

# **COMP3308/COMP3608, Lecture 2**

# **ARTIFICIAL INTELLIGENCE**

## **Problem Solving and Search.**

**Uninformed Search: BFS, DFS, UCS and IDS**

**Informed Search: Greedy Best-First**

**Reference: Russell and Norvig, ch. 3**

# Reminders

- **Tutorials start this week**
  - **All tutorials will be on Zoom live-streamed**
  - **There are 2 on Tuesday and 14 on Wednesday**
  - **Please attend your allocated tutorial**
  - **1 tutorial will be recorded and uploaded to Canvas (under “Recorded Lectures”)**
- **All students must submit their homework in Canvas by 4pm on Tuesday (no matter when your tutorial class is)**
- **Tutorial solutions will be available on Canvas on Wednesday at 6pm (after the last tutorial)**
- **Lecture slides are posted on Saturday morning at 9am**
  - **They will not contain the answers to all questions and exercises that we do during the lecture**
  - **The complete lecture slides with the answers will be available on Monday at 12noon (after the lecture time)**

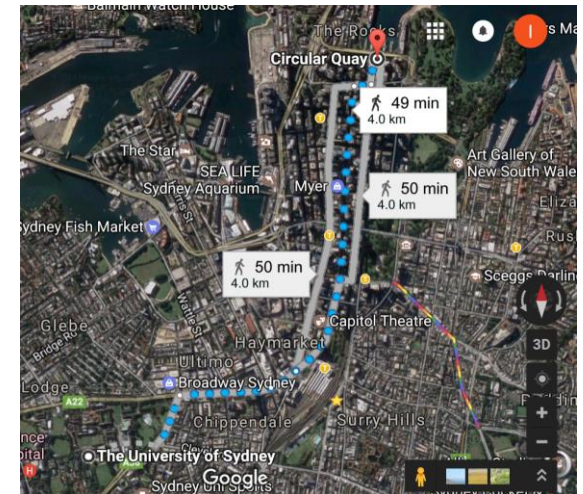
# Outline

- **Problem-solving and search**
- **Search strategies**
  - **Uninformed (blind)**
  - **Informed (heuristic)**
- **Uninformed search strategies**
  - **Breadth-first**
  - **Uniform cost**
  - **Depth-first**
  - **Depth-limited**
  - **Iterative-deepening**
- **Informed search strategies**
  - **Greedy best-first**
  - **A\* - next week**

