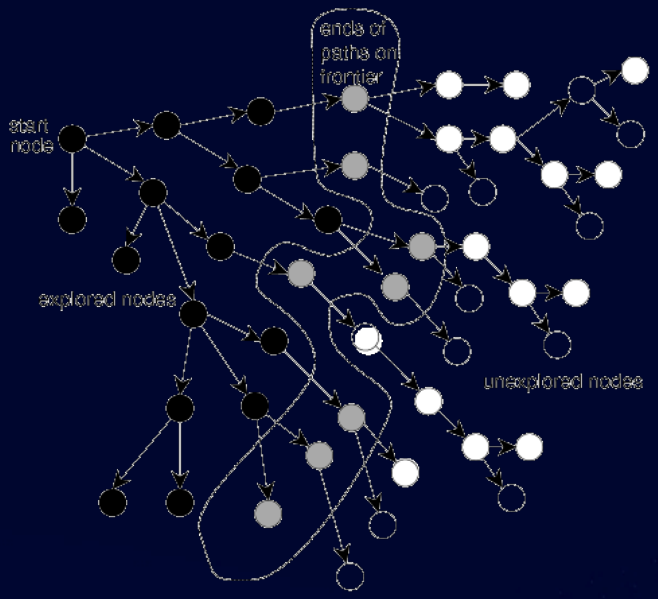
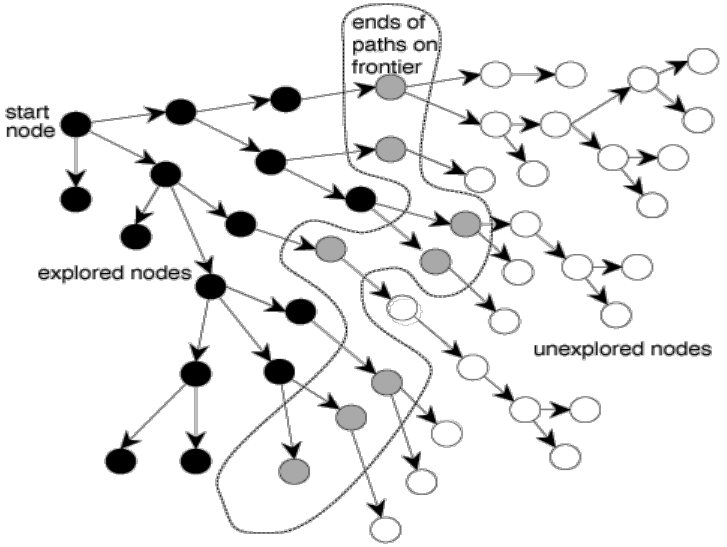


A* Tree Search

Last modified on 2022/03/25 by f.maire@qut.edu.au

Based on material by Stuart Russel,
Peter Norvig, Dan Klein and Pieter Abbeel

Informed Search



Credits: most material borrowed from AIMA or Dan Klein and Pieter Abbeel

Last modified on 2020/08/01 by f.maire@qut.edu.au

Reading for this week

- Chapter 3, Sections 3.5 to 3.6 of AIMA
  - Russell and Norvig Textbook:
    Artificial Intelligence, a Modern Approach
    3rd edition

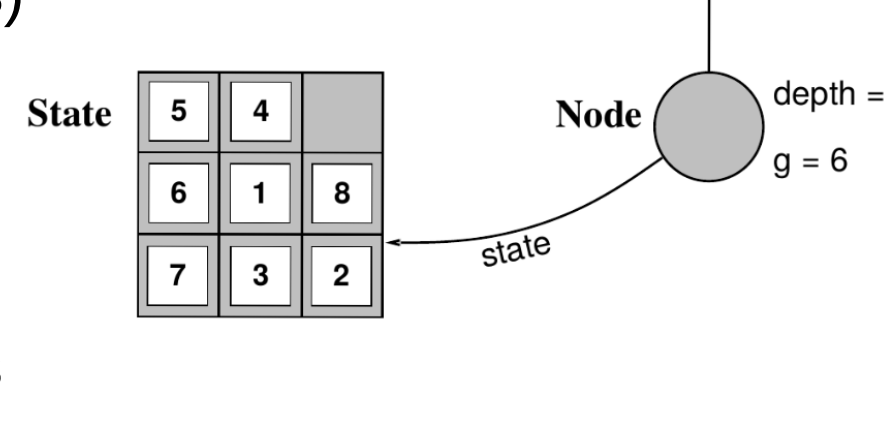
Today's Menu

- Recap
- Informed Search Methods
  - Heuristics
  - Greedy Search
  - A* Tree Search

Recap: uninformed search

Recap from last week

- Search problem
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test
- Search tree
  - Nodes:
    represent plans for reaching states
  - Plans have costs
    (sum of action costs)



- Search algorithm
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds cheapest plans (aka sequence of actions)

Fringe and *Frontier* are synonyms

Tree search algorithms

Recap from last week

Basic idea:

offline, simulated exploration of state space

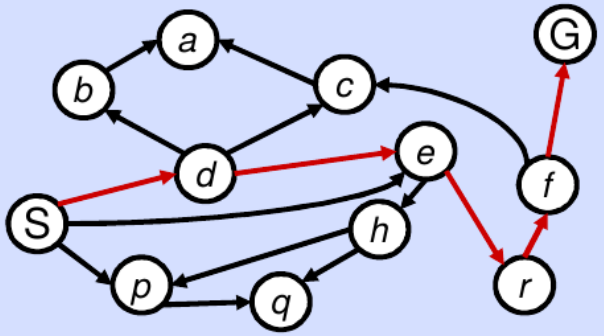
by generating successors of already-explored states

(a.k.a. expanding states)

```
function TREE-SEARCH( problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node and add the resulting nodes to the frontier
    end
```

State Graphs vs. Search Trees

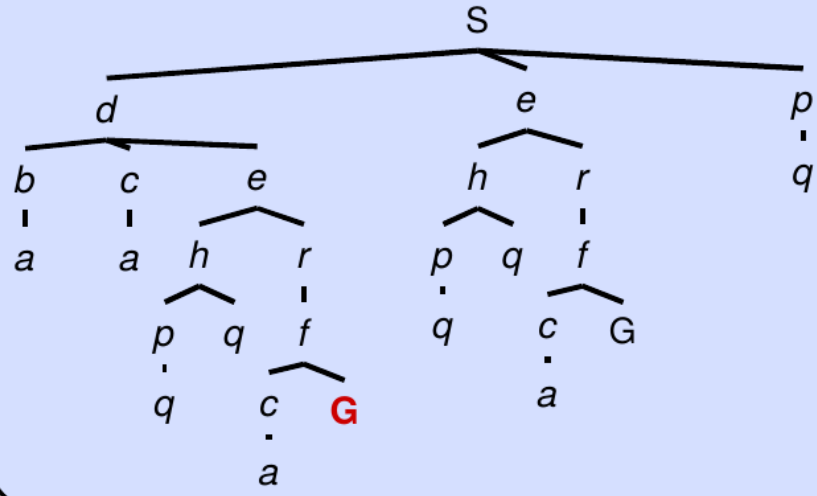
State Graph



Each NODE in in the search tree is an entire PATH in the problem graph.

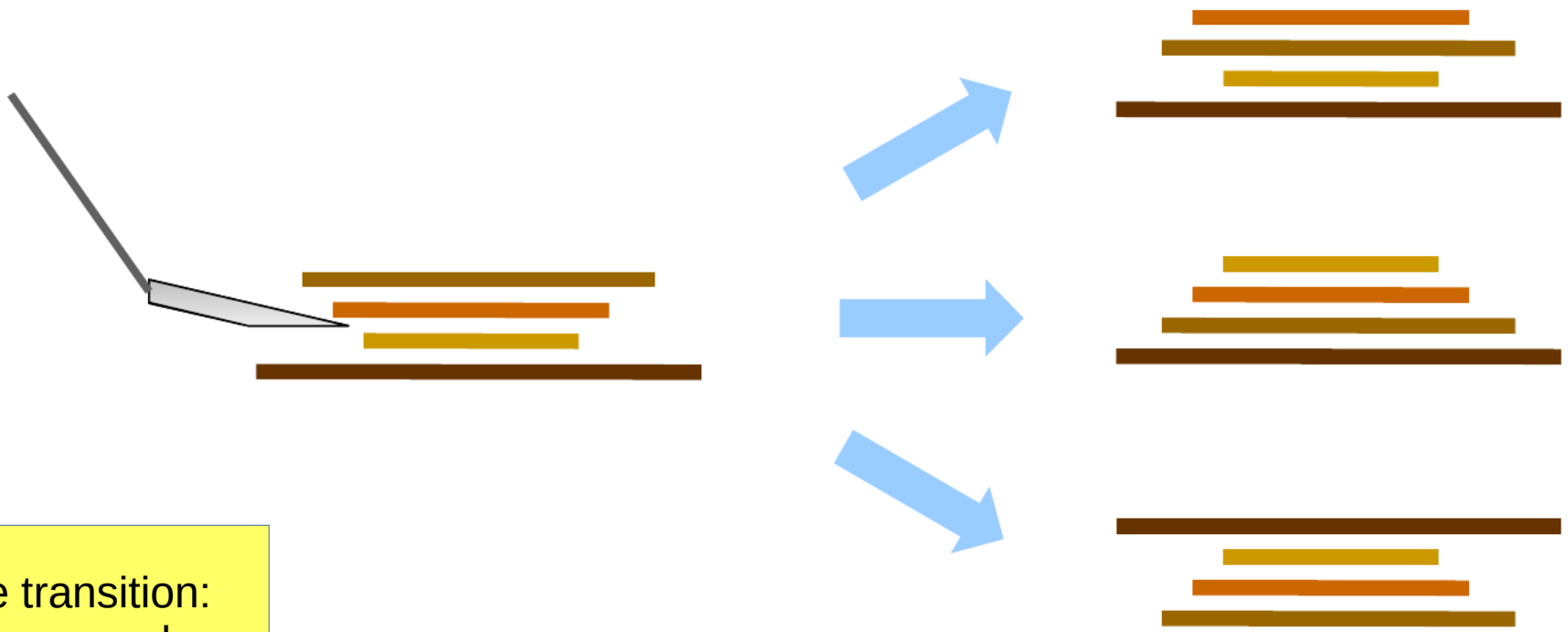
We construct both on demand – and we construct as little as possible.

Search Tree



Don’t confuse these two!

