


Searching

UI. II

Markošová Mária

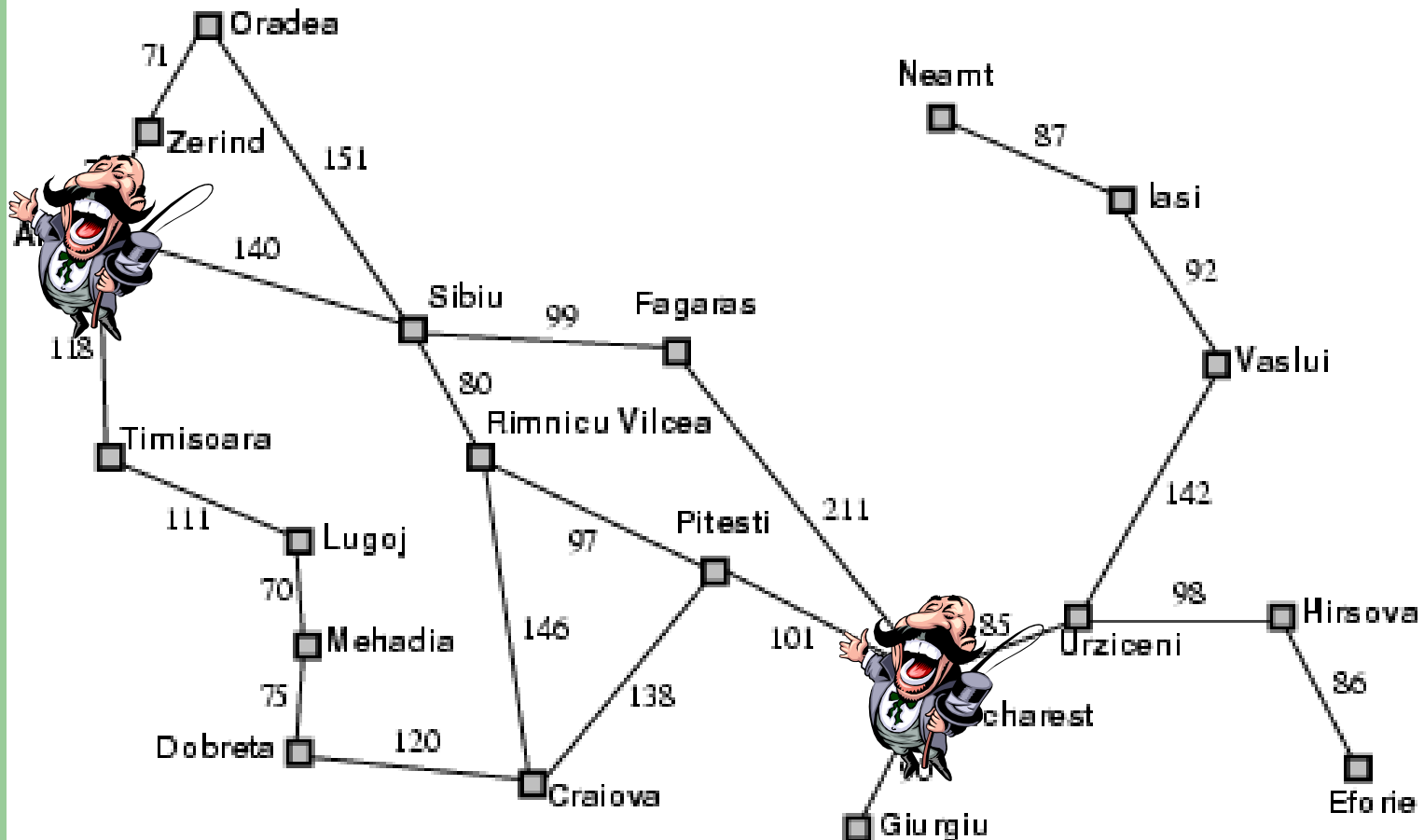
Summary

1. Agent, percept, agent function, agent program.
 2. Typology of environments.
 3. PEAS description of the agent.
 4. Types of simple agents.
 5. Model examples.
 6. Intro to searching, searching agent.
- 
1. Simple reflex agent
 2. Model based agent
 3. Goal based agent.
 4. Utility based agent

Outline of this lecture:

1. Searching tree, cost function
2. Uninformed search (BFS, DFS, uniform cost search)
3. Informed search (greedy, A^*).
4. Graph search a tree search implementation of searching algorithms

© 2015 Pearson Education, Inc. or its affiliate(s). All rights reserved.



Agent

1. Seeks the route from Arad to Bucharest in a given state space.
2. Percepts his position (town), and is able to move to the another positions (towns).
3. Can plan the route with a help of the map.

Environment ? Try to characterize the task environment

Static, dynamic?

Static.

Episodic, unepisodic?

Unepisodic.

Deterministic , undeterministic?

Deterministic.

Singleagent, multiagent?

Single agent.

Discrete, continuum?


Discrete.

Observable, unobservable?

We do not know, depends whether the agent sees the whole map or not

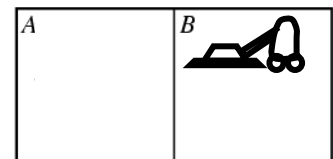
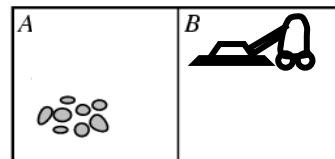
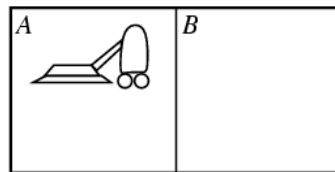
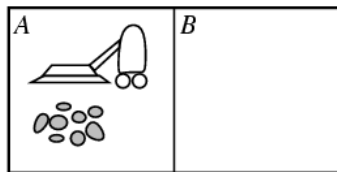
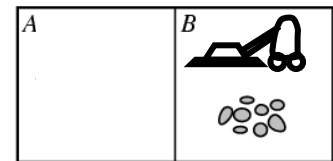
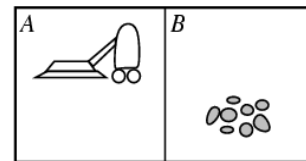
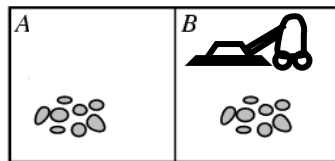
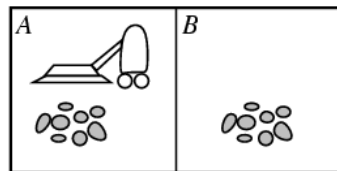
Searching algorithm:

Criteria:

1. We ask whether the algorithm finds a solution (**completeness**)
2. Path cost 
 - low (**admissible**)
 - the lowest possible (**optimal**)
3. Cost of the searching (**time and memory complexity**)

Vacuum cleaner problem as a searching problem

What are states? How many $n \cdot 2^n$ n – number of rooms
states we have for the general case (n rooms)?

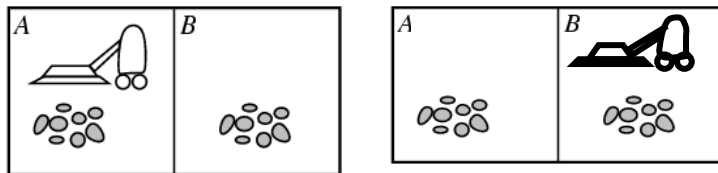


Initial state?

Whatever.

State

In AI state is a configuration of the environment and the agent in the environment.



Two different states

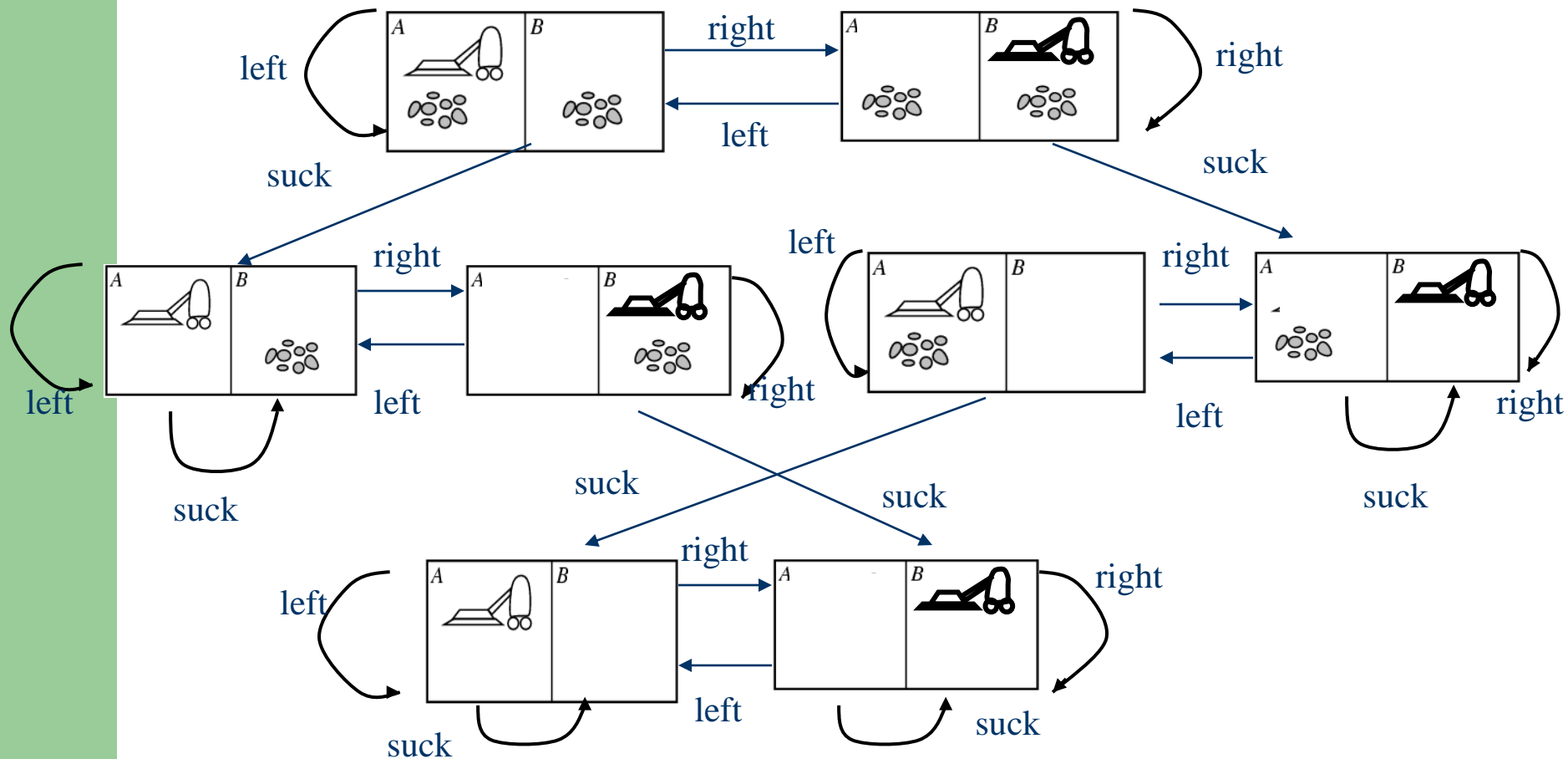
Successor function, operators ?: We generate all legal states trying all possible actions in the current state.



Goal? Both rooms clean.

Path cost? For example let each step costs 1, path cost is thus number of steps between the initial and the goal state.

State space?



Defintion of state:

In AI state is one configuration of the environment and agent in the environment.

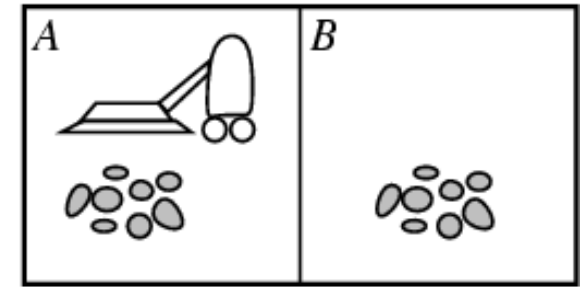
State space:

State space is created by all possible configurations of the environment.