# Problem Solving and Search

# Solving Problems by Searching

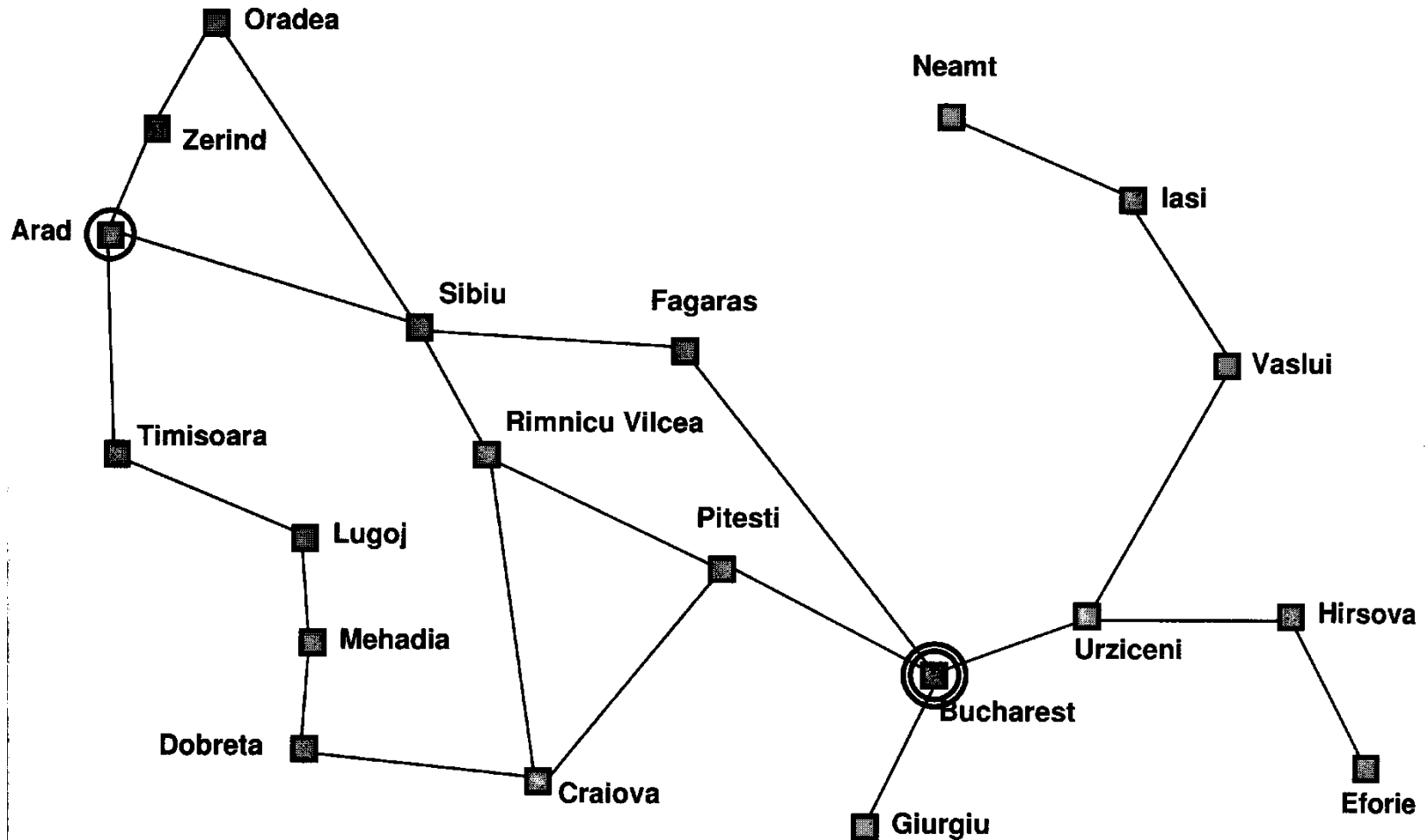
- Many tasks can be formulated as search problems
- Task: to get from an *initial problem state* to a *goal state*
  - Driving: starting address -> destination address
  - Chess: initial board position -> checkmate position
  - Reasoning: set of facts and rules + query -> answer
- Solution: a path from the initial state to the goal state
- Operators
  - possible actions, possible moves in a game
- Cost (quality) of operators
  - distance, time, money
  - strength of board position in games



# Example: Romania

- **On holiday in Romania**
  - currently in Arad
  - flight leaves tomorrow from Bucharest
- **Step 1. Formulate *goal*: be in Bucharest**
- **Step 2. Formulate search problem:**
  - *states*: various cities
  - *operators*: drive between cities, e.g. from Arad to Sibiu, from Arad to Zerind
- **Step 3. Find solution (path):**
  - a sequence of cities from Arad to Bucharest, e.g. Arad, Sibiu, Fagaras, Bucharest

# Romania Example - Map



# Romania



[http://en.wikipedia.org/wiki/File:Location\\_Romania\\_EU\\_Europe.PNG](http://en.wikipedia.org/wiki/File:Location_Romania_EU_Europe.PNG)



# Search Problem Formulation

- A search problem is defined by 4 items
  - 1) **Initial state**, e.g. Arad
  - 2) **Goal state** (1 or more)
    - Stated **explicitly**, e.g. Bucharest
    - Stated **implicitly** with a goal test, e.g. checkmate state
  - 3) **Operators** = actions
    - A set of possible actions transforming 1 state to another , e.g. Arad->Zerind, Arad->Sibiu, etc.
  - 4) **Path cost function** – assigns a numeric value to each path
    - Reflects the performance measure
    - E.g. time, path length in km, number of operators executed
- **Solution** – a path from the initial to a goal state
  - Its quality is measured by the path cost
  - Optimal solution is the one with the lowest path cost
- **State space** - all states reachable from the initial state by operators

