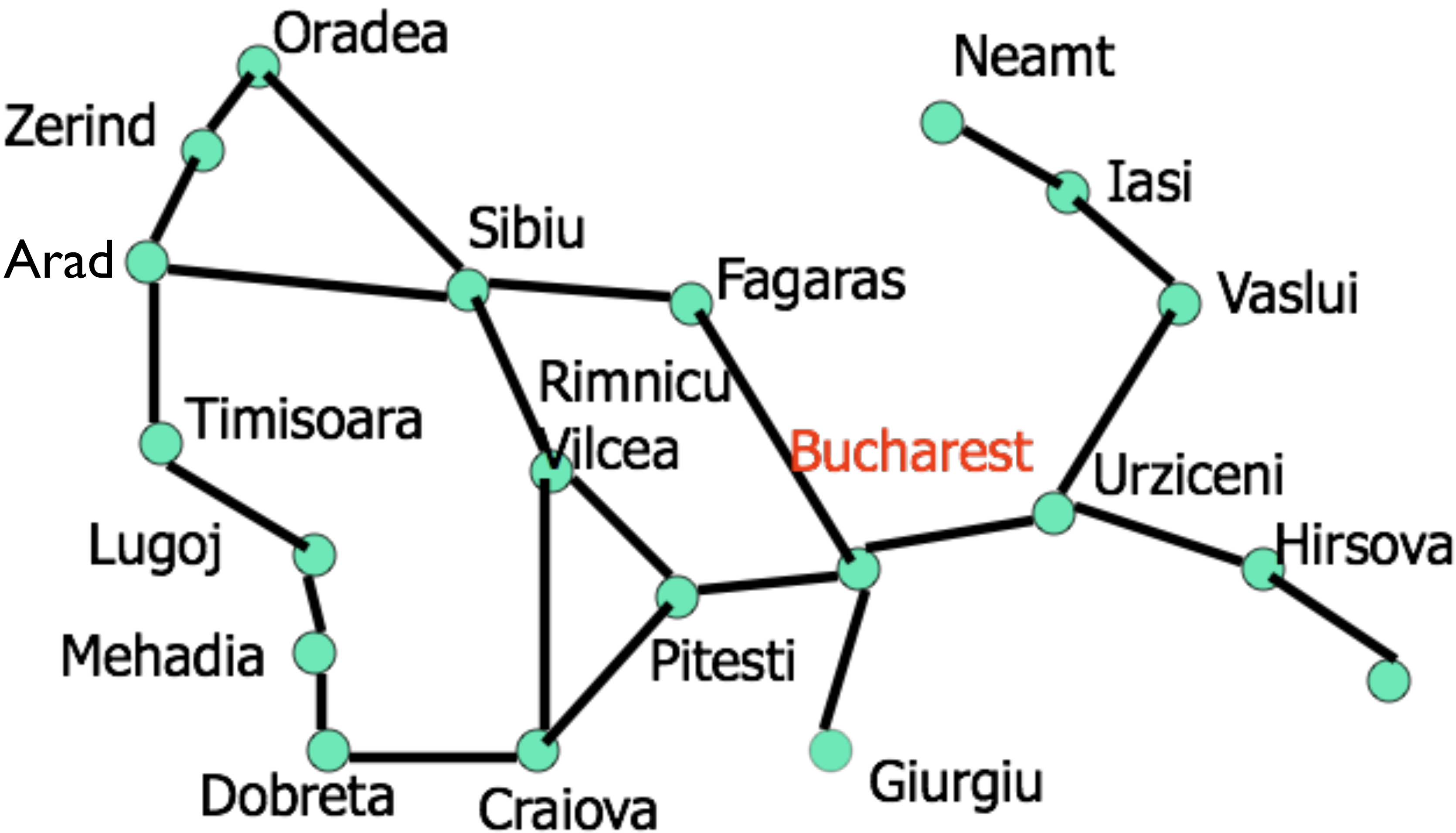
Search algorithms: UCS and A*

Devika Subramanian

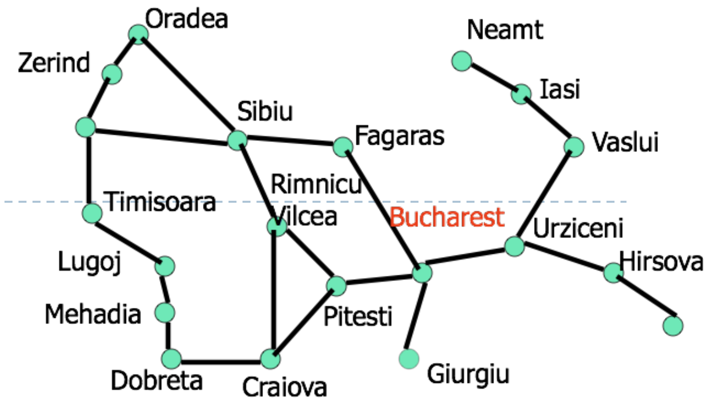
Route planning

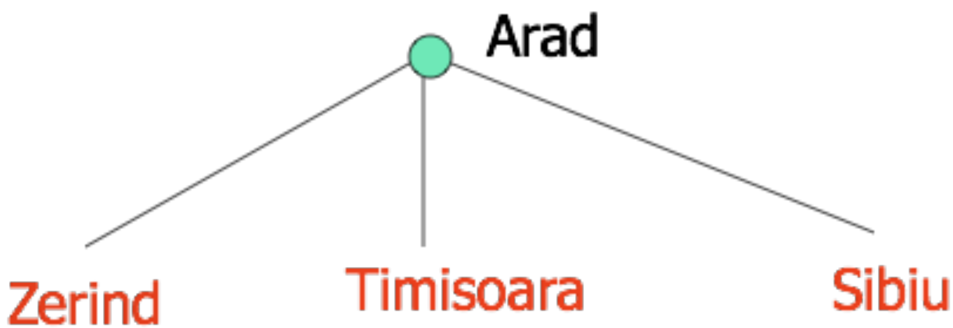


Need for search algorithms

- Dynamic programming takes time proportional to the square of the size of the state space.
  - It finds shortest paths to a goal (e.g., Bucharest) from every node in the state space (e.g., every city in Romania).
- What if all we care about is getting between a given node (e.g.,Arad) and a goal node (e.g., Bucharest)?
  - Can we solve this problem in time proportional to the size of the state space?
- This is what search algorithms are for: given a start state, a goal state and a state space graph, find a path between the two states.

Search tree





FRONTIER
OF THE
SEARCH
TREE

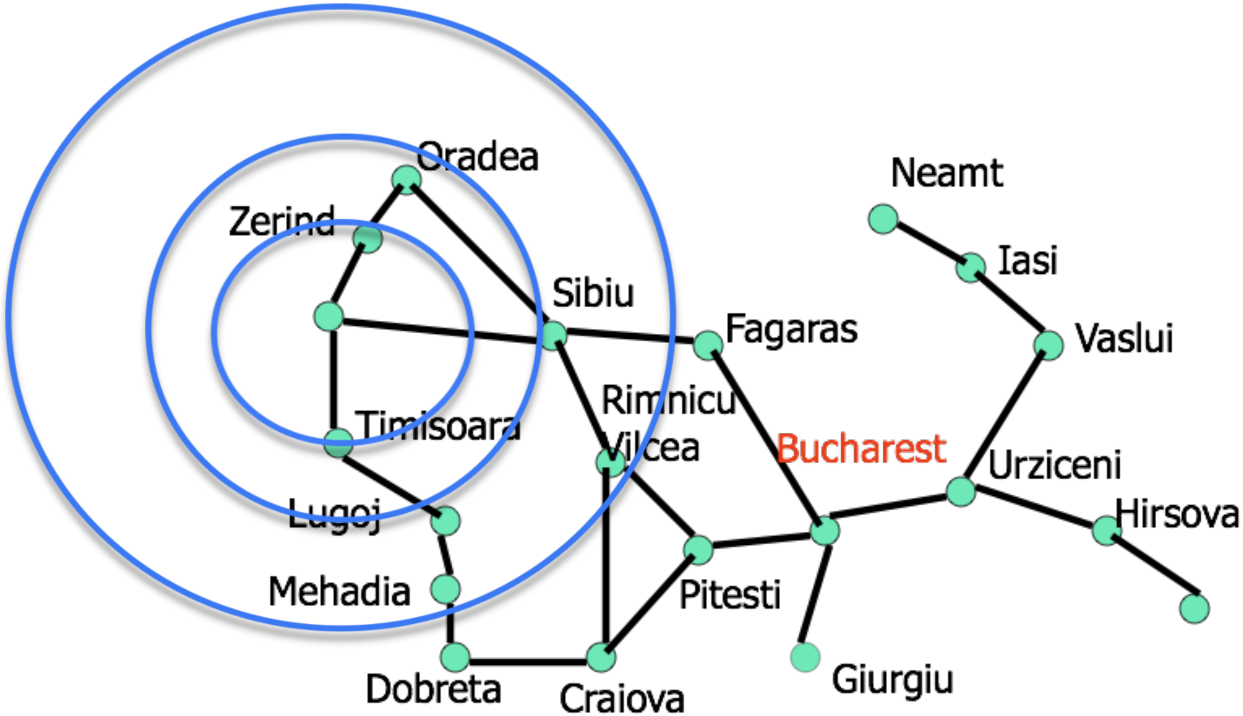
A tree is a graph in which any two vertices are connected by exactly one path.

- A what-if tree of plans and their outcomes
- The root node is the start state, children correspond to successor states
- A search frontier and visited list are maintained
- For real problems, we never build the entire tree!

Search algorithms

- Which node in the frontier of the search tree to expand next?