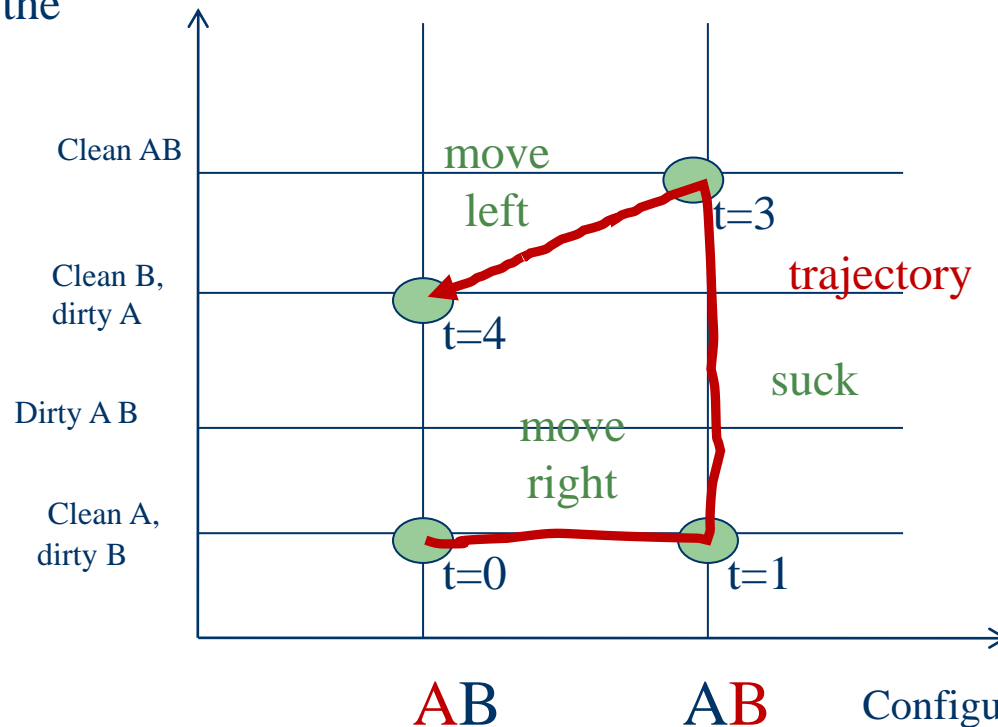
Another environment configuration is reached with a help of an agent **action**.

State (phase) space in physics

Automatic vacuum cleaner



State of the
room



Discrete state
space

Configuration
(agent is in the
red room)

State space representation in AI

State space is represented by the **oriented graph**:

- state : graph node
- action: edge of the oriented graph, which leads to another state (node)
- **successor function** – function generating successors (states reachable from certain state)
- goal state: one or more exclusive nodes in the graph

Searching the state space

We generate the searching tree: We need

1. Initial state
2. Successor function
3. Strategy for ordering successors in order to apply successor function again.

Node representation

1. Node identifier.
2. Identifiers of the successors.
3. Parent node identifier.
4. Depth of the node $d(n)$.
5. Path cost to the node $g(n)$.
6. Number of branches on the node $b(n)$.
7. Information about the action by which the node was reached.

Fringe

Fringe – nodes waiting for development.

Fringe types: **FIFO** first in first out

LIFO last in first out

fringe defined by the path cost, **BEST FIRST**

Strategy

Strategy – means the manner how the nodes are ordered .

Blind, uninformed strategy.

Informed strategy with heuristics

Difference between the state space and the searching tree (Romania example)

State space: Has 20 possible states (towns).

Searching tree: Has infinite number of nodes, thanks to the branches Arad-Sibiu, Arad – Sibiu – Arad, Arad – Sibiu – Arad – Sibiu, apod.

Difference between the node and the state

Node : data structure which helps to represent the tree

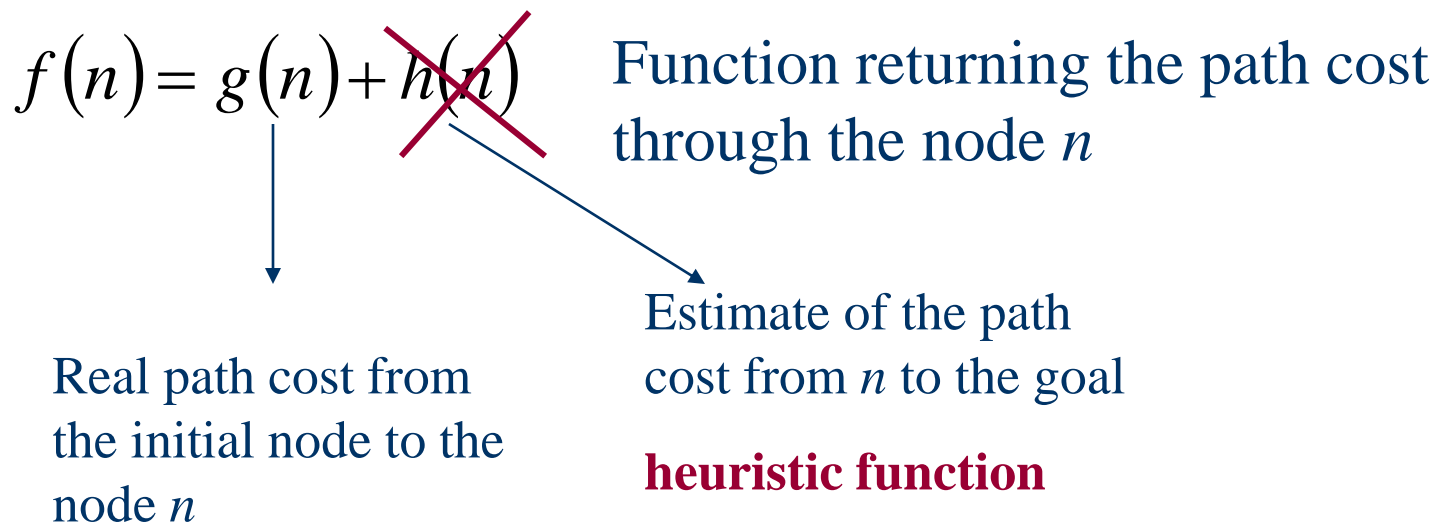
State: represents the environment configuration

Two different tree nodes can represent the same state. They can be a part of different branches.

Uninformed search (blind search)

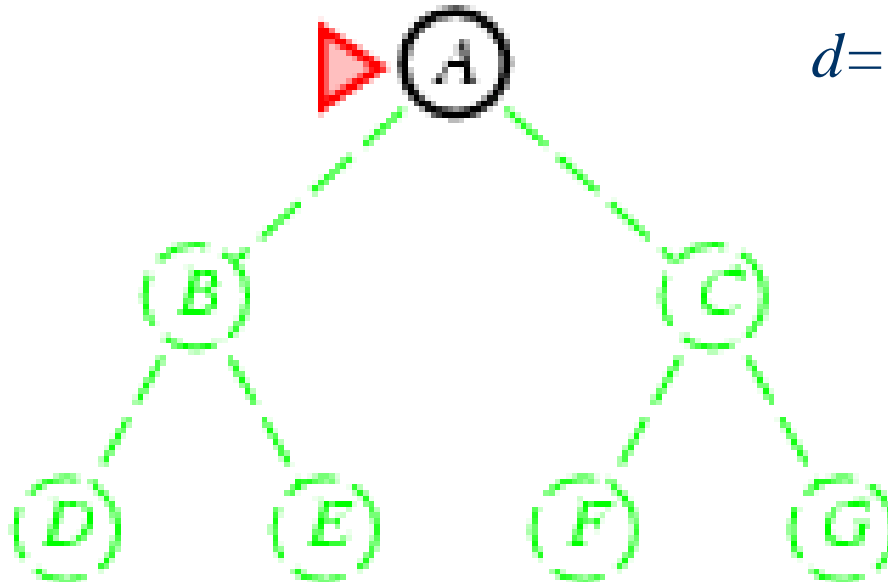
Uninformed or blind search:

We have only the problem defined, no additional information or estimates, no heuristics.



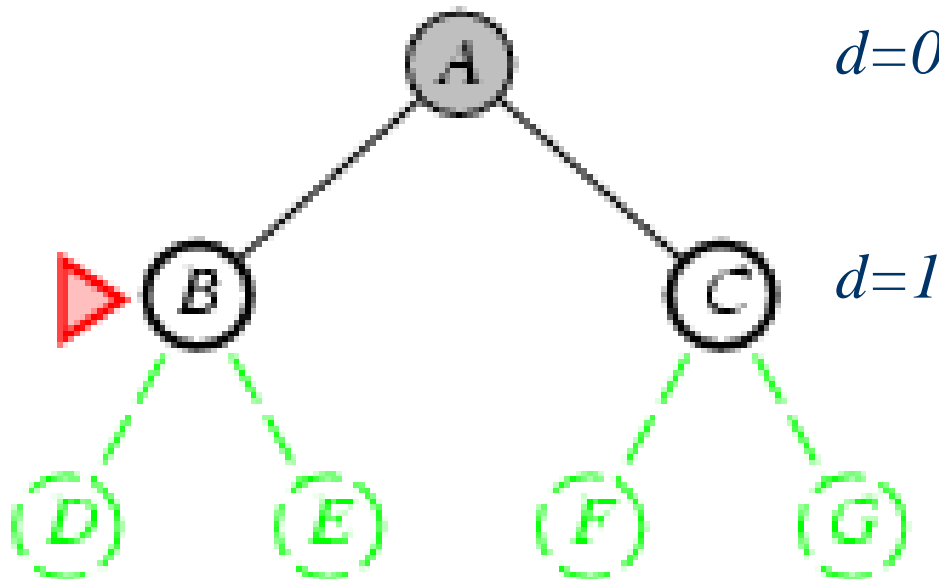
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
 - *fringe* is a FIFO (first in first out) queue, i.e., new successors go at end



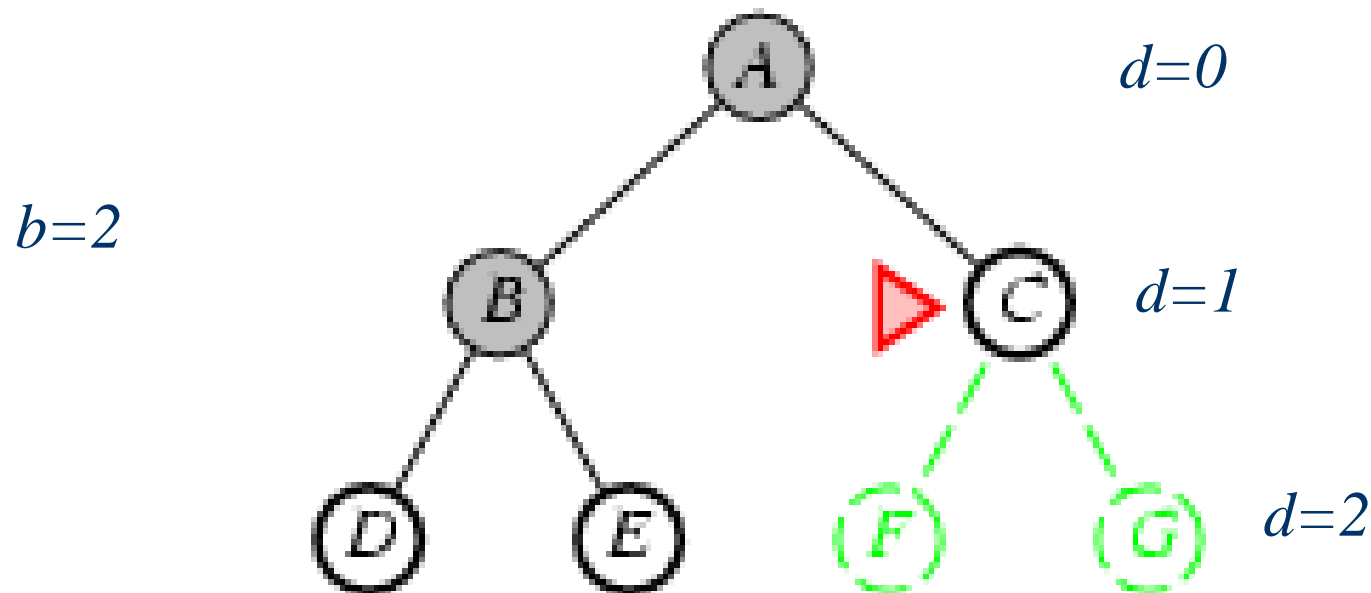
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
 - *fringe* is a FIFO (first in first out) queue, i.e., new successors go at end