# Choosing States and Actions - Abstraction

- Real problems are too complex, to solve them we need to *abstract* them, i.e. to simplify them by removing the unnecessary details
  - E.g. we ignore the weather, travel companion, scenery; they are irrelevant for the task of finding a path to Bucharest
  - The actions need to be suitably specified, e.g. “turn the steering wheel to the left by 5 degrees” is not appropriate
  - => the *level of abstraction* must be appropriate (valid and useful)
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
- (Abstract) solution = a real path that is solution in the real world

# Example Problem 1: The 8-puzzle

7	2	4
5		6
8	3	1

Start State

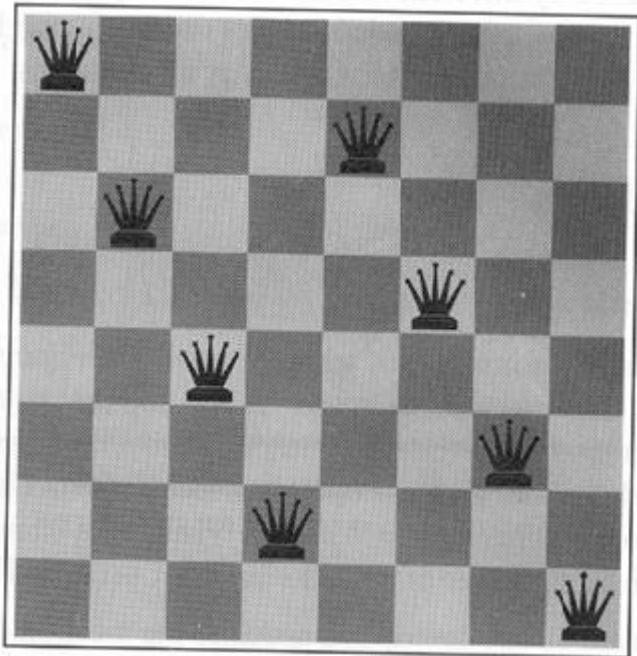
	1	2
3	4	5
6	7	8

Goal State

a sliding block puzzle

- *States?* 1 state = 1 board configuration of the tiles
- *Operators?* Move blank left, right, up, down
- *Goal state?* Given
- *Path cost?* 1 per move, i.e. path cost=length of path=number of steps

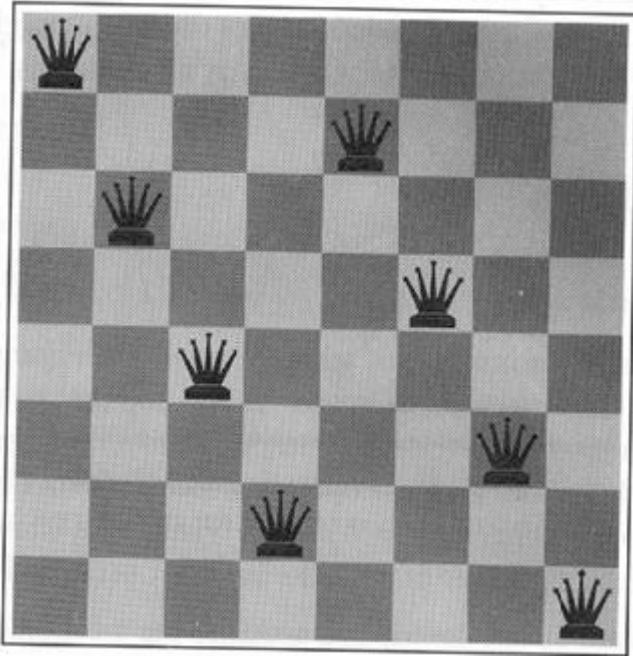
# Example Problem 2: The 8-queens



- Place 8 queens on a chessboard (8x8) so that no queen attacks each other
- Is this a solution?

- *Goal state?* 8 queens on board, none attacked
- *Path cost?* 0 (= only the goal state matters and not the number of steps). Other options are also possible, e.g. 1 per move.
- *States =?* 1 state = 1 board configuration of the tiles
- *Operators =?* Put 1 queen on the board or move 1 queen

# 8-queens - Different Types of Problem Formulation



1. **Incremental** – start with an empty board, add 1 queen at a time
2. **Complete-state** – start with all 8 queens and move them around

**Let's take a closer look at 2 possible incremental formulations!**

# 8-queens – Incremental Formulation 1

- *States?* Any arrangement of 0 to 8 queens
- *Initial state?* No queens on the board
- *Operators?* Add a queen to any square
- State space =  $1.8 \times 10^{14}$  states (=  $64 \times 63 \times \dots \times 57$ )
- Can you suggest a better formulation?
- Prohibit placing a queen in any square that is already attacked!