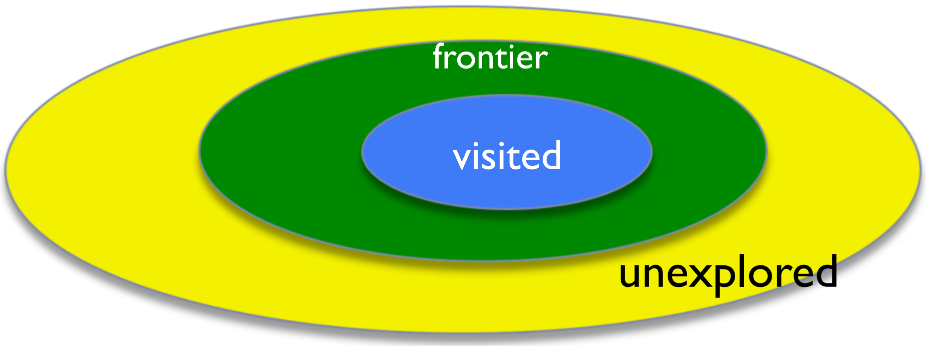
Uniform cost search (visual)



Expands nodes in order of increasing cost from start state. g(n) = distance from start to n

Uniform cost search

- Visited nodes = states for which optimal path from start state is known (key invariant maintained by algorithm)



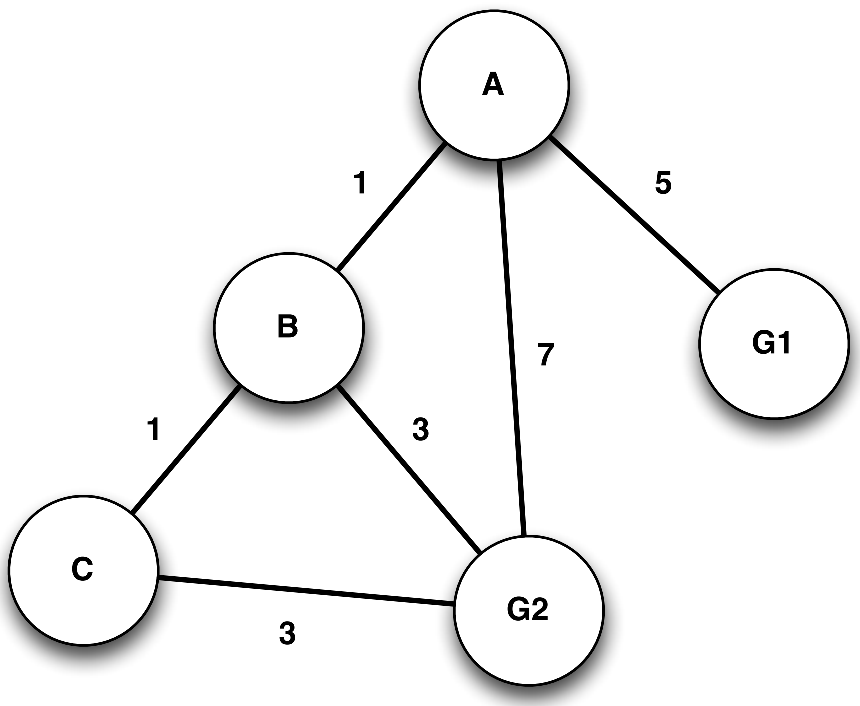
- Unexplored nodes = states which have not yet been generated
- Frontier nodes = nodes separating the explored from the unexplored, whose successors have not been generated
- When goal node moves into the visited list, we have found an optimal path from start to goal.

Edge costs cost(current, nbr) are assumed to be >= 0

UCS (uniform cost search)

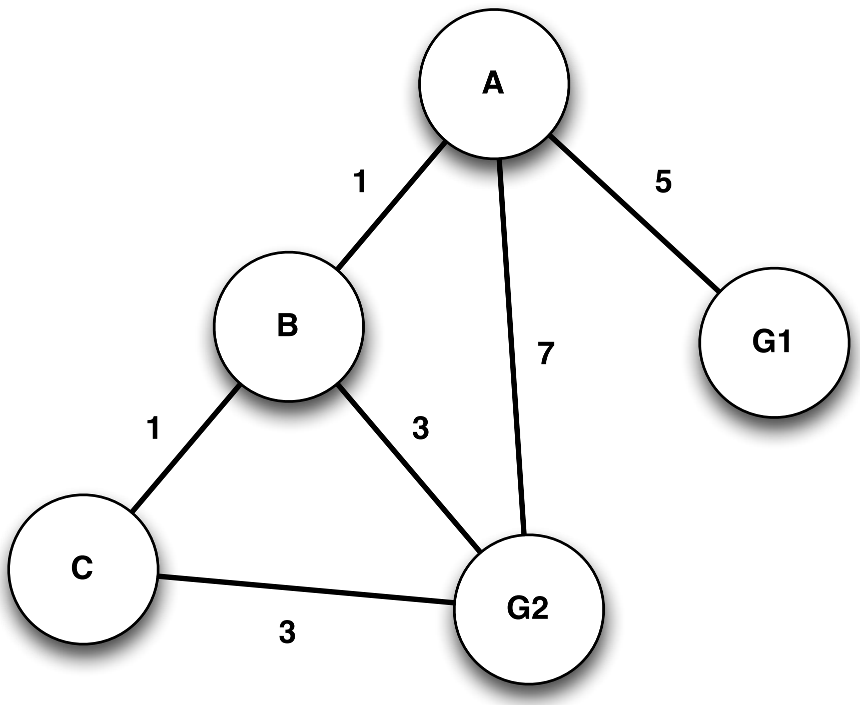
- Function UCS(start, goal, graph, frontier) returns True or False
  - Insert (start,0) into the frontier (priority queue ordered by cost from start).
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - (current,c) = **pop** node from frontier with lowest cost
    - add node (current,c) to the visited list
    - If current == goal, return True.
    - for every nbr of current node
      - If (nbr,c') not in frontier or in visited
        - insert (nbr,c+cost(current,nbr)) into frontier
      - Else if c' > c + cost(current,nbr)
        - Insert (nbr,c+cost(current,nbr)) into frontier
        - Remove (nbr,c')
  - return False.

Uniform Cost Search example



- G1 and G2 are goal nodes, and A is the start node.

Uniform Cost Search Example



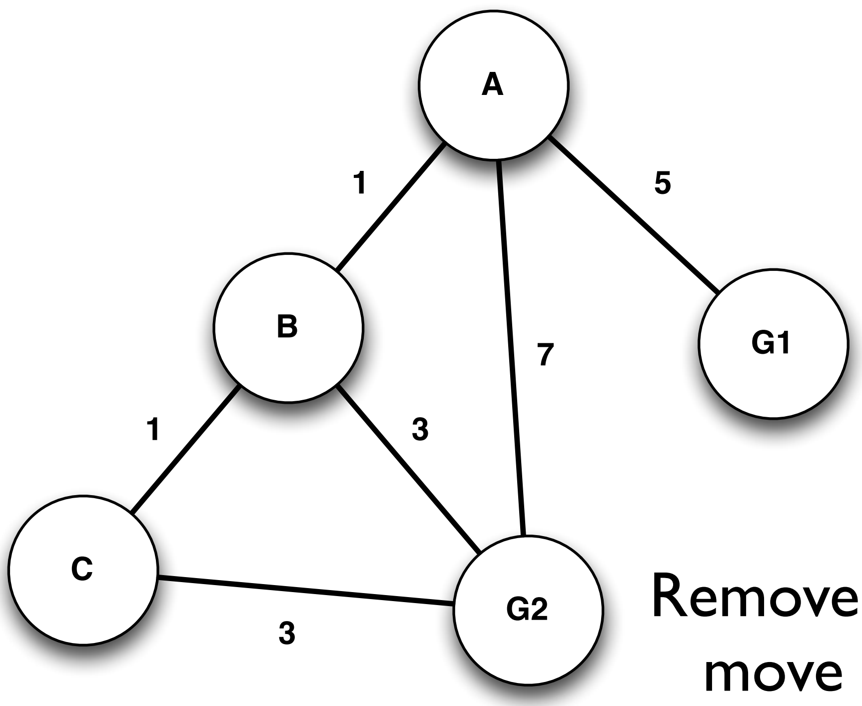
Frontier

Visited

(A,0)

Frontier starts with (A,0).
Visited is empty.

Uniform Cost Search Example

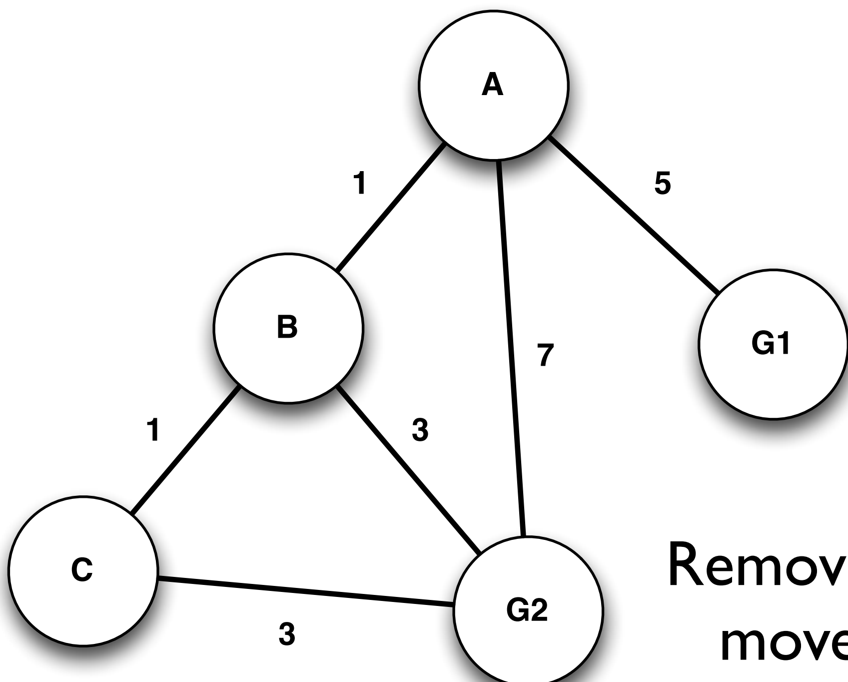


Frontier	Visited
(B,1)	(A,0)
(G1,5)	
(G2,7)	

Remove (A,0) from frontier and
move it to Visited.