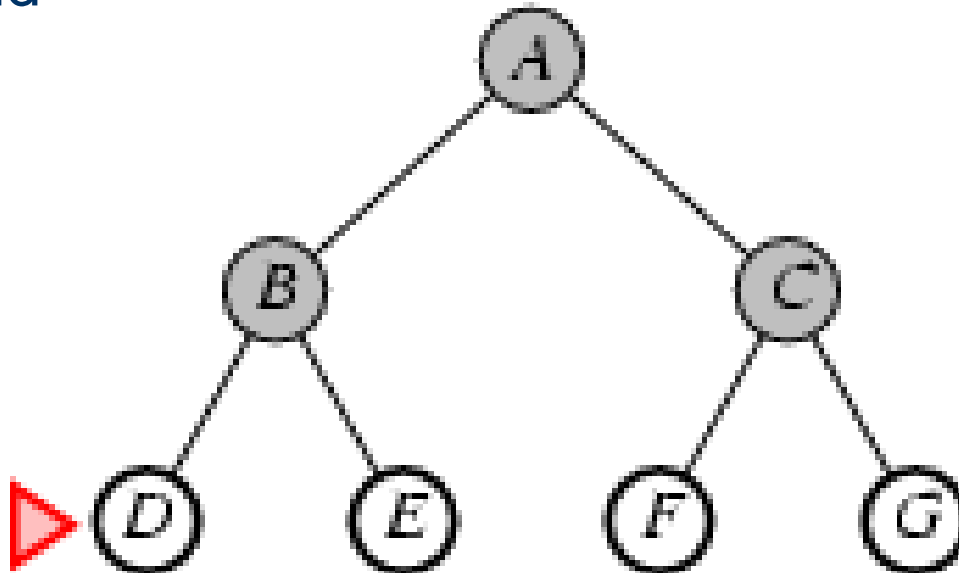
Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
 - *fringe* is a FIFO (first in first out) queue, i.e., new successors go at end

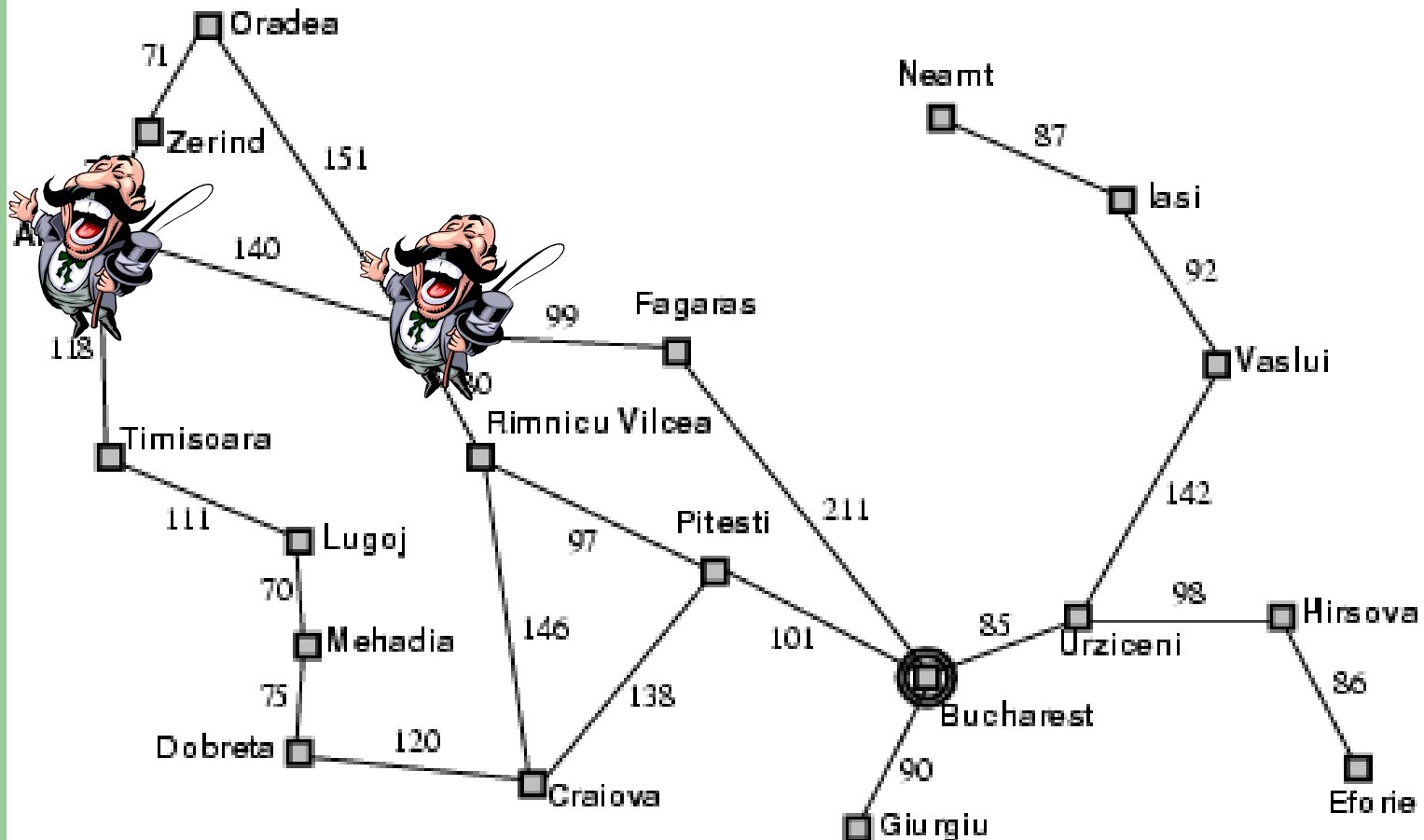


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
 - *fringe* is a FIFO queue, i.e., new successors go at end



Paradigmatic example: agent searching for the route from Arad to Bucharest



Breadth first search

Is this strategy complete ?

Yes, the solution is always found

Is this strategy optimal ?

For the constant step cost it is an optimal strategy.

Time and space (memory) complexity: exponential $O(b^d)$, b – branching factor, d depth on the tree.

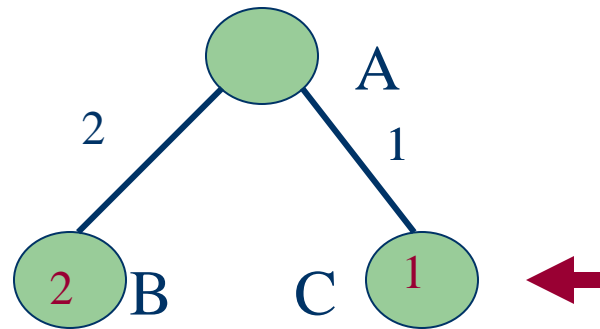
Uniform cost search

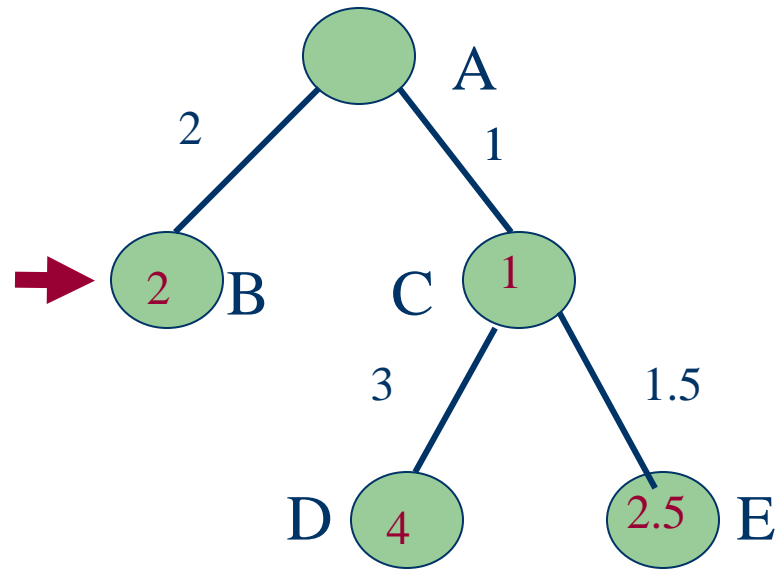
Fringe nodes are ordered according the path cost.

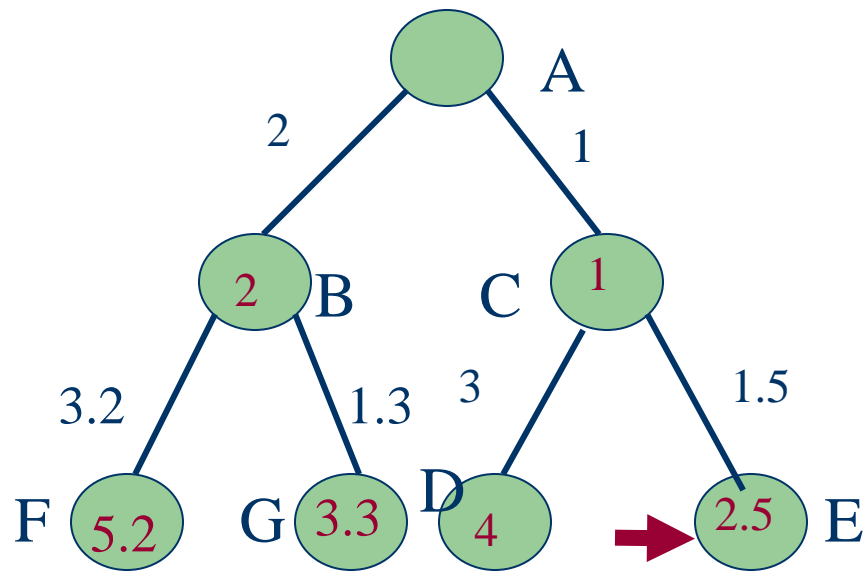
Always the node on the cheapest path is developed. If $g(u)=d(u)$, or the path cost is constant, this strategy turns to the breadth first search

path cost





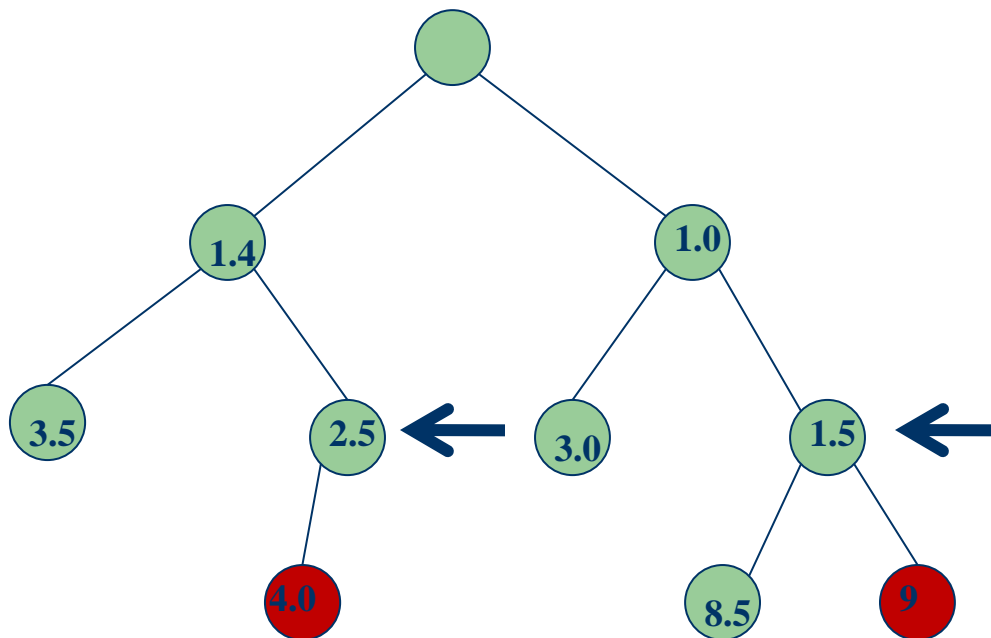




Uniform-cost search

- First the cheapest node is developed.
- **Implementation:**
 - Fringe – nodes are ordered according the path cost
- The same as breadth first for the constant step cost
- Complete? Yes, if the step cost is $\geq \epsilon$
- Time complexity? $O(b^{\text{ceiling}(C^*/\epsilon)})$ Where C^* is the cost of the optimal solution
- Space (memory) complexity? $O(b^{\text{ceiling}(C^*/\epsilon)})$
- Optimal? Yes, nodes are developed according growing $g(n)$