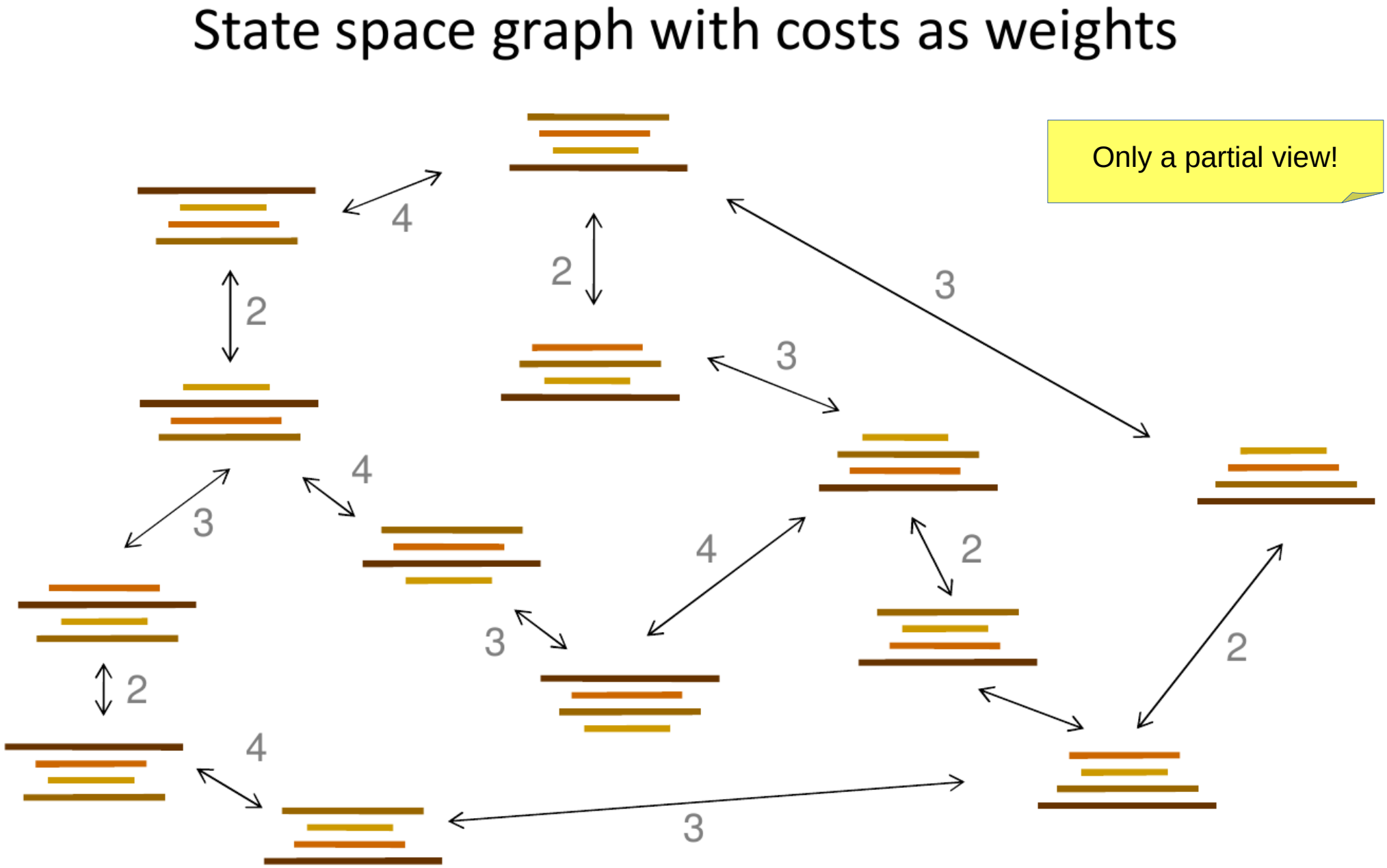
Example: pancake problem



State transition:
flip the pancakes
above the spatula

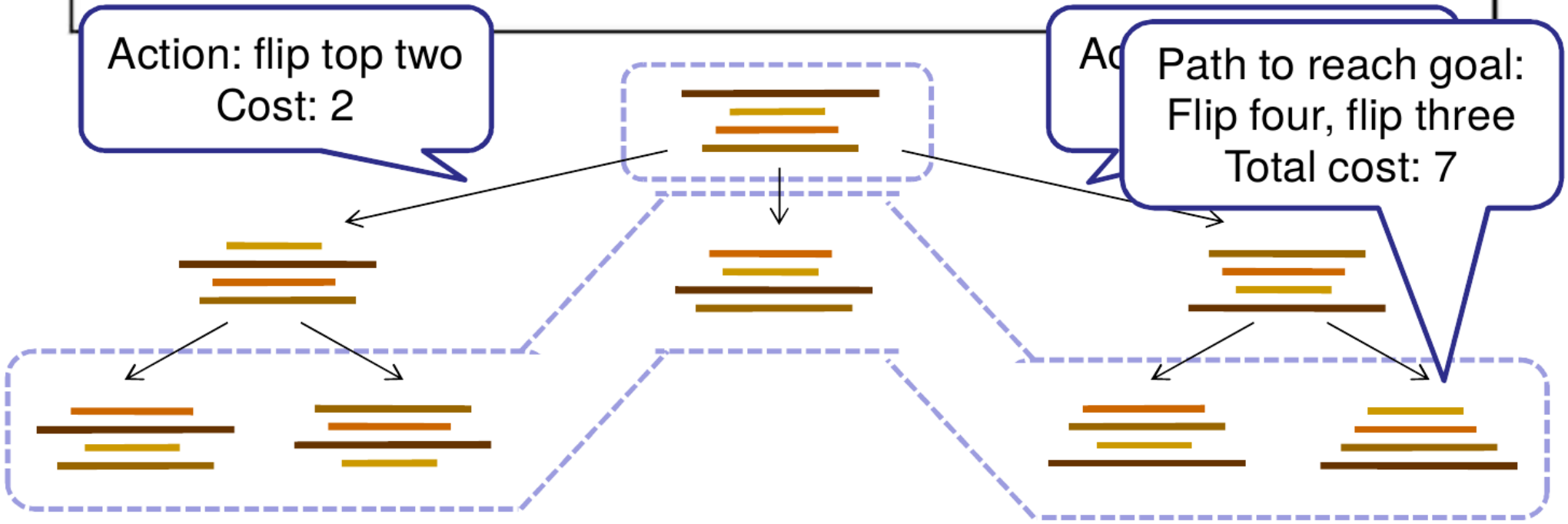
Cost: Number of pancakes flipped

Example: pancake problem



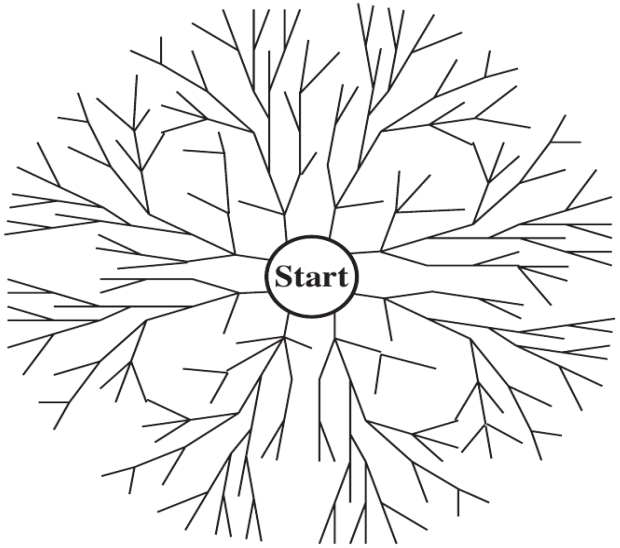
Example: pancake problem

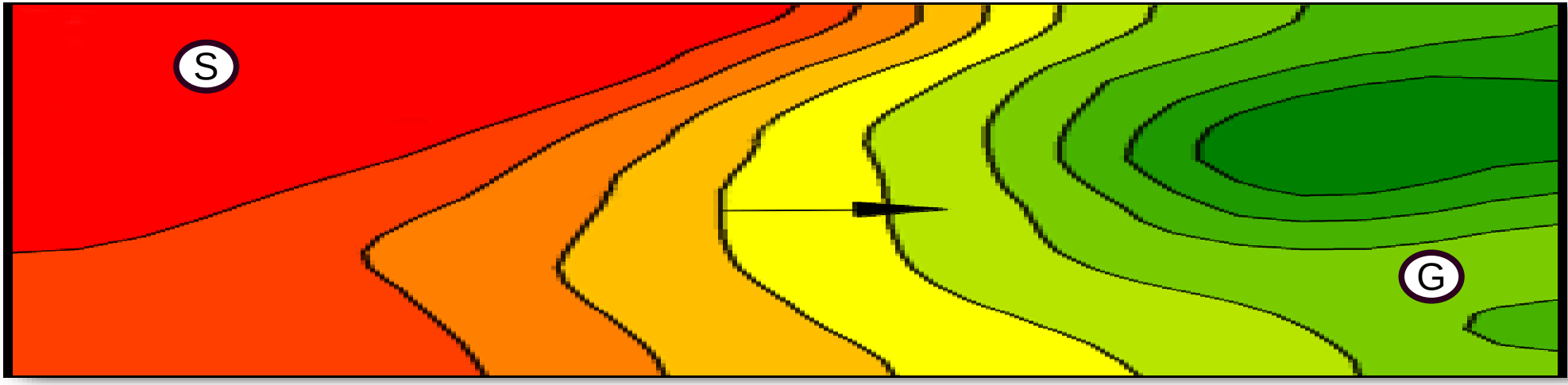
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```



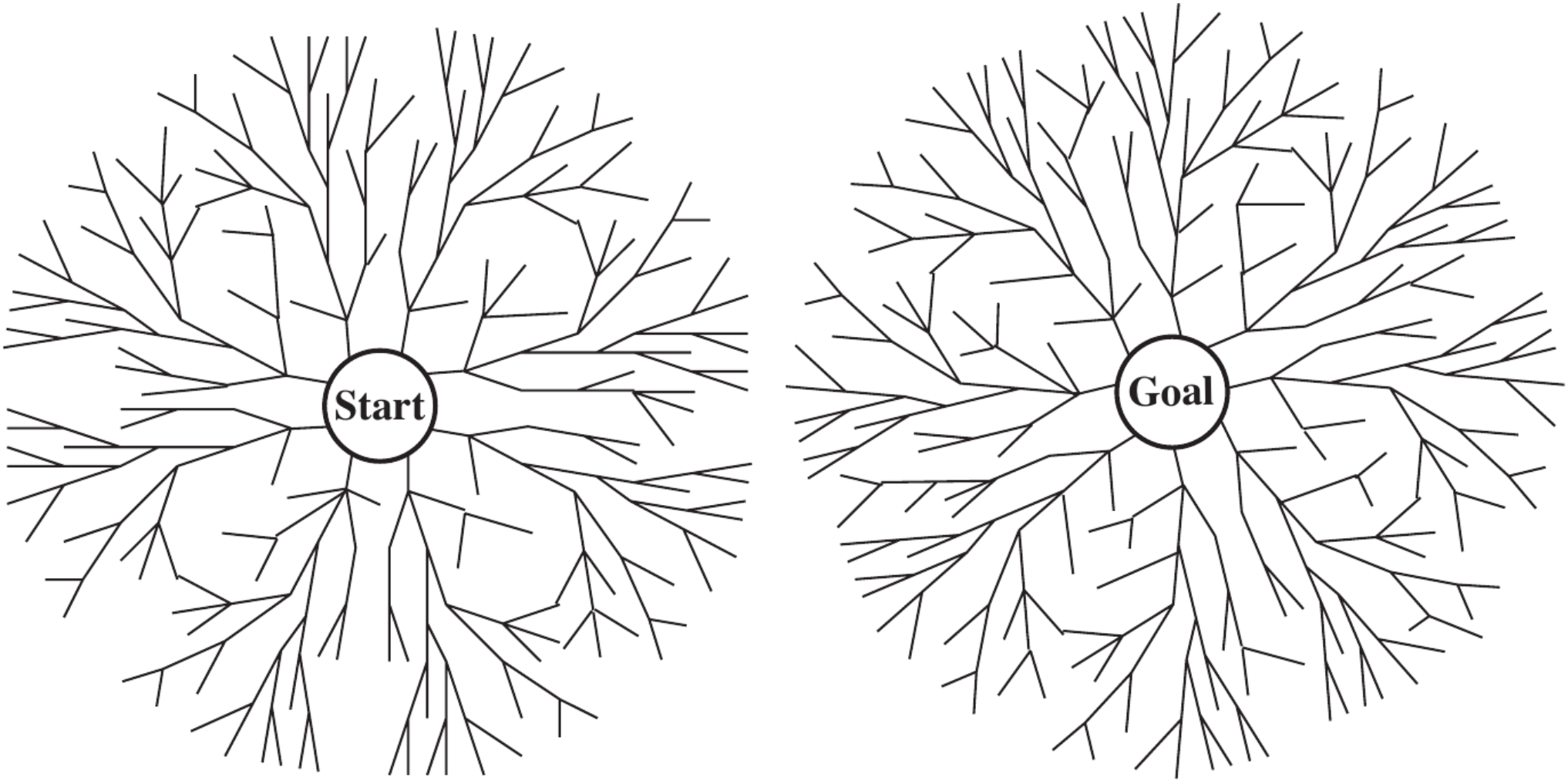
Uniform Cost Search - Issues

- UCS explores increasing cost contours
- The good news:
  - UCS is complete and optimal!
- The bad news:
  - Explores options in every “direction”
  - No information about goal location