For every nbr of A
insert (nbr,0+cost(A,nbr)) into frontier

Uniform Cost Search Example



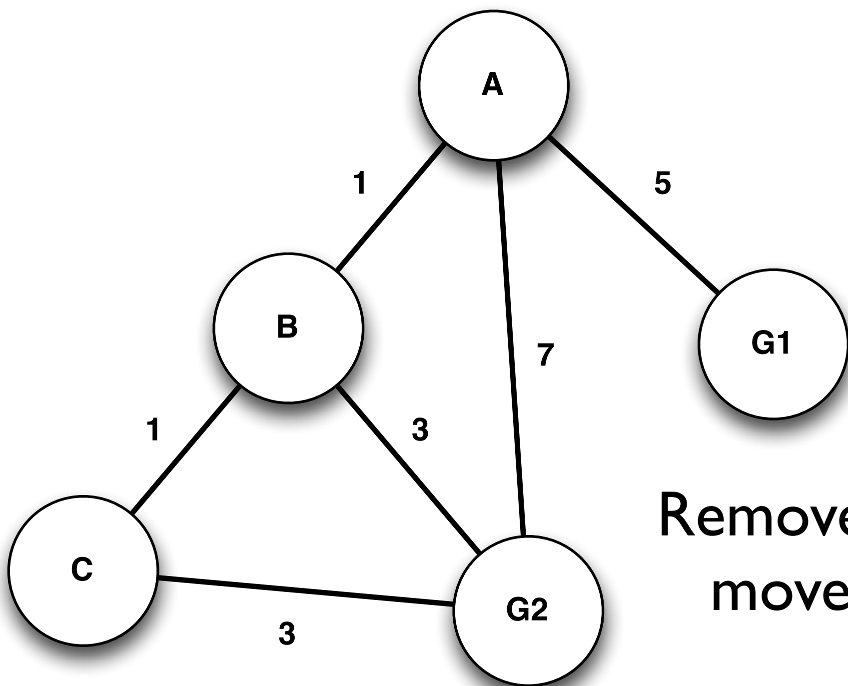
Frontier	Visited
(C,2)	(A,0)
(G2,4)	(B,1)
(G1,5)	

Remove (B,1) from frontier and move it to Visited.

For every nbr of B
insert (nbr,1+cost(B,nbr)) into frontier

Note: (G2,7) gets deleted from frontier because we have found a cheaper way via B: (G2,4)

Uniform Cost Search Example



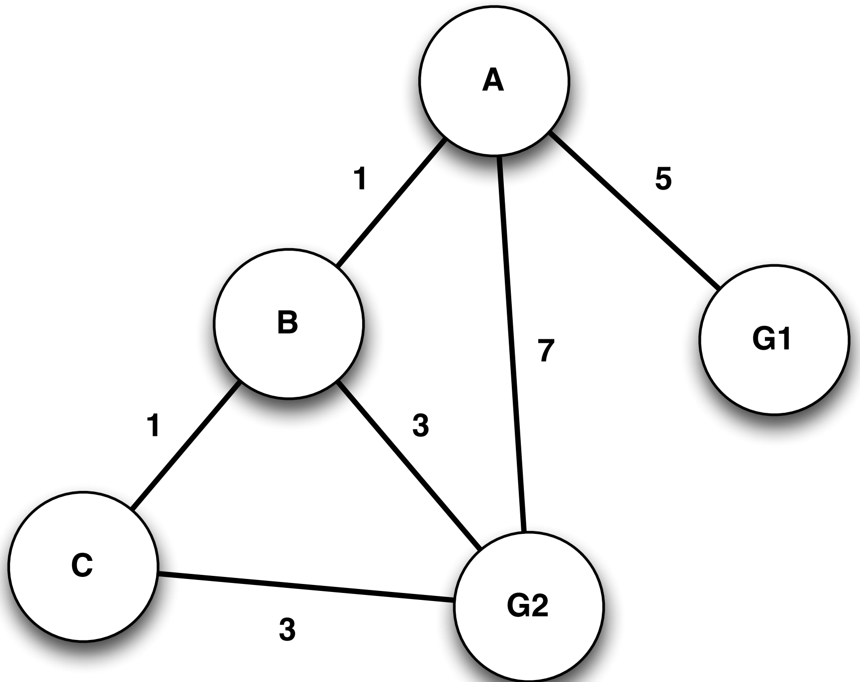
Frontier	Visited
(G2,4)	(A,0)
(G1,5)	(B,1)
	(C,2)

Remove (C,2) from frontier and
move it to Visited.

For every nbr of C
insert (nbr,2+cost(C,nbr)) into frontier

(G2,5) is not added because (G2,4) is in the frontier.

Uniform Cost Search Example



Frontier	Visited
(G2,4)	(A,0)
(G1,5)	(B,1)
	(C,2)

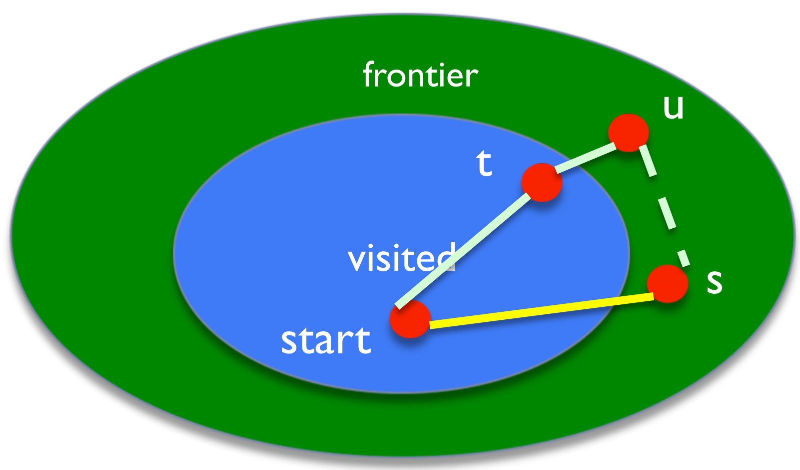
(G2,4) is the top node and it is a goal node.
Path found! A—B—G2

Properties of uniform cost search

- It is complete: if a path exists, uniform cost search will find it.
- It is optimal: uniform cost search will find a least cost path from start node to a goal node, if one exists.
- Time complexity = O(size of state space)
- Space complexity = O(size of state space)

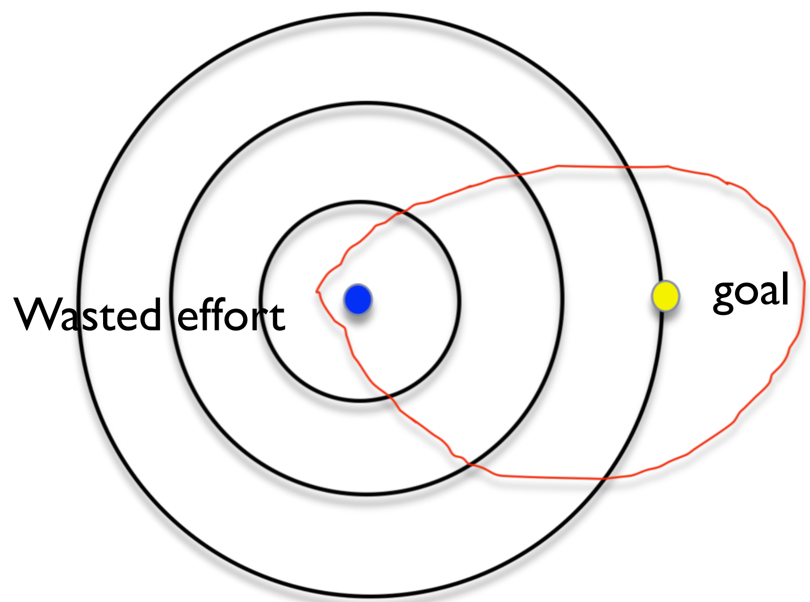
Optimality of UCS

- Theorem: when a state s is popped off the frontier, the cost associated with it is the least cost from the start state to s.
- Proof:



Informed search

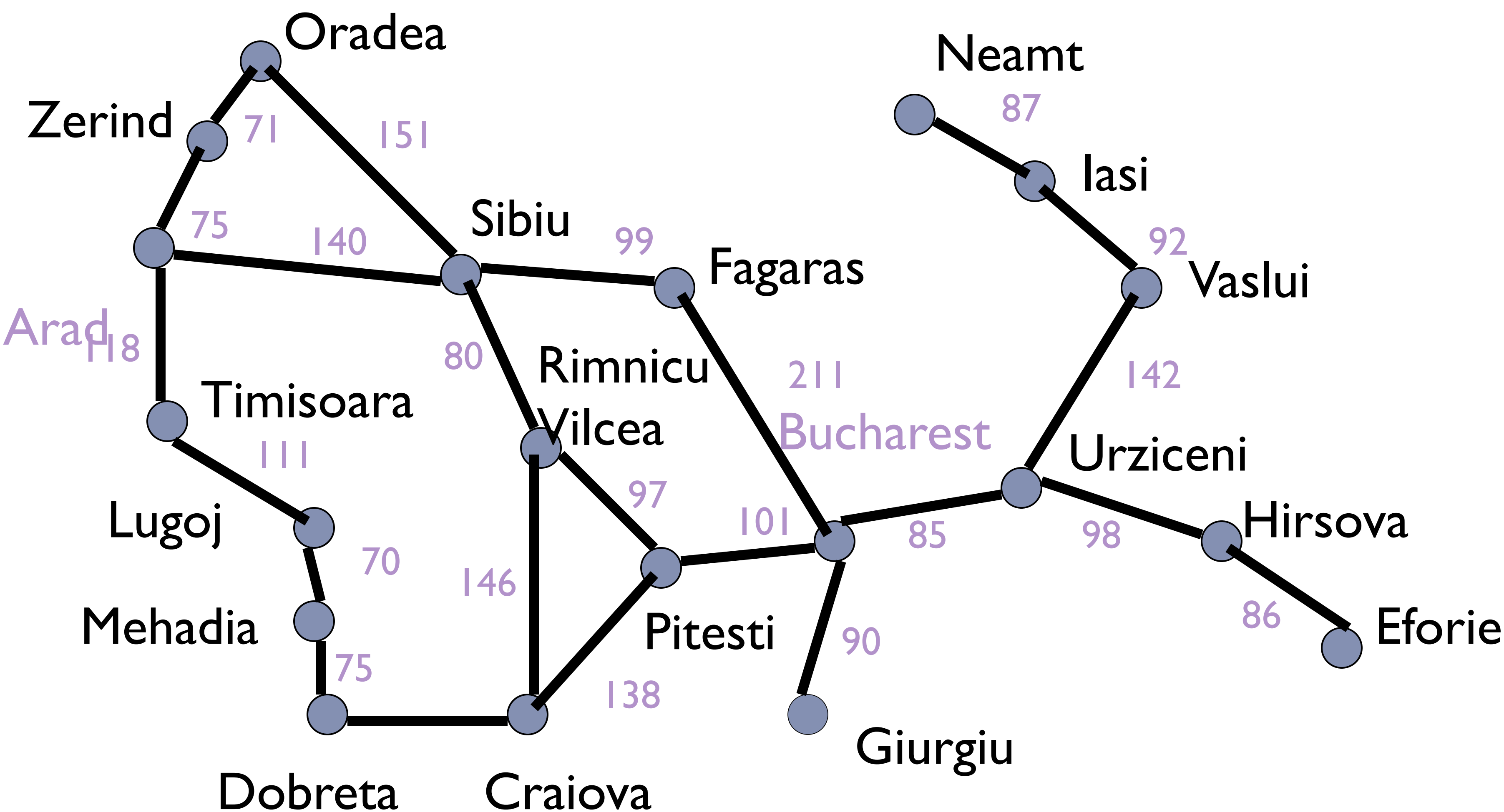
- Informed search: use estimates of distance to goal states to direct search.



Informed search: greedy search

- Idea: minimize estimated cost to reach goal.
- Define h(n) = estimated cost of the cheapest path from node n to a goal state. h(n) is called the heuristic function.
- We require $h(n) \geq 0$ for all nodes n, and h(g) = 0 for all goal nodes g.
- h(n) is problem-specific. Example: in maze navigation, Manhattan distance to goal.

Romania problem revisited

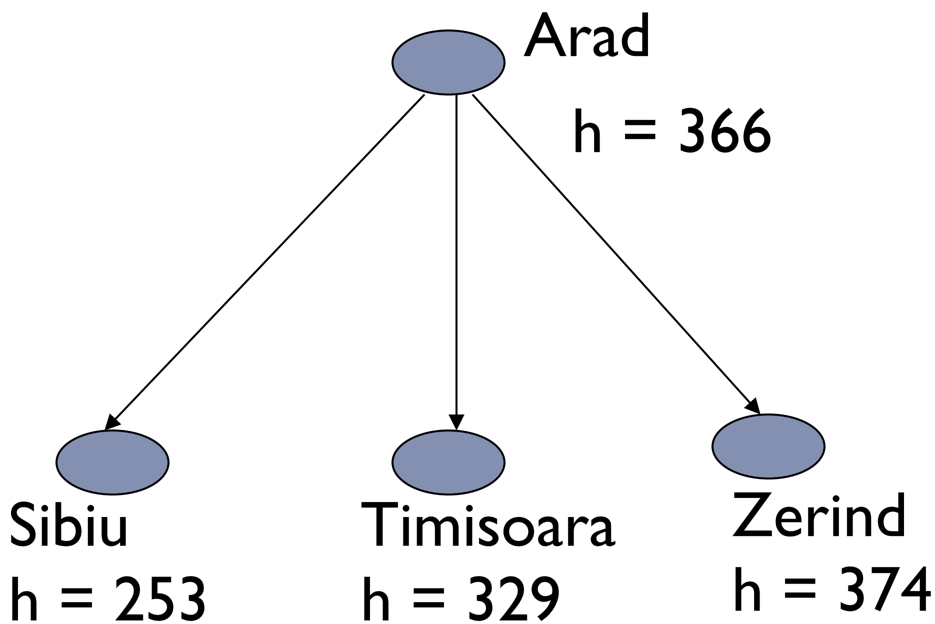


Heuristic distance estimates for the Romania problem h(n)

- 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

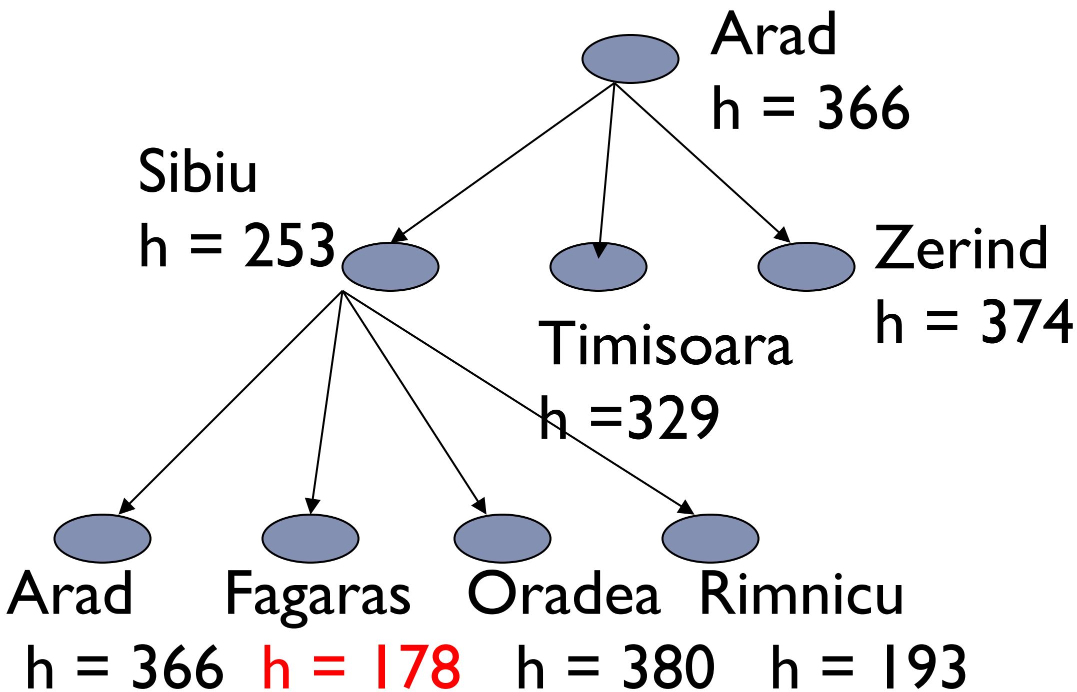
  Mehadia: 241
  Neamt: 234
  Oradea: 380
  Pitesti: 98
  Rimnicu Vilcea: 193
  Sibiu: 253
  Timisoara: 329
  Urziceni: 199
  Zerind: 374

Greedy search in action



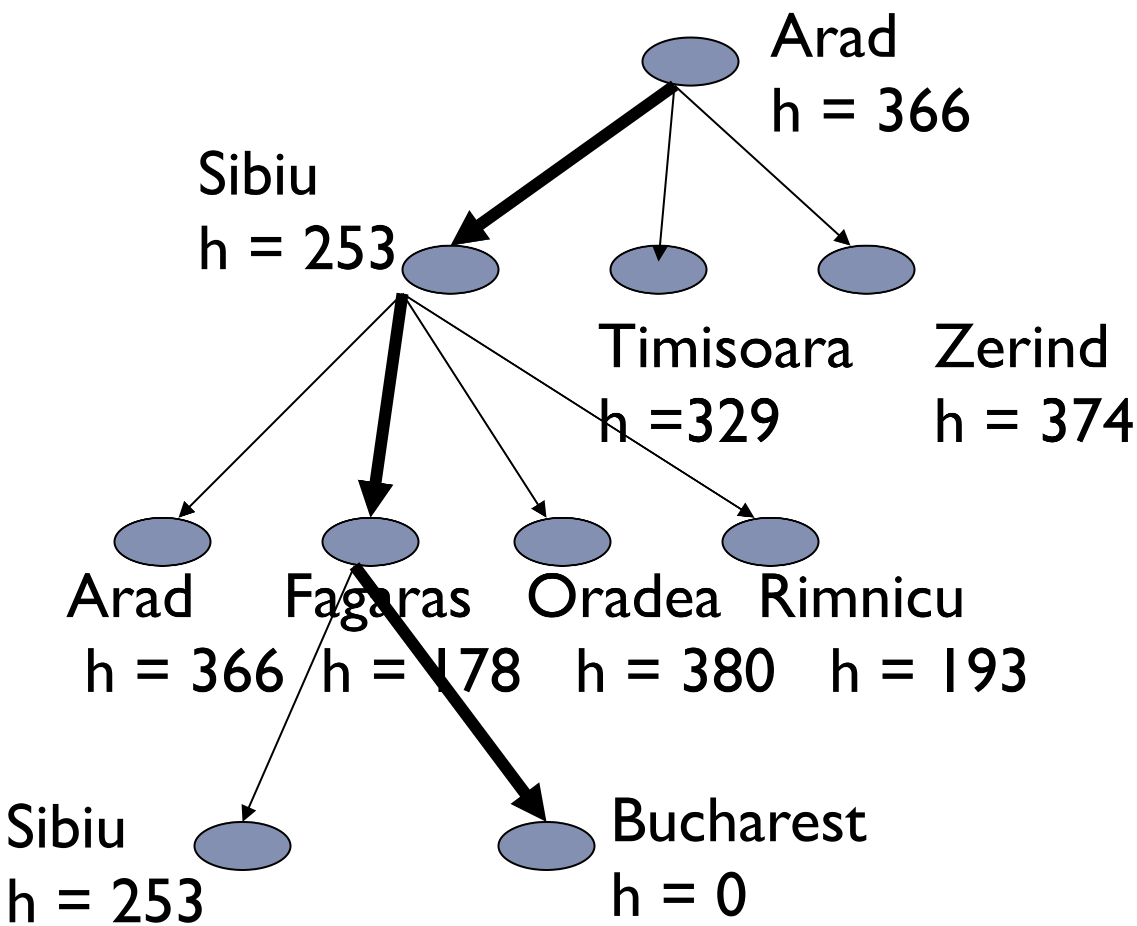
Now expand Sibiu and add its successors to search tree.