# 8-queens – Incremental Formulation 2

- *States?* Any arrangement of 0 to 8 queens, 1 in each column, with no queen attacking another
- *Initial state?* No queens on the board
- *Operators?* Place a queen in the left-most empty column such that it is not attacked by any other queen
- State space: 2057 states (has been proven)
- For 100-queens:
  - formulation 1:  $10^{400}$  states
  - formulation 2:  $10^{52}$  states (huge improvement but problem still not tractable)
- The problem formulation is very important!

# Searching for Solutions

- How can we solve the 8-queens puzzle or the Romania example?
- By *searching the state space*
- We can generate a *search tree* starting from the initial state and applying the operators
- Or we can generate a *search graph* – in a graph the same state can be reached from multiple paths



# Tree-Search algorithm – Pseudo Code

- **Basic idea:** offline exploration of the state space by generating successors of the explored states (i.e. by *expanding* states)

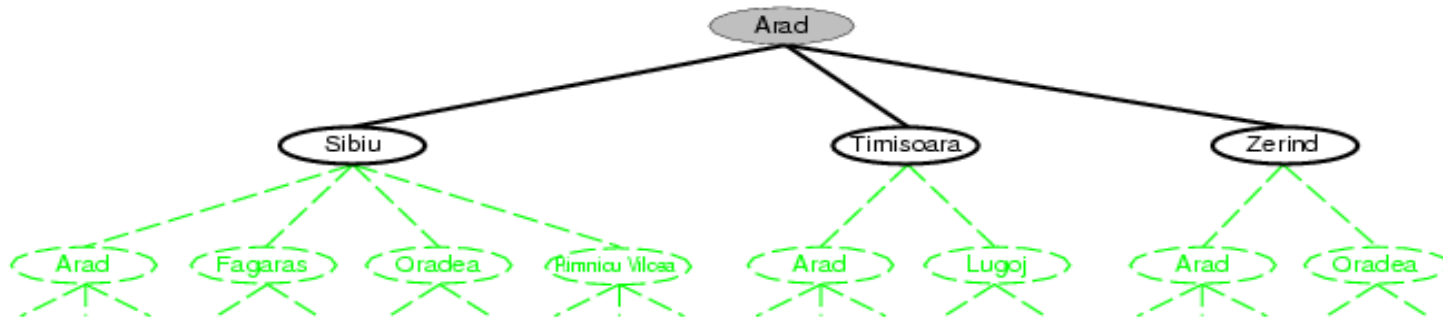
```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal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

Annotations in the pseudo-code:

- start with the initial node* (points to "the initial state of *problem*")
- the initial node at the beginning* (points to "a leaf node")

1. Choose a node for expansion based on the search strategy
2. Check if it is a goal node
  - Yes-> return solution
  - No -> expand it by generating its children

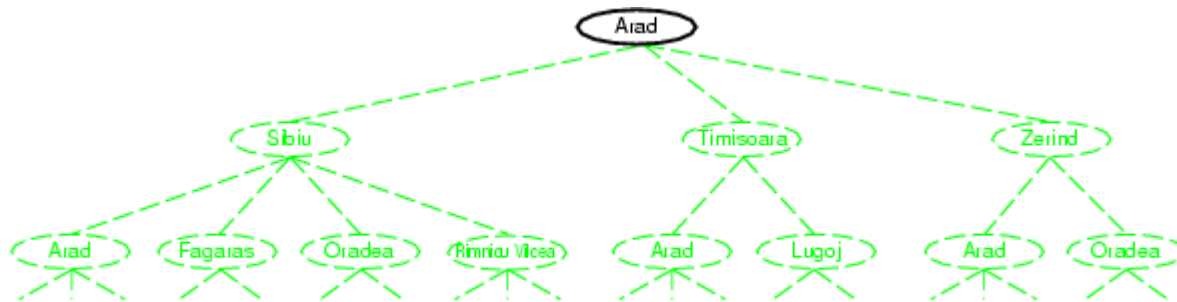
