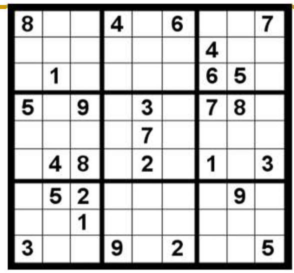
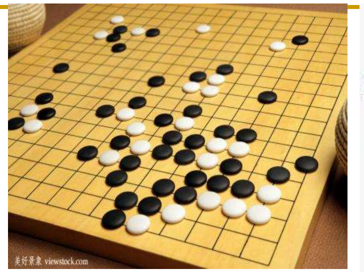
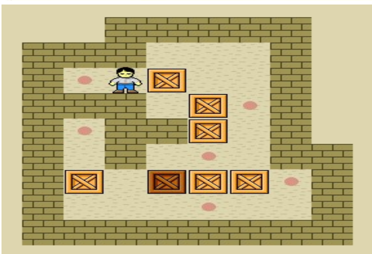
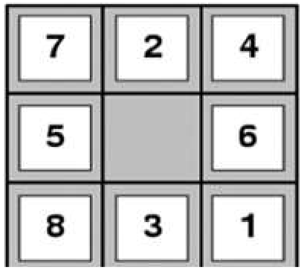
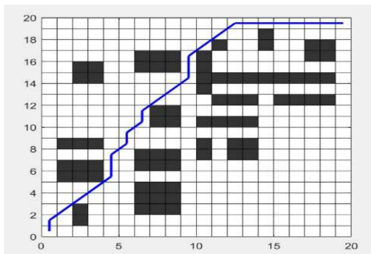
Problem Solving
- by Searching

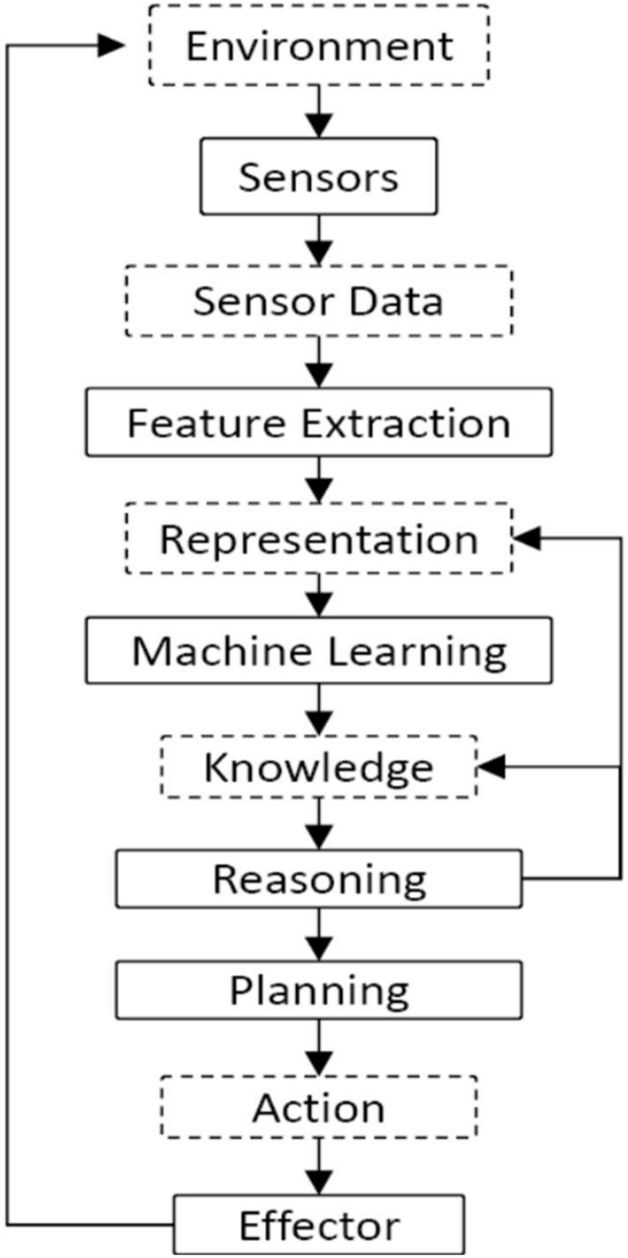



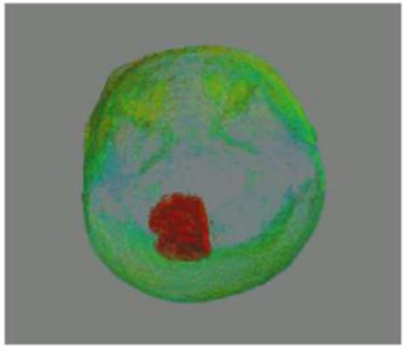


Outline

- Problem Solving by Searching
- Uninformed Search Strategies
- Informed (Heuristic) Search Strategies



Formal tasks: Playing board games, card games. Solving puzzles, mathematical and logic problems.



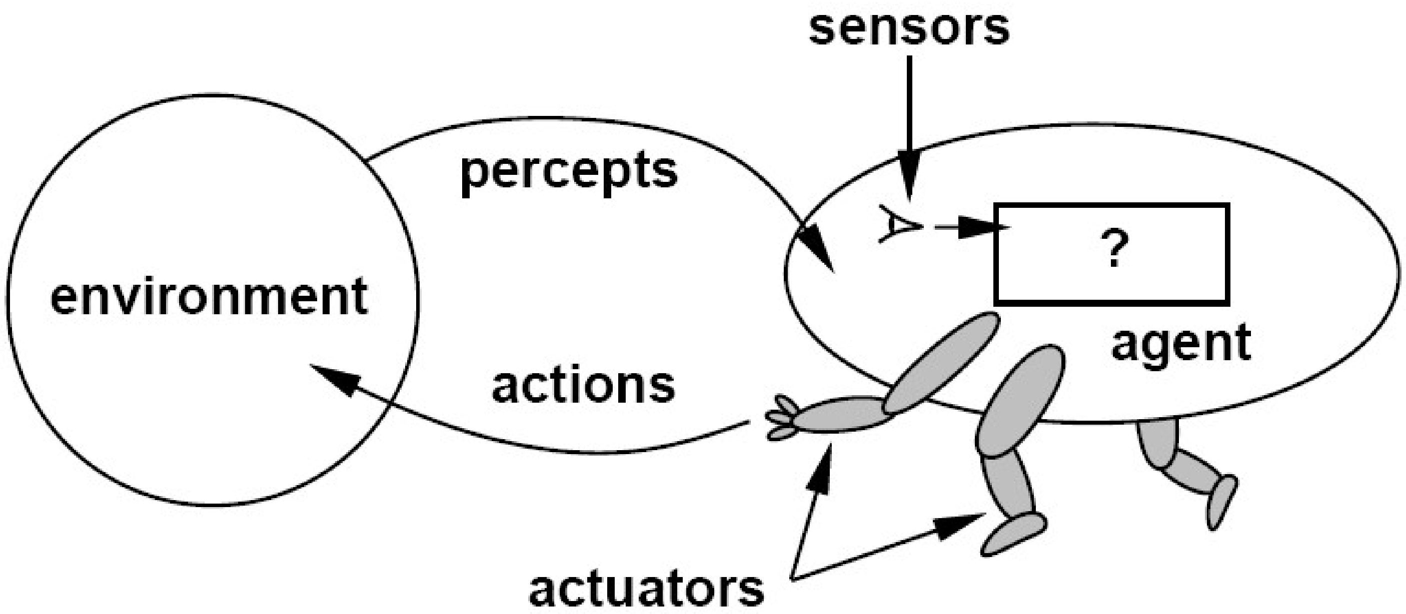
Expert tasks: Medical diagnosis, engineering, scheduling, computer hardware design.

Mundane tasks: Everyday speech, written language, perception, walking, object manipulation.

Human tasks: Awareness of self, emotion, imagination, morality, subjective experience, high-level-reasoning, consciousness.

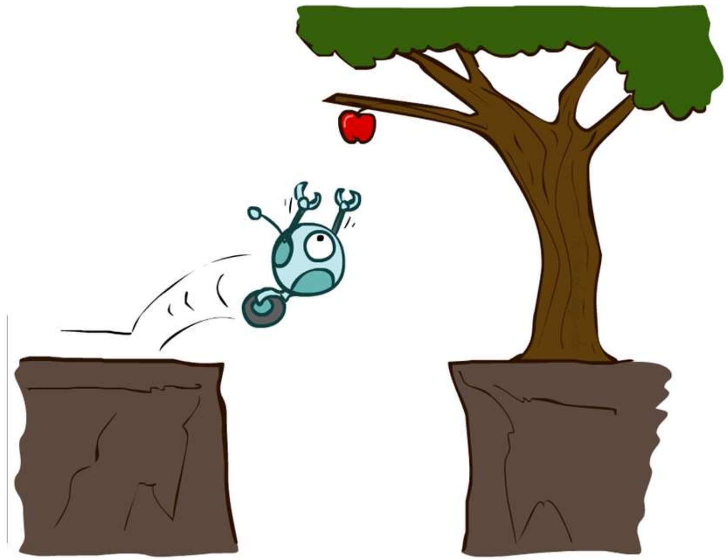
Agent

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.



Types of Agents

- **Reflex agents:**
  - Choose action based on current percept (and maybe memory)
  - May have memory or a model of the world's current state
  - Do not consider the future consequences of their actions
  - Consider how the world IS

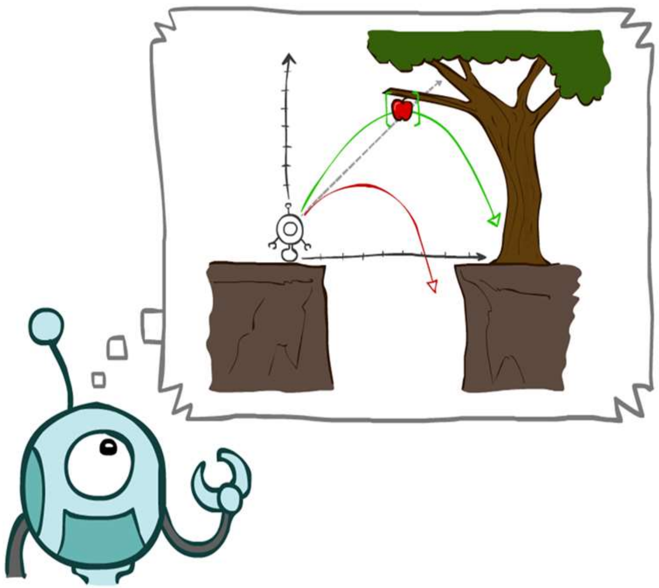


- Can a reflex agent be rational?

Types of Agents

❖ **Planning agents:**

- Ask “what if”
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal (test)
- Consider how the world WOULD BE



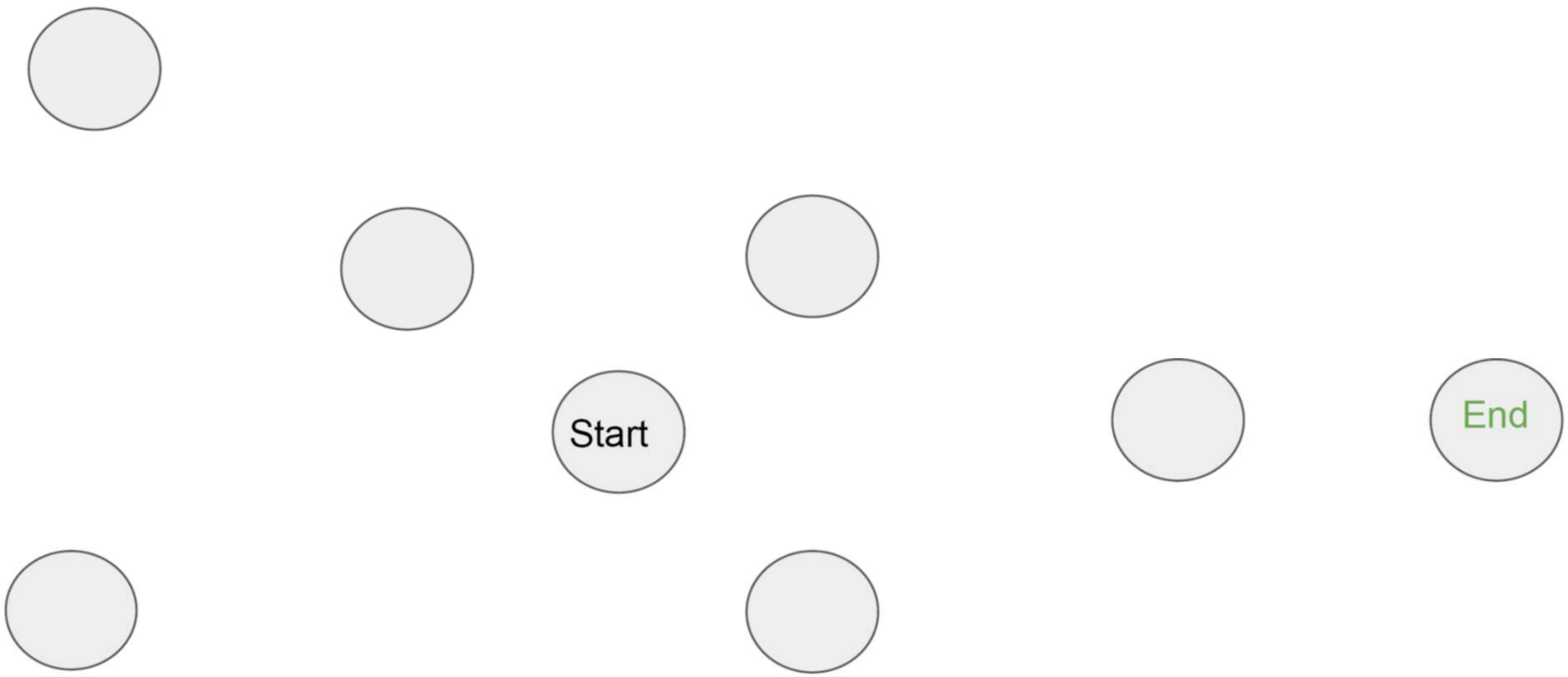
Search

- We will consider the problem of designing goal-based agents in fully observable, deterministic, discrete, static environments
  - The agent must find a *sequence of actions* that reaches the goal
  - The **performance measure** is defined by (a) reaching the goal and (b) how "expensive" the path to the goal is
  - We are focused on the process of finding the solution; while executing the solution, we assume that the agent can safely ignore its percepts (**open-loop system**)

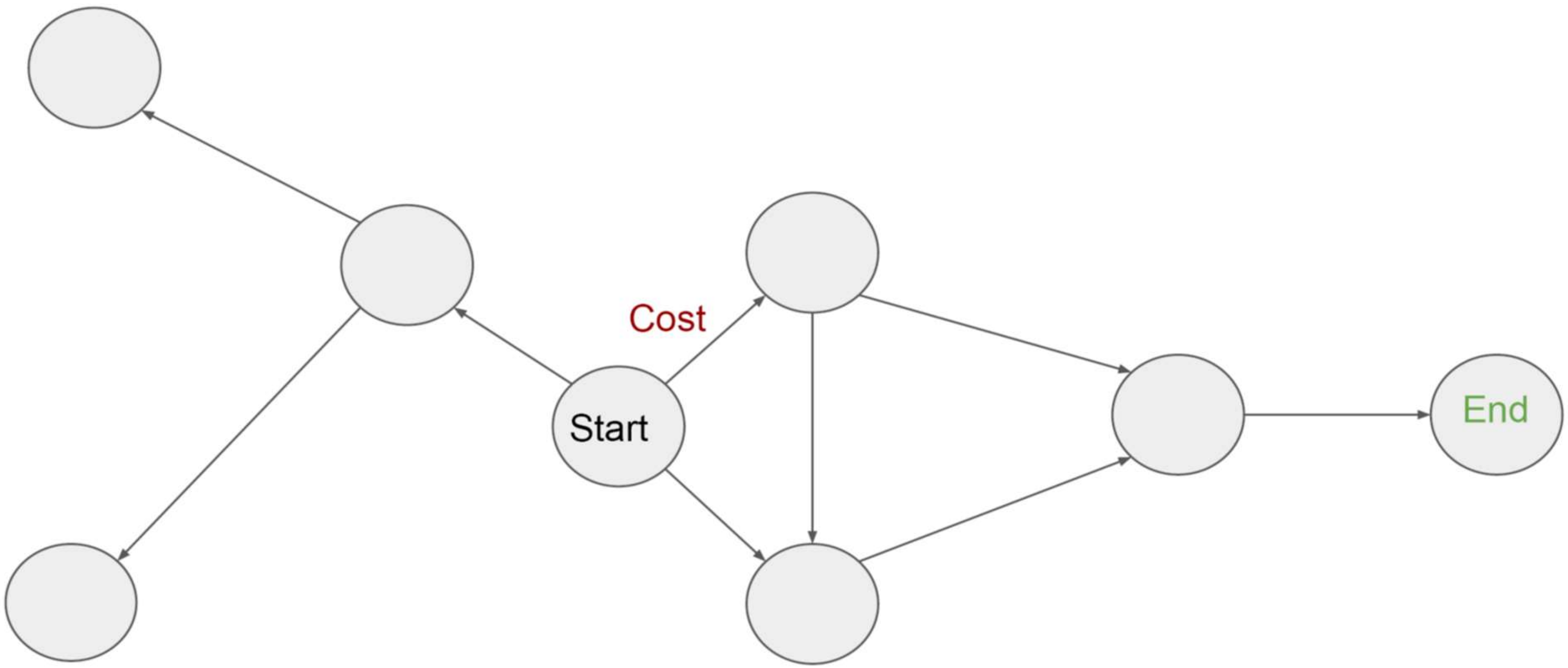
Search


How to find shortest path
to my destination?
HOME
DESTINATION

Search



Search



Search problem components

- Initial state
- Goal state
- Actions
- Transition model
  - What state results from performing a given action in a given state?
- Path cost
  - Assume that it is a sum of nonnegative *step costs*
- The optimal solution is the sequence of actions that gives the *lowest* path cost for reaching the goal

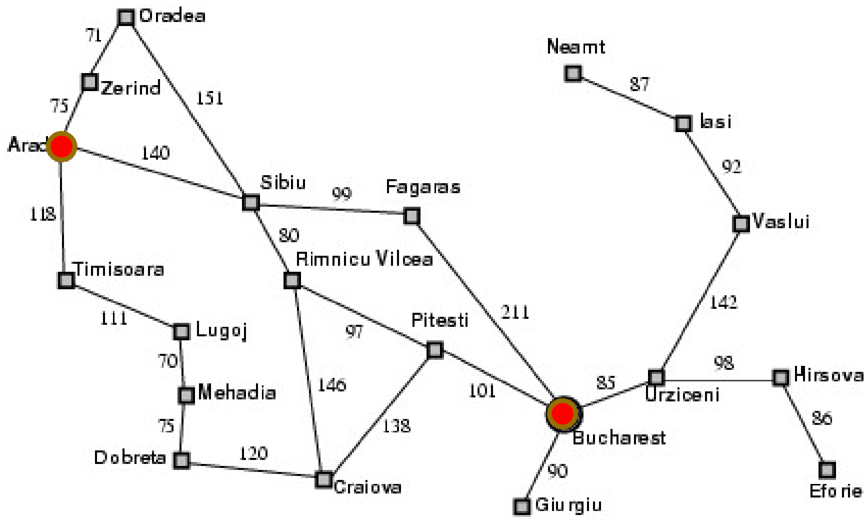
State space

- The initial state, actions, and transition model define the **state space** of the problem.
  - **The set of all states reachable** from the initial state by any sequence of actions
  - Can be represented as a **directed graph** where the nodes are states and the links between nodes are actions.