Greedy search in action (contd.)



Fagaras has the lowest h, so it is the next node to be expanded.

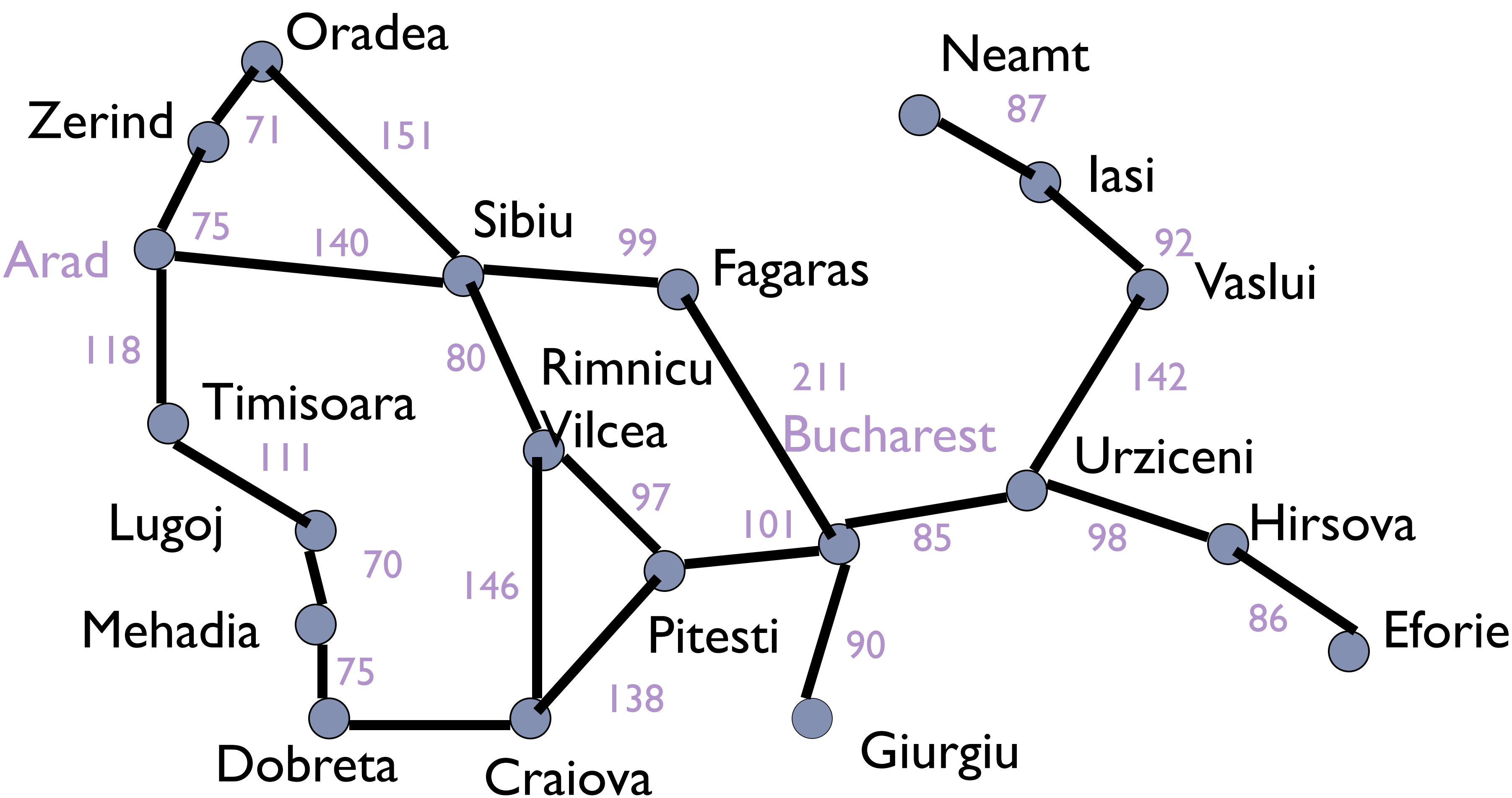
Greedy search in action (contd.)



Best-first is SUB-OPTIMAL!

Goal generated!
Path found is
Arad → Sibiu → Fagaras → Bucharest

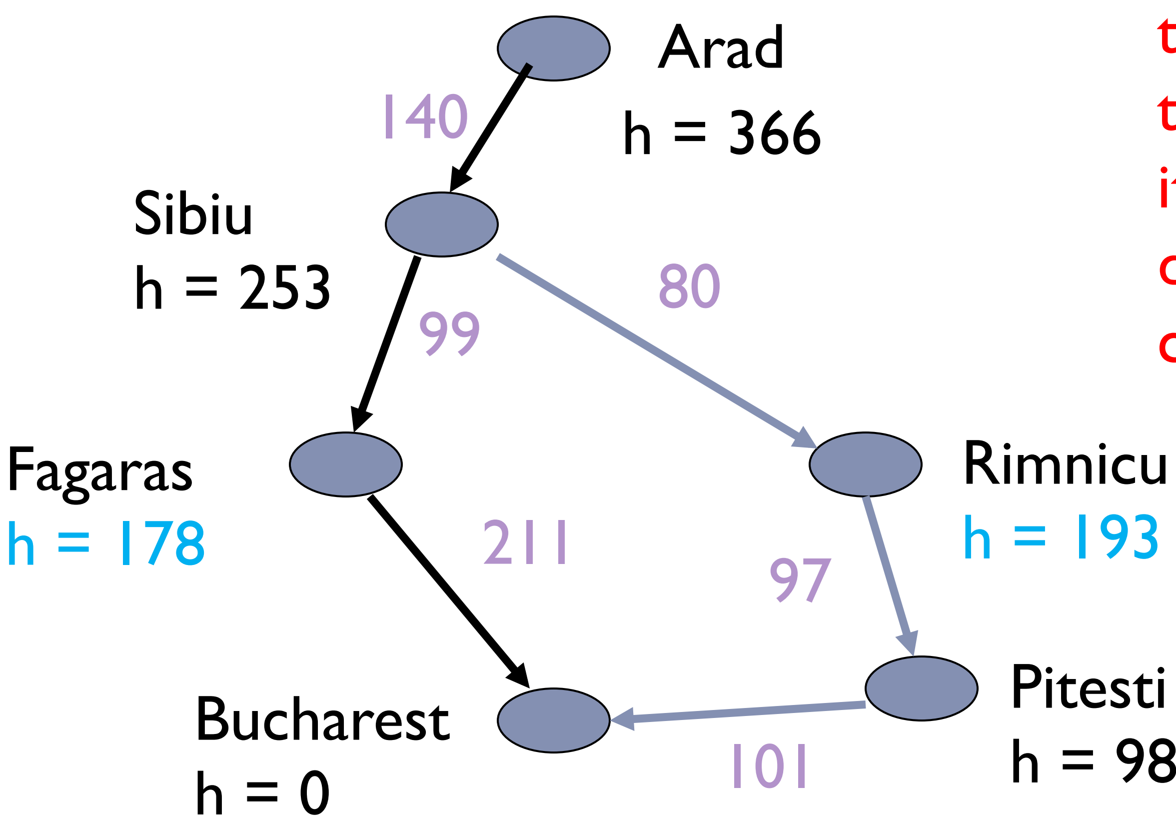
Romania problem revisited



Properties of greedy search

- It is complete, if visited list is maintained.
- It is not optimal.
- Time complexity: O(size of the state space).
- Space complexity: O(size of the state space).
- Actual performance of greedy search is a function of the accuracy of h(.).

What's wrong with greedy search

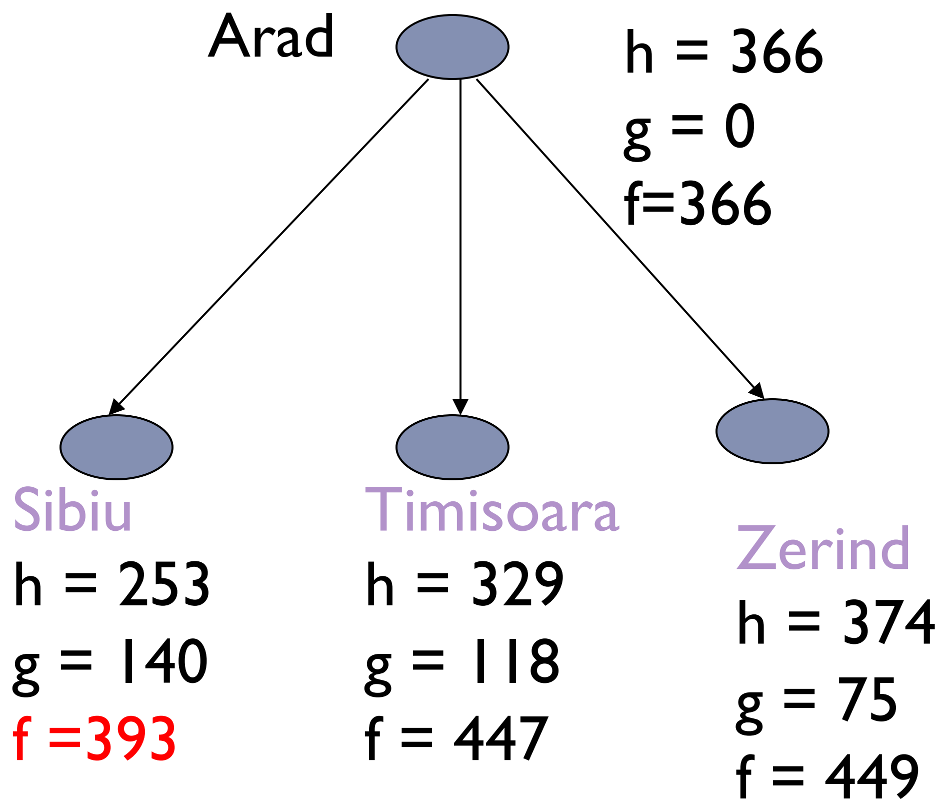


Fagaras looks better to greedy search than Rimnicu because it does not take distance already covered into account!