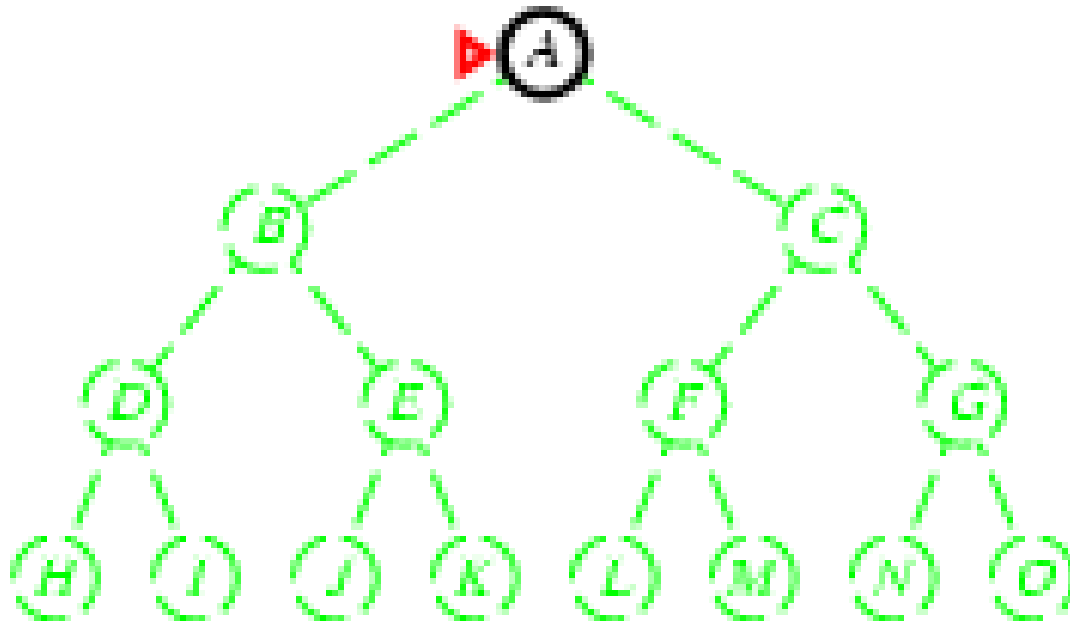
Question: What do you think, appears the best solution necessarily as the first one in the fringe ?



Question: Is the best solution chosen (and thus given for the goal test) as the first one?

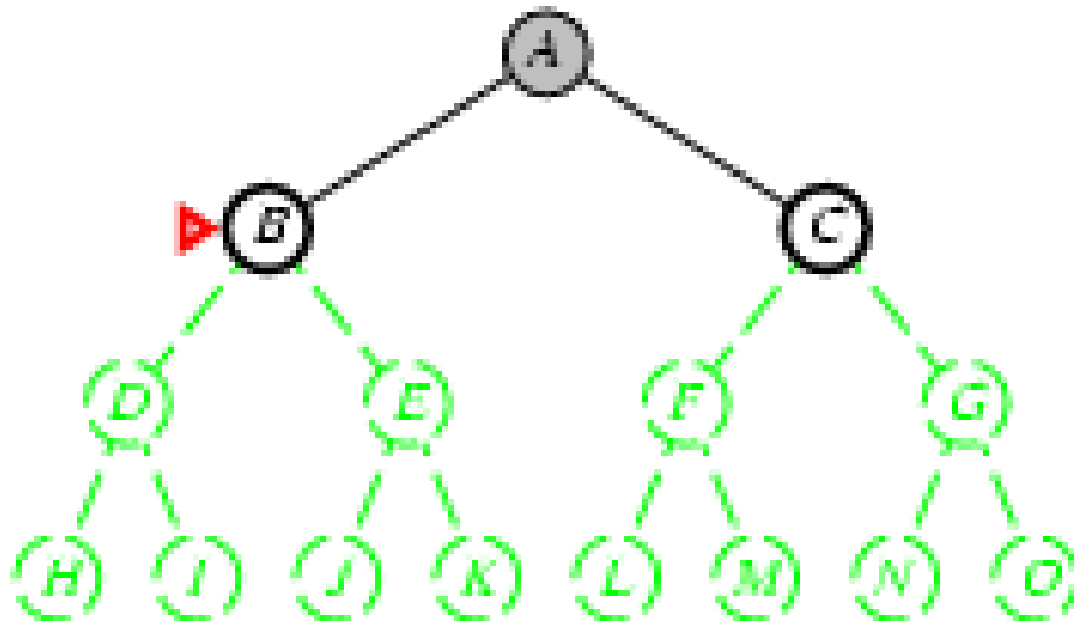
Depth-first search (hľadanie do hĺbky)

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO (last in first out) queue, i.e., put successors at front



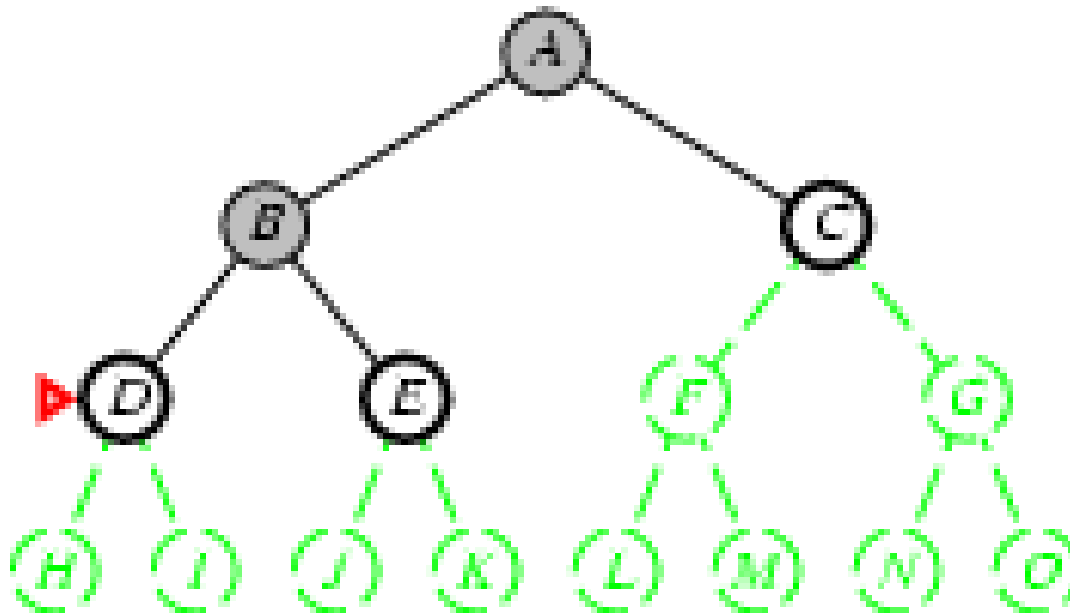
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front



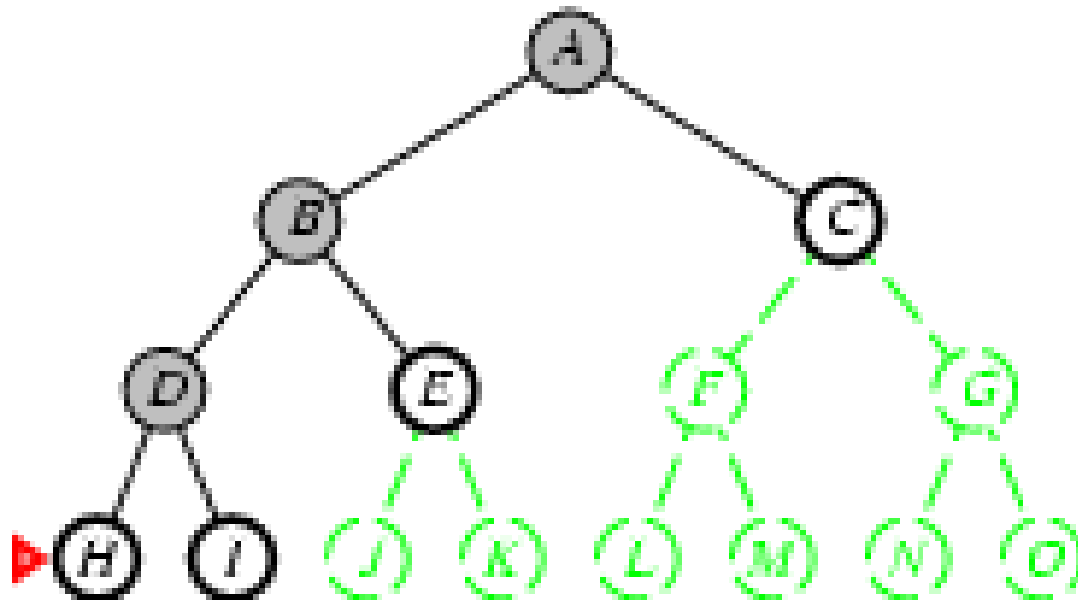
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front



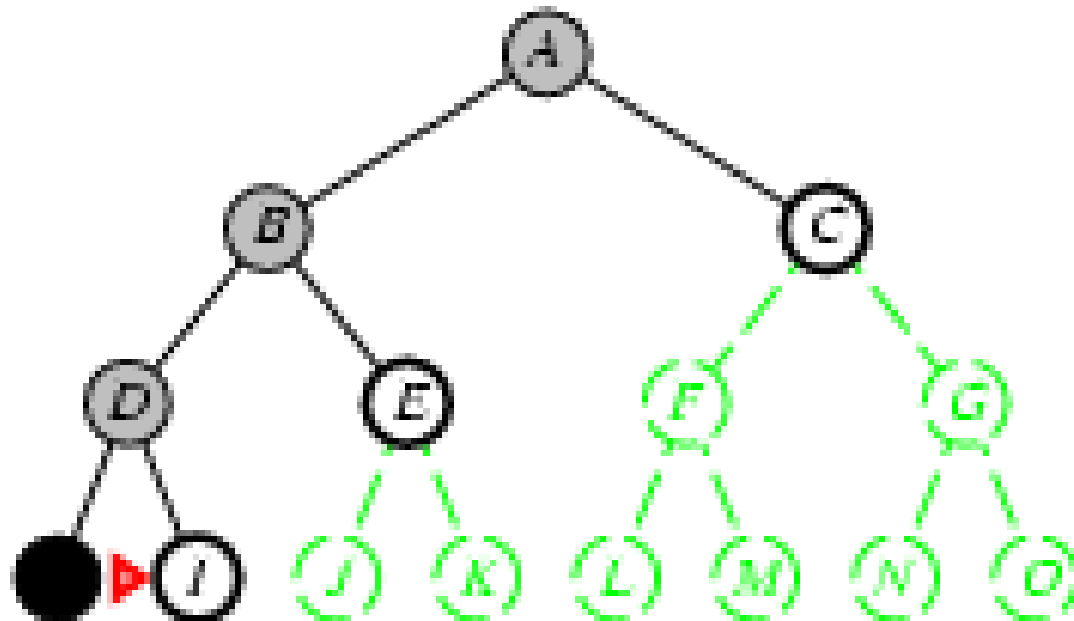
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue (last in first out), i.e., put successors at front