Example: Romania

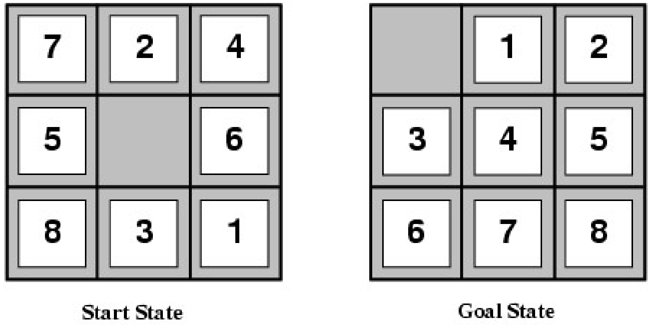
- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

- Initial state
  - Arad
- Actions
  - Go from one city to another
- Transition model
  - If you go from city A to city B, you end up in city B
- Goal state
  - Bucharest
- Path cost
  - Sum of edge costs (total distance traveled)



Example: The 8-puzzle

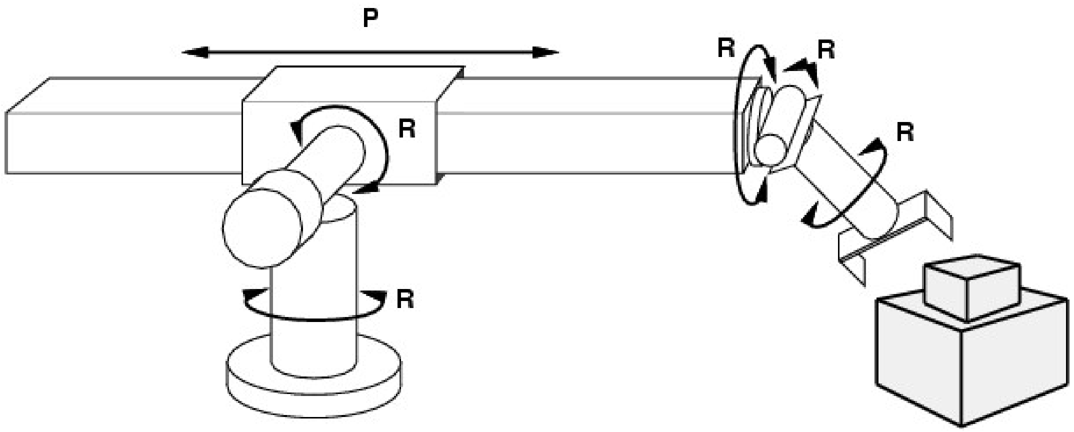
- states? locations of tiles
  - 8-puzzle: 181,440 states (9!/2)
  - 15-puzzle: ~1.3 trillion states
  - 24-puzzle: ~10^{25} states



- actions? move blank left, right, up, down
- Initial and goal states? given
- action costs? 1 per move

- Note: optimal solution of *n*-Puzzle family is NP-hard

Example: Robot motion planning

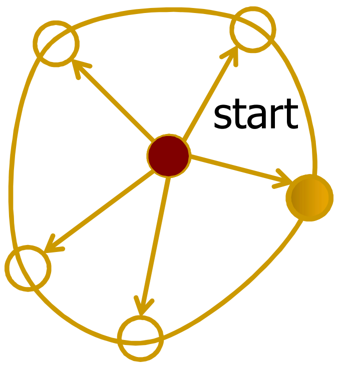


- States
  - Real-valued joint parameters (angles, displacements)
- Actions
  - Continuous motions of robot joints
- Goal state
  - Configuration in which object is grasped
- Path cost
  - Time to execute, smoothness of path, etc.

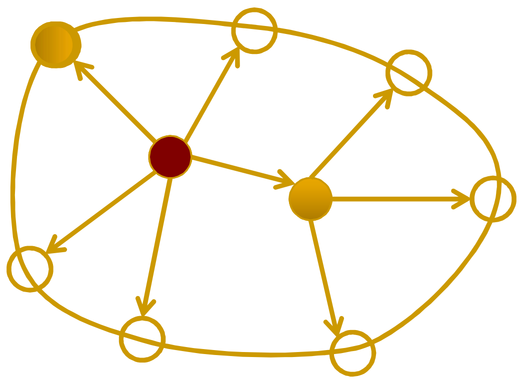
Search: Basic idea

- Let’s begin at the start state and expand it by making a list of all the possible successor states
- Maintain a frontier or a list of unexpanded states
- At each step, pick a state from the frontier to expand
- Keep going until you reach a goal state
- Try to expand as few states as possible