A* search

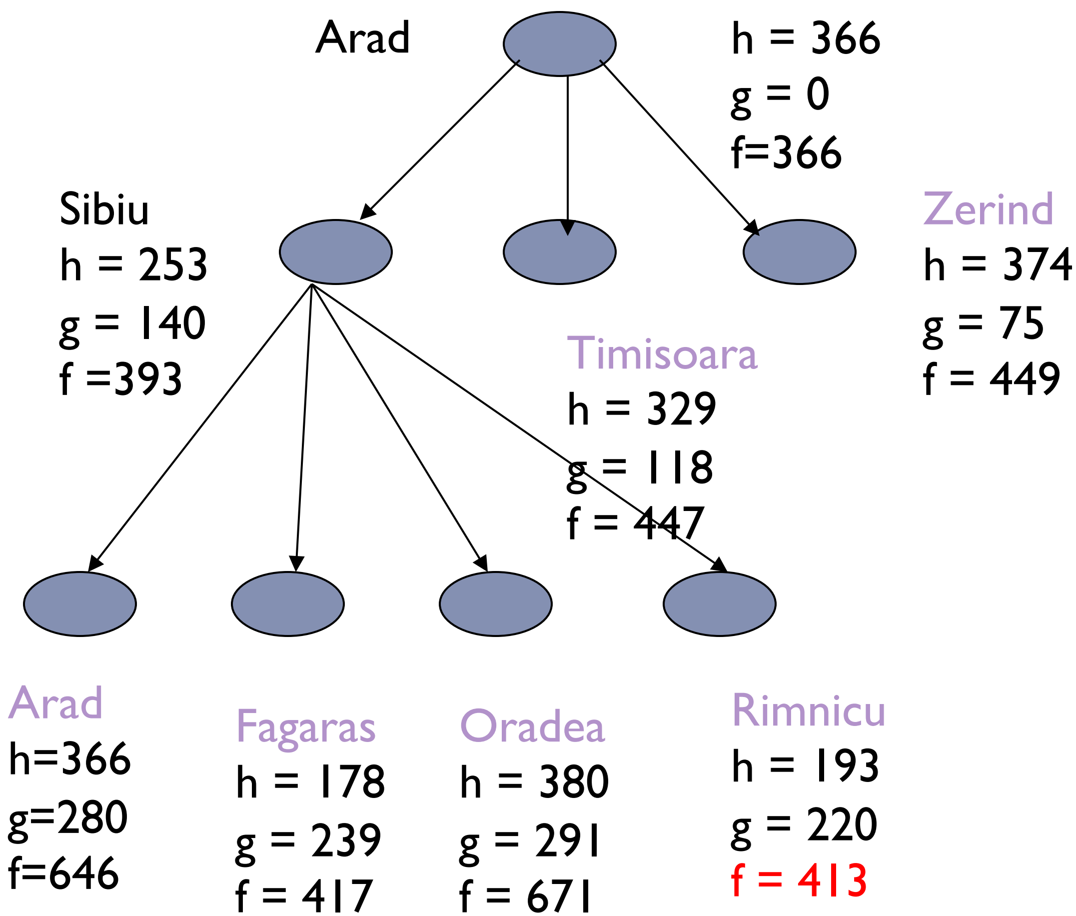
- Uses estimated cost of the cheapest solution path through node n, as a measure of the merit of node n.
- $f(n) = g(n) + h(n)$ where
  - $g(n)$ = actual path cost from start node to node n.
  - $h(n)$ = estimated cost of path from n to closest goal node.
- A* additively combines uniform cost search ($g(n)$) and greedy search ($h(n)$).

A* in action



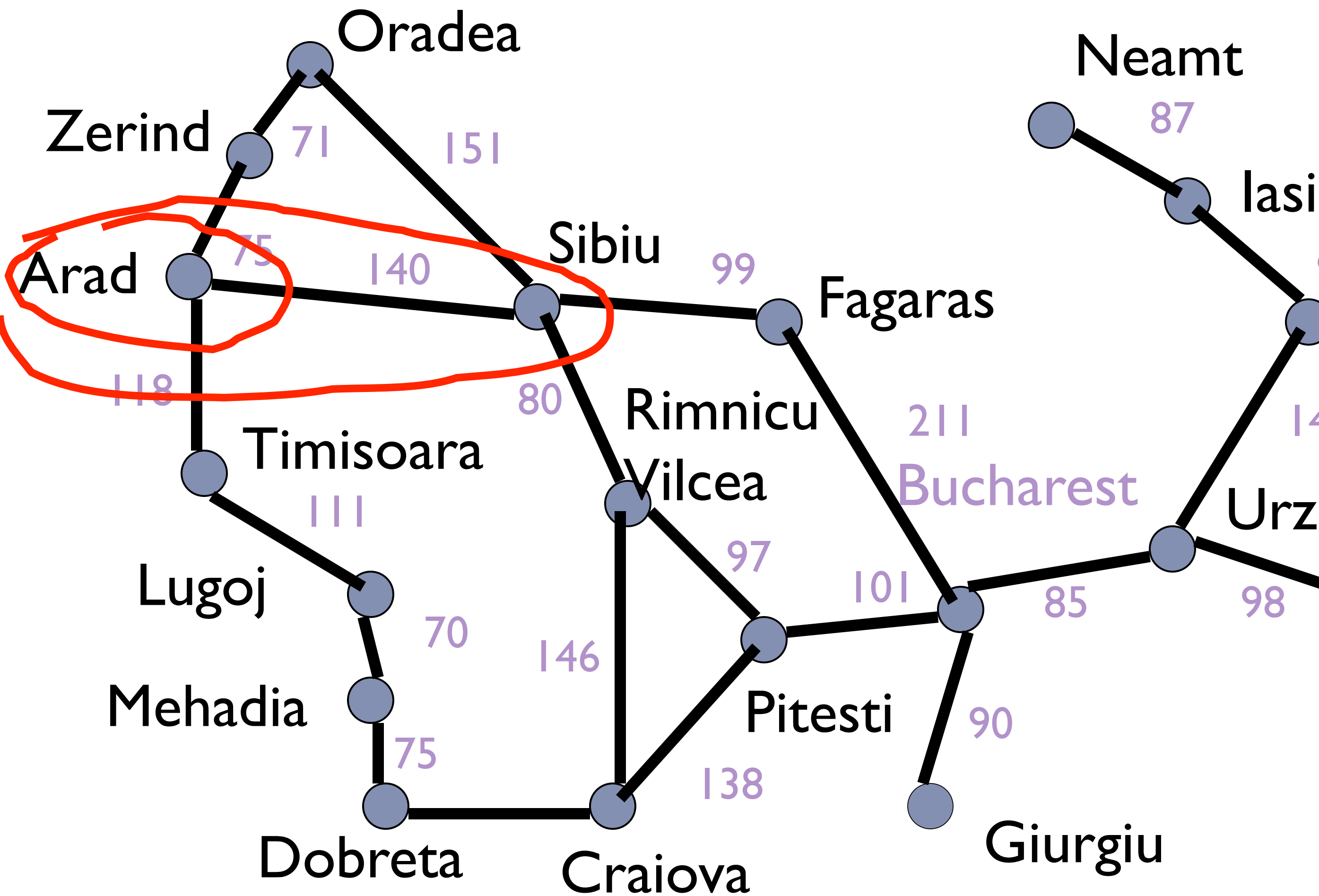
Sibiu will be expanded next.

A* in action (contd.)

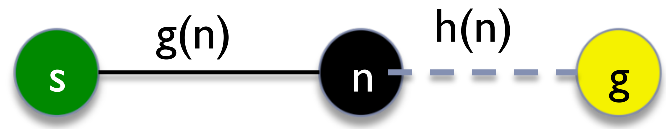


Rimnicu will be expanded next.