# Expanded and Fringe Lists



- We will keep two lists:
  - **Expanded** – for nodes that have been expanded
  - **Fringe** – for nodes that have been generated but not expanded yet.
  - We will keep the fringe ordered (implemented as a priority queue) and always select for expansion the first node of the fringe
- For the figure:
  - **Expanded:** Arad
  - **Fringe:** Sibiu, Timisoara, Zerind
  - All the other nodes have not been generated yet
  - Note the notation in the figure for the different types of nodes

# Tree Search Example

- Let's put the children of the expanded node at the end of the fringe

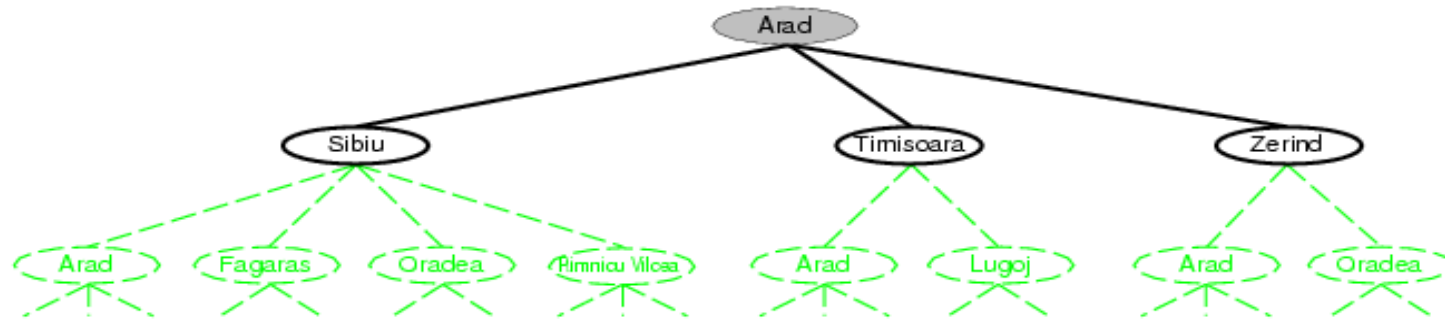


**Fringe: Arad**

**Is Arad a goal node? No => expand it**

**Expanded: none**

# Tree Search Example (2)



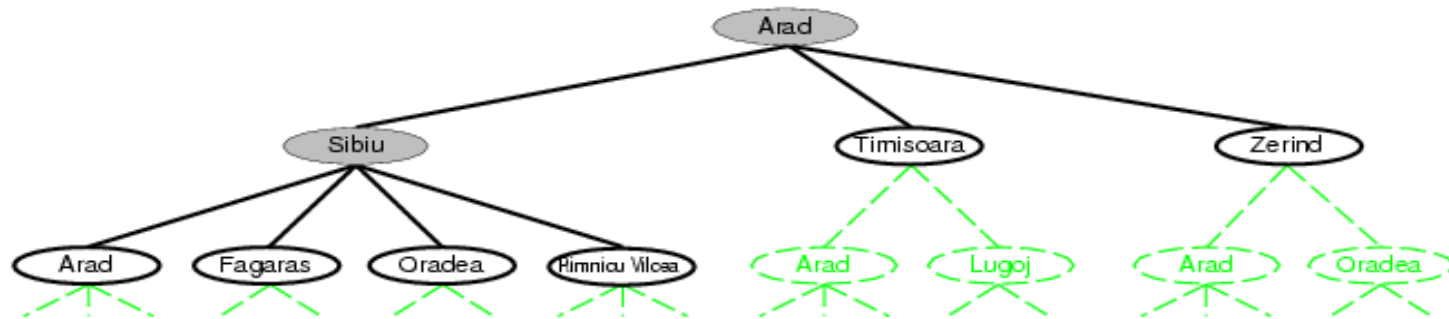
**Fringe: Sibiu, Timisoara, Zerind**

**Expanded: Arad**

**Is Sibiu a goal node?**

**No => expand it**

# Tree Search Example (3)



**Fringe: Timisoara, Zerind, Arad, Fagaras, Oradea, Rimnicu Vilcea**

**Expanded: Arad, Sibiu**

# Tree Search – More Detailed Pseudo Code

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do // nodes in fringe ordered based on their priority according to the search strategy
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe) //select node for expansion, then check if it is a goal
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

---

```
function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```

Is the first node in the fringe a goal node?