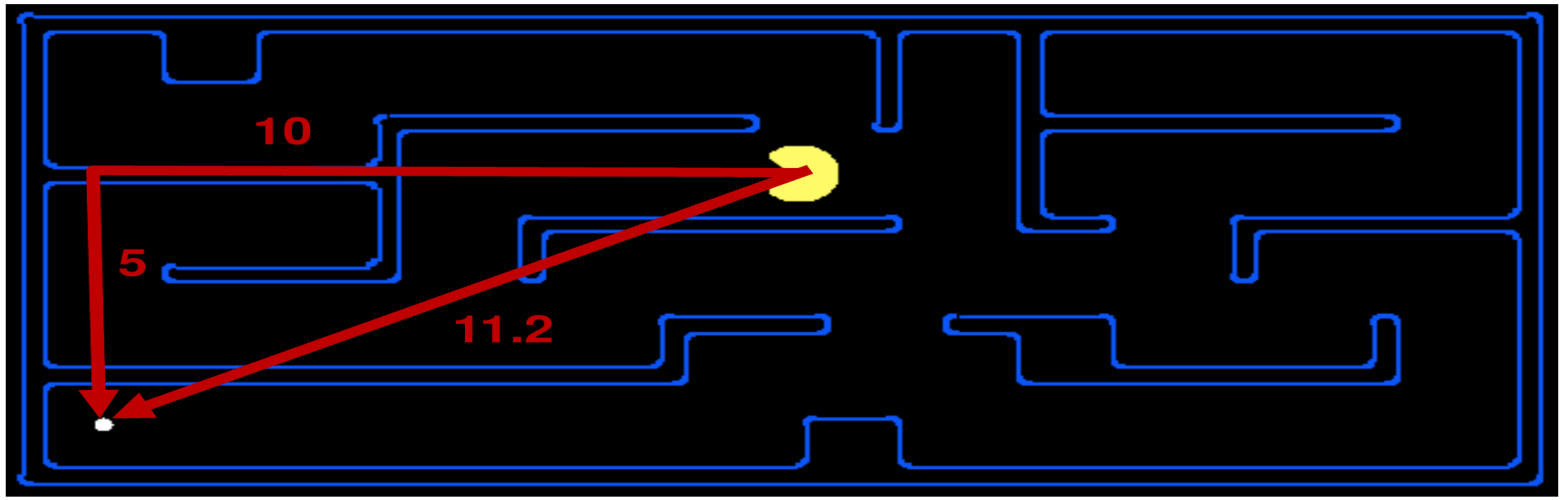


A schematic view of a bidirectional search



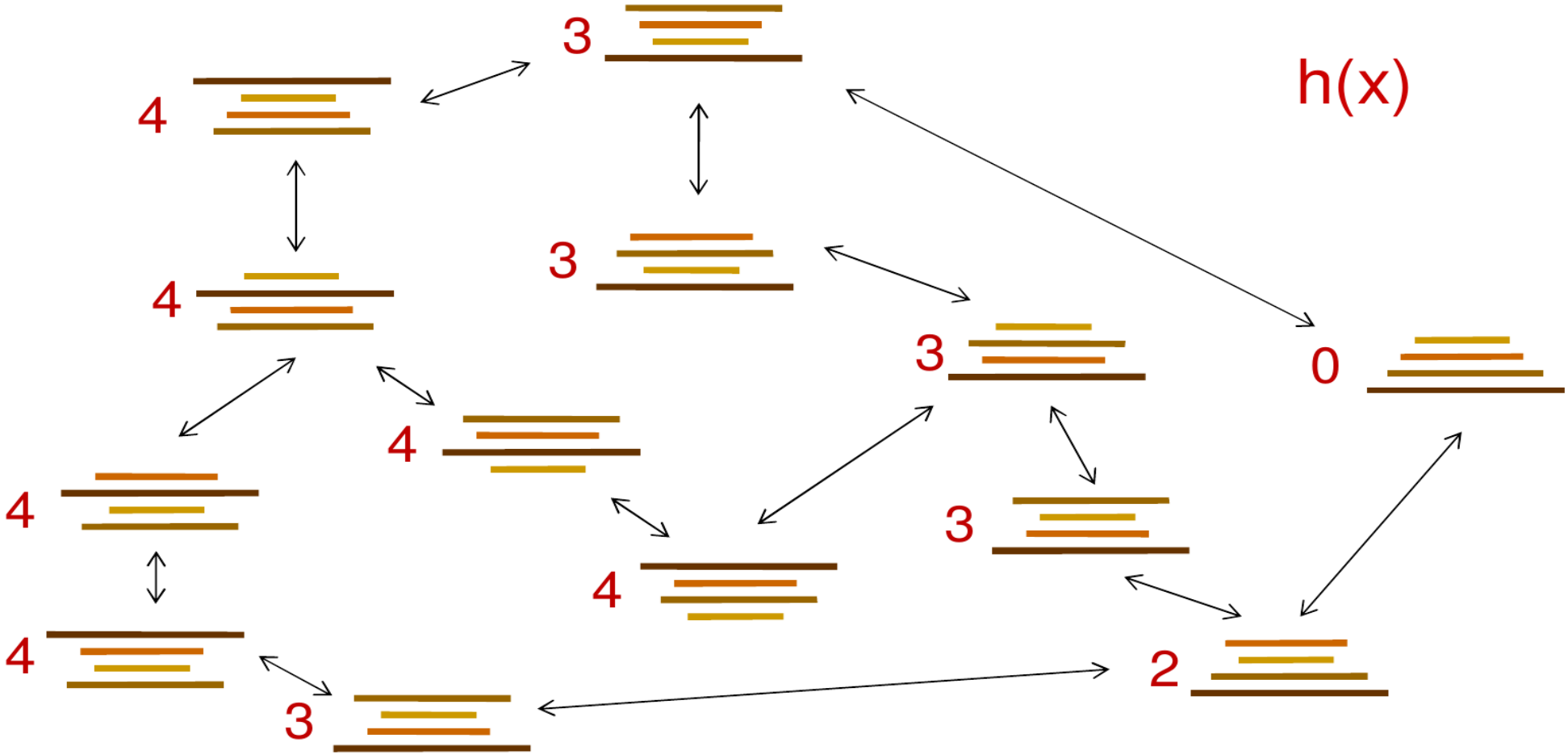
Search Heuristics

- A **heuristic** is:
  - A function that estimates how close a state is to a goal
  - Designed for a particular search problem
- Examples: Manhattan distance, Euclidean distance for shortest path finding



Example: Heuristic Function

- Heuristic: the index of the largest pancake that is still out of place (1 smallest, 4 largest)



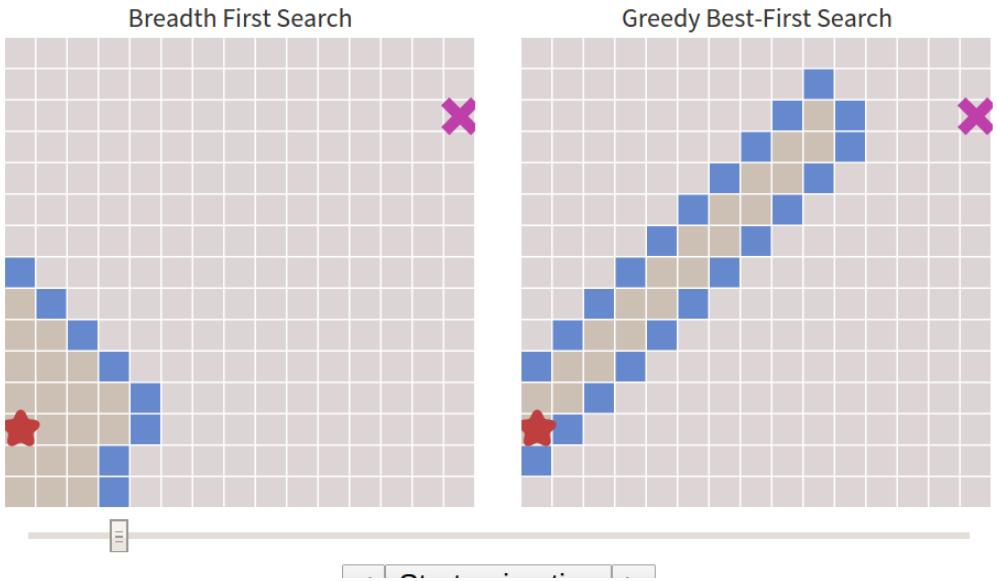
Example Heuristic: Straight-line distances to Bucharest

Arad	366
Bucharest	0
Craiova	160
Drobeta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