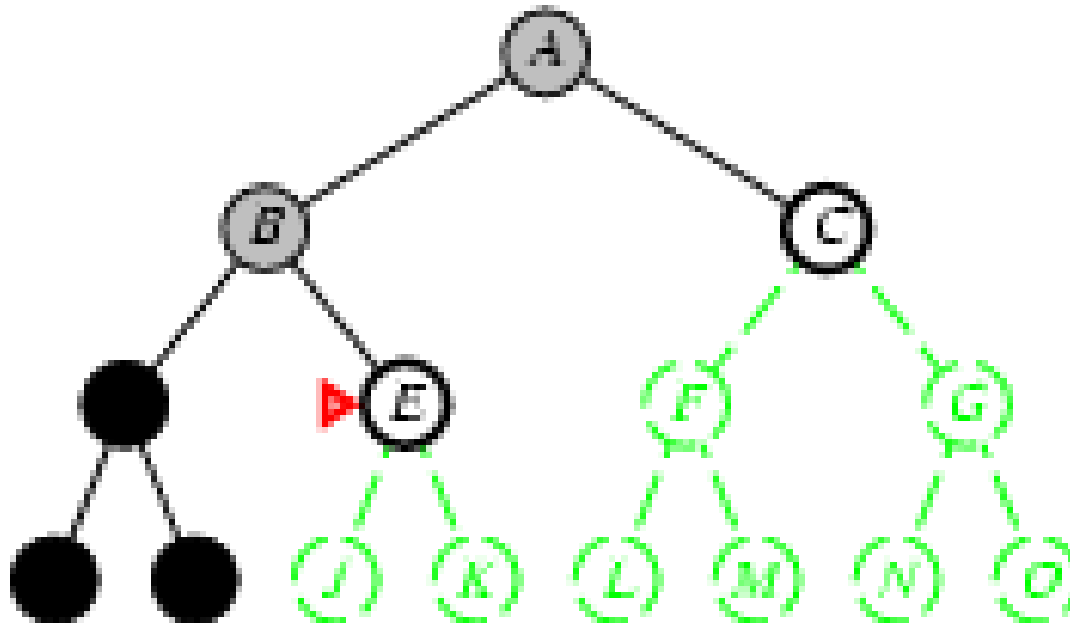
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



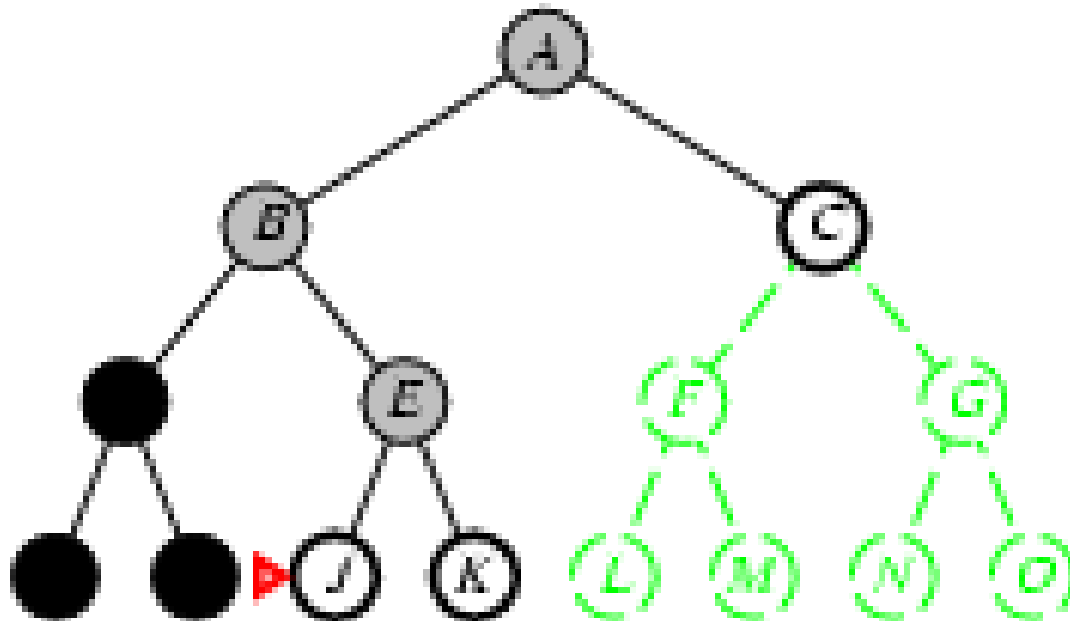
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



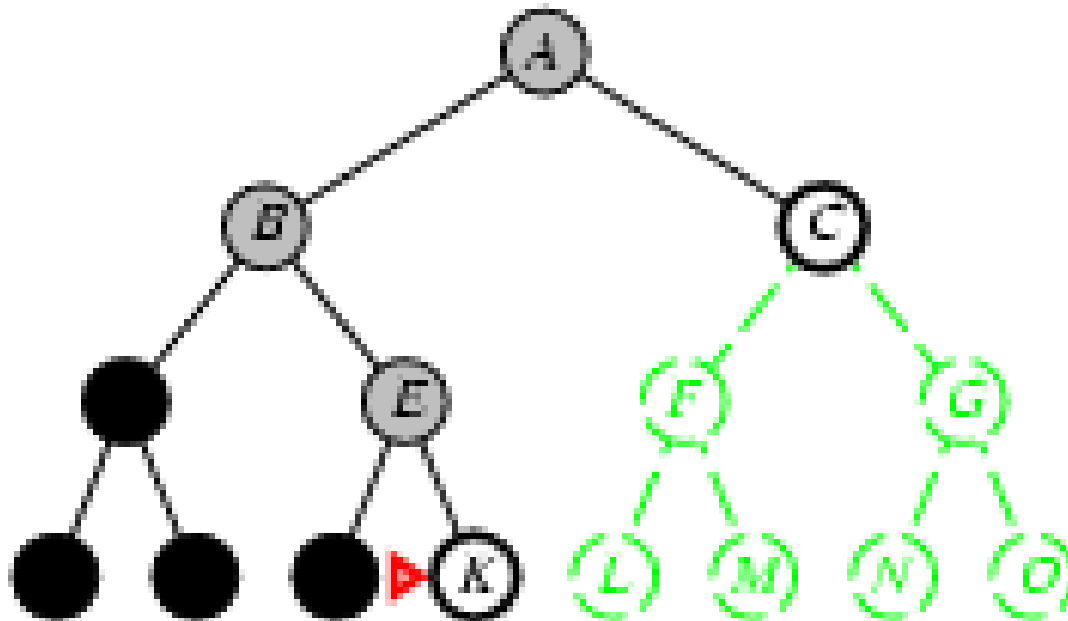
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



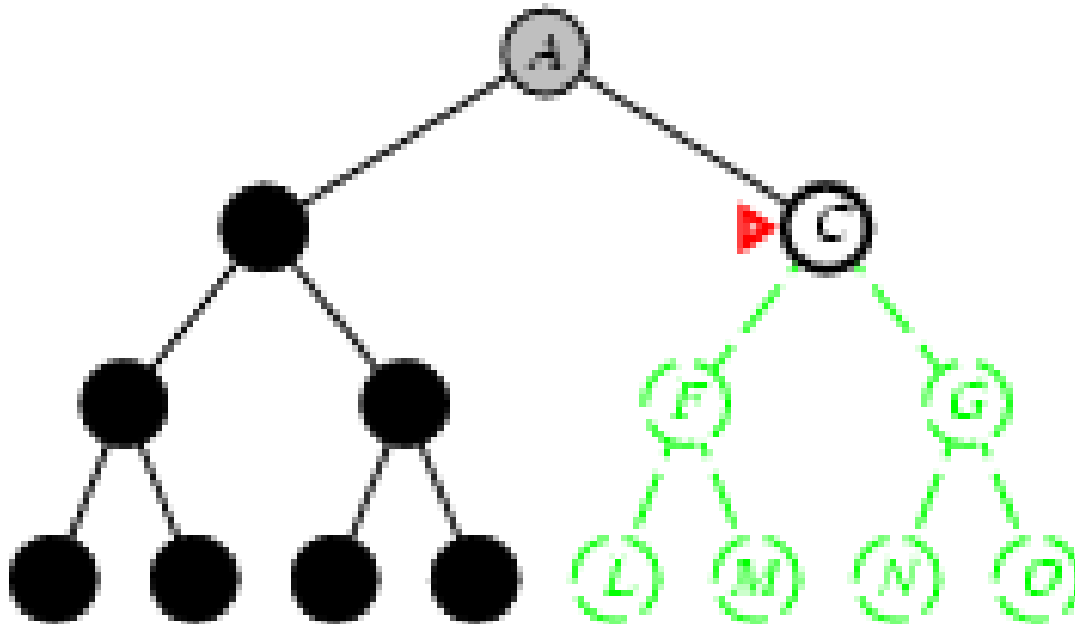
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



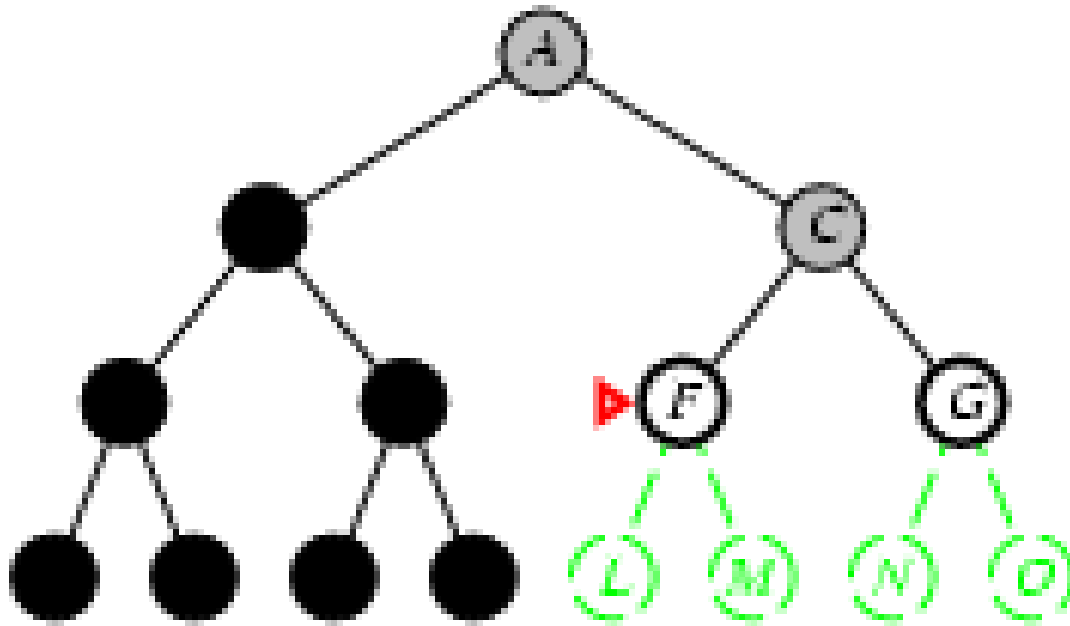
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



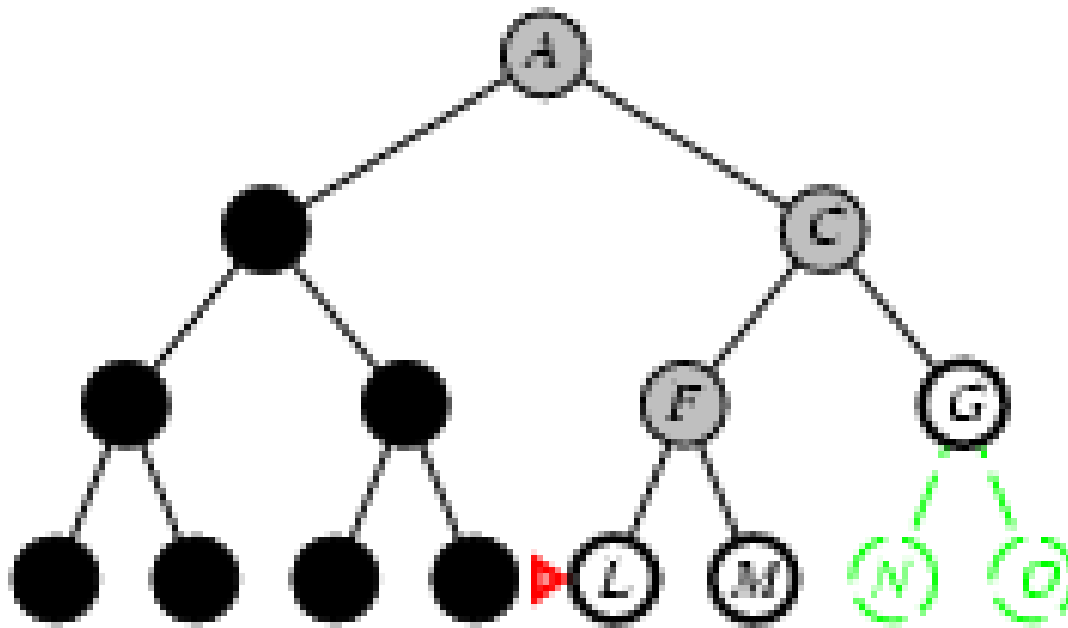
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