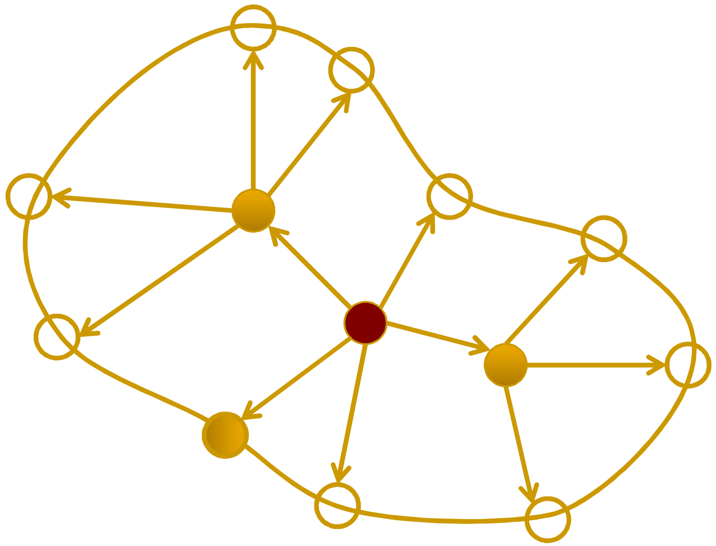
Search: Basic idea



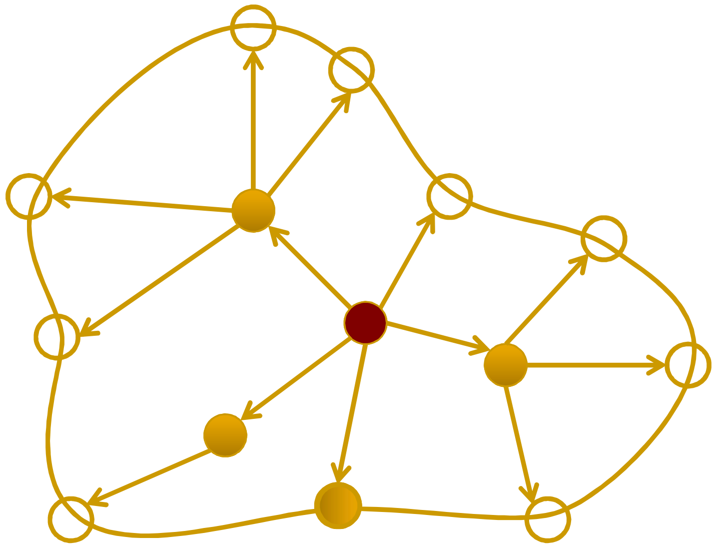
Search: Basic idea



Search: Basic idea



Search: Basic idea



General Tree-like Search

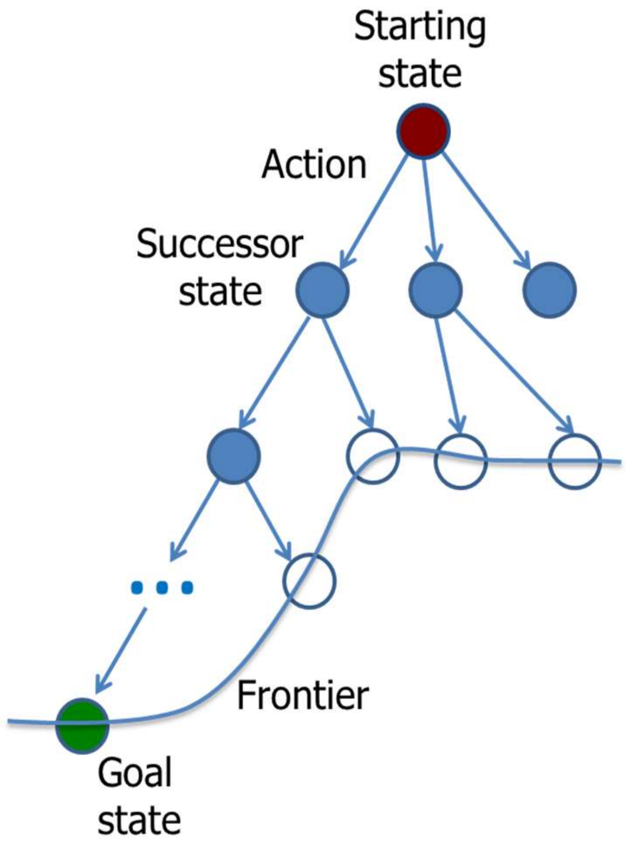
```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the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

- Important ideas:
  - Fringe/Frontier/Open List(as queue)
  - Expansion
  - Exploration strategy

- Main question: which fringe nodes to explore?

Search tree

- "What if" tree of sequences of actions and outcomes
  - ✓ The root node corresponds to the starting state
  - ✓ The children of a node correspond to the successor states of that node's state
  - ✓ A path through the tree corresponds to a sequence of actions
  - ✓ A solution is a path ending in the goal state
- Nodes vs. states
  - A state is a representation of the world, while a node is a data structure that is part of the search tree
  - A node has to keep a pointer to its parent, path cost, possibly other info

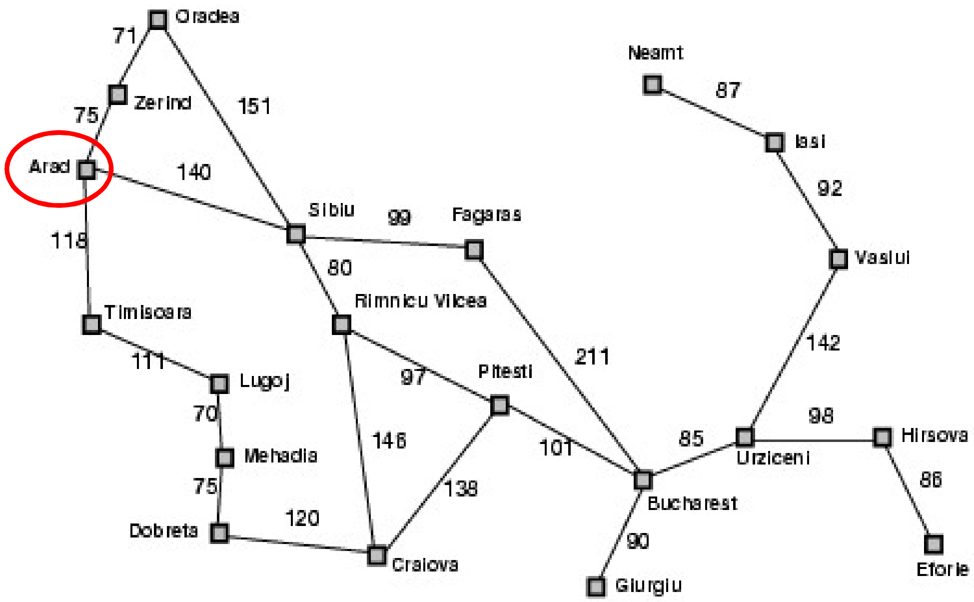


Tree-like Search Algorithm Outline

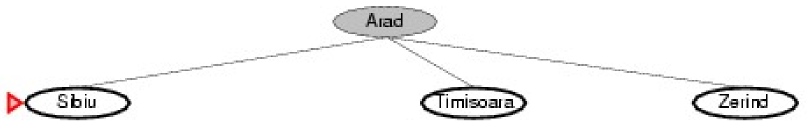
- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return the solution
  - Else **expand** the node and add its children to the frontier

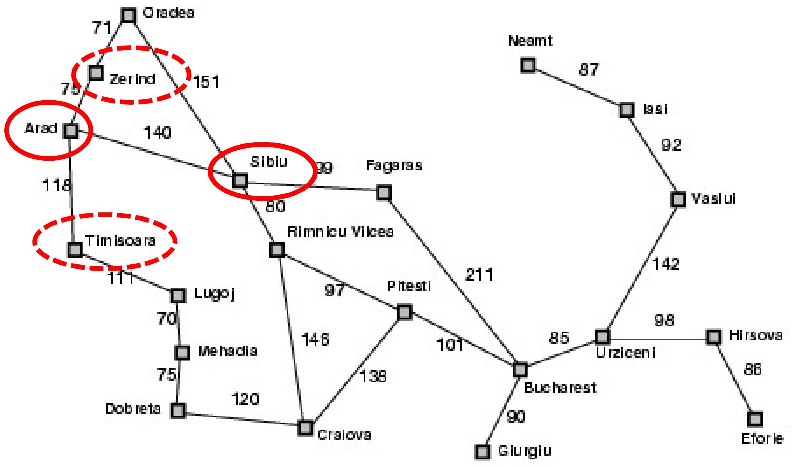
Tree-like search example

Start: Arad
Goal: Bucharest



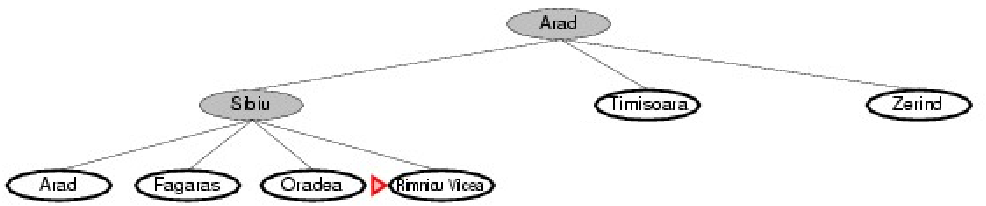
Tree-like search example



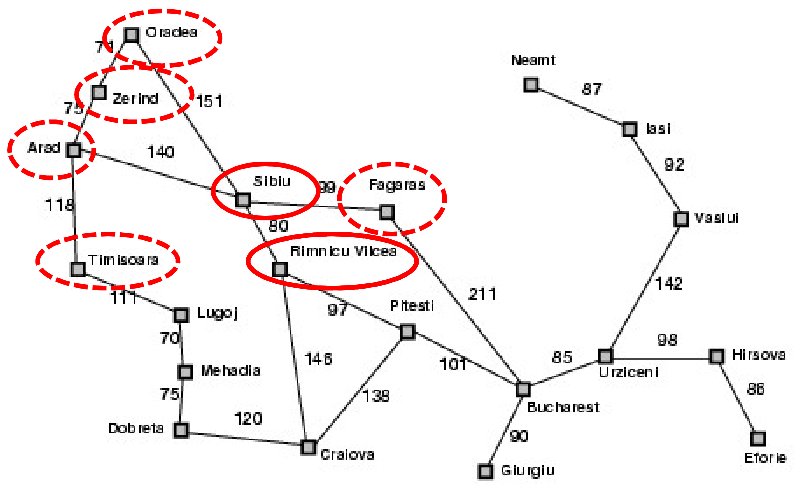


Start: Arad
Goal: Bucharest

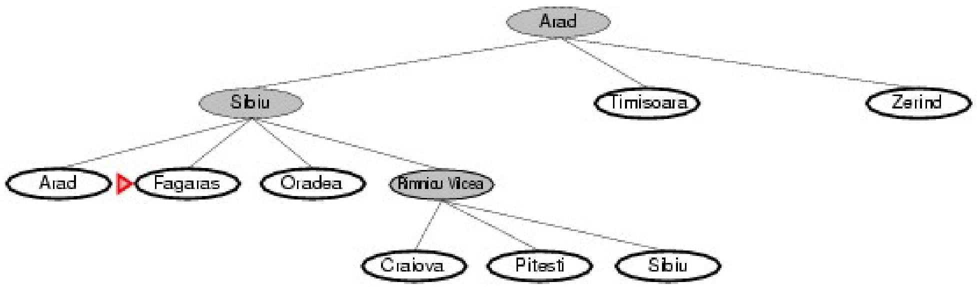
Tree-like search example



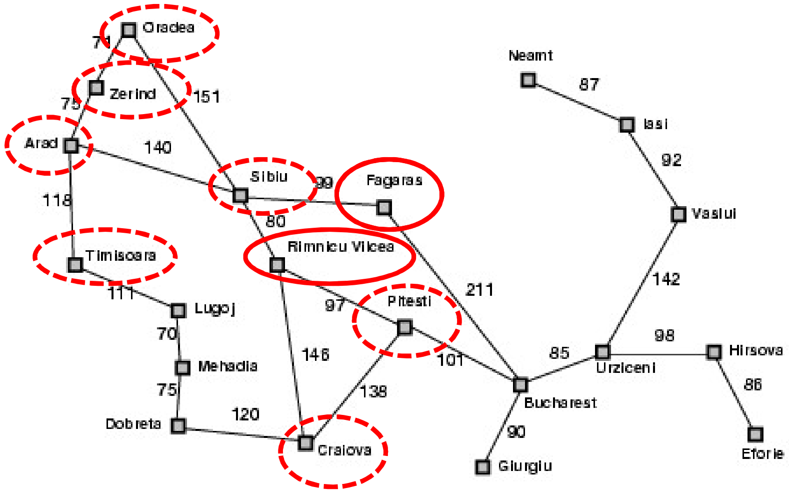
Start: Arad
Goal: Bucharest



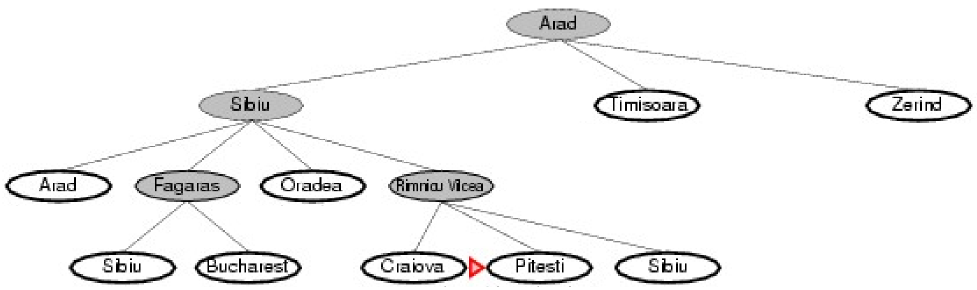
Tree-like search example



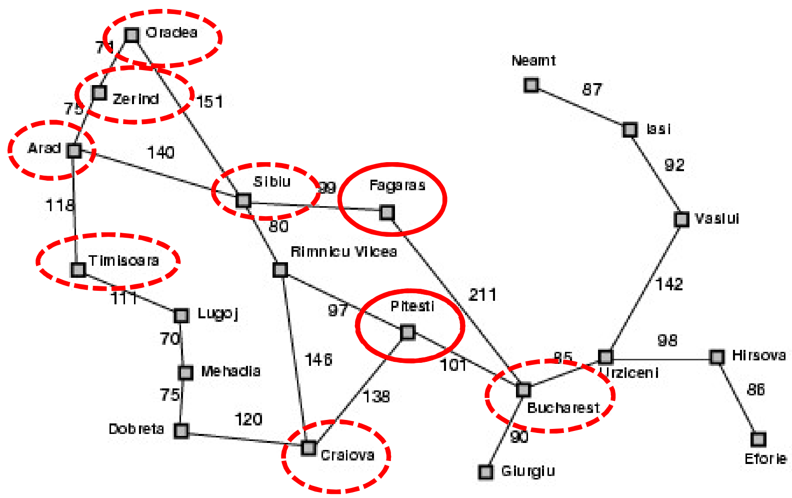
Start: Arad
Goal: Bucharest



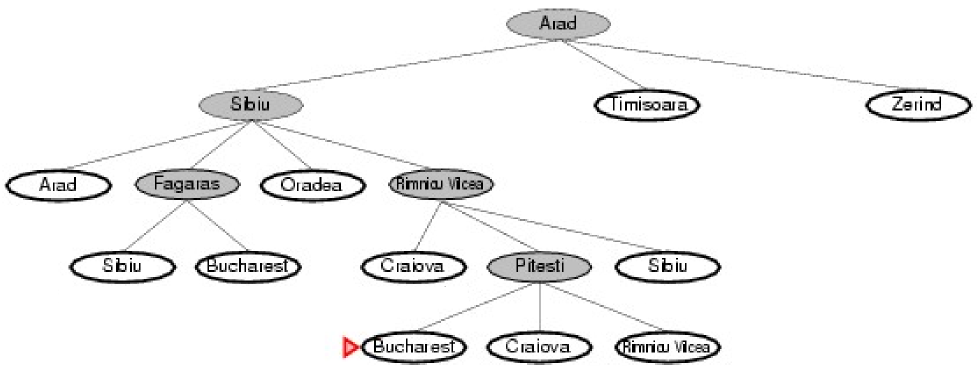
Tree-like search example



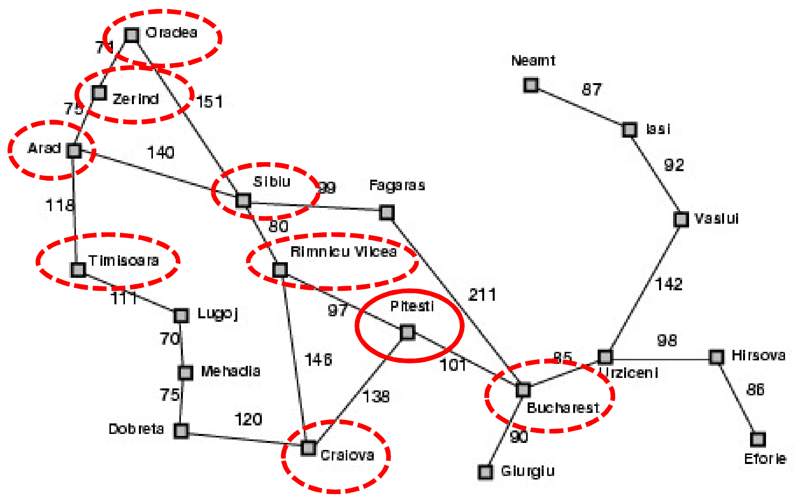
Start: Arad
Goal: Bucharest



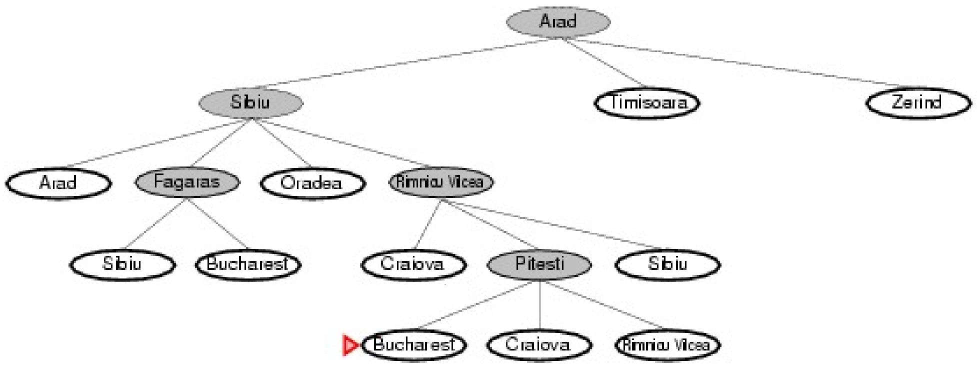
Tree-like search example



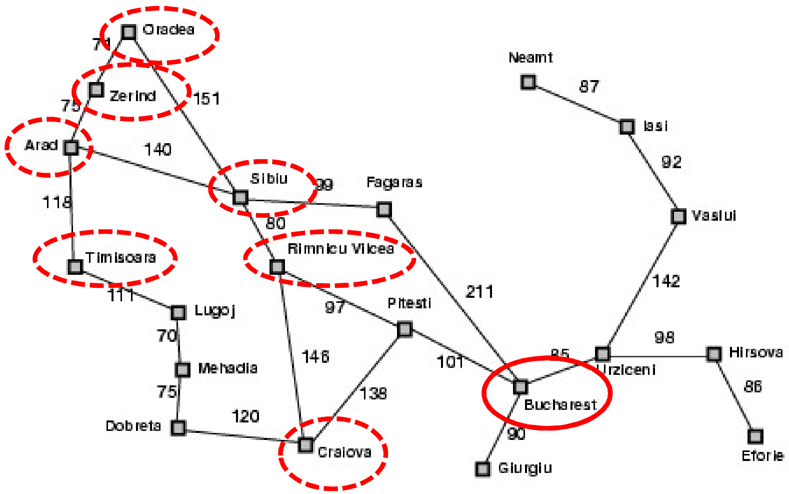
Start: Arad
Goal: Bucharest



Tree-like search example



Start: Arad
Goal: Bucharest

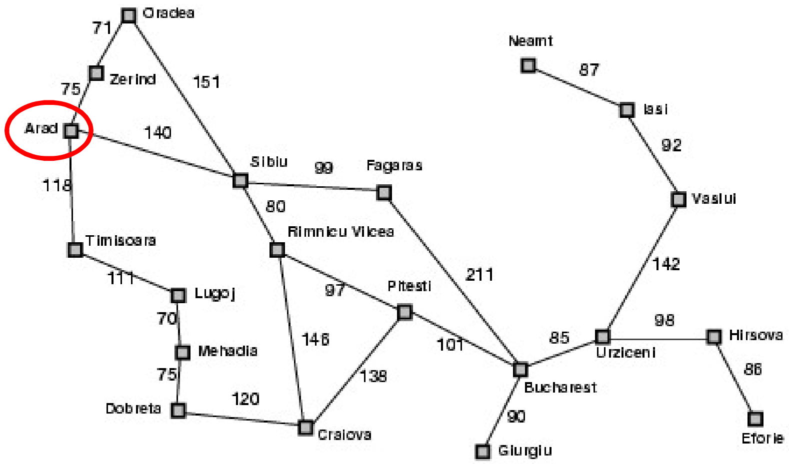


Handling repeated states

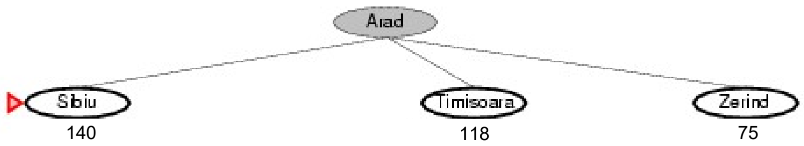
- To handle repeated states:
  - Every time you expand a node, add that state to the **explored set**; do not put explored states on the frontier again.
  - Every time you add a node to the frontier, check whether it has already existed in the frontier with a higher path cost. If yes, replace that node with the new one.

Search without repeated states

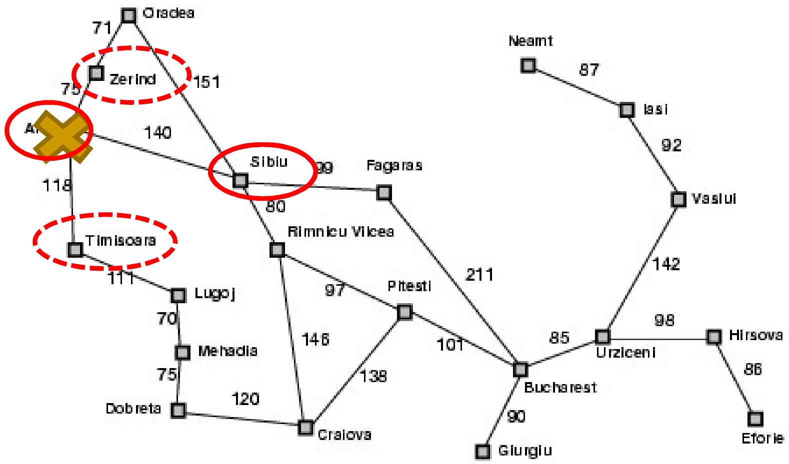
Start: Arad
Goal: Bucharest



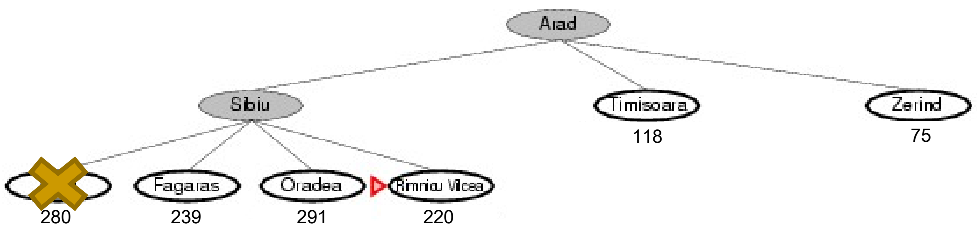
Search without repeated states

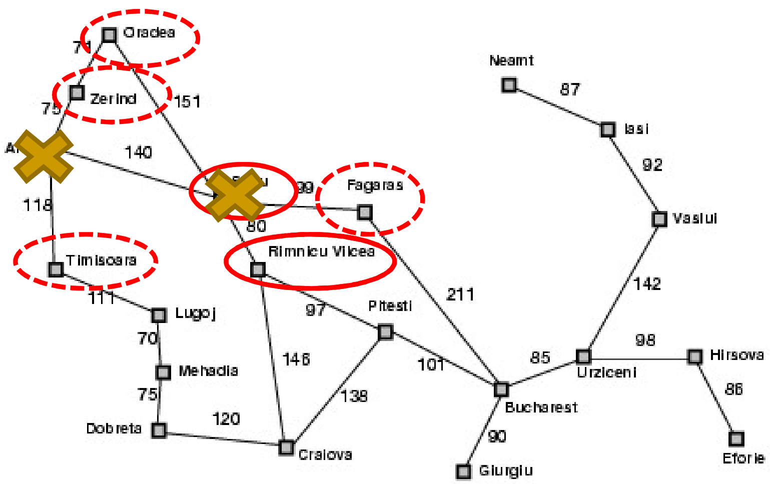


Start: Arad