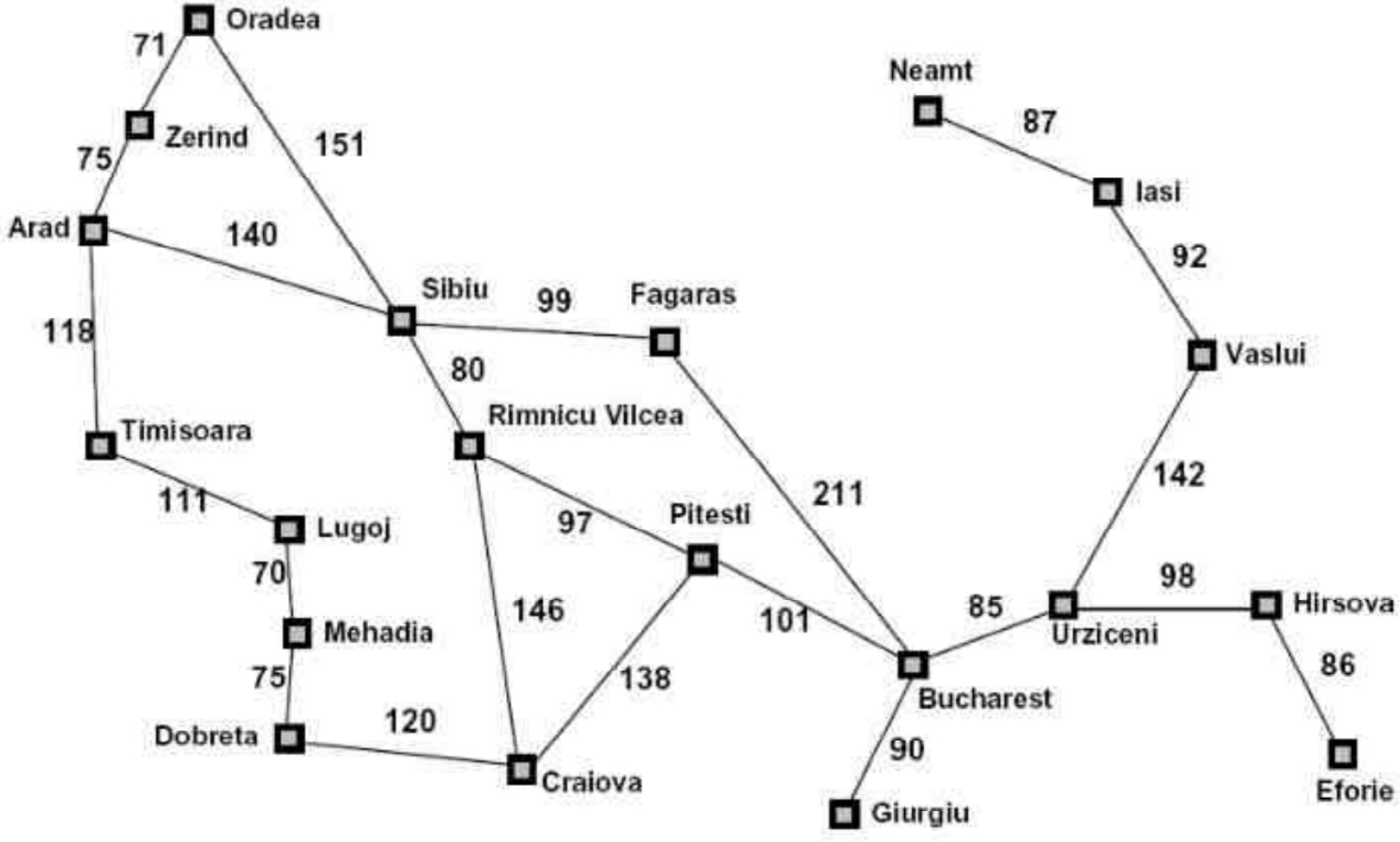
Greedy Search

- Strategy: expand a node that you think is closest to a goal state
  - Heuristic: estimate of distance to nearest goal for each state
- A case where it works well



Greedy search: expand the node that seems the closest to a goal

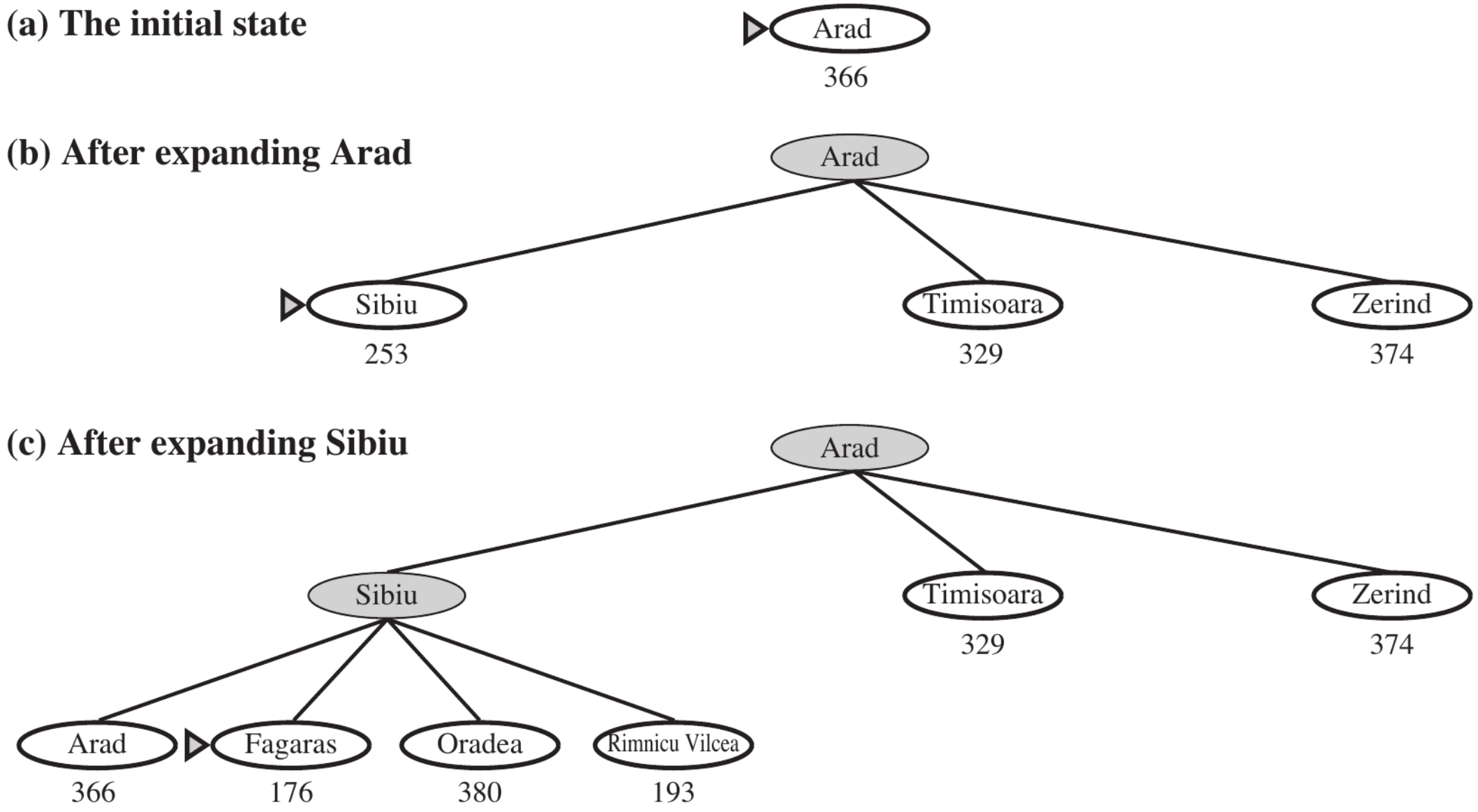
smallest h(x)



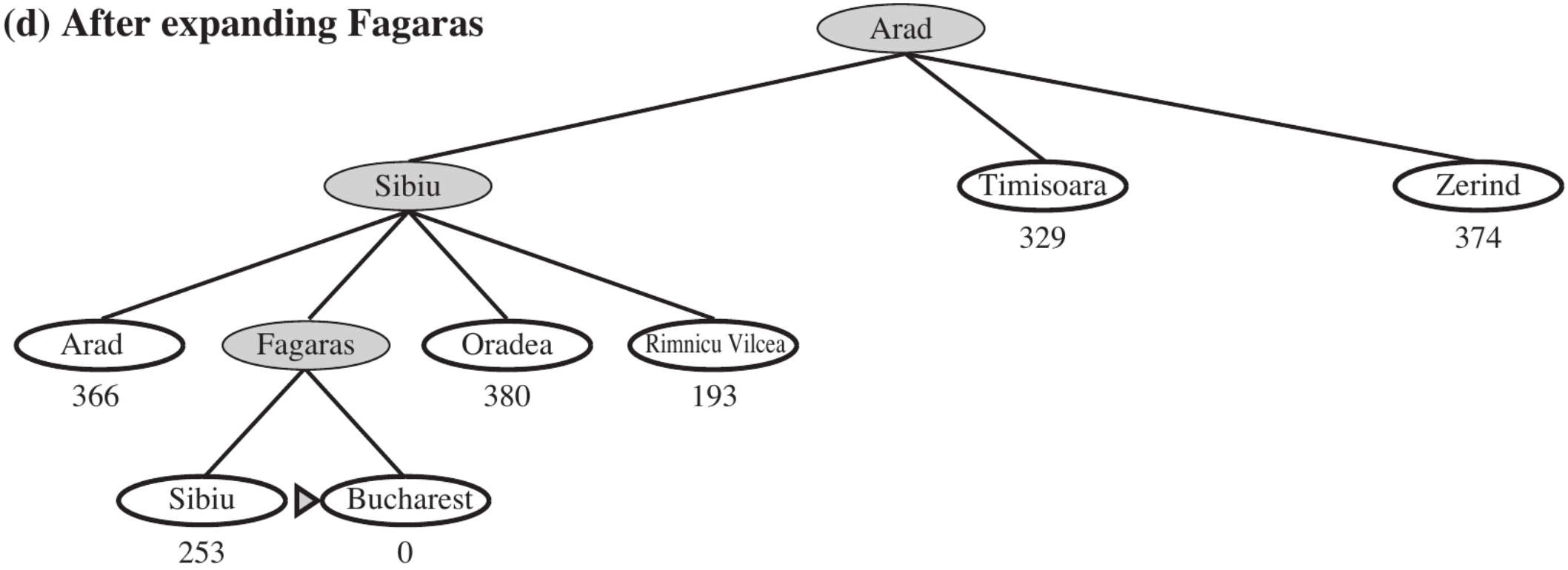
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

h(x)

Greedy best-first tree search for Bucharest with the straight-line distance heuristic

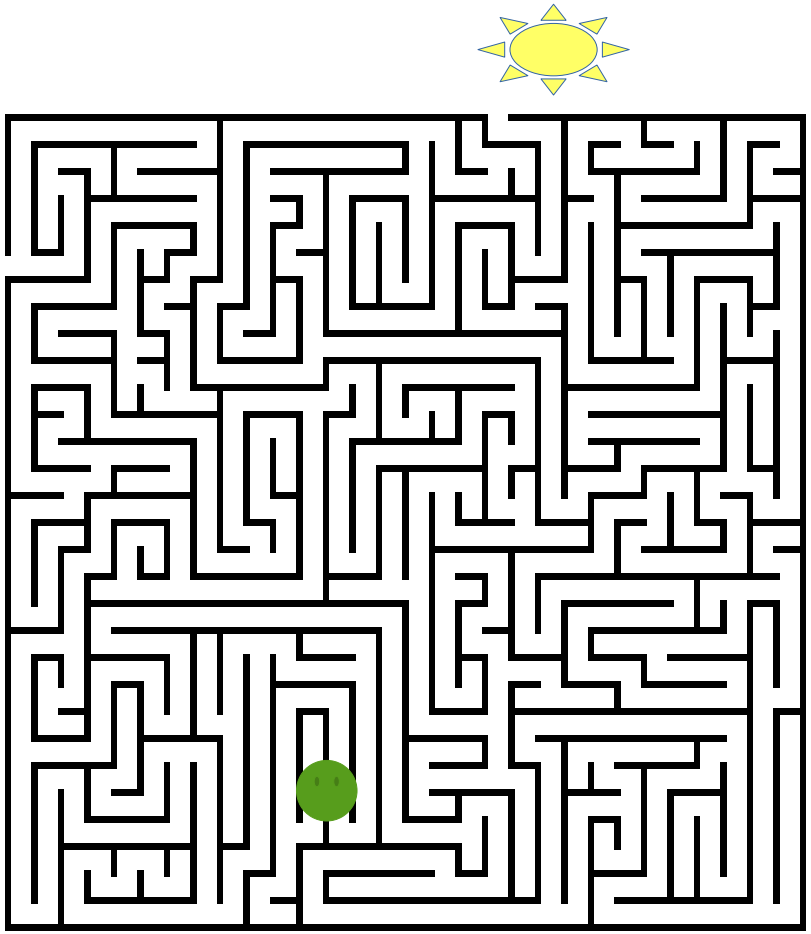


Greedy best-first tree search for Bucharest with the straight-line distance heuristic

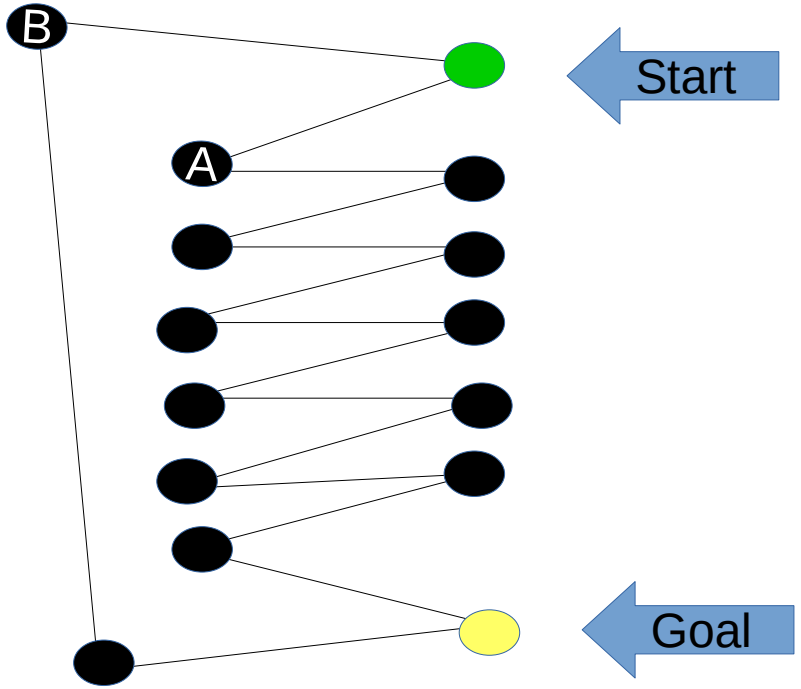


What can go wrong?