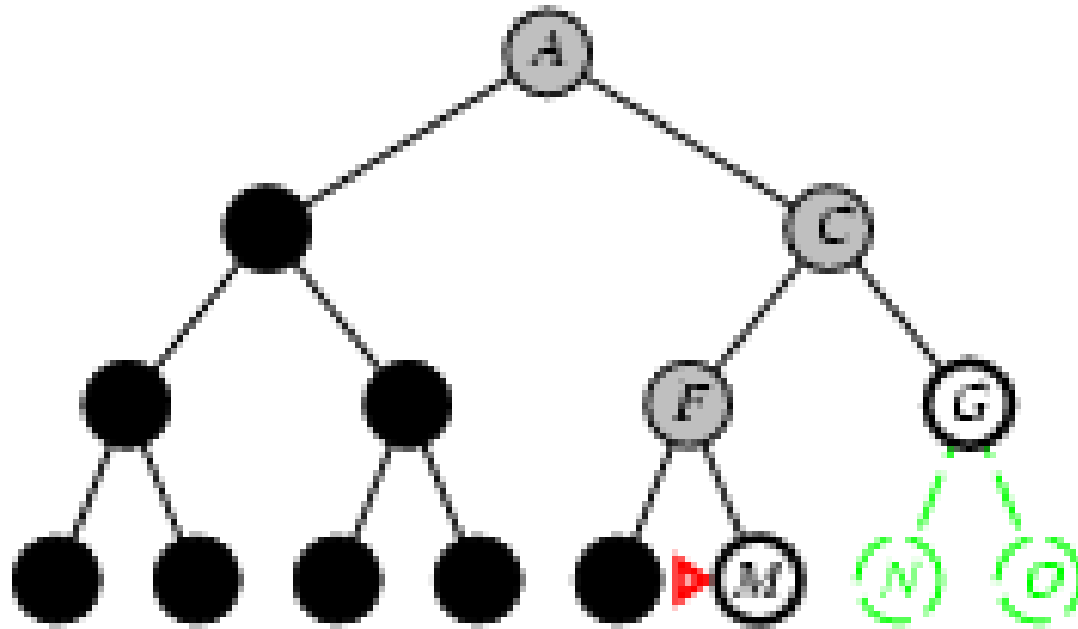
Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



Depth-first search

- Expand deepest unexpanded node
- Implementation:
 - *fringe* = LIFO queue, i.e., put successors at front
 -



Depth – first search

Is the strategy complete? No, can get stuck in an infinitely deep trees.

Is the strategy optimal? No

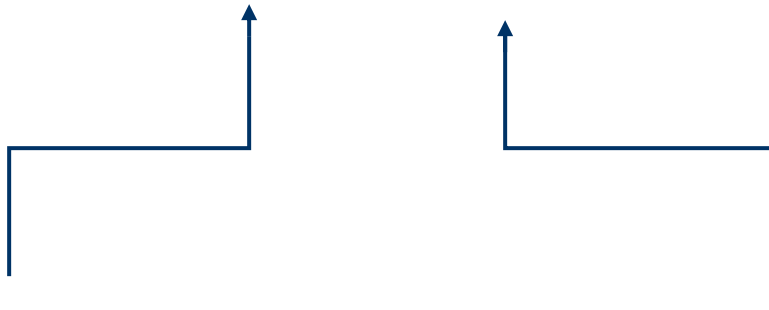
Has at least some advantages?

Yes: Linear memory complexity $O(bm)$, b – branching factor, m – maximal depth

Time complexity: exponential $O(b^m)$, but more efficient then the breadth first

Best first strategy in an informed search

$$f(n) = g(n) + h(n) \quad \longleftarrow \quad \text{Path cost through the node } n$$



Real path cost from the initial node to the node n .

Heuristic estimate of the path cost from n to the goal

Best first search makes fringe according growing $f(n)$.

Informed search

1. Best first search strategy is used.
2. Function $f(n)$ has the $h(n)$ part.
3. Fringe, called OPEN list too, is according growing f

Informed strategies

$$f(n) = \cancel{g(n)} + h(n)$$

A* Greedy best first search
algorithm

Greedy best first search

Only heuristic estimate $h(n)$ is used.

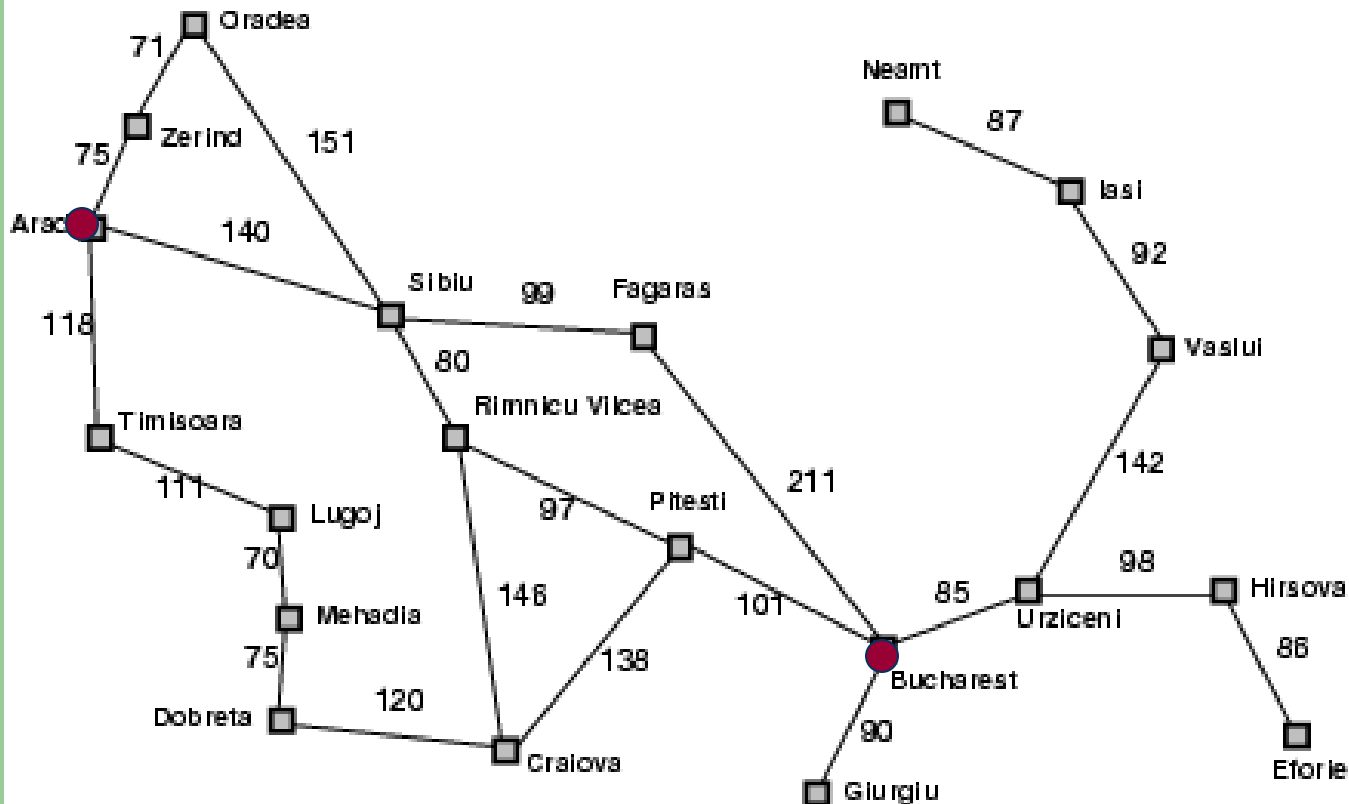
$$f(n) = h(n)$$

$h(c)=0$, c is a goal node

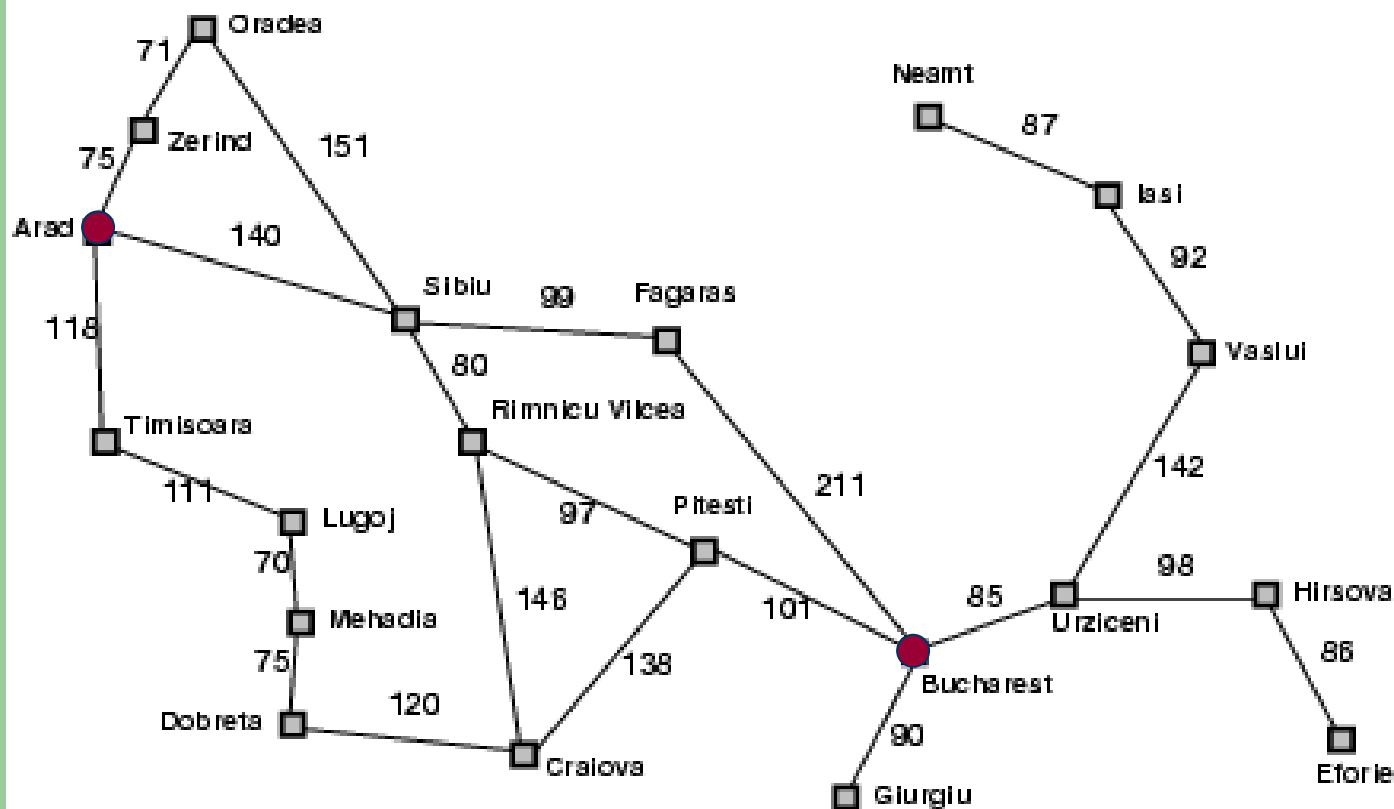
If $h(n)$ is correctly done, better nodes, closer to the goal, have better estimate.

How would you estimate the distance from the current town to the goal town in this case?