Yes => stop and return solution

No => expand it:

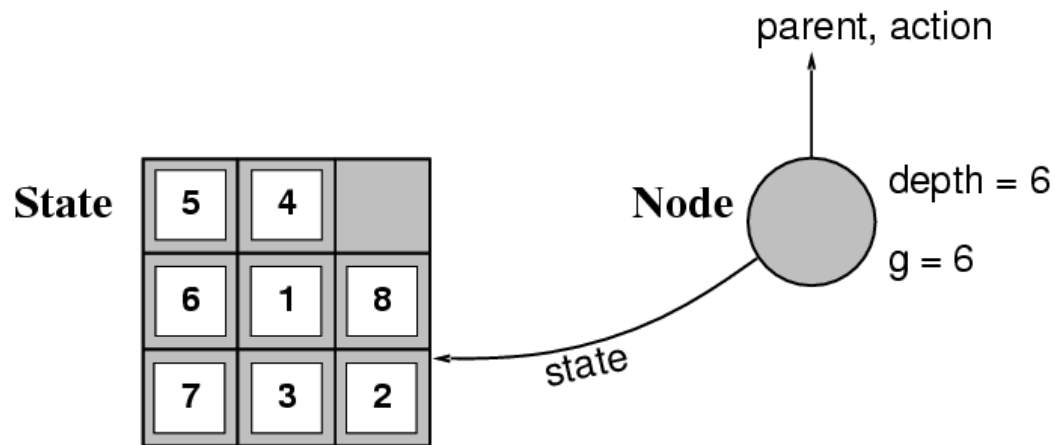
- Move it to the expanded list
- Generate its children and put them in the

fringe in a order defined by the search strategy  
(top of the fringe = most desirable child)

- We will keep the fringe ordered

# Nodes vs States

- A *node* is different than a *state*
- A node:
  - represents a state
  - is a data structure used in the search tree
  - includes *parent*, *children*, and other relevant information e.g. *depth* and *path cost g*



# Search Strategies

- A *search strategy* defines which node from the fringe is most promising and should be expanded next
- We will keep the nodes in the fringe ordered based on the search strategy and always expand the first one (i.e. the best one)
- Evaluation criteria
  - *Completeness* – is it guaranteed to find a solution if one exists?
  - *Optimality* – is it guaranteed to find an *optimal (least cost path)* solution?
  - *Time complexity* – how long does it take to find the solution? (measured as number of generated nodes)
  - *Space complexity* – What is the maximum number of nodes in memory?
- Time and space complexity are measured in terms of:
  - $b$  – max branching factor of the search tree (we can assume that it is finite)
  - $d$  – depth of the optimal (least cost) solution
  - $m$  – maximum depth of the state space (can be finite or not finite, i.e.  $\infty$ )
- Two types of search methods: *uniformed (blind)* and *informed (heuristic)*



# Uninformed (Blind) Search Strategies

# Uninformed (Blind) Search Strategies

- **Uninformed strategies:**
  - **Do not use problem-specific heuristic knowledge**
  - **Generate children in a systematic way, e.g. level by level, from left to right**
  - **Know if a child node is a goal or non-goal node**
  - **Do not know if one non-goal child is better (more promising) than another one. Exception: UCS, but this is not based on heuristic knowledge.**
  - **In contrast, informed (heuristic) use heuristic knowledge to determine the most promising non-goal child**
- **5 uninformed search strategies**
  - **Breadth-first**
  - **Depth-limited**
  - **Uniform-cost**
  - **Iterative deepening**
  - **Depth-first**

# Breadth-First Search (BFS)

- Expands the shallowest unexpanded node
- Implementation: insert children at the end of the fringe