Goal: Bucharest



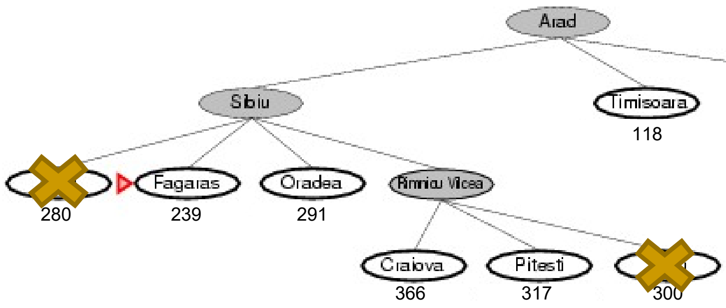
Search without repeated states

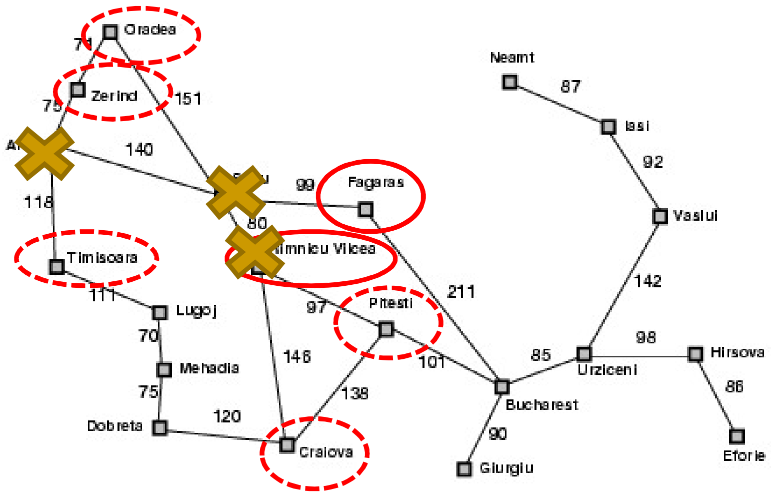




Start: Arad
Goal: Bucharest

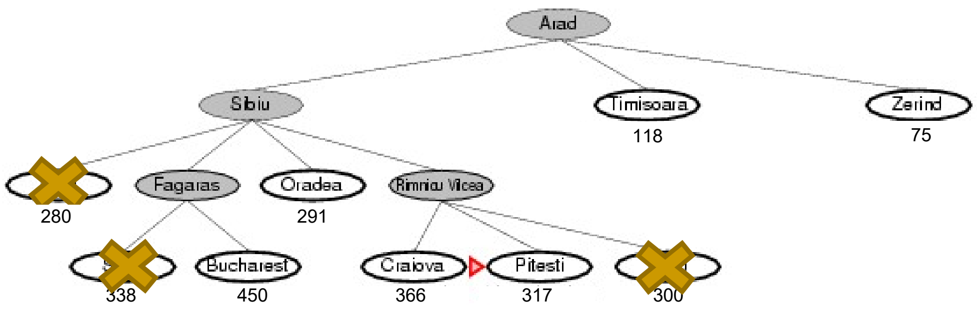
Search without repeated states



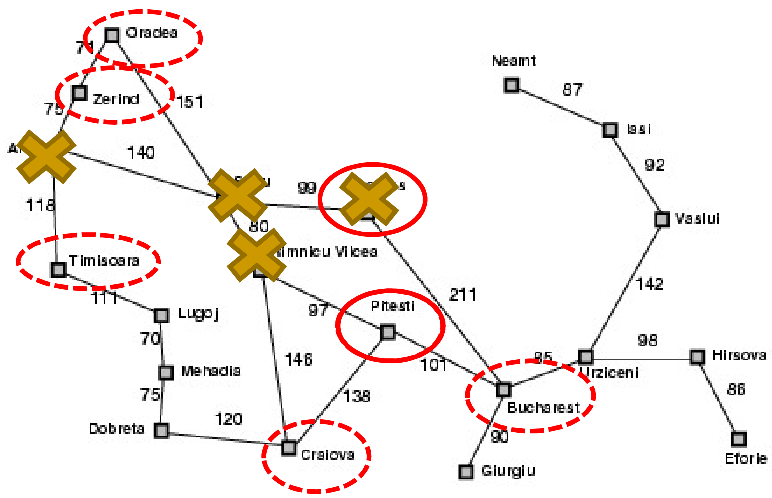


Start: Arad
Goal: Bucharest

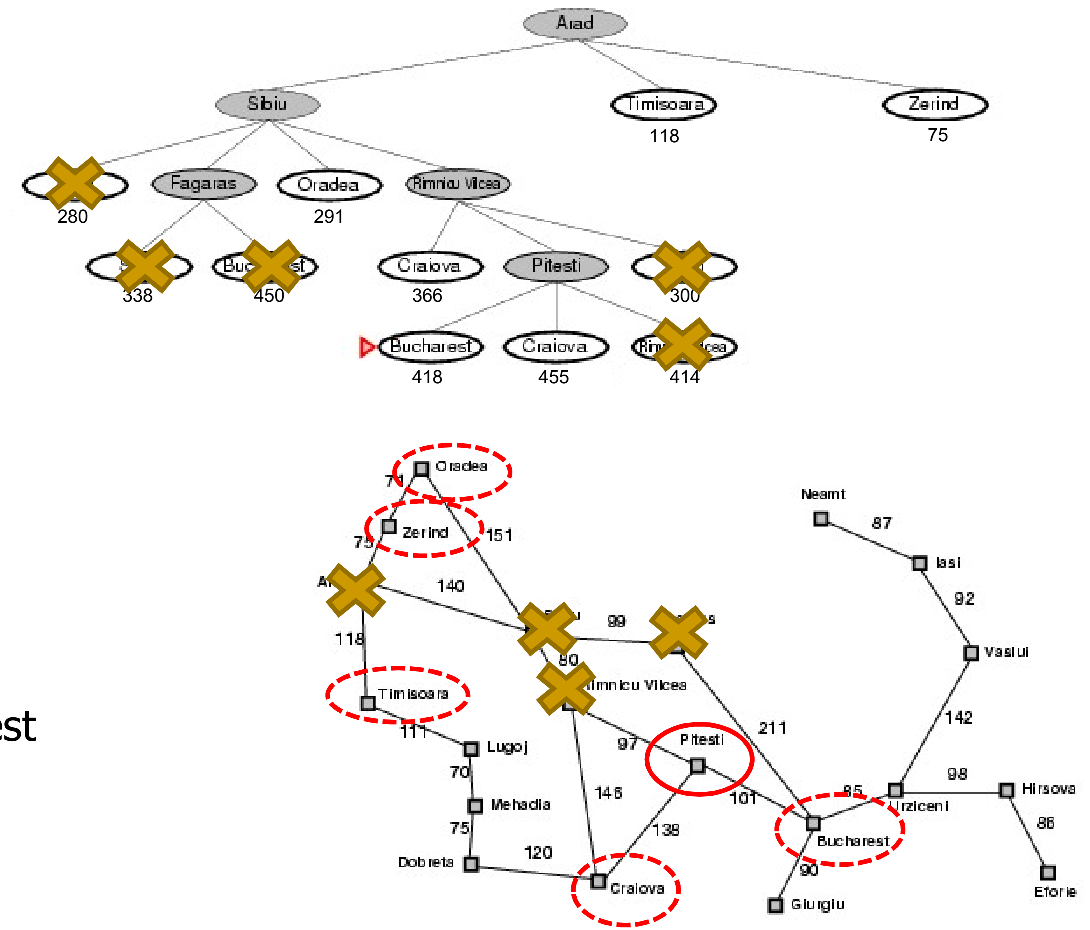
Search without repeated states



Start: Arad
Goal: Bucharest

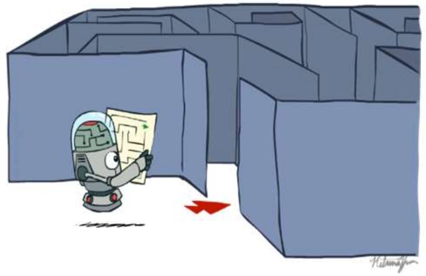


Search without repeated states



Start: Arad
Goal: Bucharest

Search



- **Search problem:**
  - States (configurations of the world)
  - Actions and costs（Plans have costs , sum of action costs)
  - Successor function (world dynamics)
  - Start state and goal test
- **Search algorithm:**
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: find the least-cost plans

Search Strategy

- A search strategy is defined by picking the order of node expansion
  - **Uninformed** search (or blind search) strategies
    - given no clue about how close a state is to the goal(s)
  - **Informed** search (or heuristic search) strategies
    - use domain-specific hints about the location of goals
    - use an ***evaluation function*** to rank nodes and select the most promising one for expansion

Outline

- Problem-Solving by Searching
- Uninformed Search Strategies
- Informed (Heuristic) Search Strategies

Uninformed search strategies

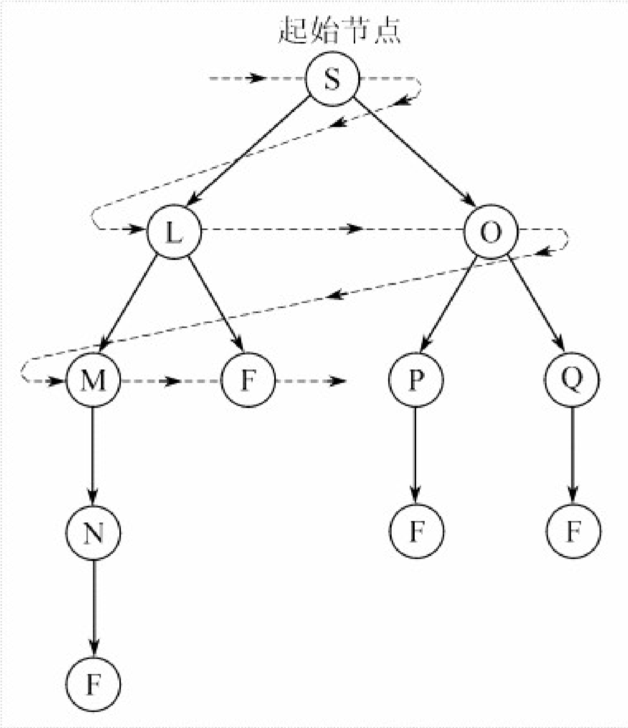
- Breadth-first search
- Depth-first search
- Iterative deepening search
- Uniform-cost search

宽度优先搜索（Breadth-first search， BFS）

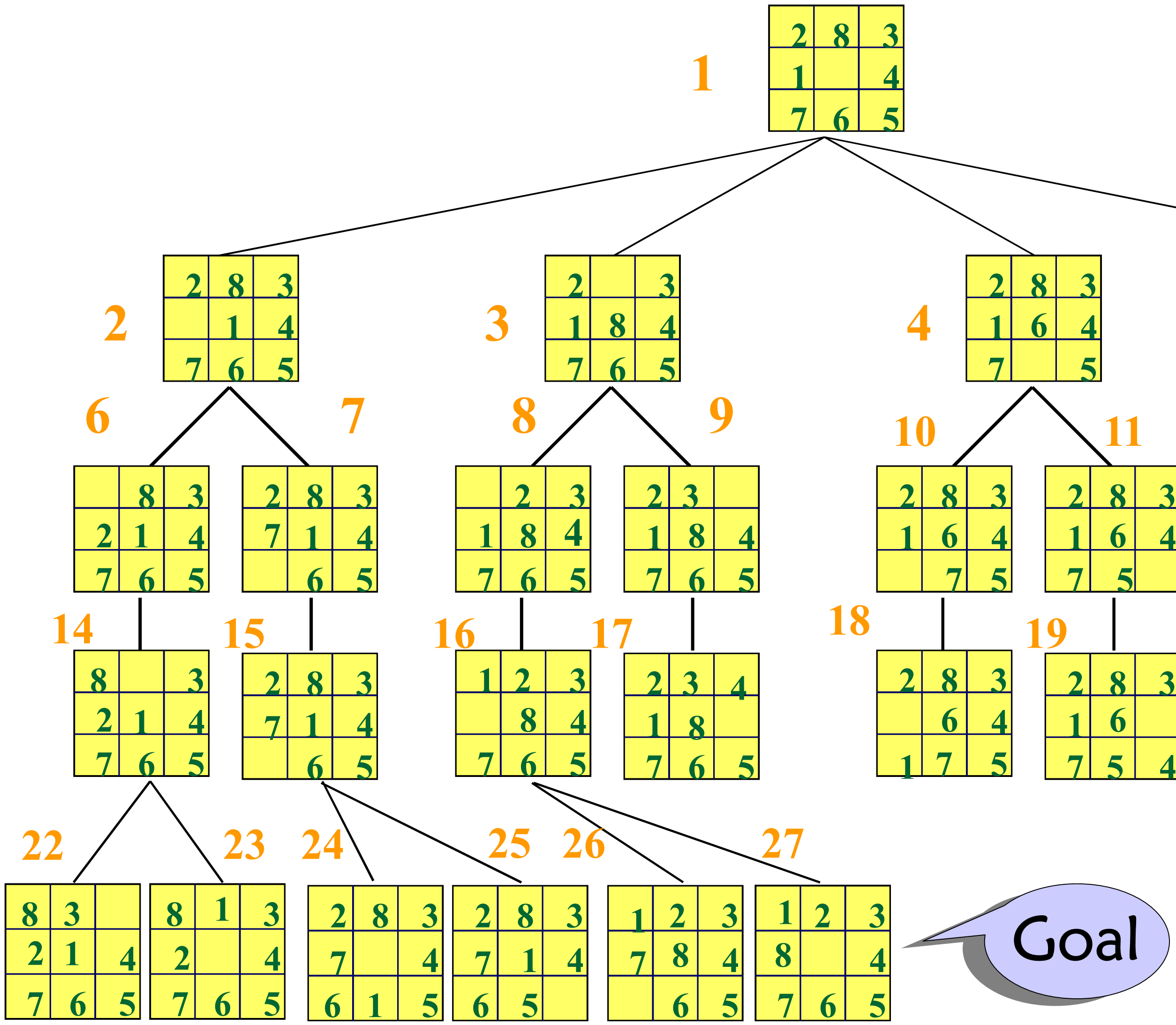
- 基本思想:

  从初始节点出发，逐层对节点进行扩展

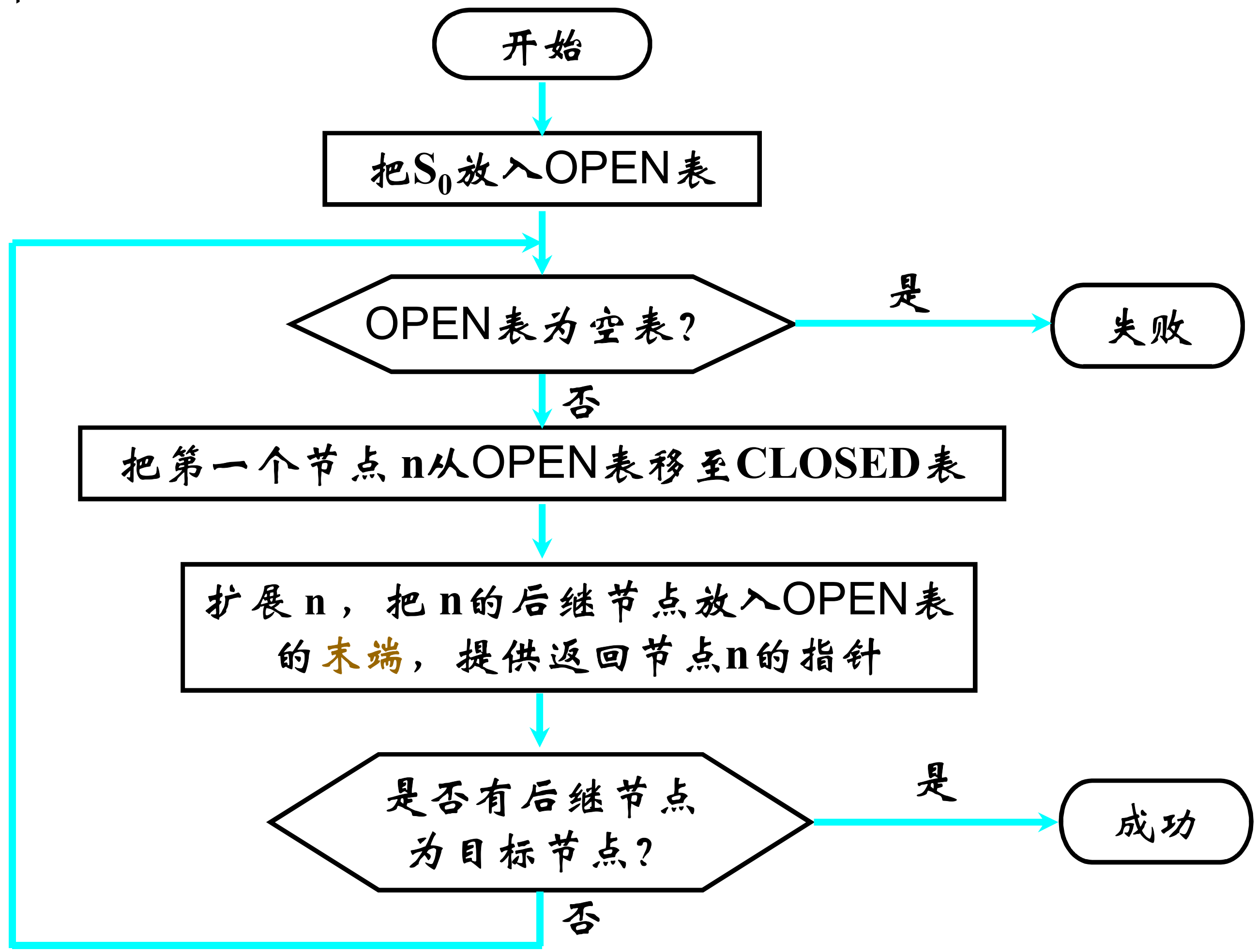
- 特点:
  - OPEN表是一个队列结构，即先进先出的数据结构
  - 搜索代价高
  - 有解时必能找到解



BFS



宽度优先算法框图


开始
把S_0放入OPEN表
OPEN表为空表?
是
失败
否
把第一个节点n从OPEN表移至CLOSED表
扩展n，把n的后继节点放入OPEN表的末端，提供返回节点n的指针
是否有后继节点为目标节点?
是
成功
否

搜索轨迹的记录

- OPEN表：
  用于存放刚生成的节点

状态节点	父节点

- CLOSED表：
  用于存放将要扩展或者已扩展的节点

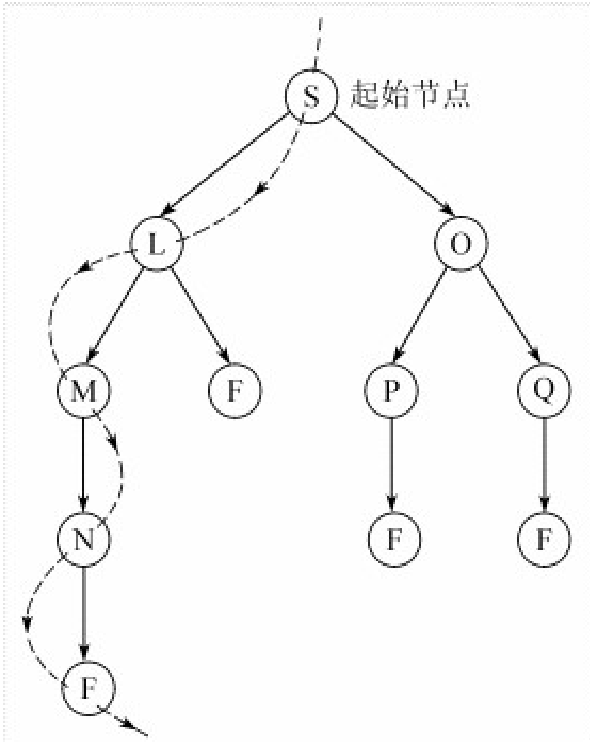
编号	状态节点	父节点

深度优先搜索
（Depth-first search，DFS）

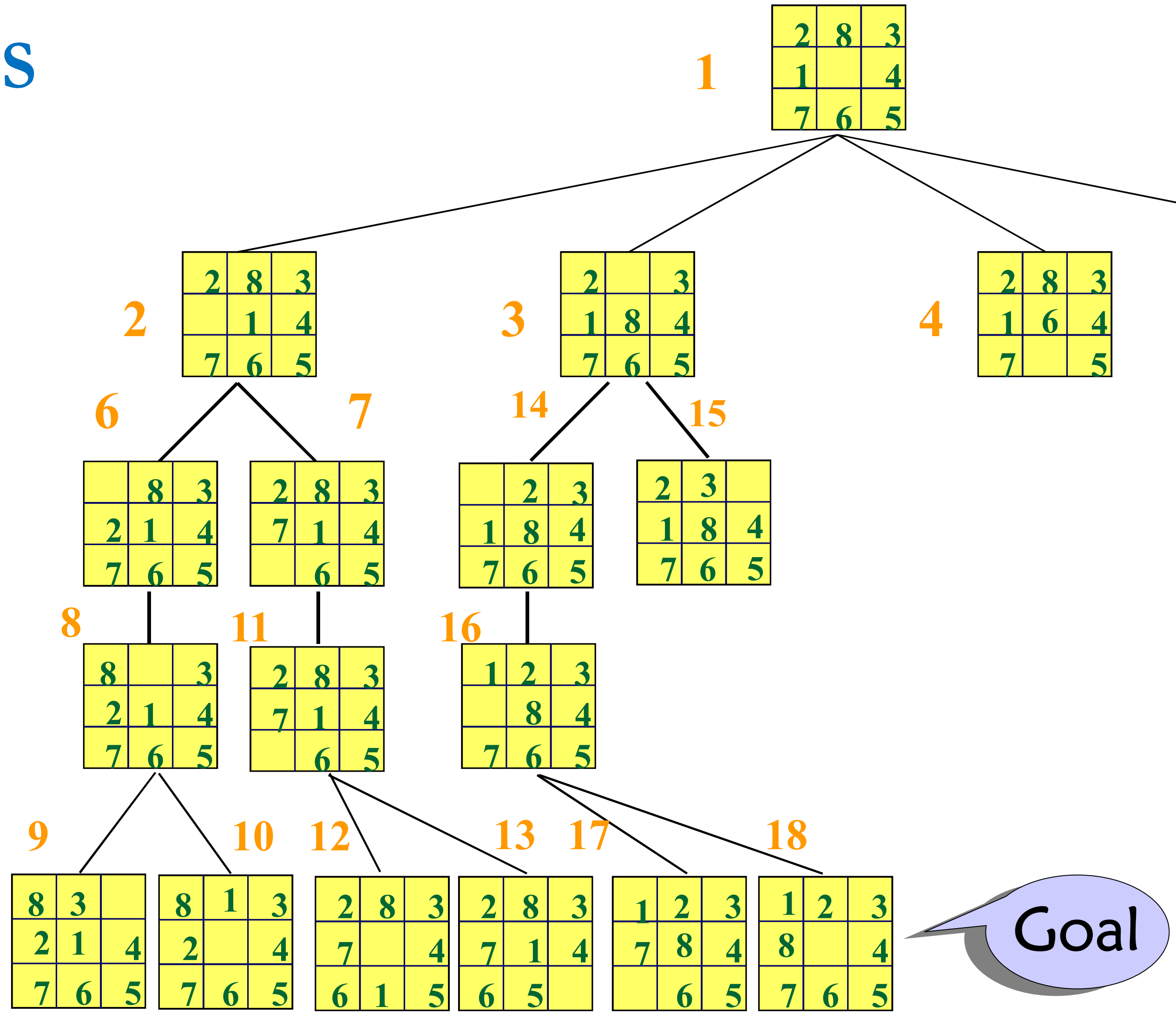
- 基本思想

　　首先扩展最新产生的(即最深的)节点

- 有界深度优先搜索 (Depth-limited search ,DLS)
- 特点
  - OPEN表是一个堆栈结构，即先进后出的数据结构
  - 效率较高
  - 无法保证找到解

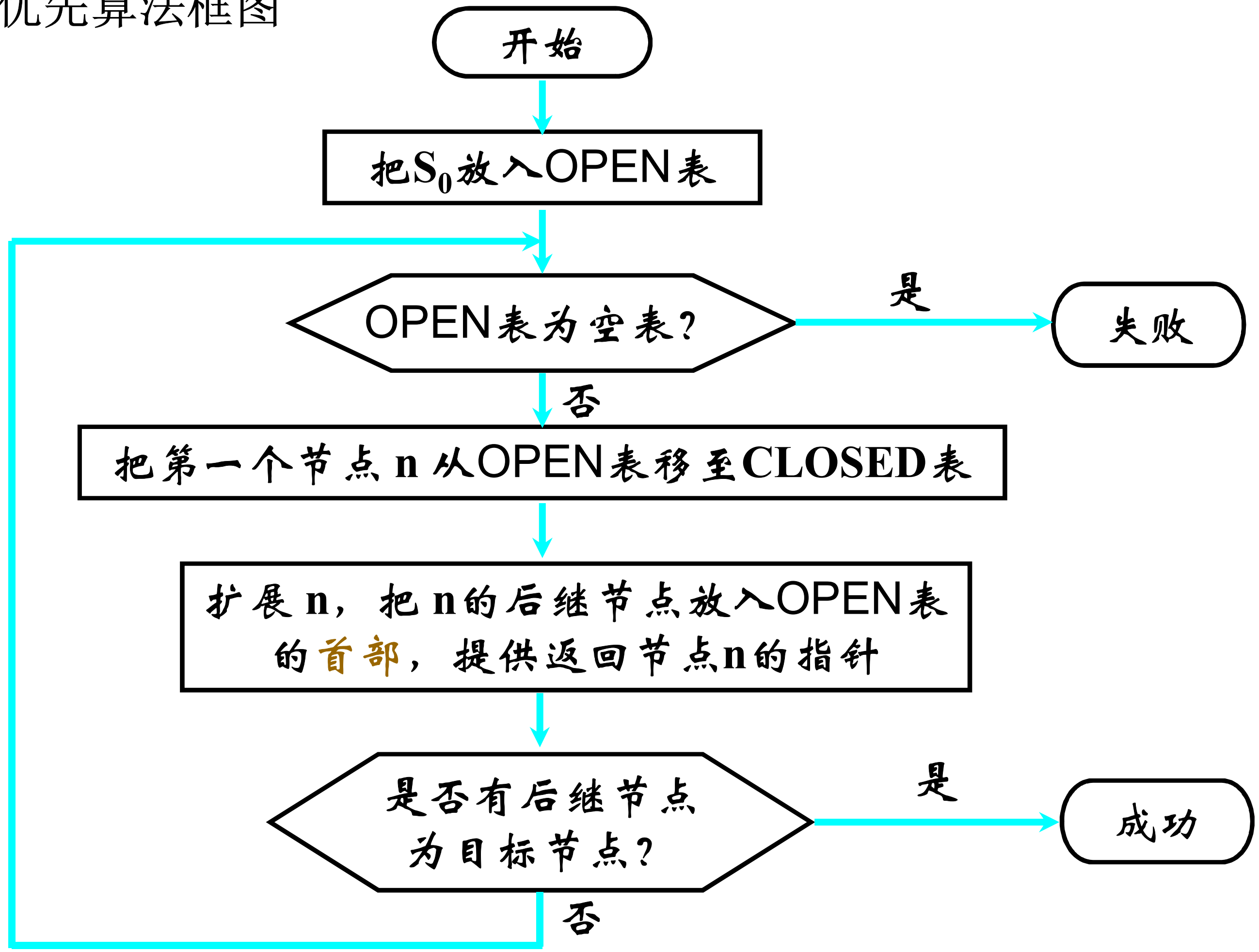


DFS



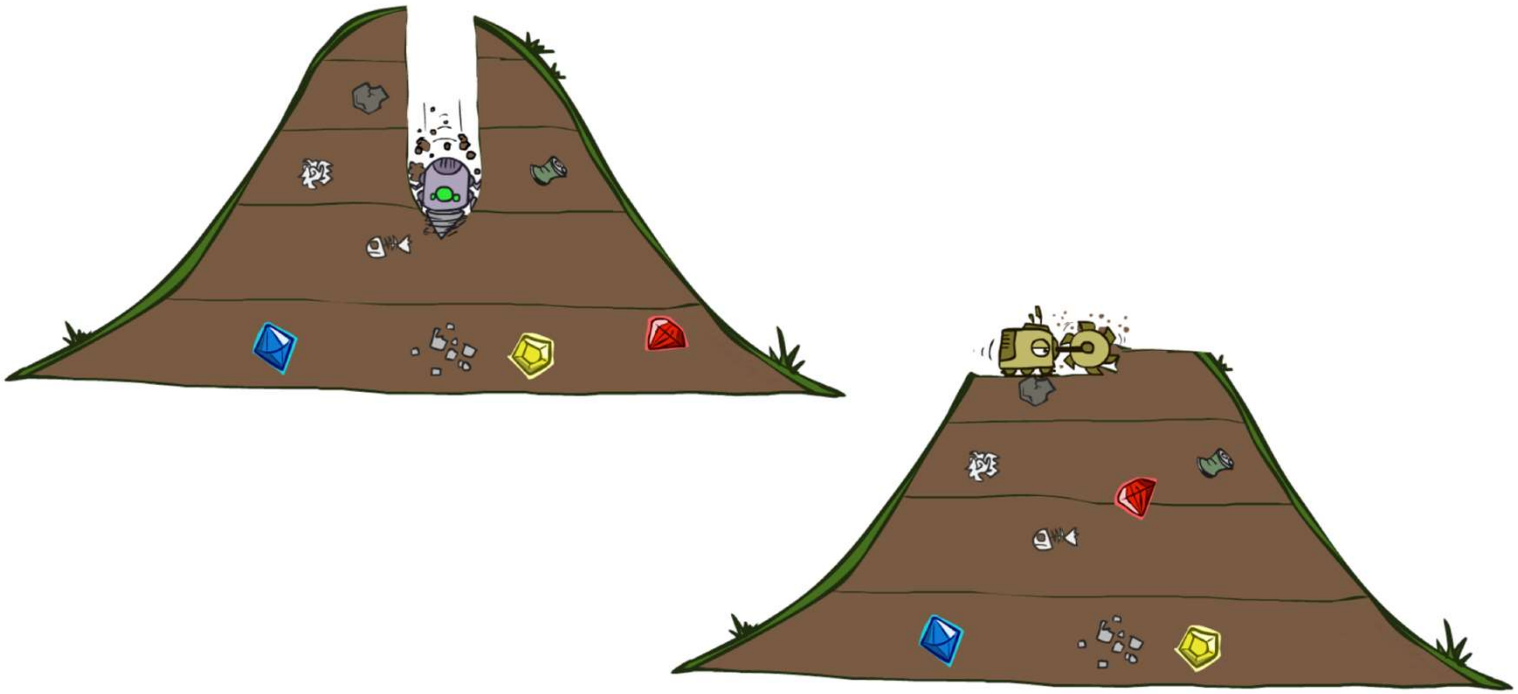
八数码难题的有界深度搜索树

深度优先算法框图



开始
把S_0放入OPEN表
OPEN表为空表？
是
失败
否
把第一个节点n从OPEN表移至CLOSED表
扩展n，把n的后继节点放入OPEN表的首部，提供返回节点n的指针
是否有后继节点为目标节点？
是
成功
否

DFS vs BFS



Measures of performance

Completeness

Guaranteed to find a solution when there is one?

Optimality

Finds an optimal solution?

Time

How long does it take to find a solution?

Space

How much memory is needed to conduct the search?

Search cost is about time cost and space cost.

Domain cost = path cost

Total cost = domain cost + search cost

Analysis of search strategies