Greedy Search



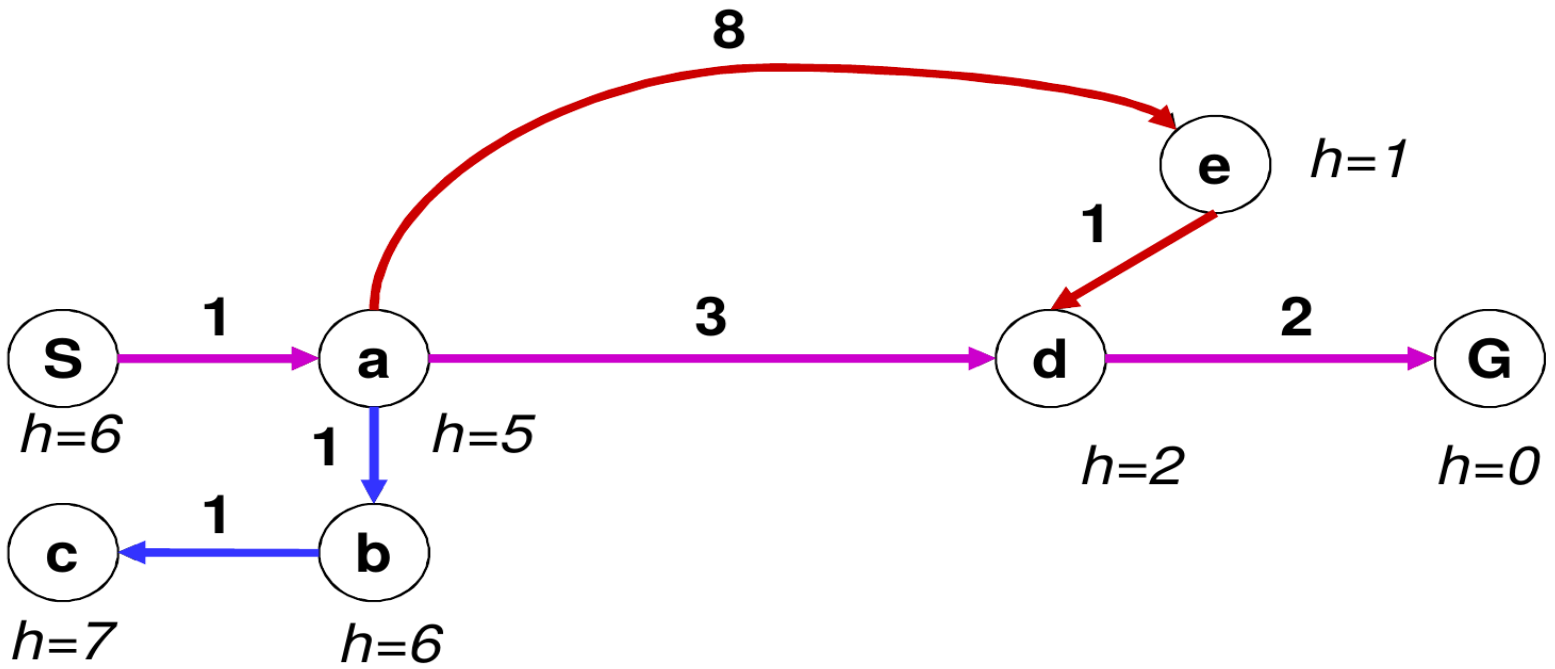
Here h(x) is the straight line distance.



Being greedy (going via A) does not always pays off!
The path via B has a lower cost

Combining UCS and Greedy

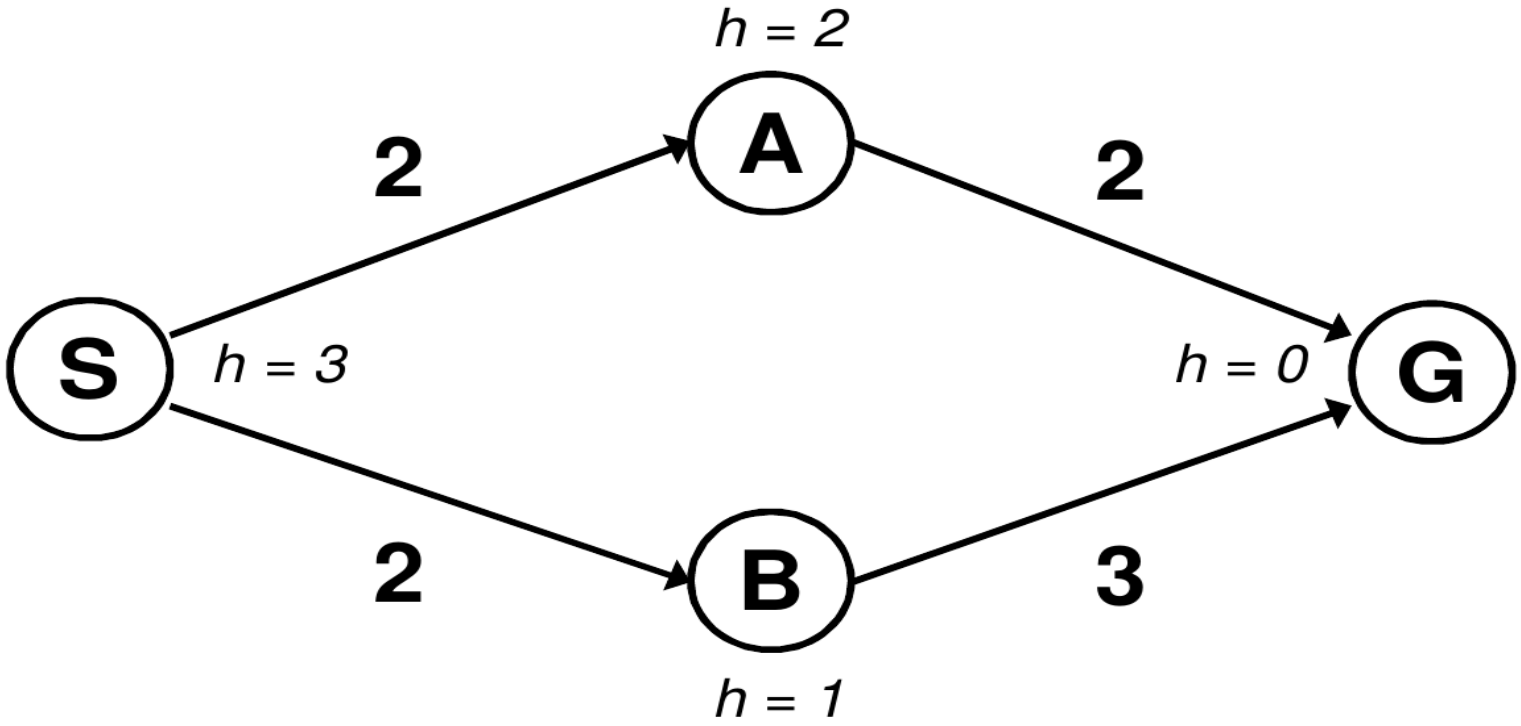
- Uniform-cost orders by path cost, or backward cost g(n)
- Greedy orders by goal proximity, or forward cost h(n)



A* Search orders by the sum: f(n) = g(n) + h(n)

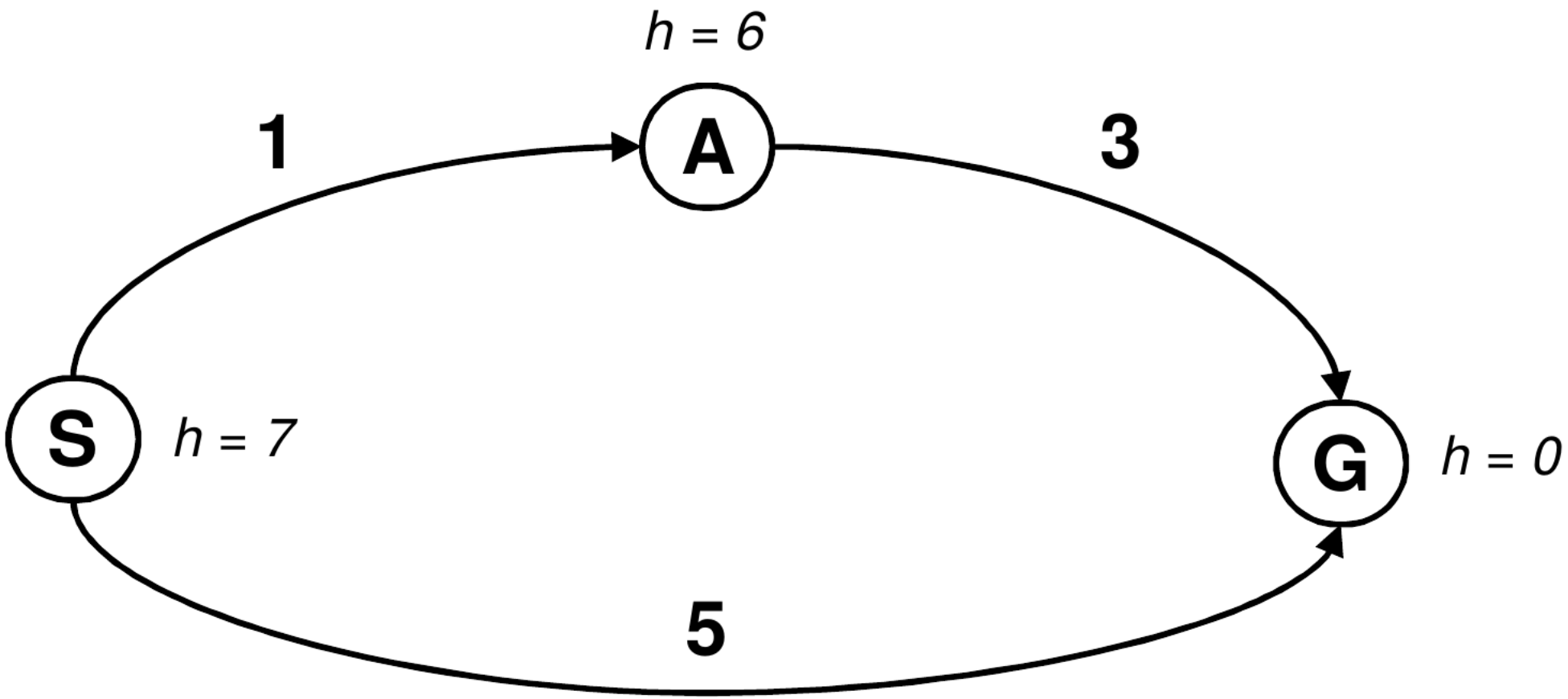
When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Is A* Optimal?