**Fringe:** A

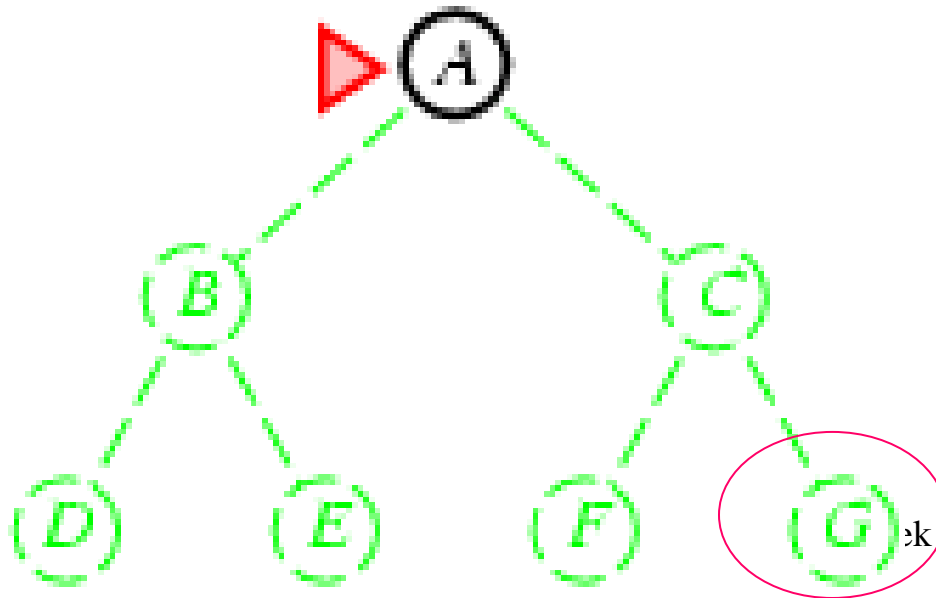
**Expanded:** none

Is the first node in the fringe a goal node?

Yes => stop and return solution

No => expand it:

- Move it to the expanded list
- Generate its children and put them in the fringe in a order defined by the search strategy



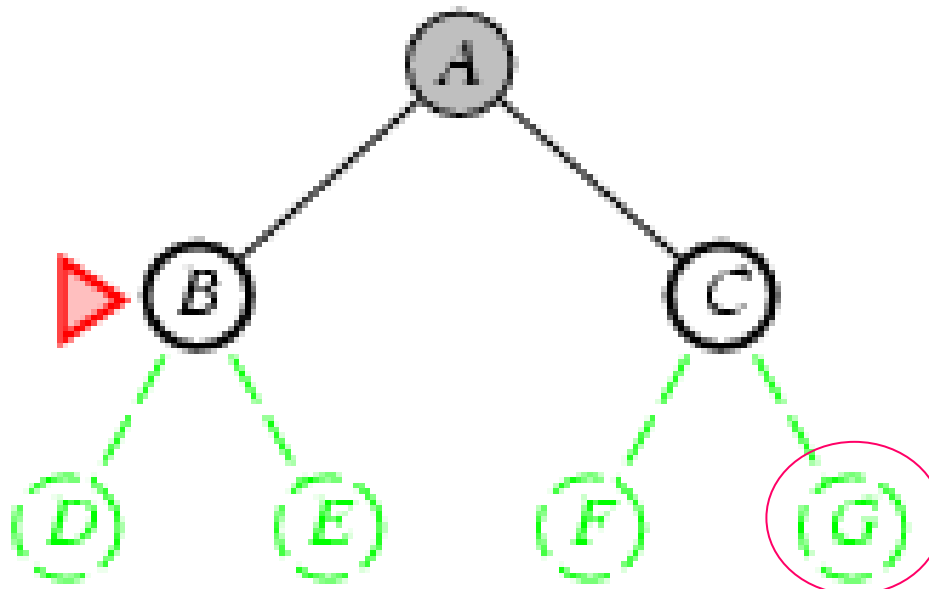
**Goal node**

# Breadth-First Search (BFS)

- Expands the shallowest unexpanded node
- Implementation: insert children at the end of the fringe