- Evaluation criteria of strategies:
  - **Completeness**
  - **Optimality**
  - **Time complexity**
  - **Space complexity**
- Time and space complexity are measured in terms of
  - ***b****:* maximum branching factor of the search tree
  - ***d****:* depth of the optimal solution
  - ***m***: maximum length of any path in the state space (may be infinite)

Properties of breadth-first search

- **Complete?**

  Yes (if branching factor b is finite)
- **Optimal?**

  Yes – if cost = 1 per step
- **Time?**

  Number of nodes in a b-ary tree of depth d: $O(b^d)$

  (d is the depth of the optimal solution)
- **Space?**

  $O(b^d)$

- Space is the bigger problem (more than time)

Properties of depth-first search

- **Complete?**

  Fails in infinite-depth spaces, spaces with loops
- **Optimal?**

  No – returns the first solution it finds
- **Time?**

  Could be the time to reach a solution at maximum depth m: $O(b^m)$

  Terrible if m is much larger than d

  But if there are lots of solutions, may be much faster than BFS
- **Space?**

  $O(bm)$, i.e., linear space!

Iterative deepening search

- Use DFS as a subroutine
  1. Check the root
  2. Do a DFS searching for a path of depth 1
  3. If there is no path of depth 1, do a DFS searching for a path of depth 2
  4. If there is no path of depth 2, do a DFS searching for a path of depth 3…

Properties of iterative deepening search

- **Complete?**

  Yes, when the branching factor is finite

- **Optimal?**

  Yes, when the path cost is a nondecreasing function of the depth of the node

- **Time?**

  $d\,b^1 + (d\text{-}1)b^2 + \ldots + b^d = O(b^d)$

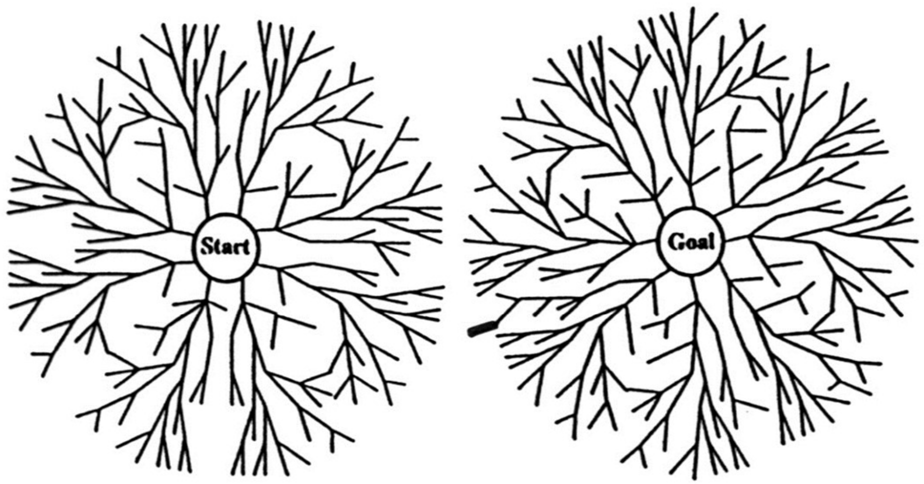
- **Space?**

  $O(bd)$

Quiz

- When to use BFS / DFS / IDS？

Bi-Directional Search



A schematic view of a bidirectional breadth-first search that is about to succeed, when a branch from the start node meets a branch from the goal node.

Bi-Directional Search

- **Complete?**

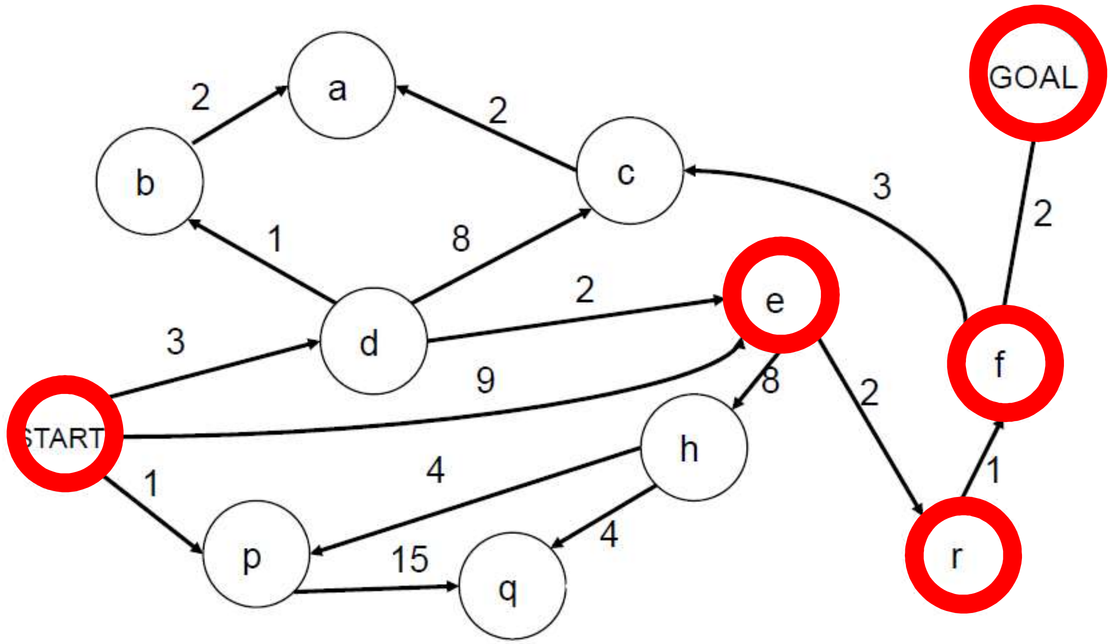
  It depends
- **Optimal?**

  It depends
- **Time?**

  $O(b^{d/2})$
- **Space?**

  $O(b^{d/2})$

Search with varying step costs

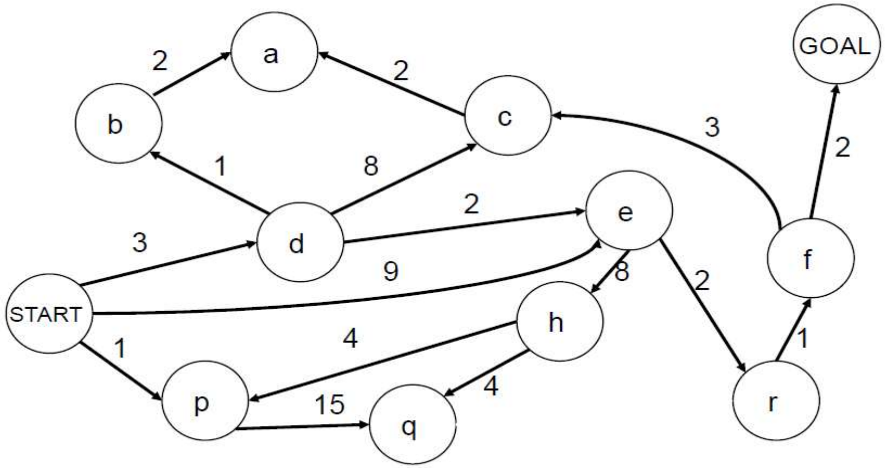


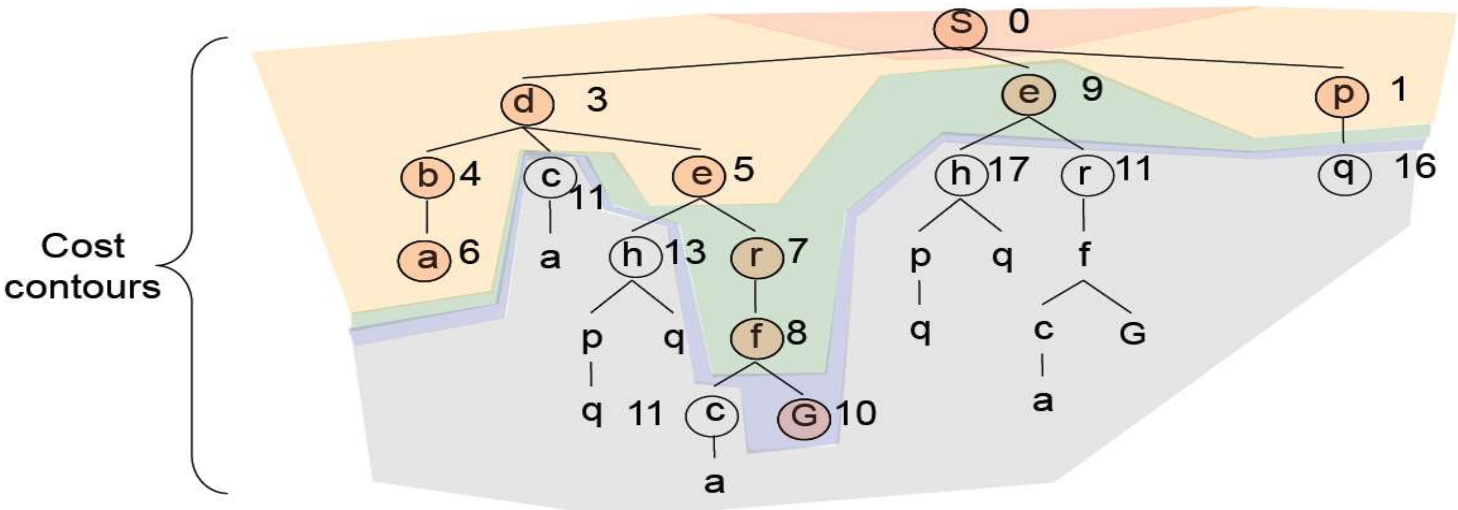
- BFS finds the path with the fewest steps, but does not always find the cheapest path

Uniform-cost search

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Implementation: *frontier* is a priority queue ordered by the path cost
- Equivalent to Dijkstra's algorithm in general

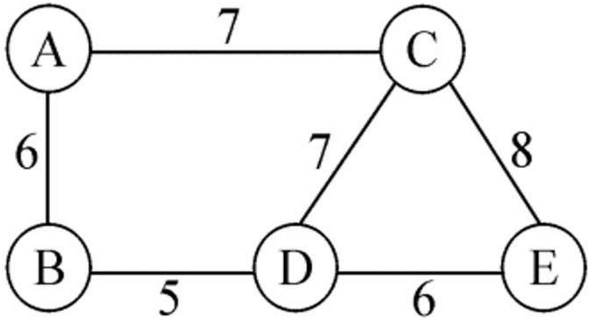
Uniform-cost search example





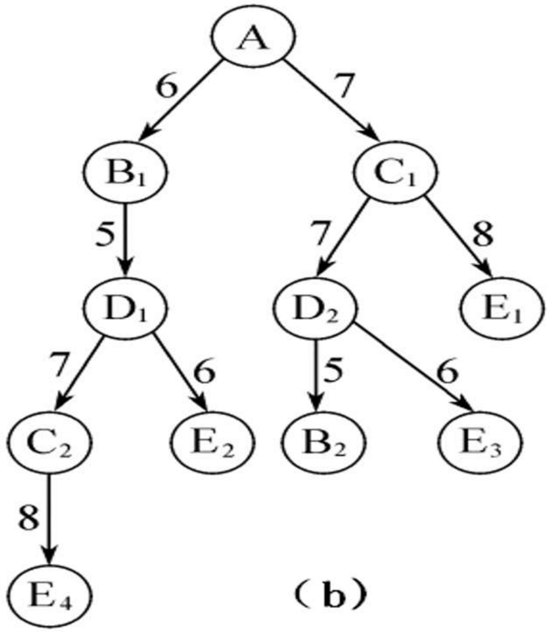
Exp：推销员旅行问题

设A、B、C、D和E是五个城市，推销员从城市A出发到城市E，已知5个城市间的交通图和每两个城市间的旅行费用，问推销员该走怎样的路线费用最省？



(a) 旅行交通图

(b) 旅行交通图的代价树



（b）

Properties of uniform-cost search

- **Complete?**

  Yes, if step cost is greater than some positive constant ε