What went wrong?
Actual bad goal cost < estimated good goal cost
We need estimates to be less than actual costs!

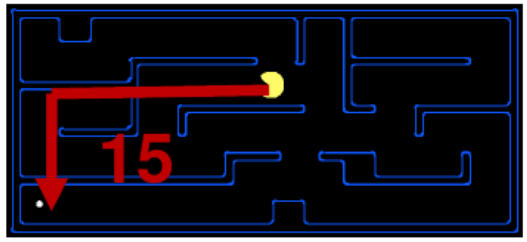
Admissible Heuristics

A heuristic h is *admissible* (optimistic) if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost to a nearest goal

Examples:



4

Coming up with admissible heuristics is most of what's involved in using A* in practice.

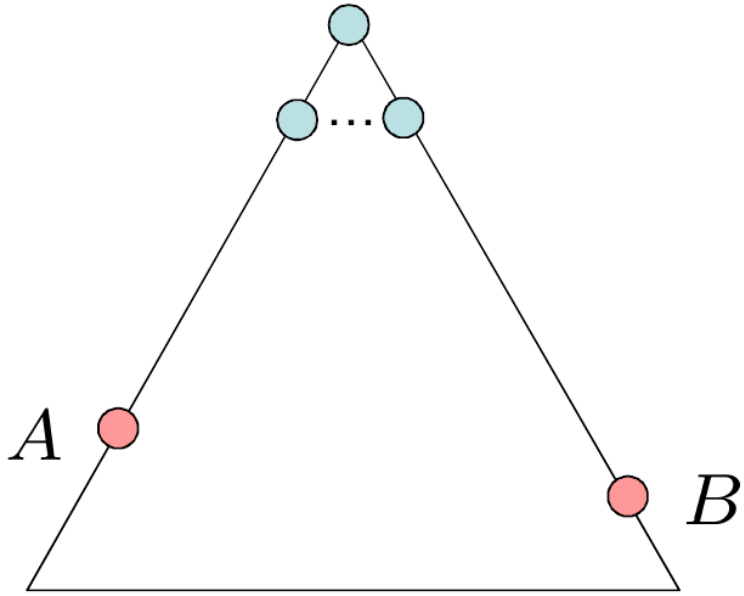
Optimality of A* Tree Search

Assume:

- A is an optimal goal node
- B is a suboptimal goal node
- h is admissible

Claim:

- A will exit the fringe before B

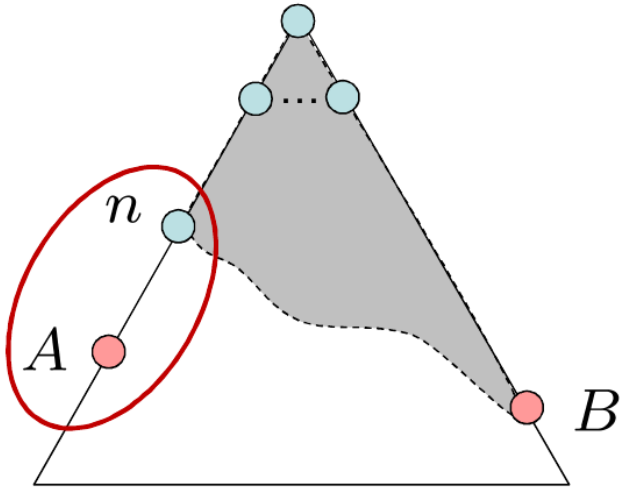


‘fringe’ and ‘frontier’ are synonyms

Optimality of A* Tree Search

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)

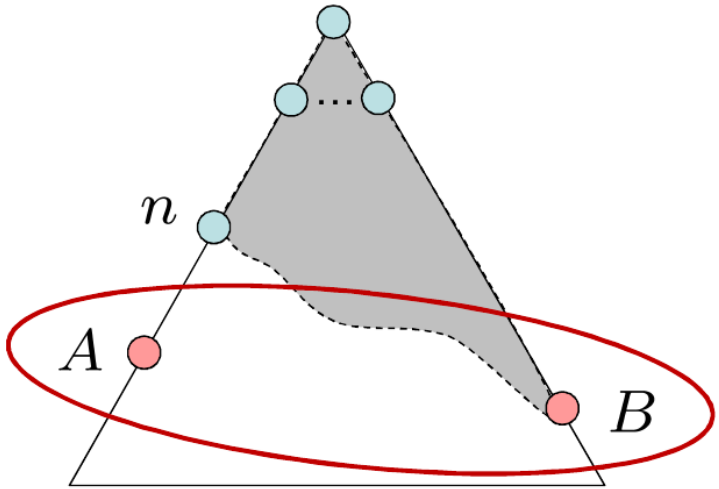


$$f(n) = g(n) + h(n) \quad \text{Definition of f-cost}$$
$$f(n) \leq g(A) \quad \text{Admissibility of h}$$
$$g(A) = f(A) \quad \text{h = 0 at a goal}$$

Optimality of A* Tree Search

Proof:

- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)



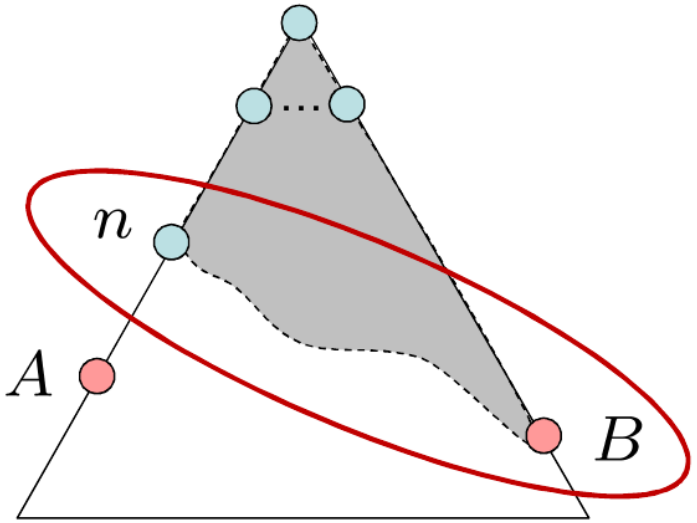
$g(A) < g(B)$ B is suboptimal

$f(A) < f(B)$ h = 0 at a goal

Optimality of A* Tree Search

Proof:

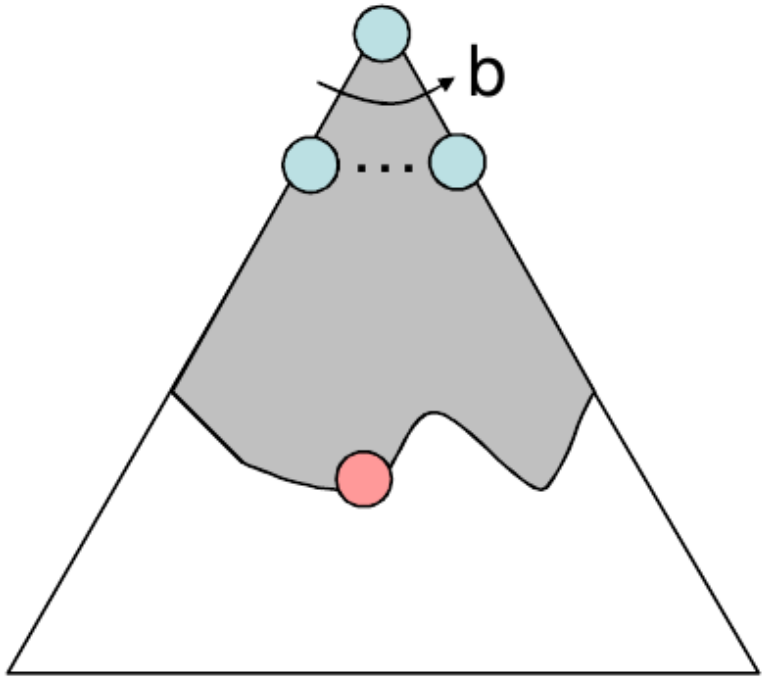
- Imagine B is on the fringe
- Some ancestor *n* of A is on the fringe, too (maybe A!)
- Claim: *n* will be expanded before B
  1. f(n) is less or equal to f(A)
  2. f(A) is less than f(B)
  3. *n* expands before B
- All ancestors of A expand before B
- A expands before B
- A* search is optimal



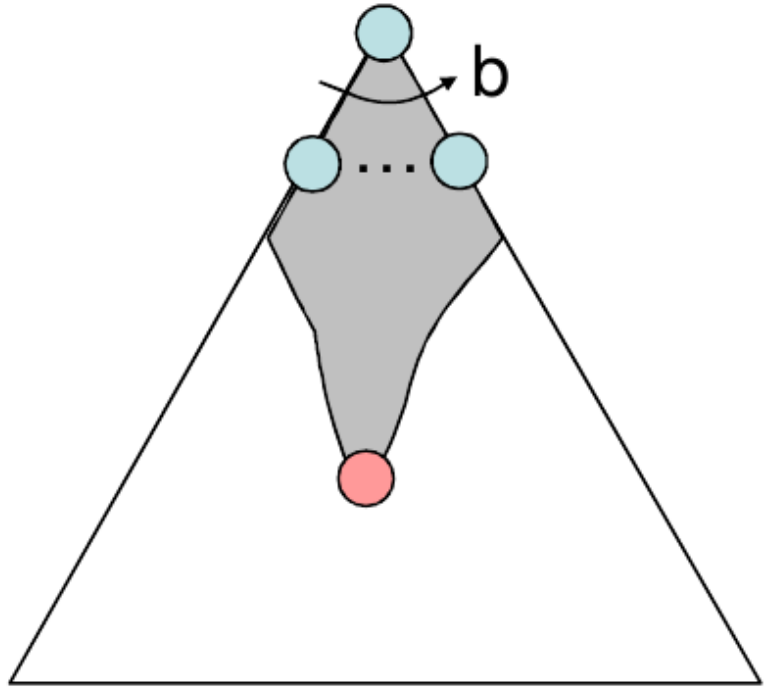
$$f(n) \leq f(A) < f(B)$$

Properties of A*

Uniform-Cost

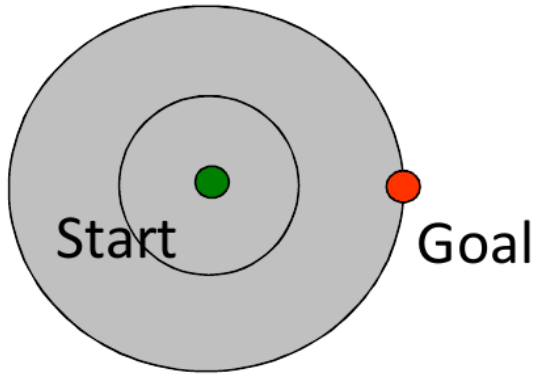


A*

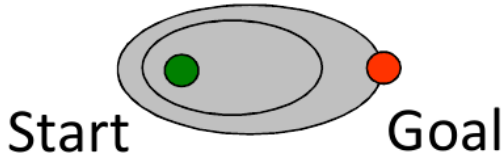


Properties of A*

- Uniform-cost expands equally in all "directions"



- A* expands mainly toward the goal, but does hedge its bets to ensure optimality