Romania with step costs in km



Romania with step costs in km



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

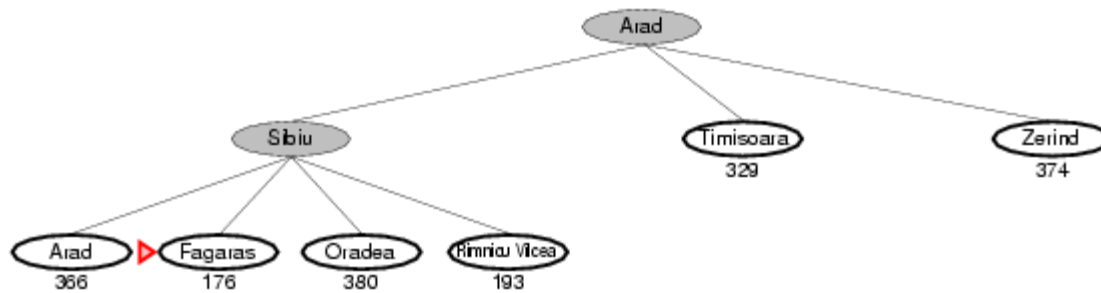
Greedy best-first search example



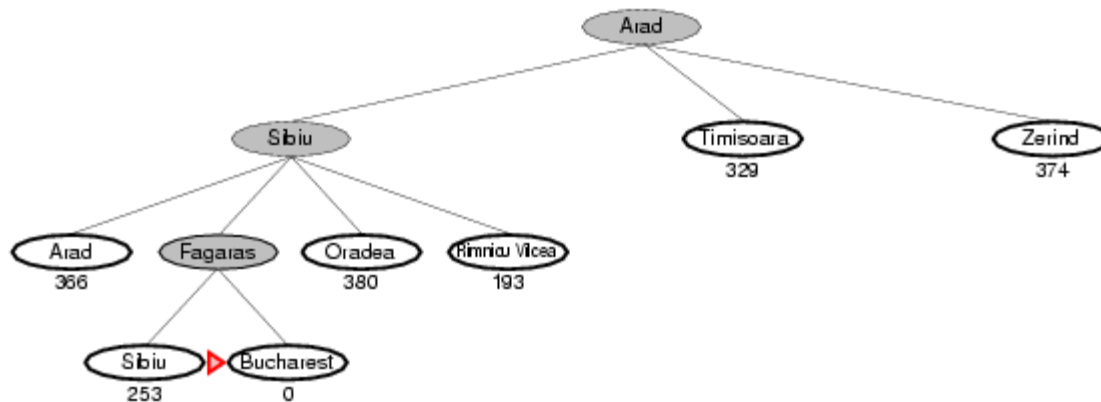
Greedy best-first search example



Greedy best-first search example



Greedy best-first search example

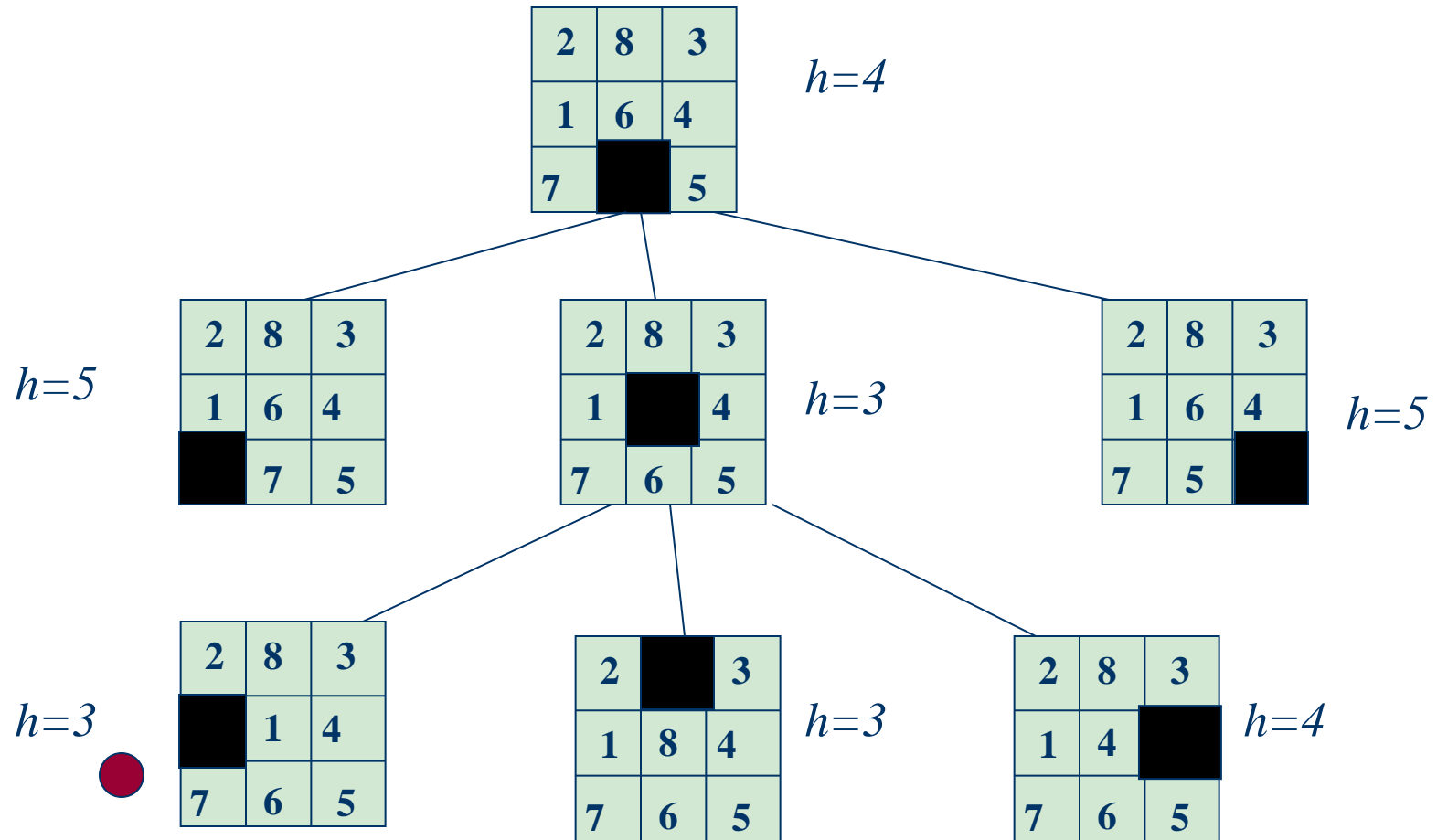


1	2	3
4		4
7	6	5

Correct ordering
in the 8 puzzle
problem

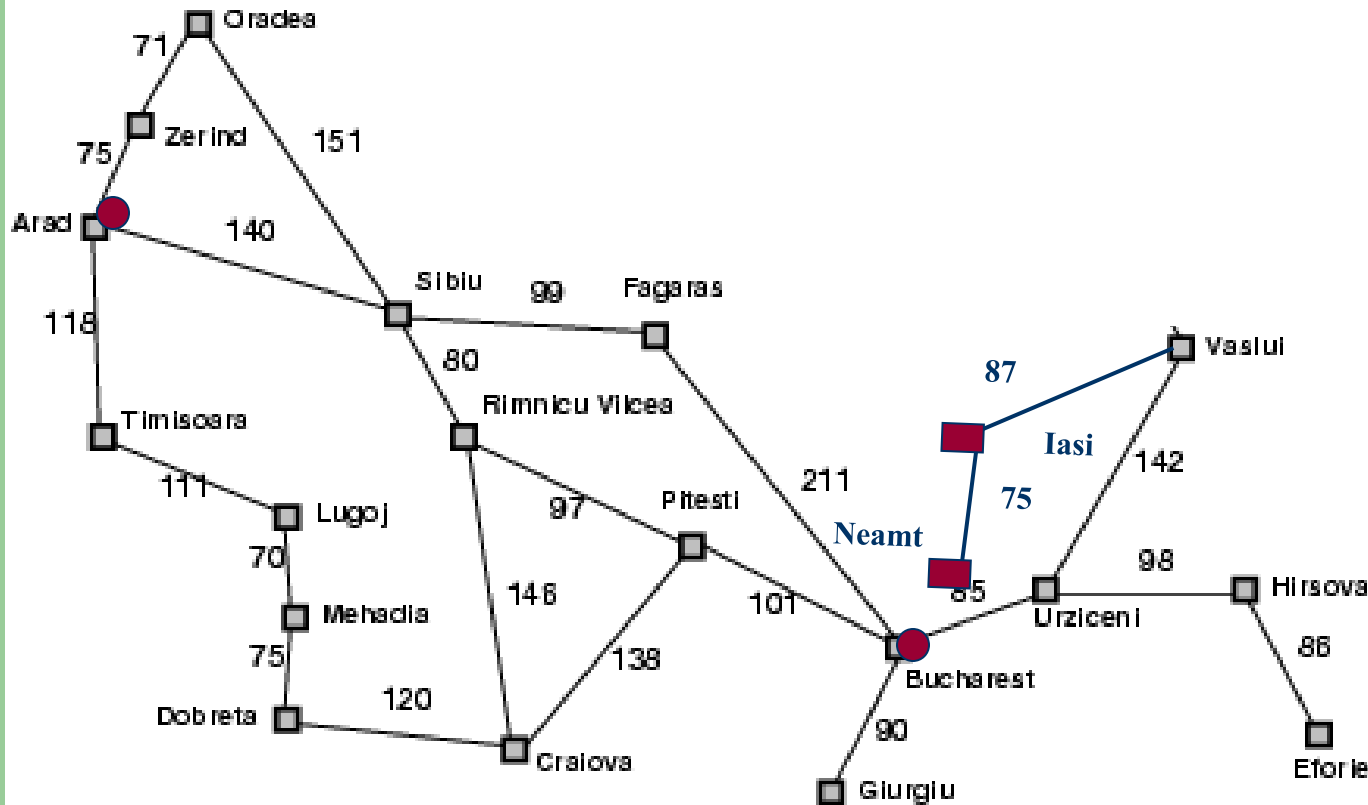
What is the state space in this
case? Can you draw it, at least
part of it? Heuristic function? Number of tiles
on bad positions.

Greedy search / another example



Is greedy search complete?

Romania with step costs in km



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 the greedy best-first search

- Complet? No, possibility of the loops like lasi
→ Neamt → lasi → Neamt →
- Time complexity? $O(b^m)$, but a good heuristics
can do a dramatical improvement (m – max.
search depth)
- Memory complexity ? $O(b^m)$
- Optimal? No

Algorithm A*

Cost function $f(n)=g(n)+h(n)$; $g(n)$ and $h(n)$ are not zero

$g(n) \neq 0$ - real cost from the initial state to n

$h(n) \neq 0$ - estimate of the cost to the goal

$f(n)$ – estimate of the cheapest cost from the initial state to the goal state through the node n . The node with **the lowest f** is developed as the first one.