- **Optimal?**

  Yes – nodes expanded in increasing order of path cost

- **Time?**

  Number of nodes with path cost ≤ cost of optimal solution (C^*), $O(b^{C^*/\varepsilon})$

  This can be greater than $O(b^d)$: the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps

- **Space?**

  $O(b^{C^*/\varepsilon})$

Uninformed search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	

b: maximum branching factor of the search tree
d: depth of the optimal solution
m: maximum length of any path in the state space
C*: cost of optimal solution
g(n): cost of path from start state to node n

Outline

- Problem-Solving by Searching
- Uninformed Search Strategies
- Informed (Heuristic) Search Strategies

棋局的穷举

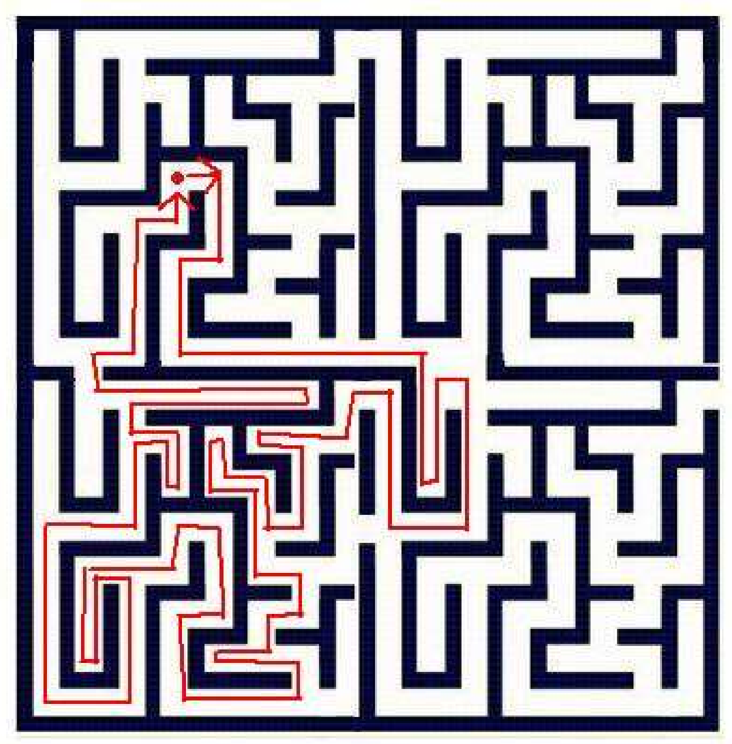
棋局数：

- 一字棋：$9! \approx 3.6 \times 10^5$
- 西洋棋：10^{78}
- 国际象棋：10^{120}
- 围棋：10^{761}

假设每步可以选择一种棋局,用并行速度(10^{-104}秒/步)计算,国际象棋的算法需用10^{16}年,即1亿亿年才可以算完。

How does Theseus find the way out of Minotaur's labyrinth?

Ariadne’s clew:



Informed search strategies

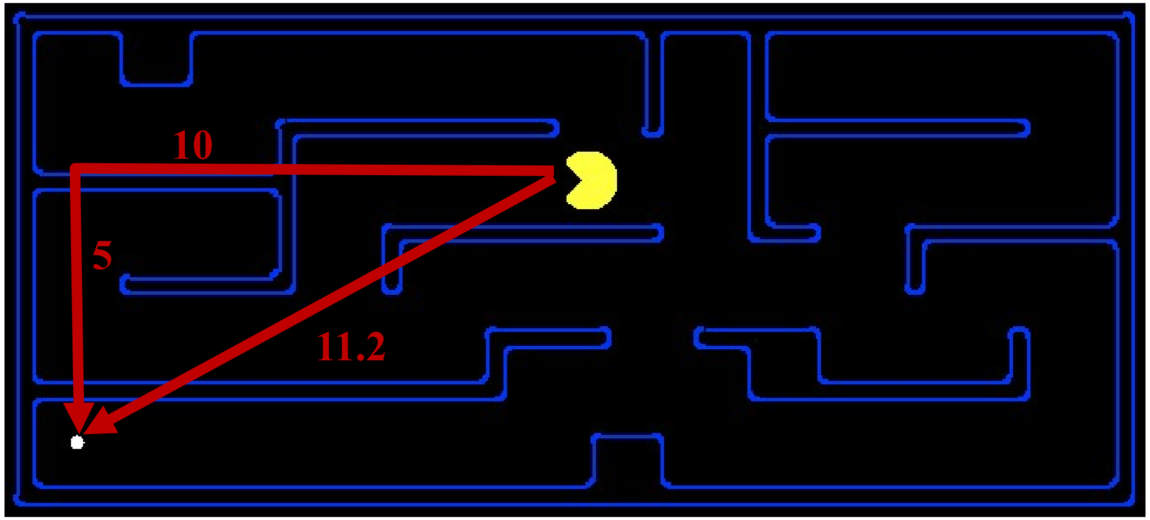
- Idea: give the algorithm “hints” about the desirability of different states
- Use an ***evaluation function*** to rank nodes and select the most promising one for expansion

Informed search strategies

- Greedy best-first search
- A* search

Greedy Best-first Search

- Strategy: expand a node that you think is closest to a goal state
  - **heuristic function**: *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing

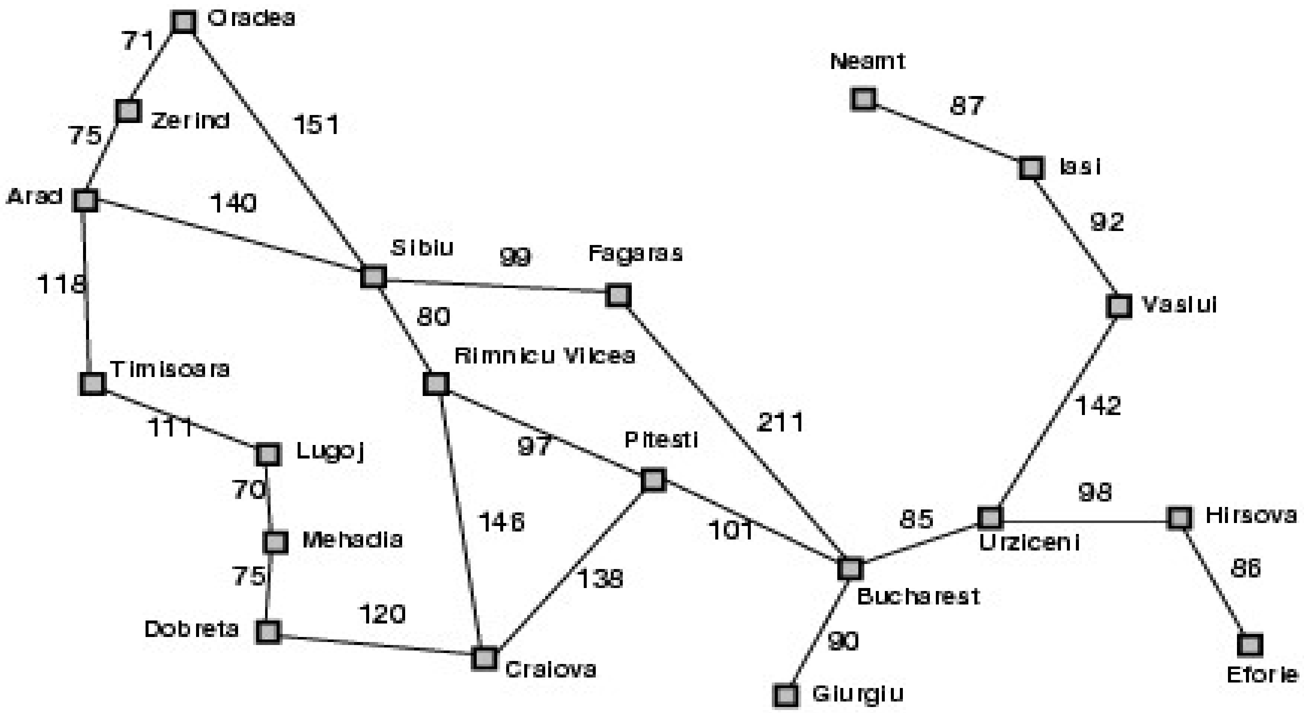


Greedy best-first search

- Expand the node that has the lowest value of the heuristic function $h(n)$
  - Try to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.

Heuristic for the Romania problem

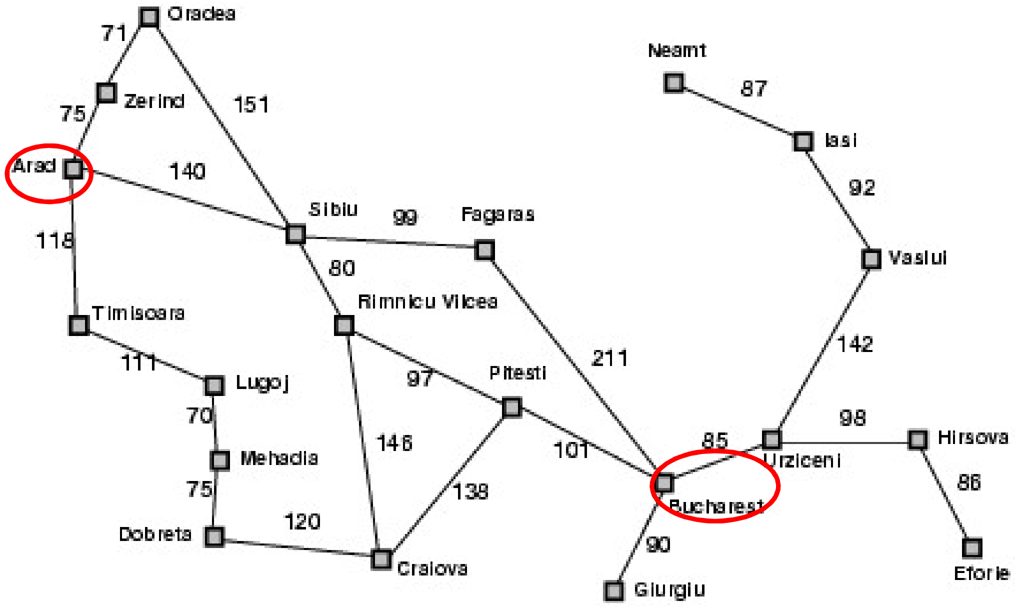
h(n)——straight-line distances to Bucharest



Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

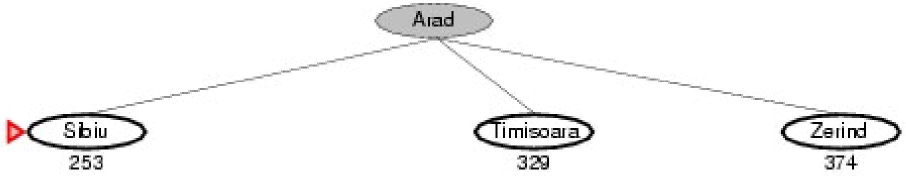


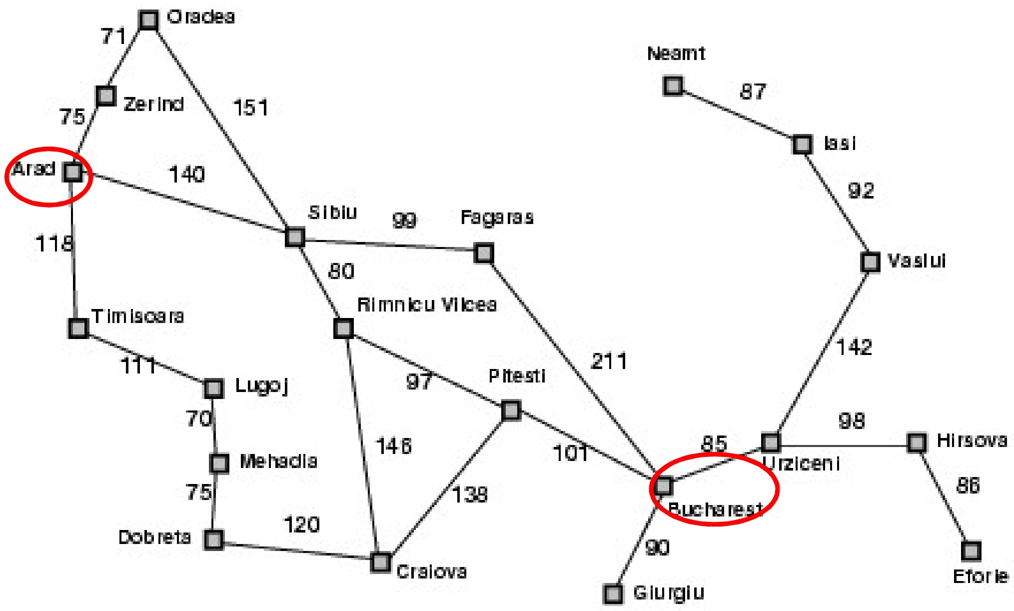
h(x)

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

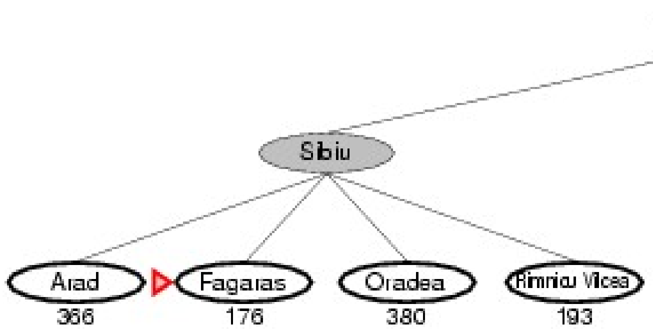
- Expand the node that seems closest…

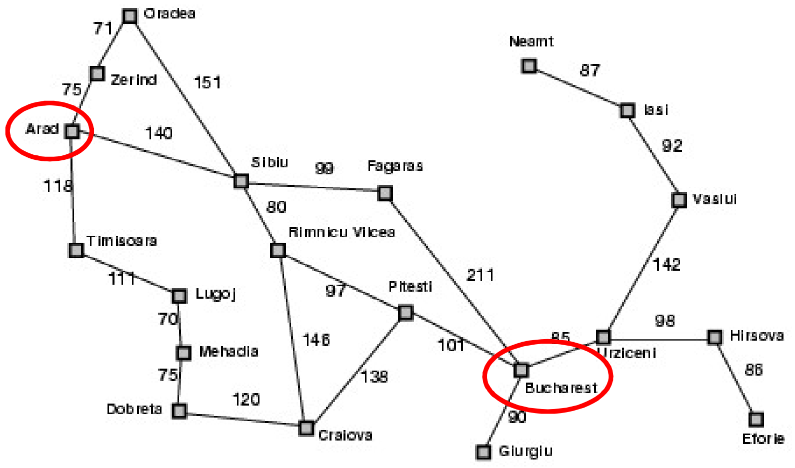




Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example

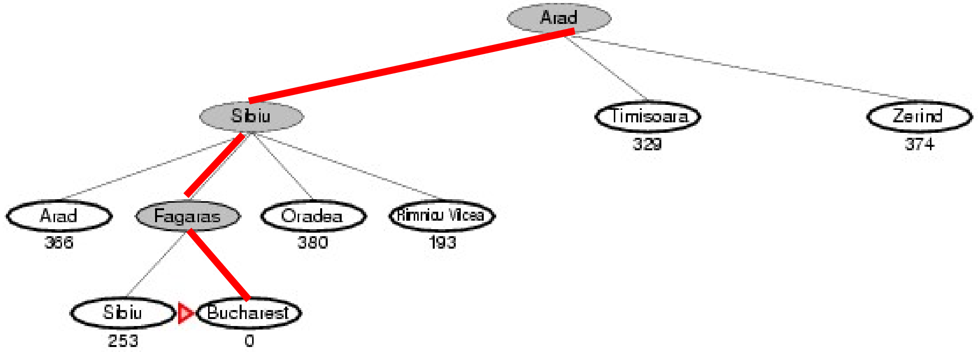


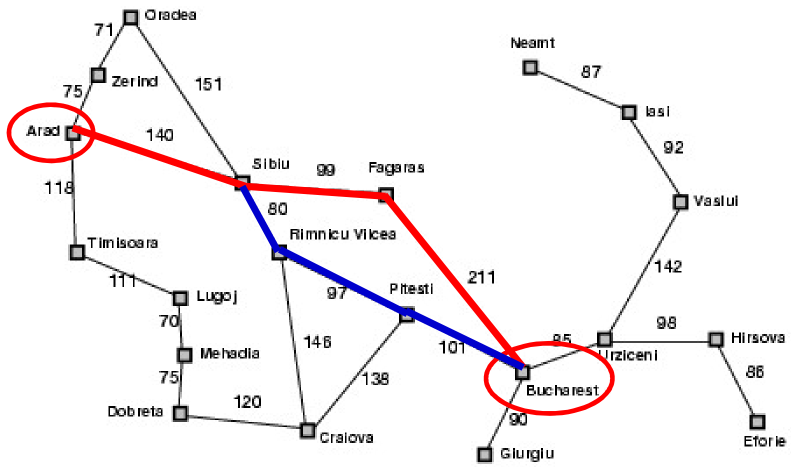


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy best-first search example





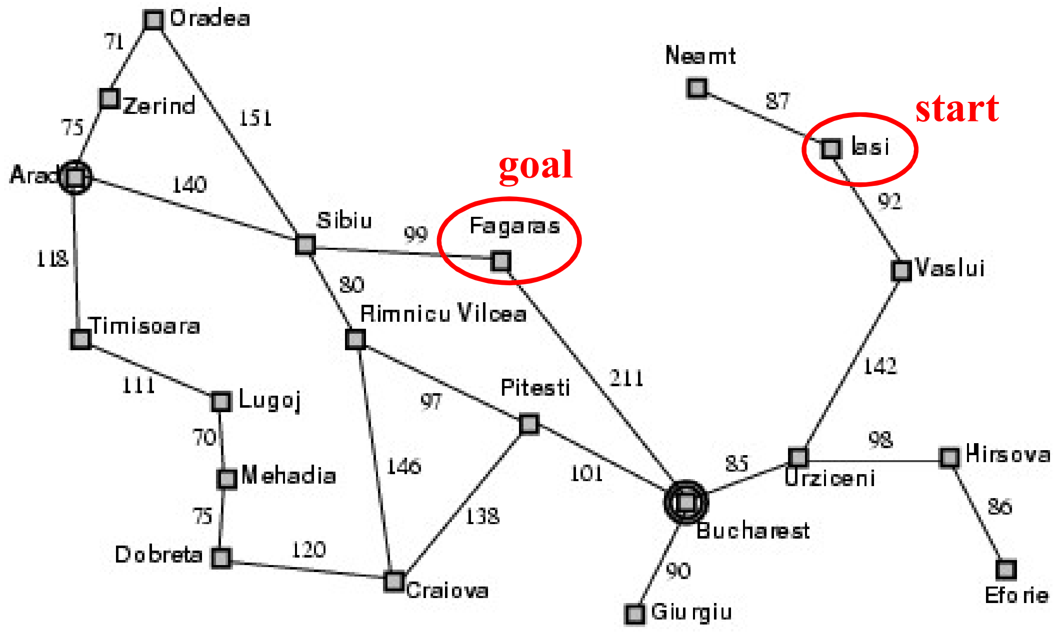
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops



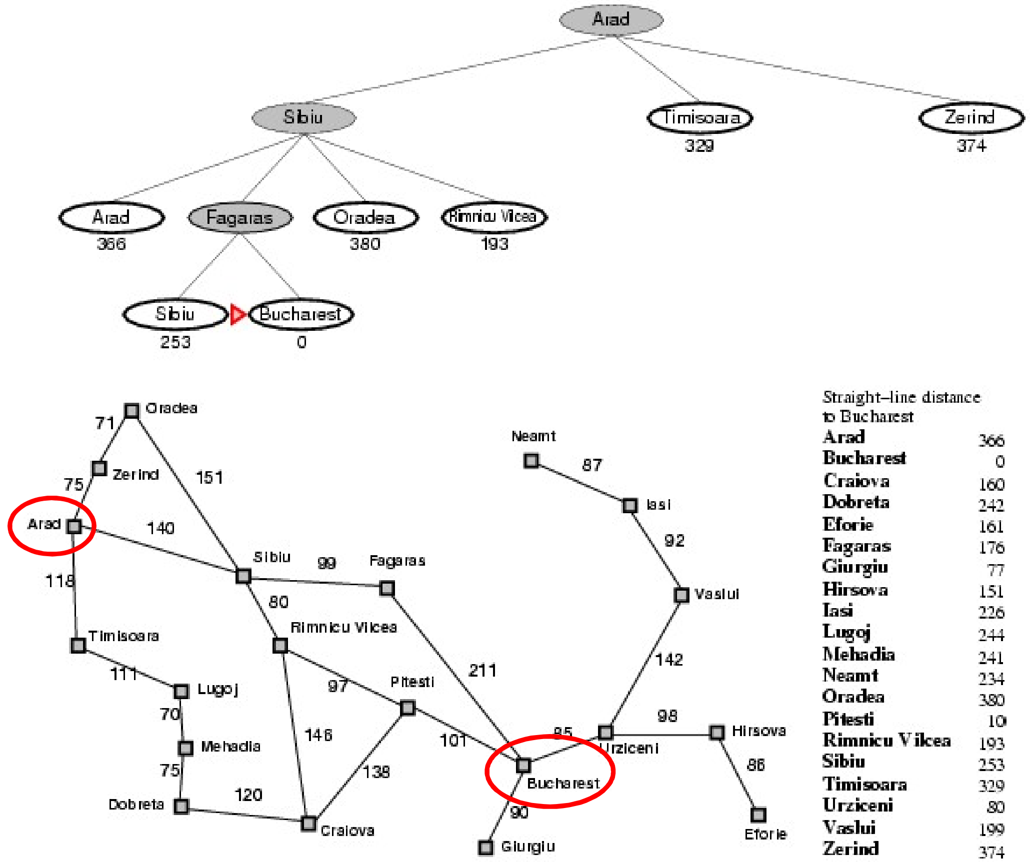
Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops

- **Optimal?**

  No



Properties of greedy best-first search

- **Complete?**

  No – can get stuck in loops
- **Optimal?**

  No
- **Time?**

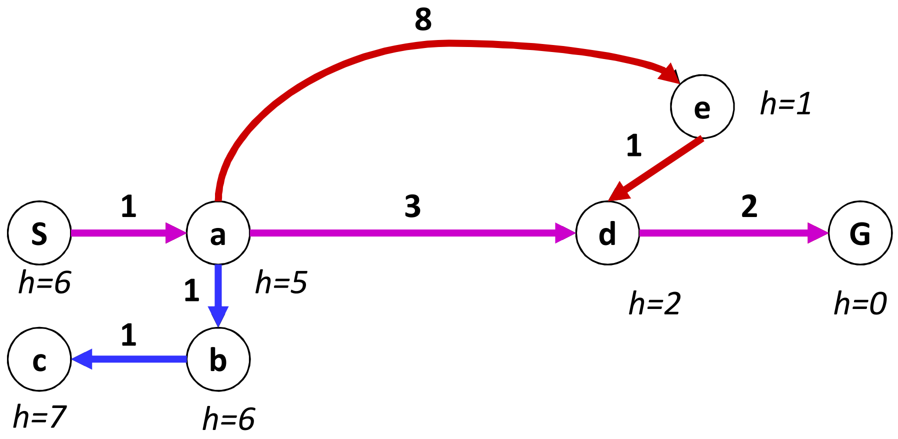
  Worst case: $O(b^m)$