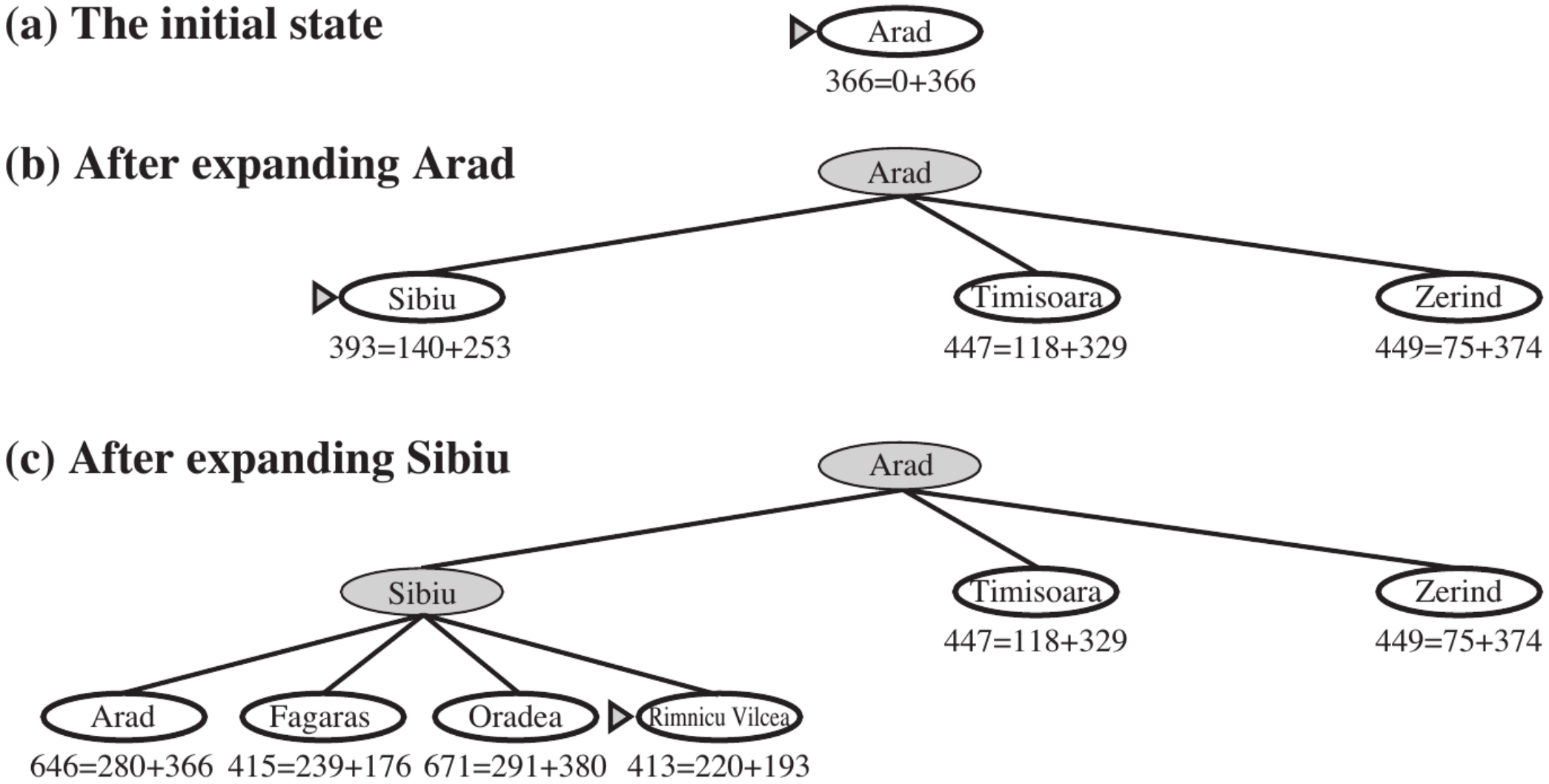
Creating Admissible Heuristics

- Most of the work in solving hard search problems optimally is in coming up with admissible heuristics

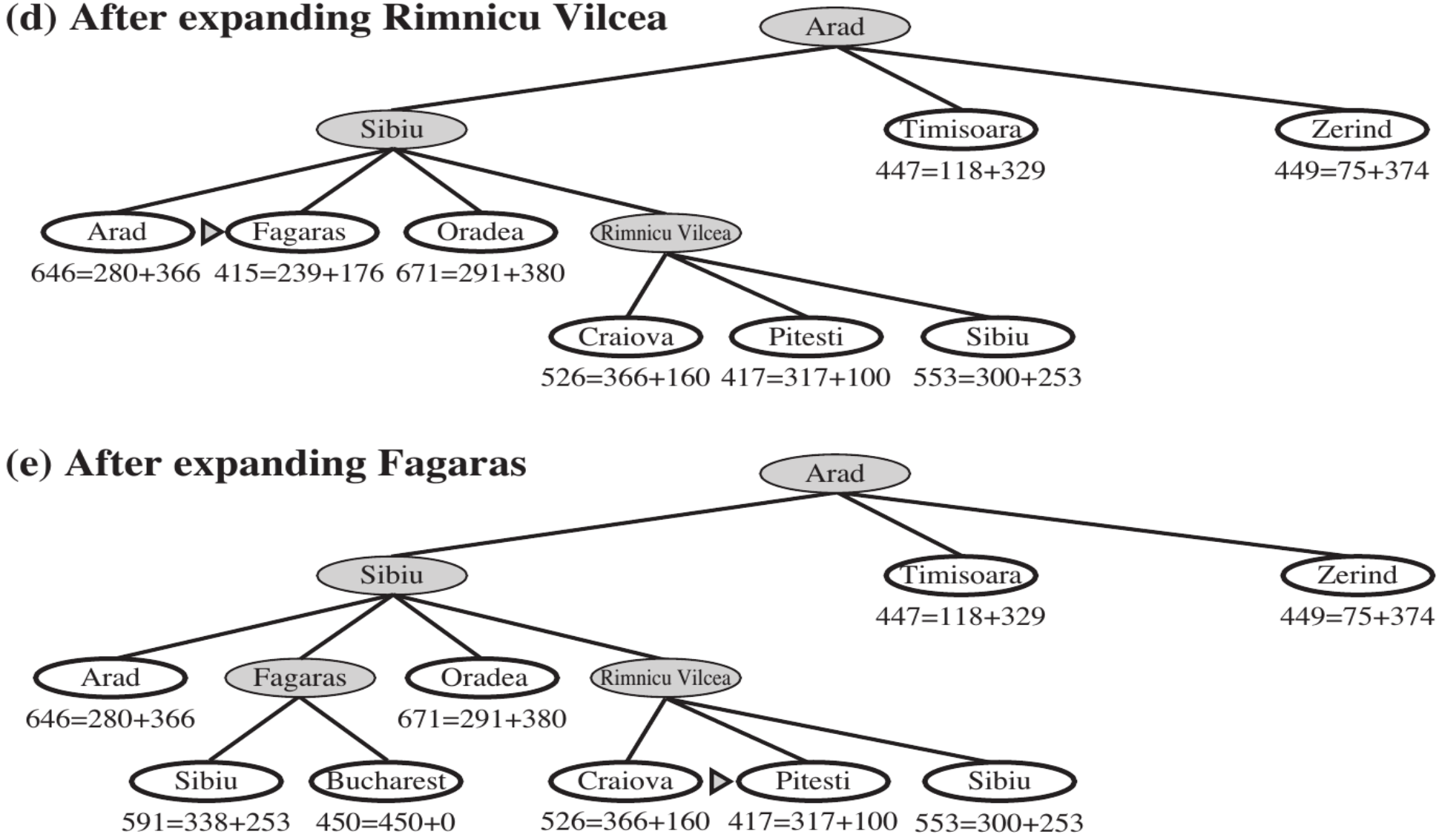
- Often, admissible heuristics are solutions to relaxed problems, where new actions are available

Relaxing a problem ~ removing constraints

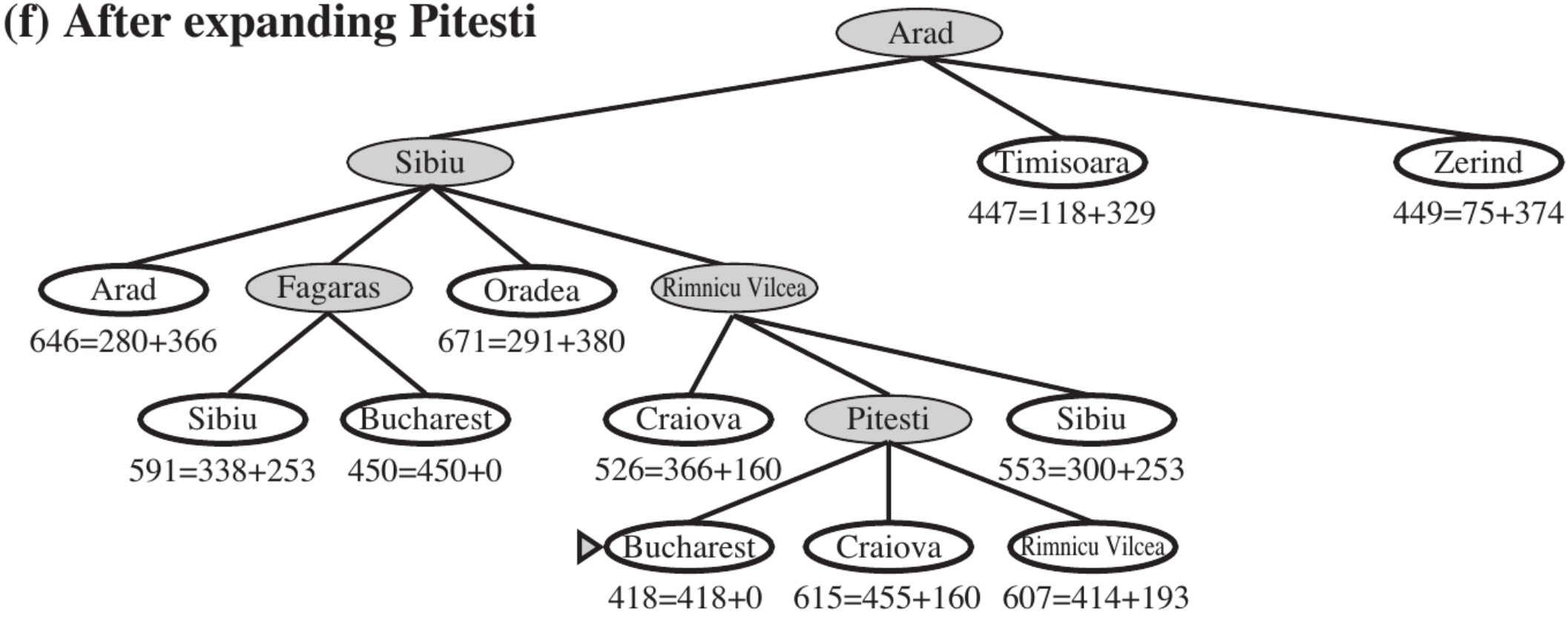
Stages in A* search
Nodes labelled with $f = g+h$