A* search example

▶ Arad
366=0+366

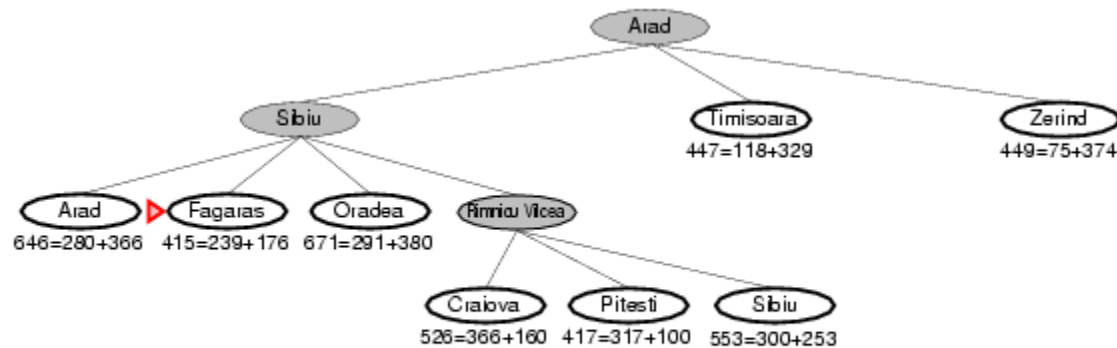
A* search example



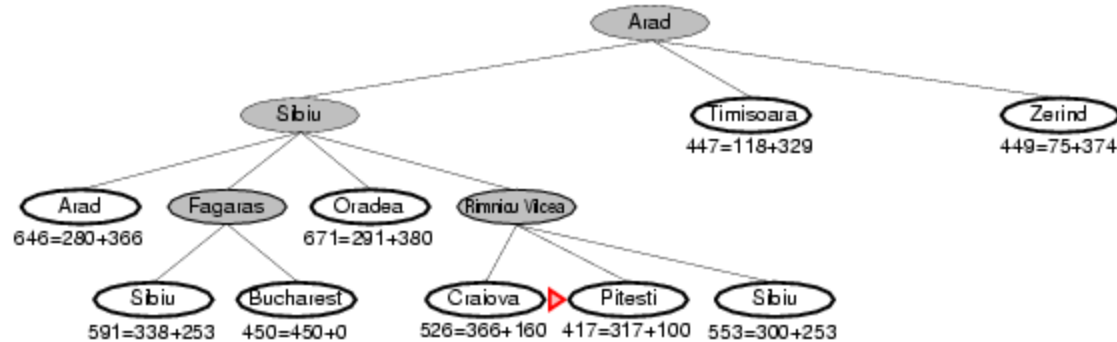
A* search example



A* search example



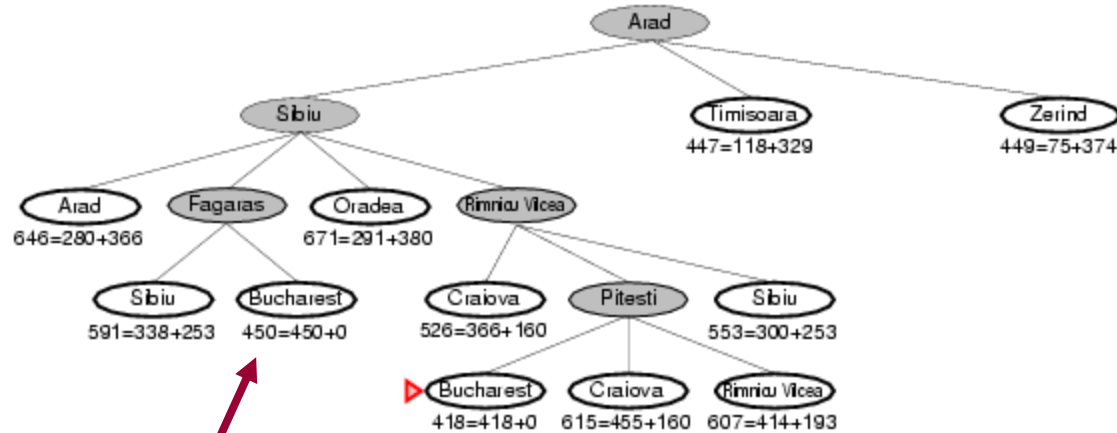
A* search example



In spite of putting goal node into the fringe, there are still better nodes .

Suboptimal goal is
in a fringe

A* search example



Suboptimal goal

Optimal goal

Algorithm found an optimal goal,
there are no better nodes to develop

Tree search and Graph search A*

- Complication : loops such as Arad – Sibiu, Arad – Sibiu
- Solution: implement Graphsearch algorithm, except of Treesearch
- Difference between Graphsearch and Treesearch: Treesearch does not take the *CLOSED* list into account.
- The *OPEN* list : nodes waiting to be developed in a fringe
- The *CLOSED* list: expanded nodes

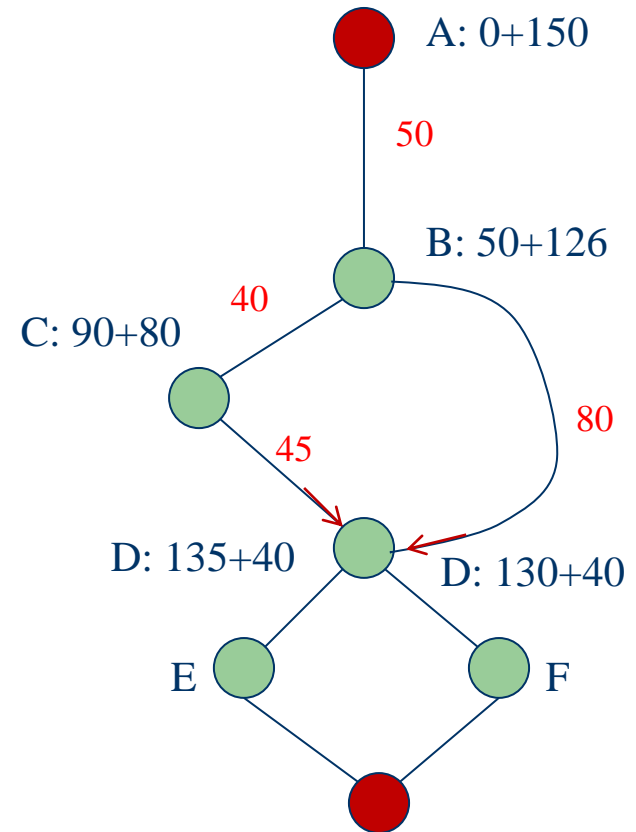
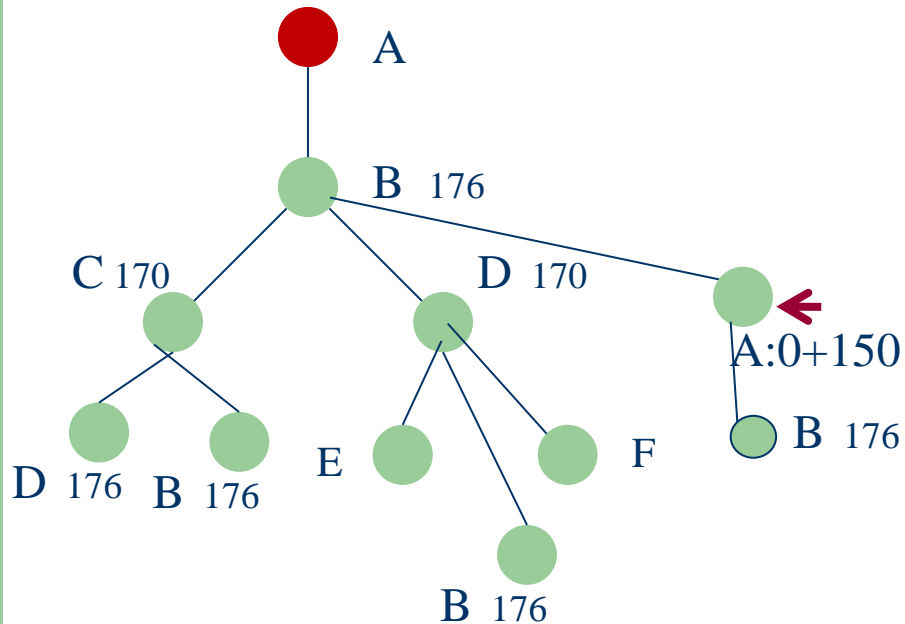
Tree search and Graph search A*

- Complication : loops such as Arad – Sibiu, Arad – Sibiu
- Solution: implement Graphsearch algorithm, except of Treesearch
- Difference between Graphsearch and Treesearch: Treesearch does not take the *CLOSED* list into account.
- The *OPEN* list : nodes waiting to be developed in a fringe
- The *CLOSED* list: expanded nodes

Tree search in general

1. Start with n_0 , put it to the ordered list *OPEN*. *OPEN* is empty at the beginning.
3. **If** *OPEN* is empty, **then** end and mistake announcement
4. Choose the first node from *OPEN*, delete it from *OPEN*. This is a node n .
5. **If** n is the goal, **then** end. Solution is the path from n_0 to n .
7. Develop n . Reorganize an *OPEN* list according the chosen strategy
8. Go to 3.

Tree search implementation:



Graphsearch in general

- 1. Start with n_0 , put it to the ordered list *OPEN*.
- 2. Create *CLOSED* list, empty at the beginning
- 3. **If** *OPEN* is empty, **then** end and mistake announcement
- 4. Choose the first node from *OPEN*, delete it from *OPEN* and put it to the *CLOSED*. This is a node n .
- 5. **If** n is goal, **then** end. Solution is the path from n_0 to n .
- 6. **Expand n and create the set of successors M , create the edge representation between n and the members of M .**
- 7. Reorganize *OPEN* according the chosen strategy
- 8. Go to 3.

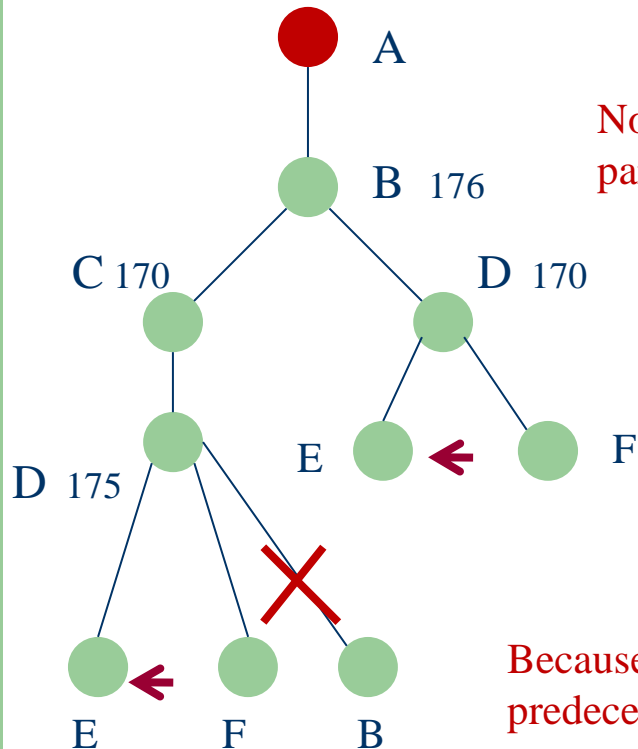
This can be used for the uninformed strategies as well, depending on the fringe in the *OPEN* list. For the bread first search new nodes are given at the end of the *OPEN* list, for depth first at the beginning of the *OPEN* list.

Graphsearch – implementation of 6

To avoid repeated searching in the small or the greater loops, 6 is implemented like this:

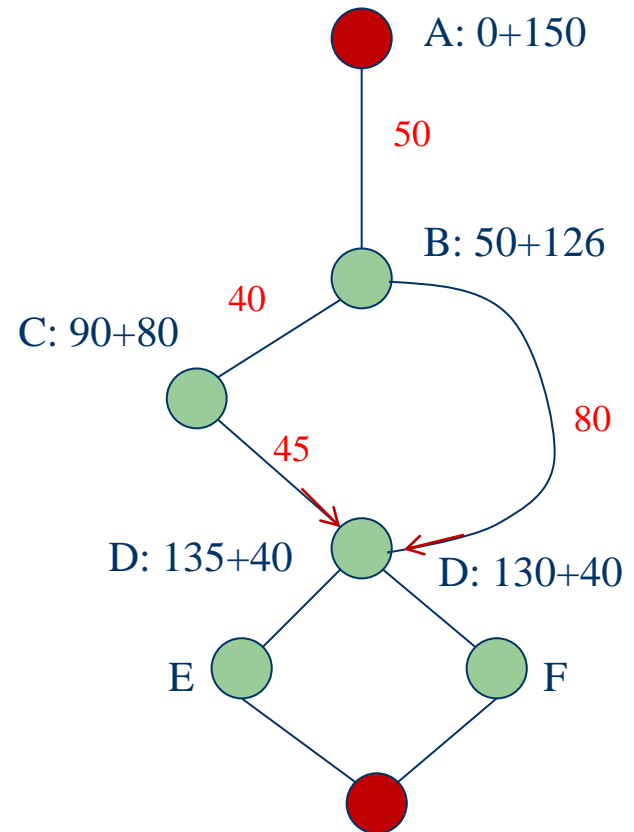
- 6a)** Expand the node n , create the successor set M ,
put there all successors which are not **parents** of n . Create the
edge representation between n and the members of M .
- 6b)** Expand the node n , create the successor set M ,
put there all successors which are not **predecessors** of n .
Create the edge representation between n and the members of
 M .

Example of the 6a) and 6b) situation:



No A, because A is a parent of B.

Because B is a predecessor of D.



What it is about?

1. 6a) implementation eliminates short loops.
2. 6b) implementation eliminates greater loops.
3. 6b) is often ignored, because of time consuming control whether the node is not a predecessor.
4. There is still a possibility of repeated searching of some sub trees, as we have seen in the example.

Completeness of A*

A* is complete if it has these properties:

1. Each node has a finite number of successors.
2. In each step the path cost grows at least as $\delta > 0$.
3. Heuristics is **admissible** (tree search implementation).
4. Heuristics is consistent (graph search implementation).

Admissible heuristics $h(n)$ **cannot overestimate** the path cost from the initial node to the goal through the n node.

Having two A_1^* and A_2^* algorithms, differing in the heuristic functions only, and if for all no goal nodes holds $h_1 < h_2$, we say, that A_2^* is more informed than A_1^*

It is possible to prove, that each node developed by the more informed algorithm is developed by the less informed one too.

Summary

- Function for the best first search is $f(n)=g(n)+h(n)$
- Greedy, A^*
- Properties of A^* .