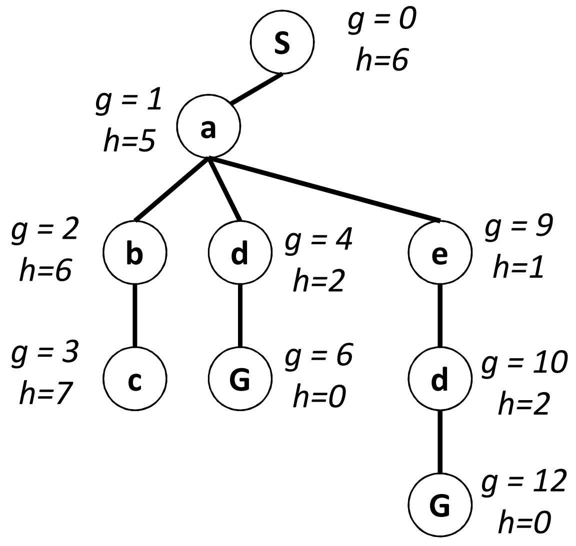
  Can be much better with a good heuristic
- **Space?**

  Worst case: $O(b^m)$

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)

Admissible heuristics

- A heuristic $h(n)$ is admissible if

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

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

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.
- Example: straight line distance never overestimates the actual road distance.

A^* search

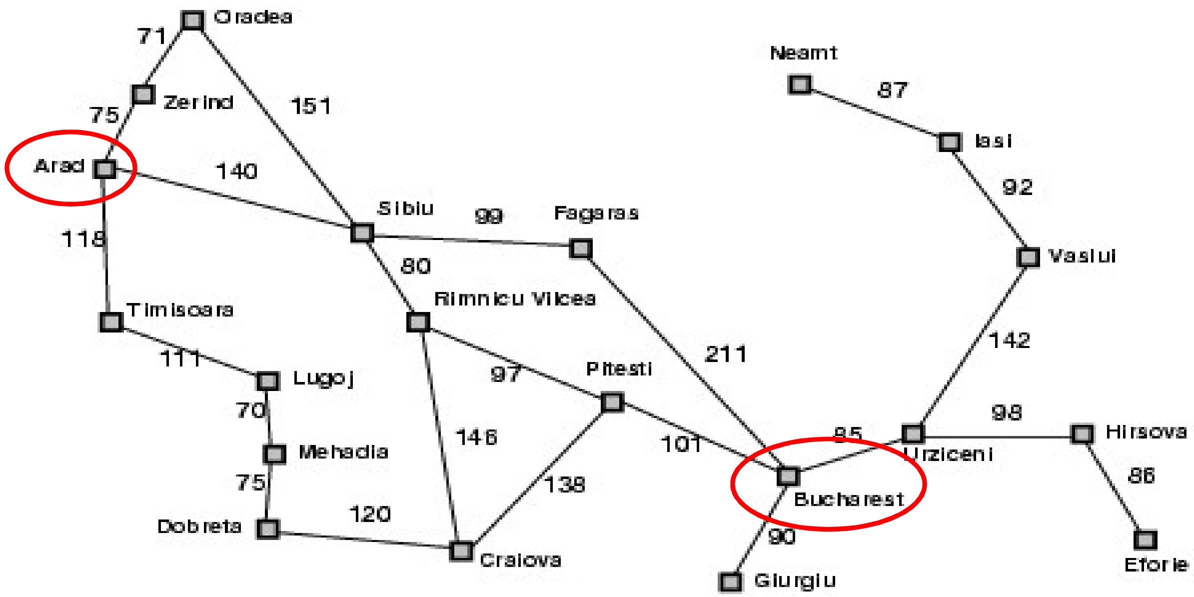
- Idea: avoid expanding paths that are already expensive
- The **evaluation function** $f(n)$ is the estimated total cost of the path through node n to the goal:

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

$g(n)$: cost so far to reach n (path cost)

$h(n)$: estimated cost from n to goal (heuristic)

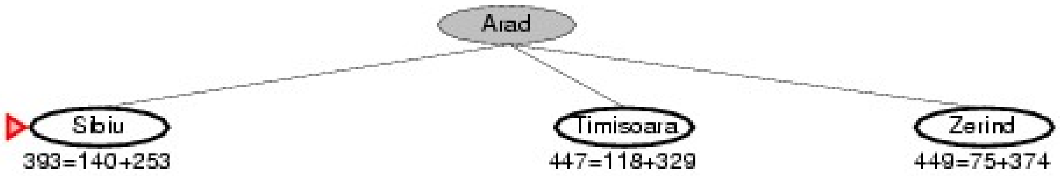
A* search example

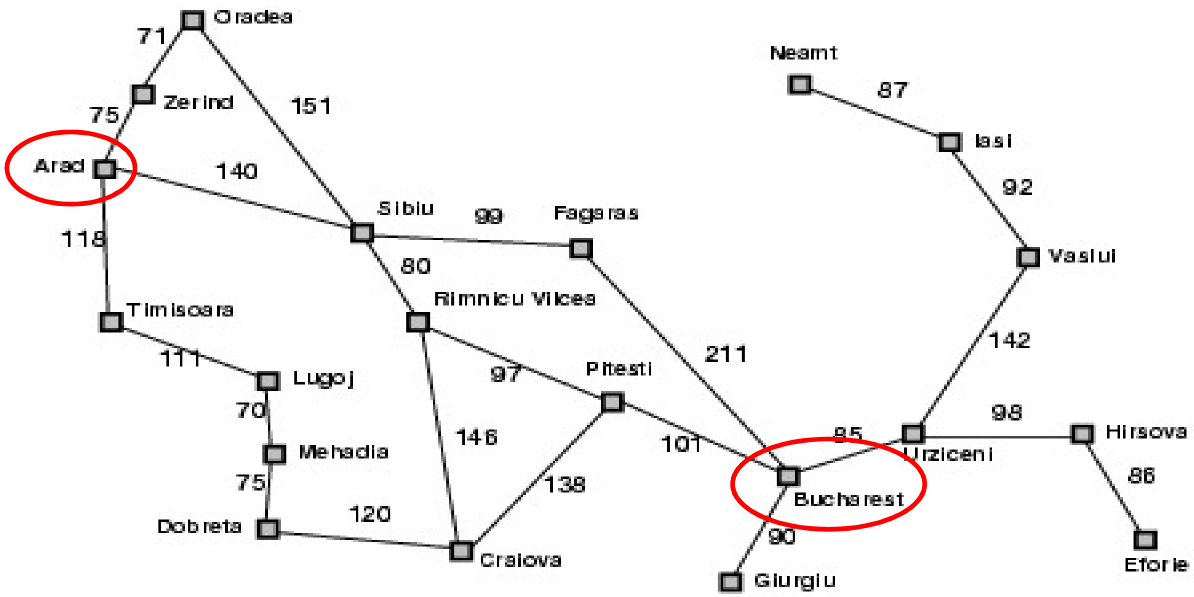


Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

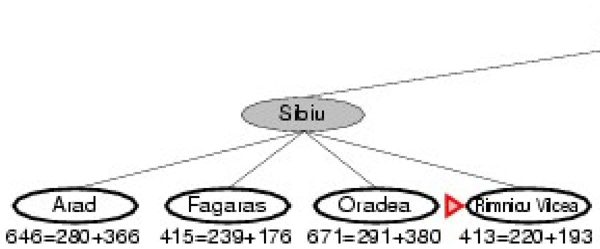


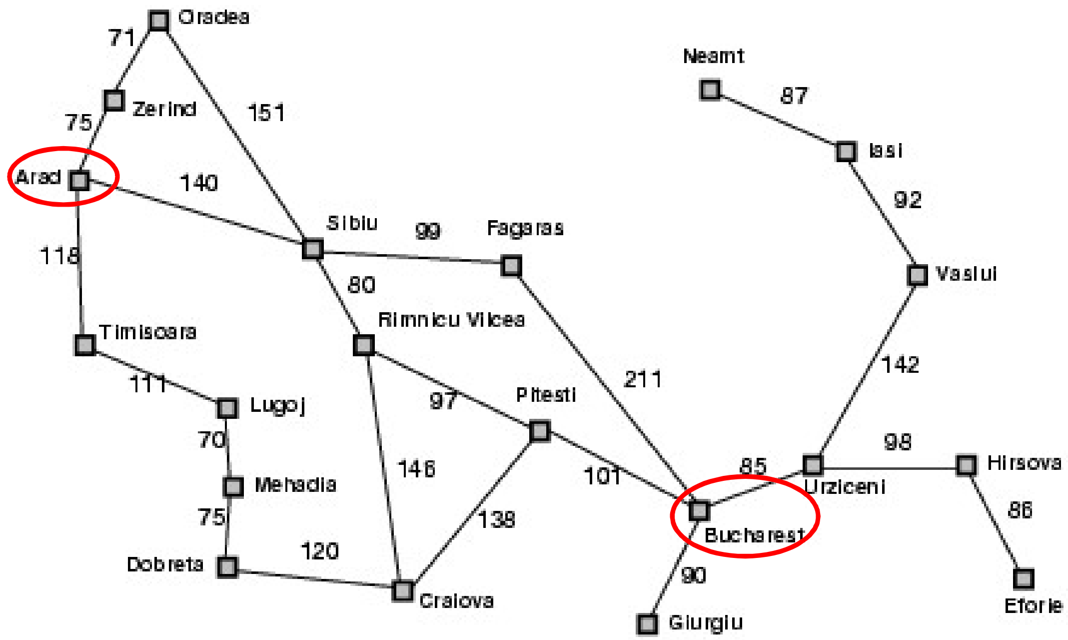


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

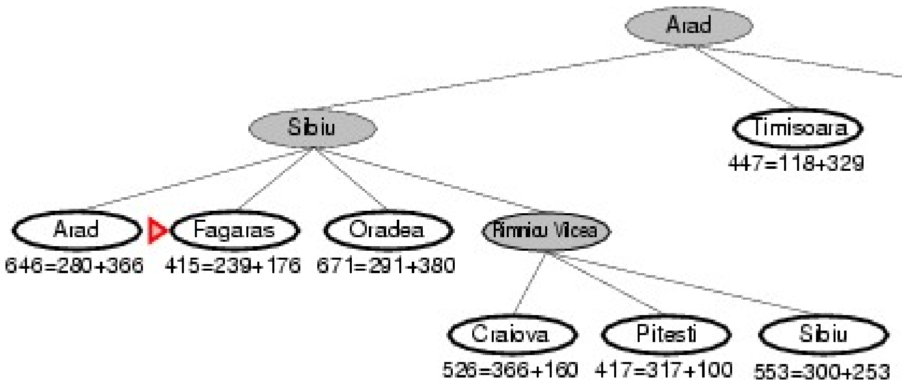


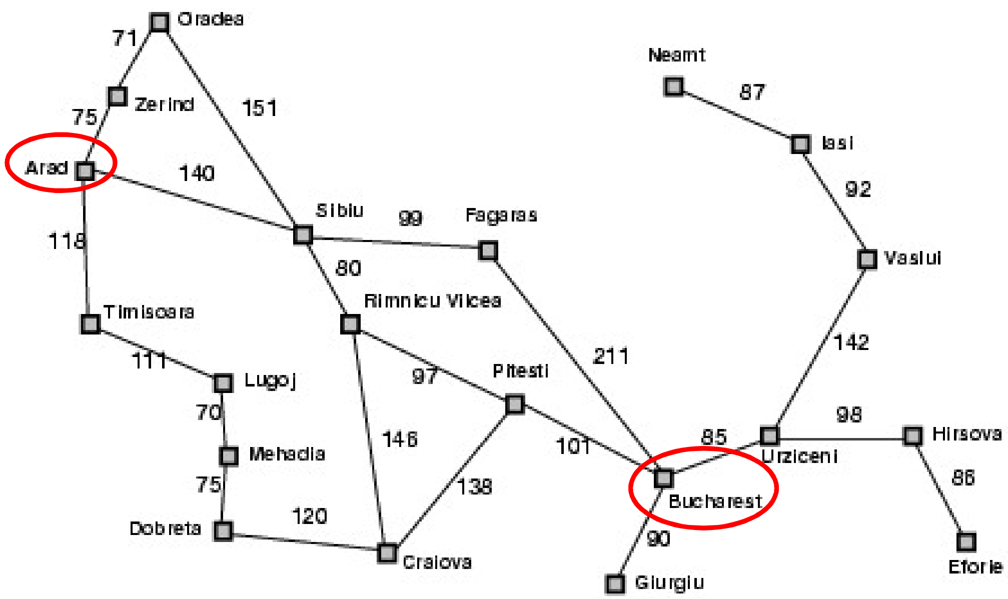


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

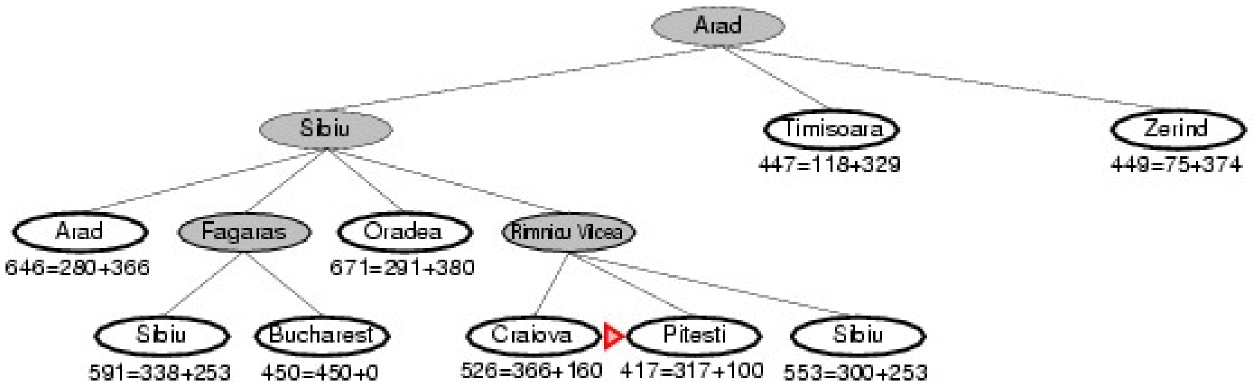


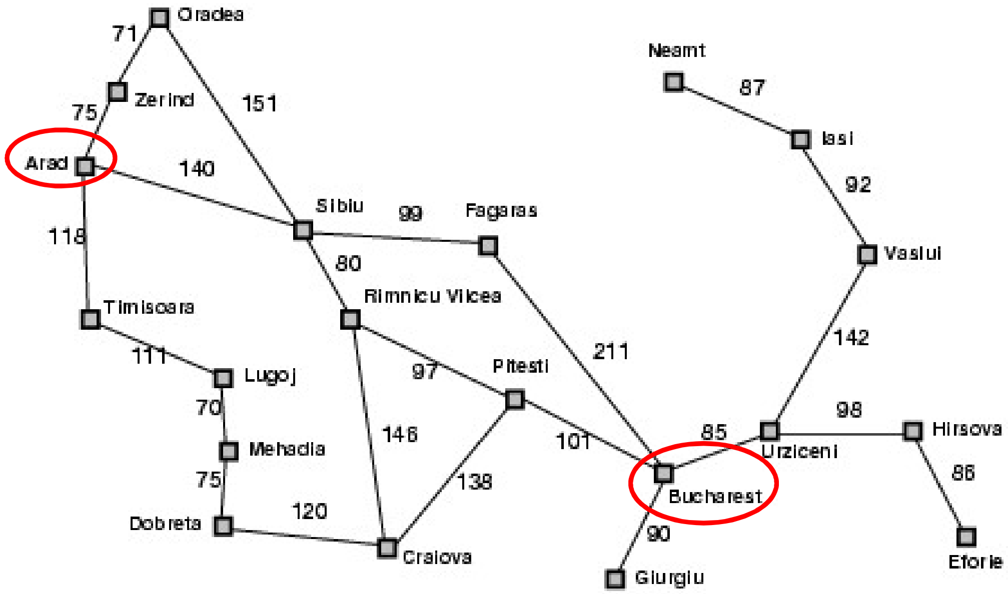


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

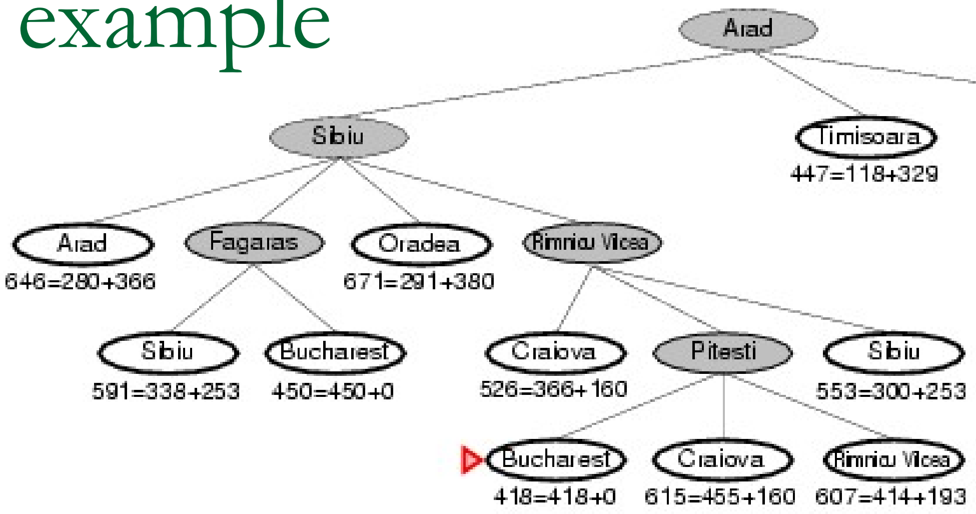


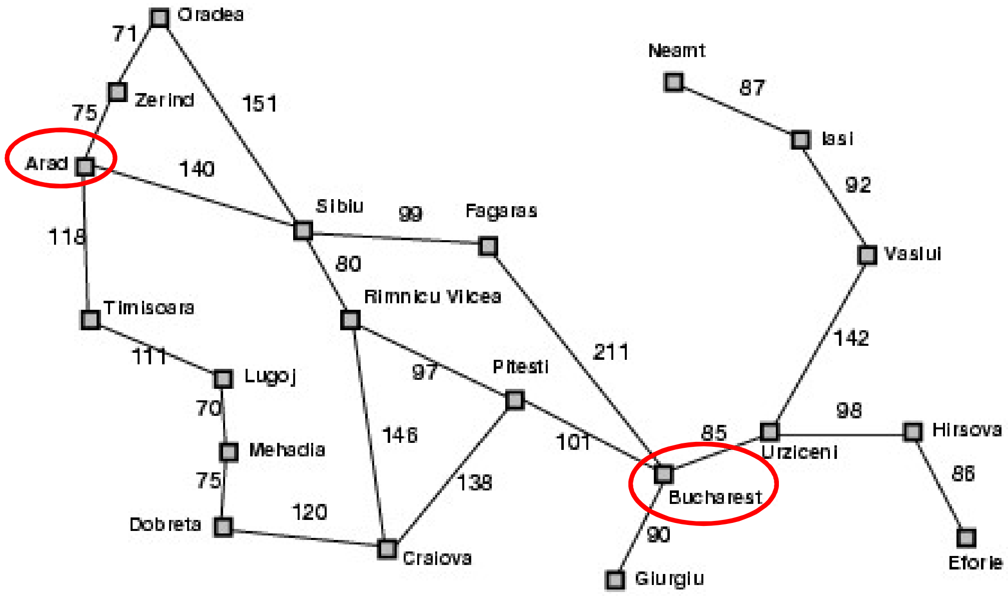


Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

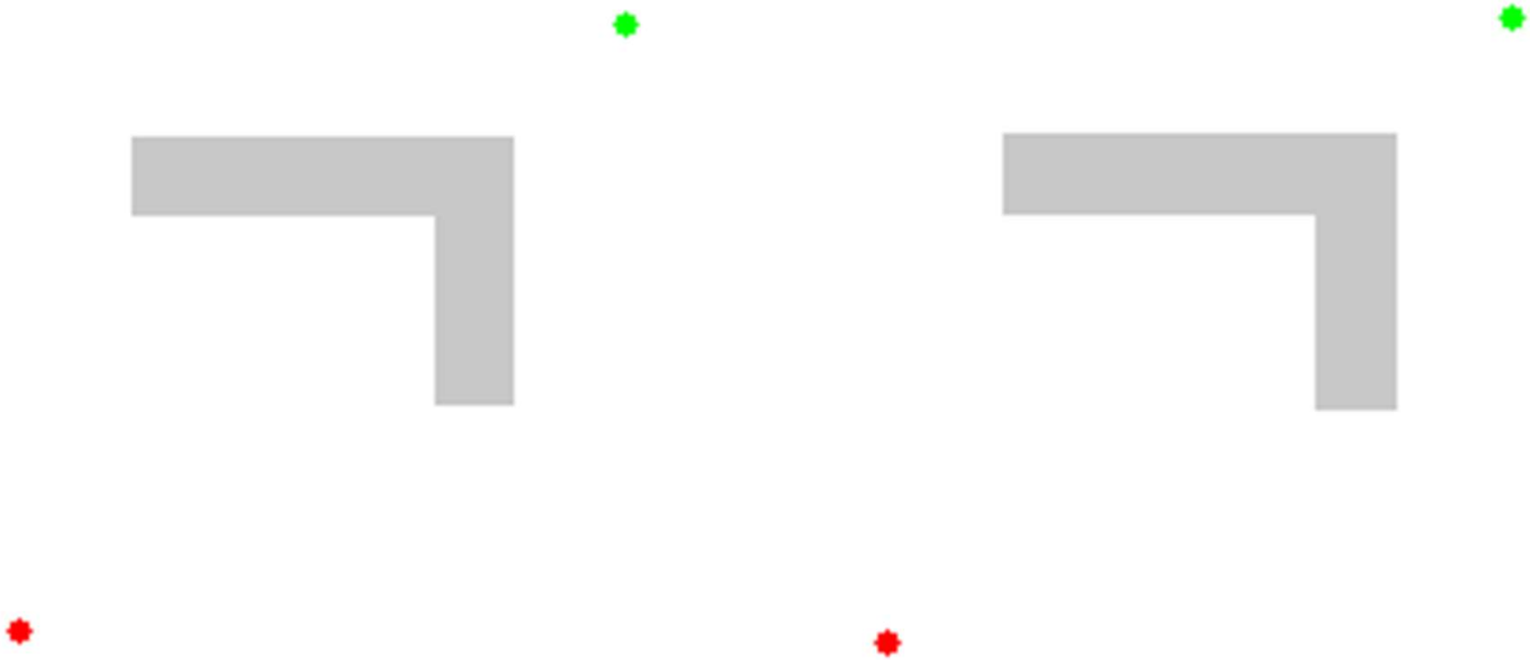




Straight–line distance
to Bucharest

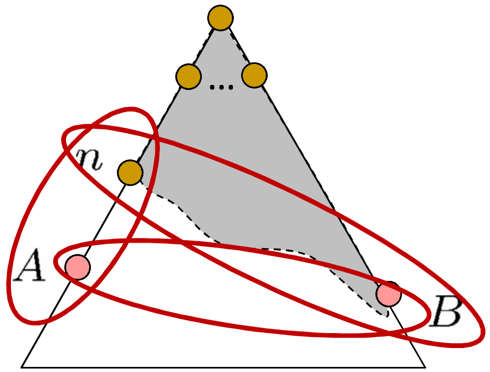
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Uniform cost search vs. A* search



Source: Wikipedia

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

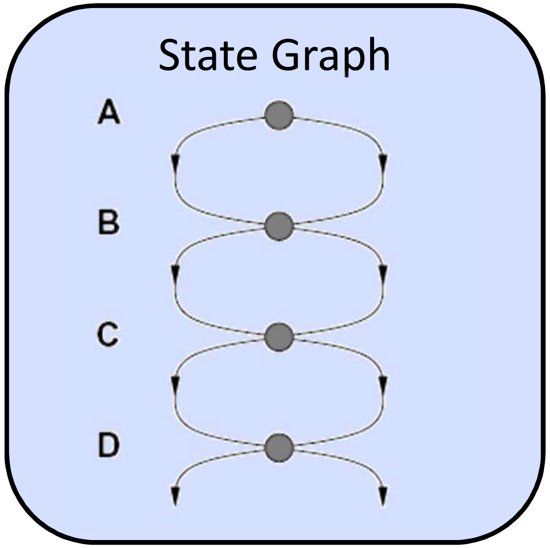
$f(n) = g(n) + h(n)$	Definition of f-cost
$f(n) \leq g(A)$	Admissibility of h
$g(A) = f(A)$	h = 0 at a goal

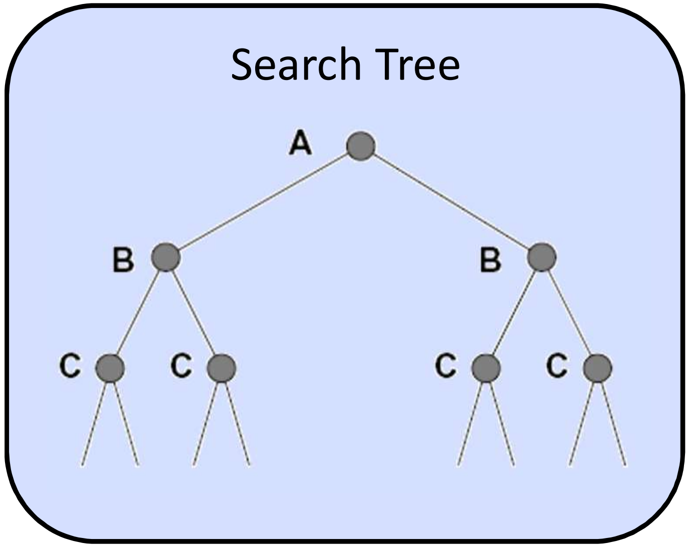
$g(A) < g(B)$	B is suboptimal
$f(A) < f(B)$	h = 0 at a goal

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

Tree-like Search: Extra Work!

- Failure to detect repeated states can cause exponentially more work.



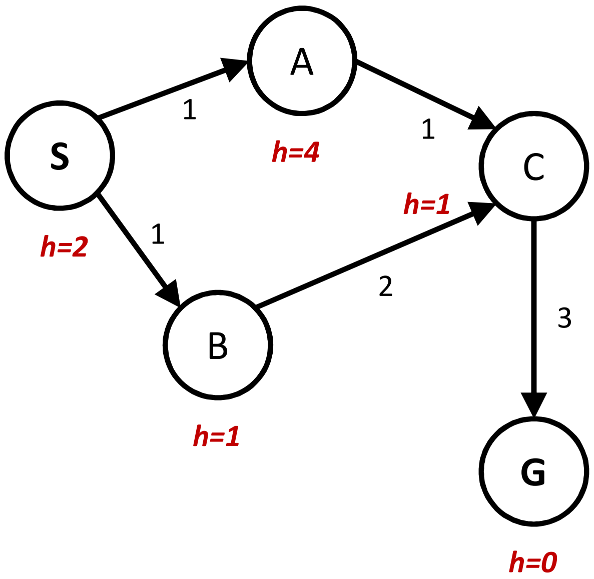


Graph Search

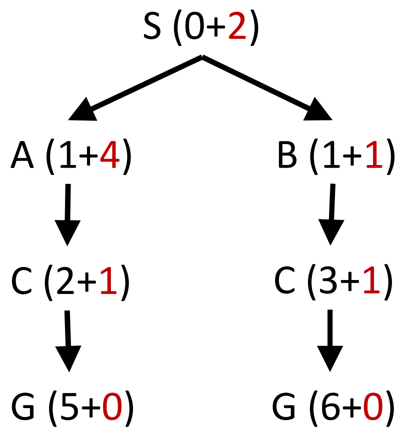
- Idea: never expand a state twice
- How to implement:
  - Tree search + set of expanded states ("closed set")
  - Expand the search tree node-by-node, but…
  - Before expanding a node, check to make sure its state has never been expanded before
- How about optimality?

A* Graph Search Gone Wrong?

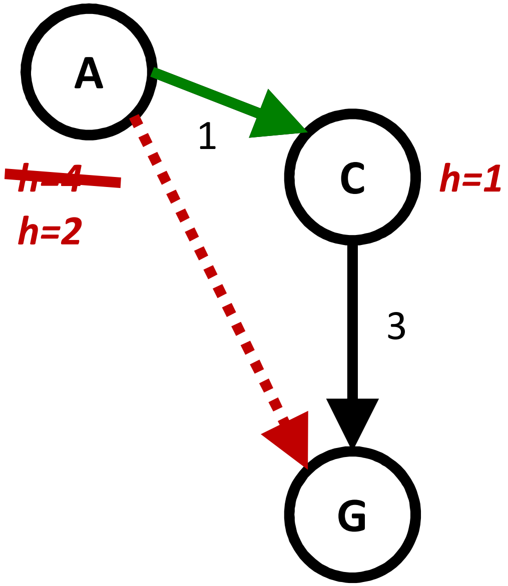
State space graph



Search tree



Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G
  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)
- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)
  - A* graph search is optimal

Optimality of A*

- **Tree-like search** (i.e., search without repeated state detection):
  - A* is optimal if heuristic is ***admissible*** (and non-negative)
- **Graph search** (i.e., search with repeated state detection)
  - A* optimal if heuristic is ***consistent***
- Consistency implies admissibility
  - In general, most natural admissible heuristics tend to be consistent, especially if they come from relaxed problems.

Source: Berkeley CS188x

Properties of A*

- **Complete?**

  Yes – unless there are infinitely many nodes with $f(n) \leq C^*$

- **Optimal?**

  Yes

- **Time?**

  Number of nodes for which $f(n) \leq C^*$ (exponential)

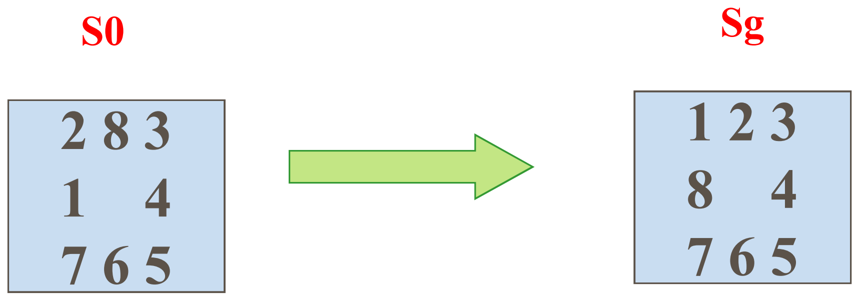
- **Space?**

  Exponential

Example: 8-puzzle

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

  (i.e., no. of squares from desired location of each tile)

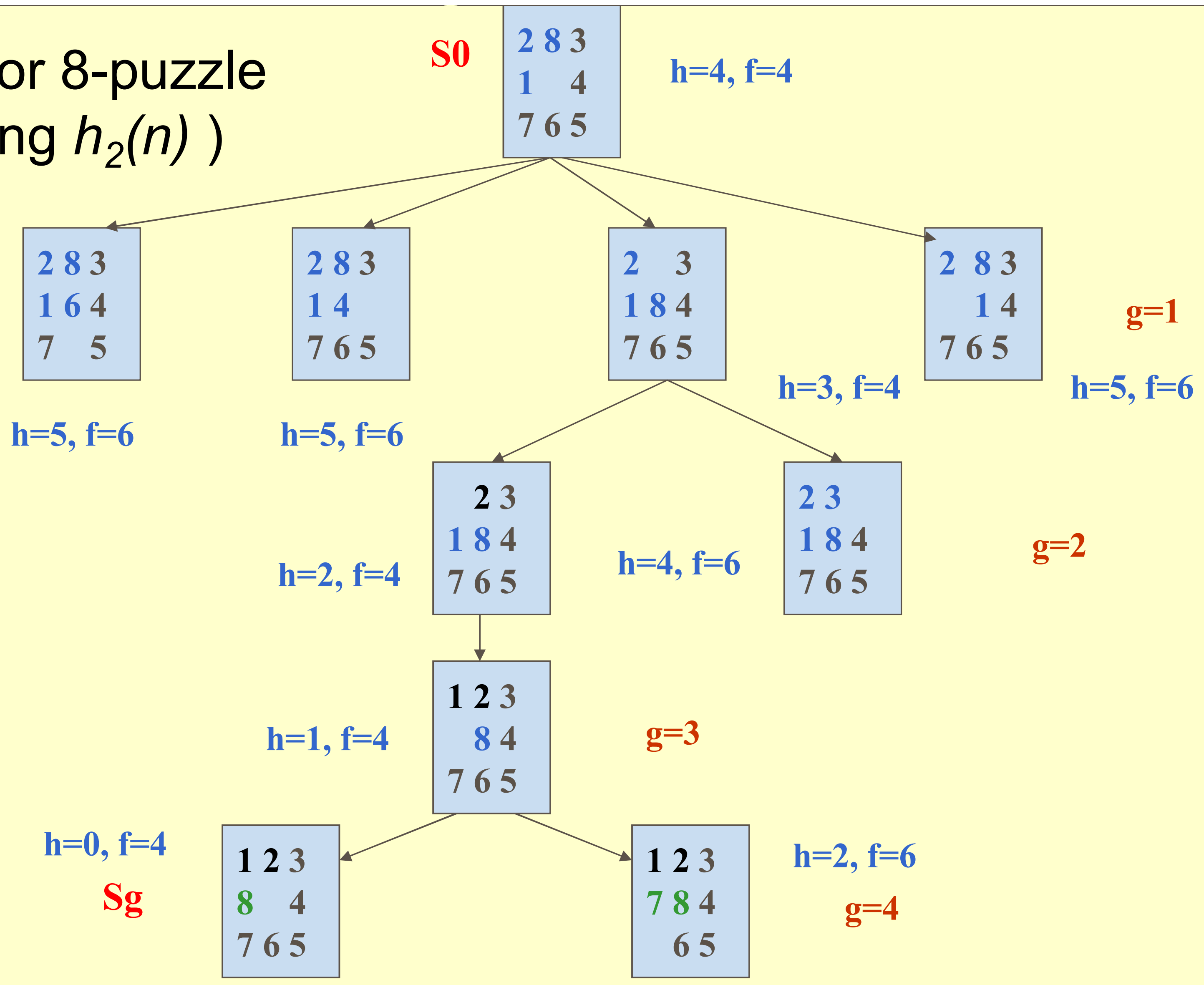


- $h_1(S) = ?$ 3
- $h_2(S) = ?$ 1+1+2 = 4
- If $h_2(n) \geq h_1(n)$ for all n (both admissible), then h_2 dominates h_1 , *and* h_2 is better for search

Dominance

- If h_1 and h_2 are both admissible heuristics and $h_2(n) \geq h_1(n)$ for all n, then h_2 dominates h_1
- Which one is better for search?
  - A* search expands every node with $f(n) < C^*$ or $h(n) < C^* - g(n)$
  - Therefore, A* search with h_1 will expand more nodes

A* for 8-puzzle (using $h_2(n)$)



Heuristics from relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem.
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution.
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution.

Combining heuristics

- Suppose we have a collection of admissible heuristics $h_1(n)$, $h_2(n)$, …, $h_m(n)$, but none of them dominates the others
- How can we combine them?

$$h(n) = \max\{h_1(n), h_2(n), \ldots, h_m(n)\}$$

Weighted A* search

- **Idea:** speed up search at the expense of optimality
- Take an admissible heuristic, “inflate” it by a multiple $\alpha > 1$, and then perform A* search as usual.
- Fewer nodes tend to get expanded, but the resulting solution may be suboptimal (its cost will be at most α times the cost of the optimal solution).

Example of weighted A* search

Heuristic: 5 * Euclidean distance from goal

Source: Wikipedia

Compare: Exact A*

All search strategies (different fringe strategies)

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes (if heuristic is admissible)	Number of nodes with $g(n)+h(n) \leq C^*$	

A note on the complexity of search

- We said that the worst-case complexity of search is exponential in the length of the solution path
  - But the length of the solution path can be exponential in the number of “objects” in the problem!
- Example: towers of Hanoi

Q & A