How A* searches



Contour lines of equal f values

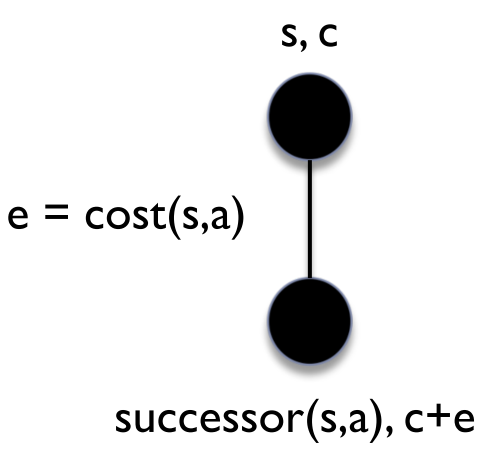


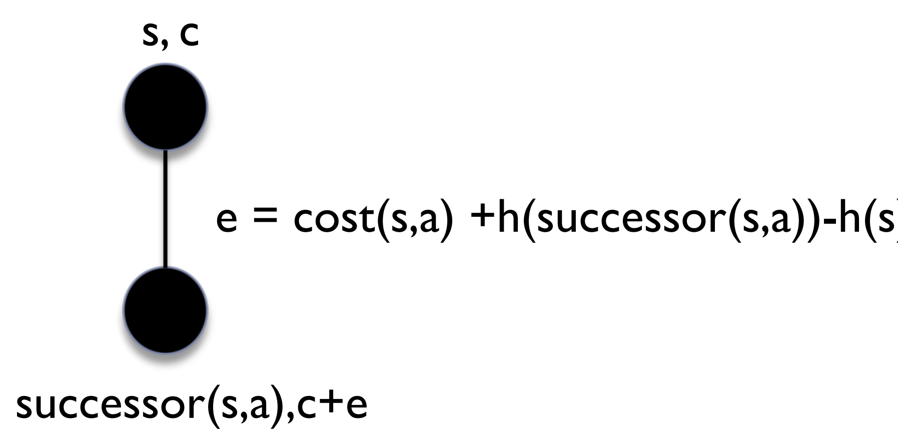
The A* algorithm

- Function A*(start, end, graph, frontier) returns True or False
  - Insert (start,0+h(start)) into the frontier (priority queue ordered by f() = h()+g()).
  - Initialize the visited list to empty.
  - while frontier is nonempty:
    - (current,c) = **pop** node from frontier with least f-cost
    - add (current,c) to the visited list
    - If current == end, return True.
    - for every nbr of current node
      - If (nbr,c') not in frontier or visited
        - **insert** (nbr,c+cost(current,nbr)+h(nbr)-h(current)) into frontier
      - Else if c' > c+ cost(current,nbr)+h(nbr)-h(current)
        - **insert** (nbr,c+cost(current,nbr)+h(nbr)-h(current)) into frontier
        - Remove (nbr,c')
  - return False.

A* and UCS

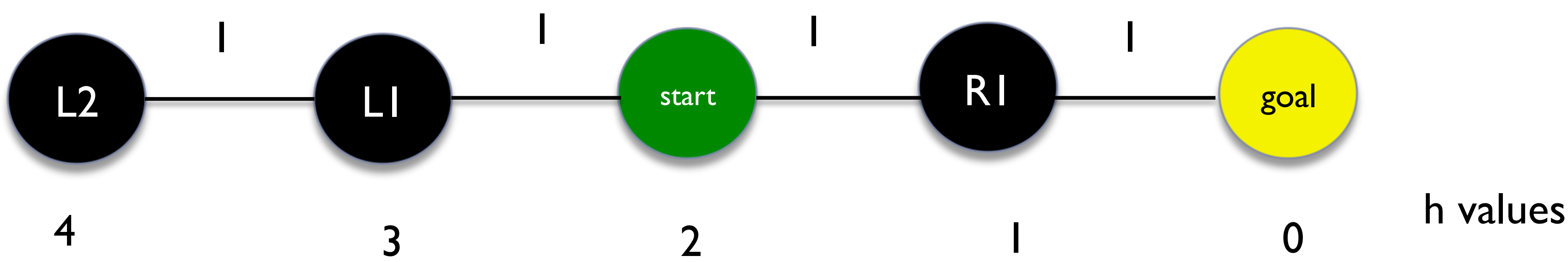
- We can simulate A* with UCS by
  - A* has priority function f(n) = g(n)+h(n) while UCS has priority function g(n). Cost of start node for UCS is 0, for A* is h(start)
- Simply modify the edge cost: cost(s,a) by
  - cost(s,a) + h(successor(s,a)) – h(s)





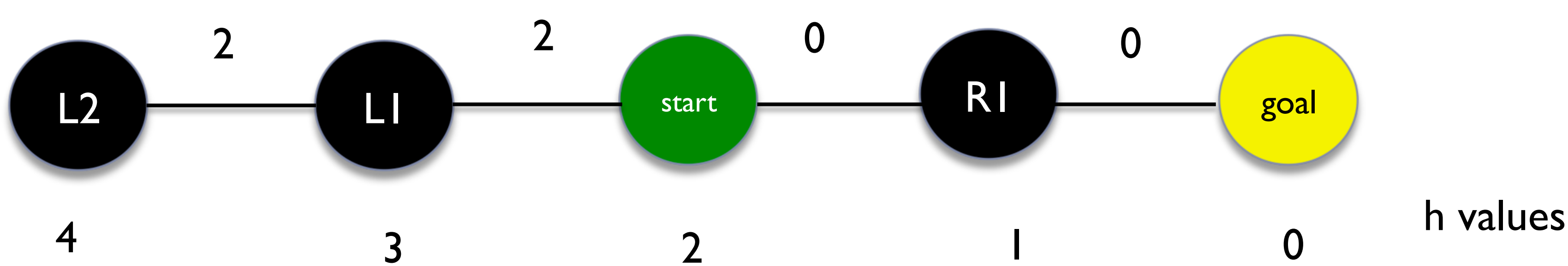
A* vs UCS example

UCS view



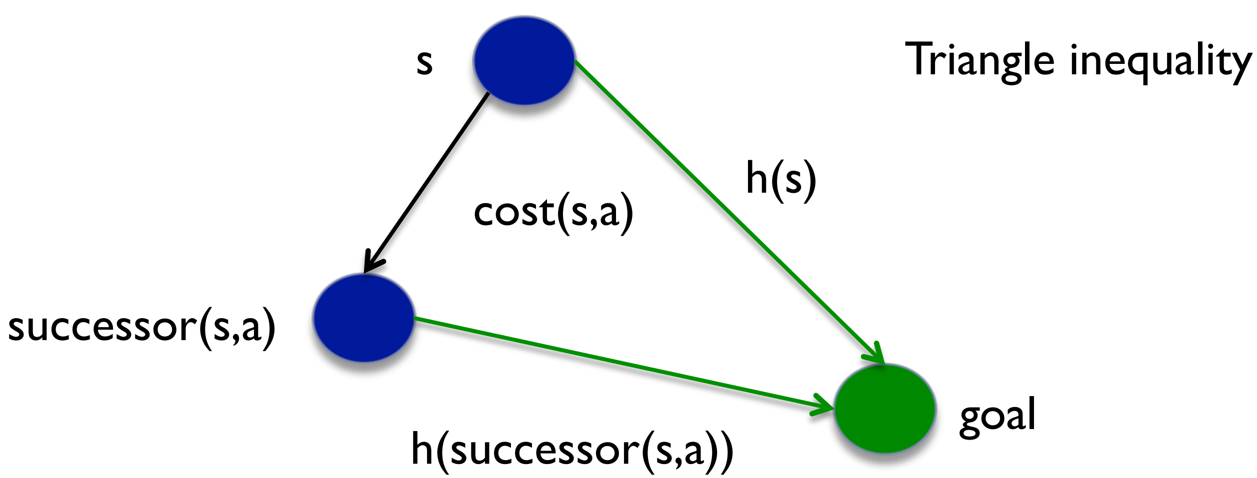
cost(s,a) + h(successor(s,a)) – h(s)

A* view



Consistent heuristic

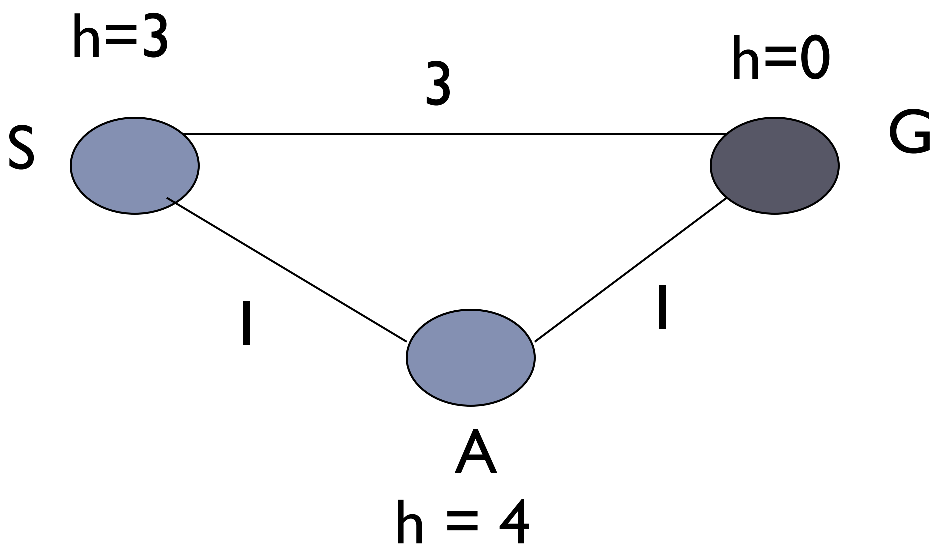
- A heuristic h is consistent if
  - cost(s,a) + h(successor(s,a)) >= h(s)
  - h(goal) = 0
- A* with a consistent heuristic is guaranteed to find the shortest path between a start and goal state.



A* properties

- It is complete.
- It is optimal provided modified edge costs >= 0
  - cost(s,a) + h(successor(s,a)) – h(s) >=0, i.e., h is consistent
- Time complexity: O(size of state space)
- Space complexity: O(size of state space)
- A* is optimally efficient – there is no algorithm that expands fewer nodes than A* with a given h that guarantees completeness and optimality.
  - A* expands all nodes n with the property that f(n) <= cost of optimal path between start and goal
- A* runs out of memory before it runs out of time.

Is this h consistent?