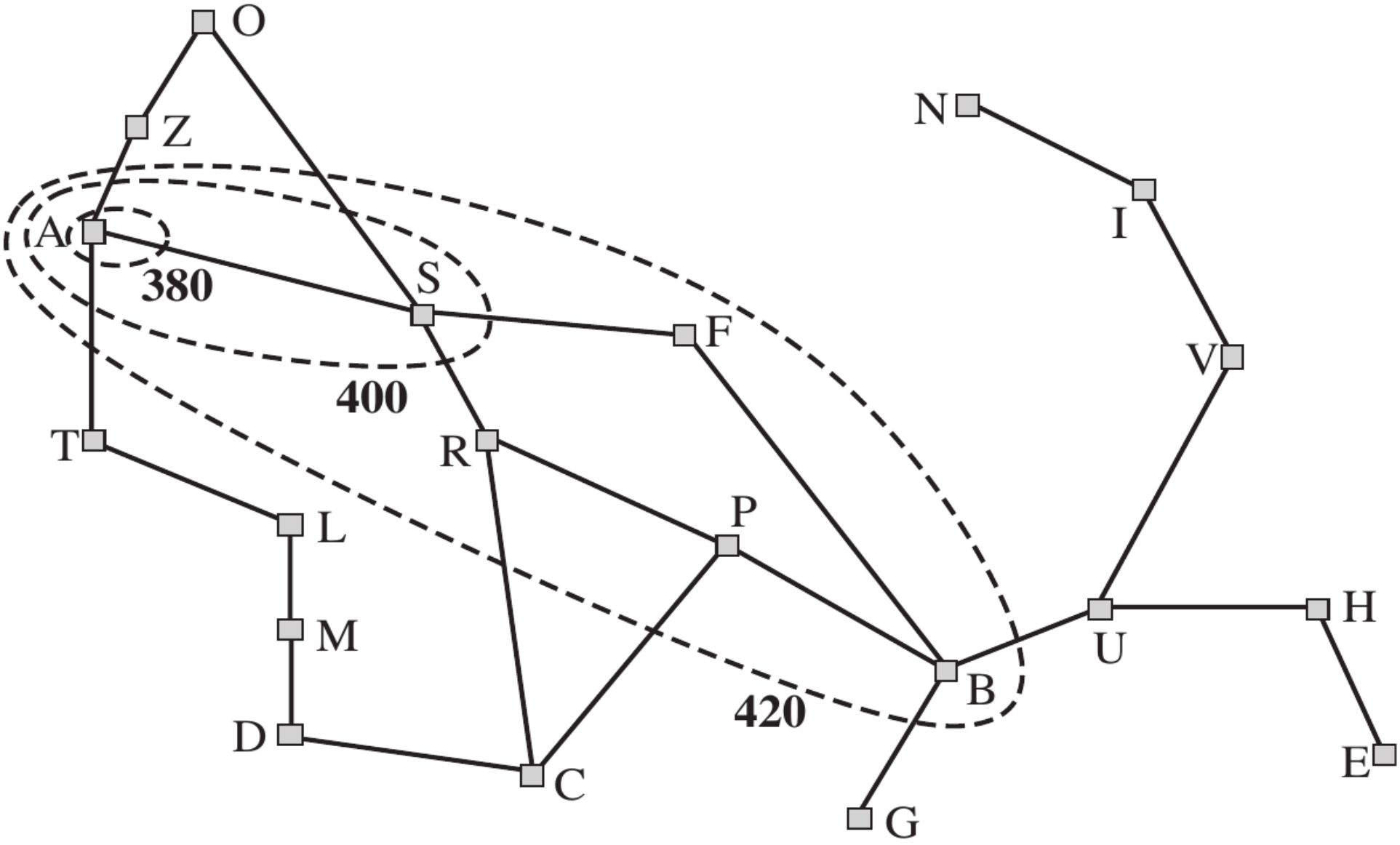
Stages in A* search
Nodes labelled with $f = g+h$



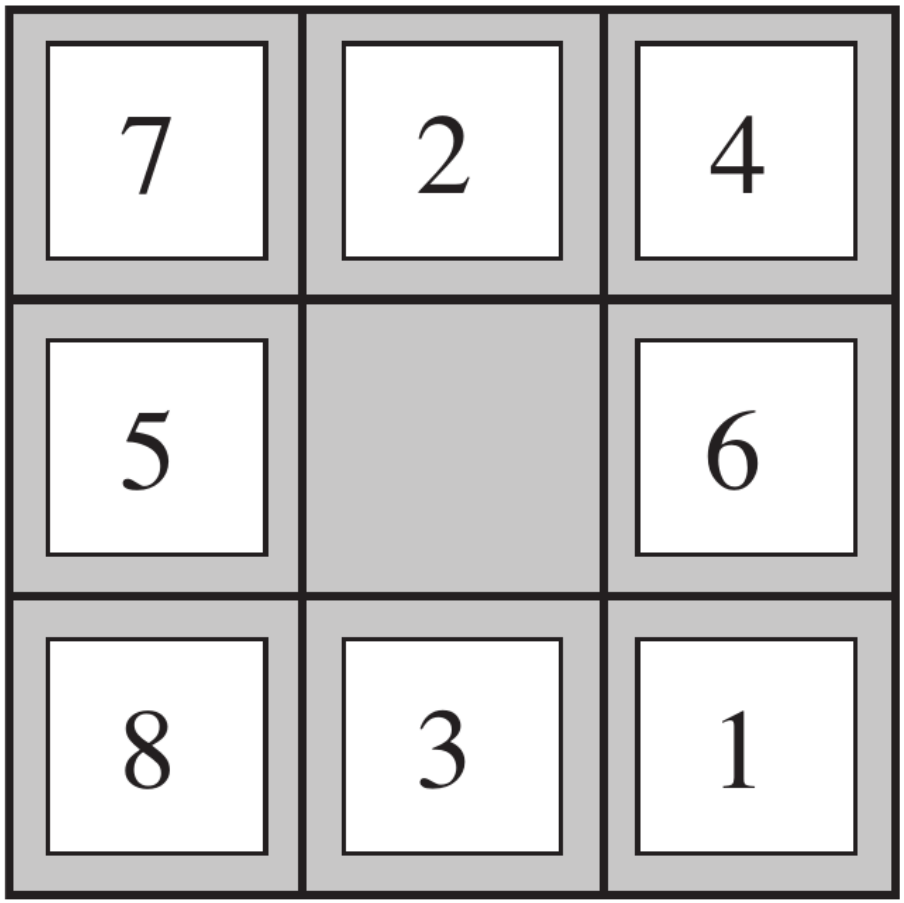
Stages in A* search
Nodes labelled with *f = g+h*



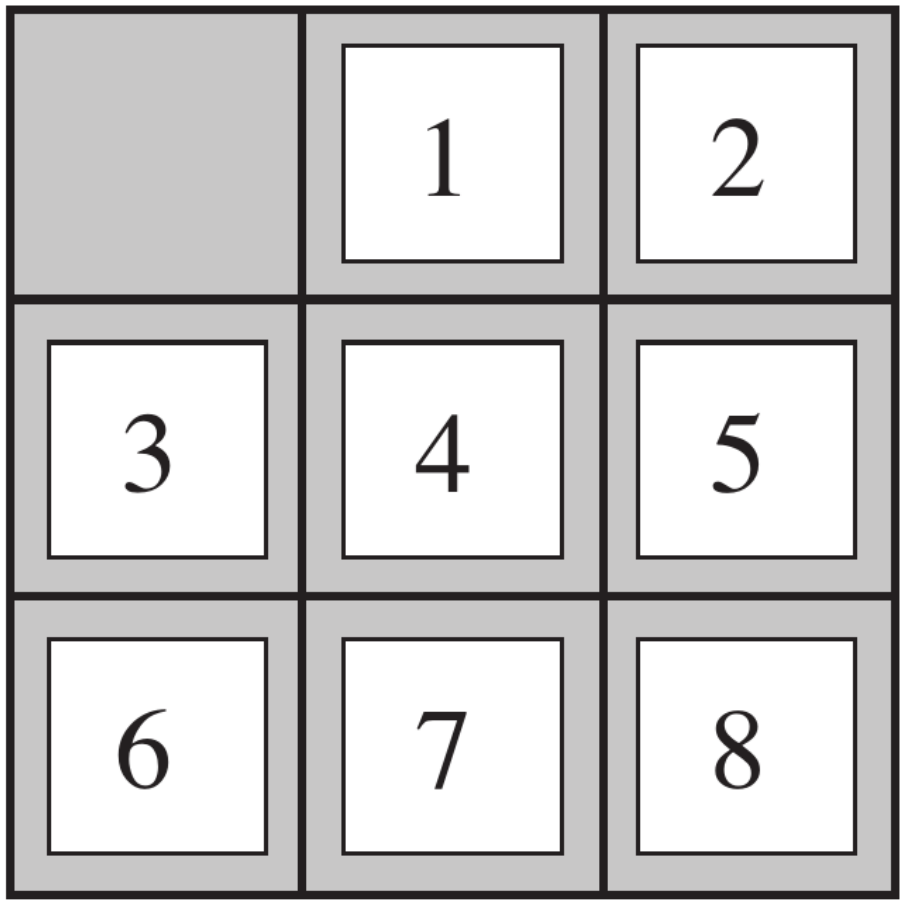
Map of Romania with contours of f = g+h



A typical instance of the 8-puzzle



Start State



Goal State

Heuristics for the sliding puzzle

- Heuristic: Number of tiles misplaced
  - Why is it admissible?
  - This is relaxed-problem heuristic
- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles?
  - Total Manhattan distance
  - Why is it admissible?

More remarks on heuristics

- How about using the actual cost as a heuristic?
  - Would it be admissible?
  - Would we save on nodes expanded?
  - What's wrong with it?
- With A*: a trade-off between quality of estimate and work per node
  - As heuristics get closer to the true cost, you will expand fewer nodes but usually do more work per node to compute the heuristic itself

Trivial Heuristics, Dominance

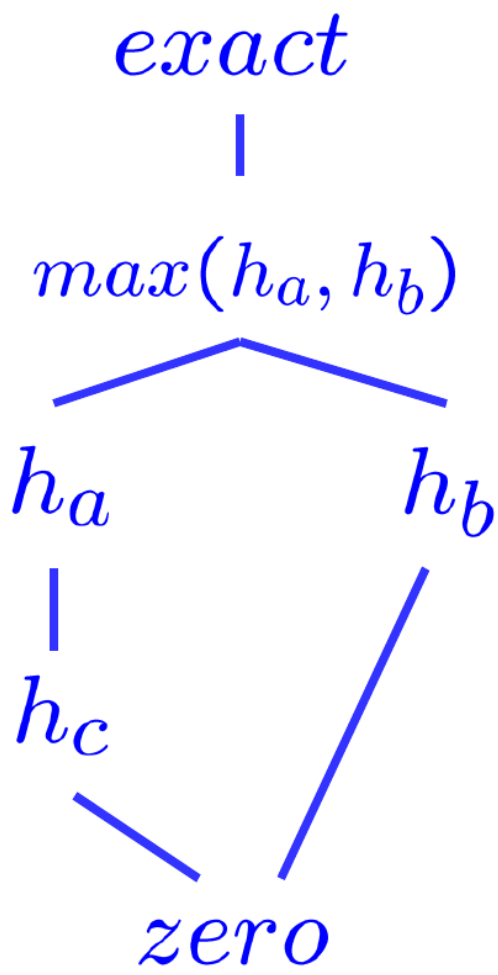
- Dominance: $h_a \geq h_c$ if

$$\forall n : h_a(n) \geq h_c(n)$$

- Heuristics form a semi-lattice:
  - Max of admissible heuristics is admissible

$$h(n) = max(h_a(n), h_b(n))$$

- Trivial heuristics
  - Bottom of lattice is the zero heuristic (what does this give us?)
  - Top of lattice is the exact heuristic



Optimality of A* Tree Search

- A* tree search is optimal if the heuristic is admissible

- UCS is a special case (h=0)