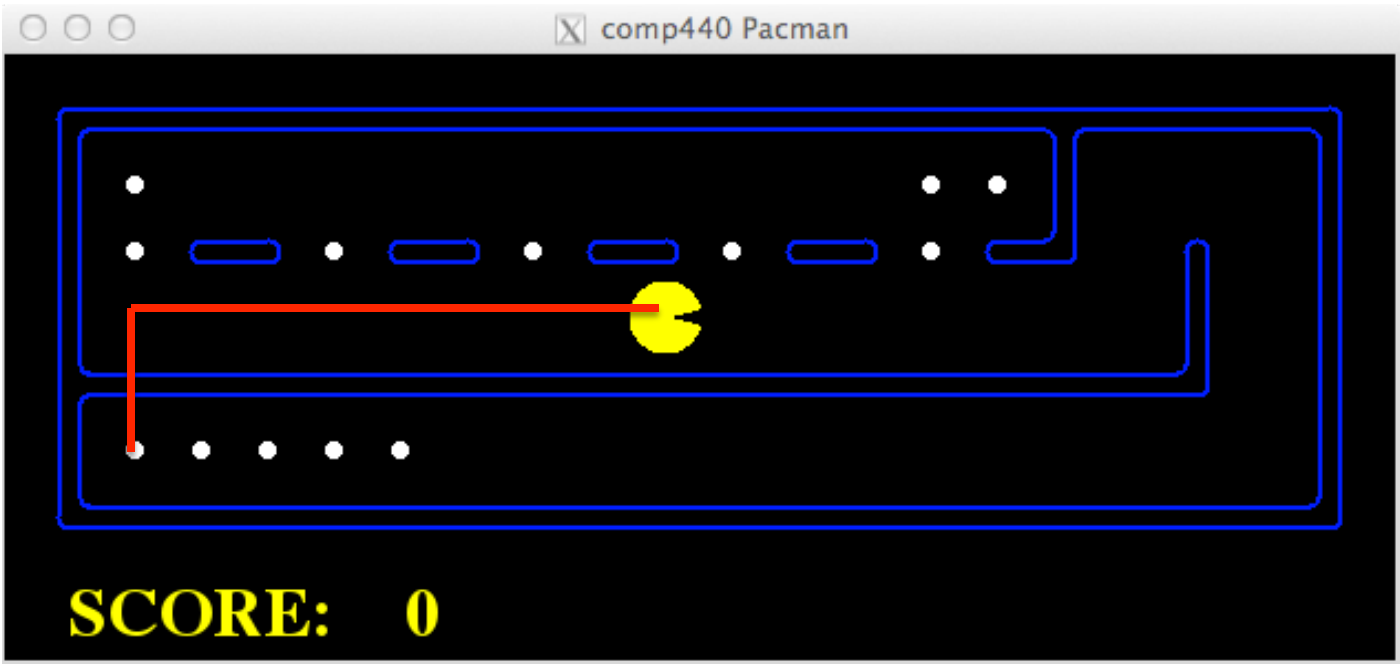
Admissible heuristic

- Let h*(n) = the true minimal cost to goal node from n.
- We will call h an admissible heuristic if $h(n) \leq h^*(n)$ for all nodes n.
- An admissible heuristic never overestimates the remaining distance to the goal.
- An admissible heuristic is optimistic.
- Designing admissible heuristics is where the work is in using A*.

Consistency and admissibility

- If a heuristic h(n) is consistent, then it is admissible.
- Proof: exercise!

Example of a consistent heuristic



Relax constraints on the original problem. Knock down walls!
A consistent heuristic for the original problem is an exact solution for the relaxed problem. Here h(n) = Manhattan distance from n to goal.

Relaxed problem

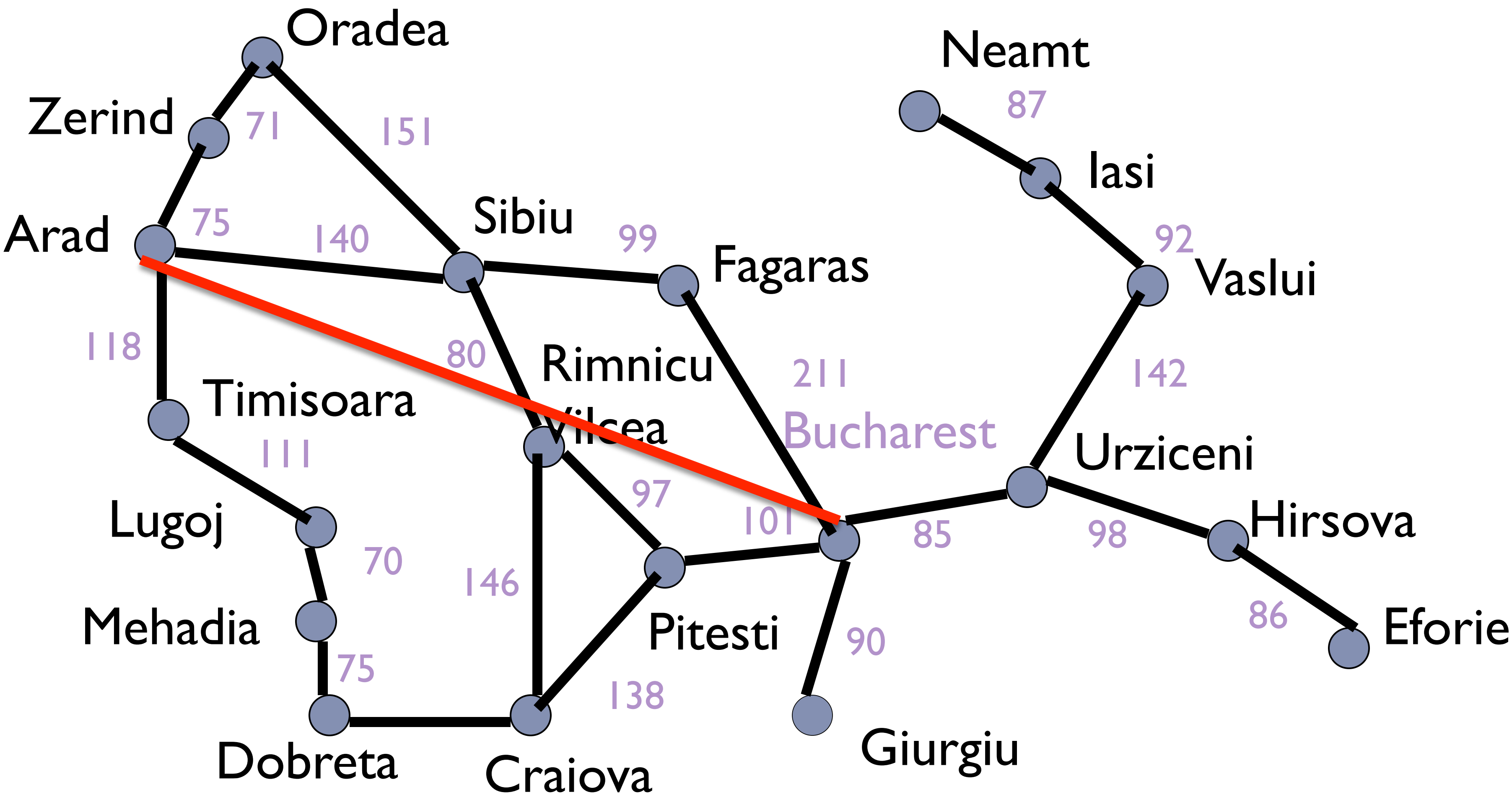
- P' is a relaxation of search problem P if P and P' have the same states and actions (same state graph), and edge costs in P' are lower that those in P
  - cost'(s,a) <= cost(s,a), for every s,a

- Given a relaxed search problem P', the relaxed heuristic h(n) for P is the shortest path from n to g in the graph for P' with reduced cost. It is a consistent heuristic for P.
  - h(s) <= cost'(s,a)+h(successor(s,a)) [triangle inequality]
  - h(s) <= cost(s,a)+h(successor(s,a)) [relaxation]

Designing consistent heuristics

- Consistent heuristics are often solutions to relaxed versions of the original problem.
  - Manhattan distance in a maze is a relaxed version of the original problem where we allow the agent to move through maze walls.
  - Euclidean distance in route planning is a relaxed version of the original problem where we allow the agent to travel in a straight line between two nodes regardless of whether there is a road between the nodes.
- Few general recipes for making consistent heuristics; many of them problem-specific and require deep understanding of the search space.

Straight line distance



Using relaxation to design h(n)

7	2	4
5		6
8	3	1

Start state

	1	2
3	4	5
6	7	8

Goal state

h(n) = number of moves to move from state n to goal state

Idea # 1: h(n) = number of tiles out of place

1. h(start) = ?
2. Relax original problem into a set of 8 independent subproblems.

How good is the heuristic?

- We measure effectiveness of a heuristic by comparing the number of nodes expanded by A* using that heuristic against the number of nodes expanded by UCS.
-

	8 step solution	12 step solution
A*+misplaced tiles	39	227
Uniform cost	6300	3.6×10^6

Slide adapted from P. Abeel

Using relaxation to design h(n)

7	2	4
5		6
8	3	1

Start state

	1	2
3	4	5
6	7	8

Goal state

$$h(\text{start}) = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$$

Idea # 2: h(n) = sum of Manhattan distance each tile has to move to get to final position

1. What problem relaxation is it based on?

How good is the heuristic?

- We can also compare the effectiveness of two heuristics by comparing the number of nodes expanded by A* using each heuristic.
-

	8 step solution	12 step solution
A*+total manhattan distance	25	73
A*+misplaced tiles	39	227

Slide adapted from P. Abeel

Another consistent heuristic

- h(n) = actual cost of moving from state n to goal state
- Is it a practical heuristic?
  - Tradeoff between work to estimate h(n) and the gains obtained in reduction of number of nodes expanded by A*

Combining heuristics

- If $h_1(s)$ and $h_2(s)$ are consistent heuristics, is $h_1(s)+h_2(s)$ consistent?
- If $h_1(s)$ and $h_2(s)$ are consistent heuristics, is $\max(h_1(s),h_2(s))$ consistent?

Summary

- Uniform cost search is complete, optimal, O(size of state space) in space and time complexity
- Informed search: using heuristics
  - Greedy search is complete, not optimal, O(size of state space) in space and time complexity
  - A* is complete, optimal (with consistent heuristic), O(size of state space) in space and time complexity. Is a special case of UCS with a modified edge cost function.
  - Actual performance: function of h(n)
- Heuristic design: relaxation of original problem
  - The closer the heuristic is to the actual cost of getting to the goal, while still a remaining an underestimate, the fewer nodes A* expands in the search for a plan/sequence of actions.