Fringe: B, C

Expanded: A

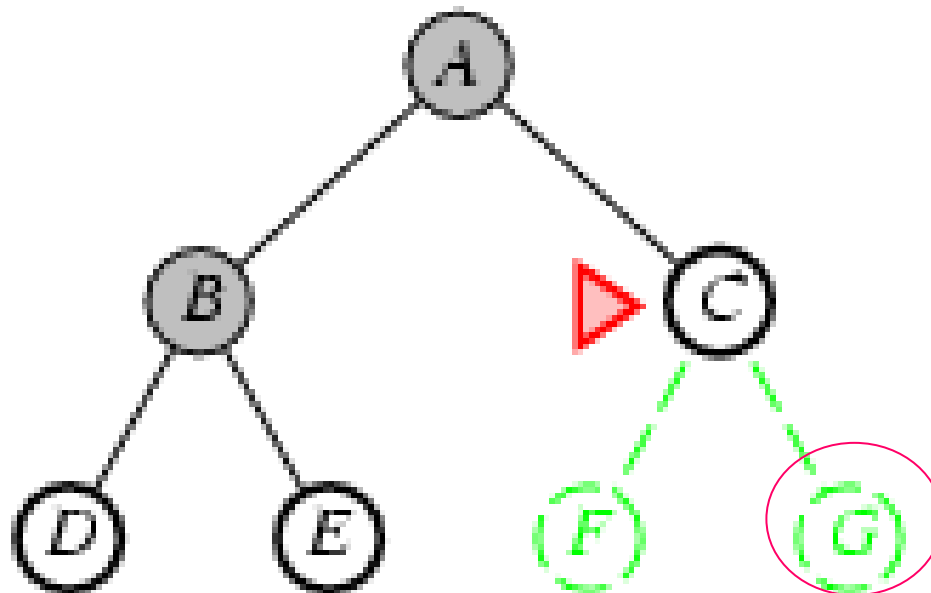


# Breadth-First Search (BFS)

- Expands the shallowest unexpanded node
- Implementation: insert children at the end of the fringe

**Fringe:** C, D, E

**Expanded:** A, B

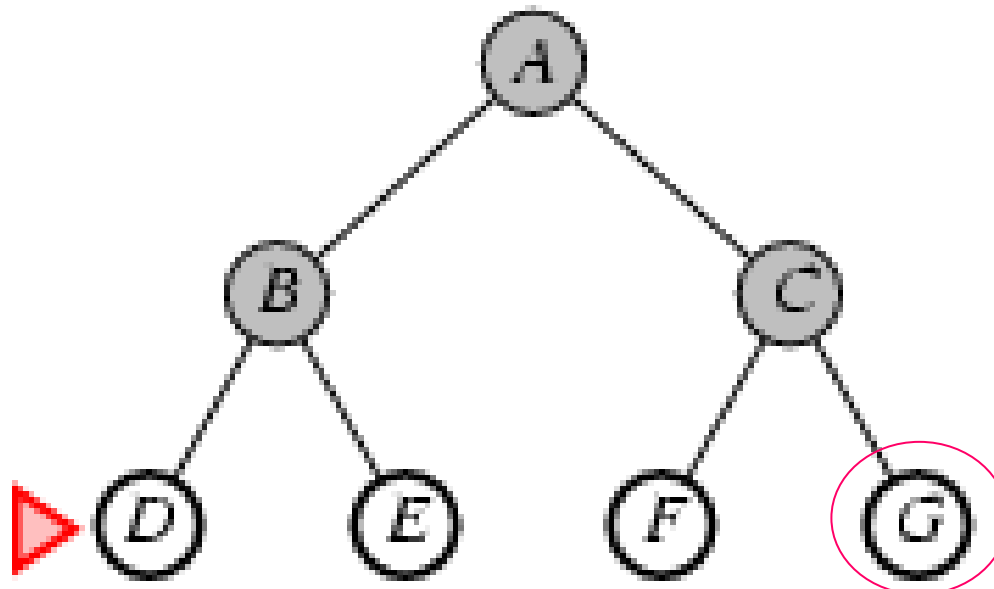


# Breadth-First Search (BFS)

- Expands the shallowest unexpanded node
- Implementation: insert children at the end of the fringe

**Fringe: D, E, F, G**

**Expanded: A, B, C**



# Breadth-First Search (BFS)

- Expands shallowest unexpanded node
- Implementation: insert children at the end of the fringe

**Fringe:** E, F, G

**Expanded:** A, B, C, D

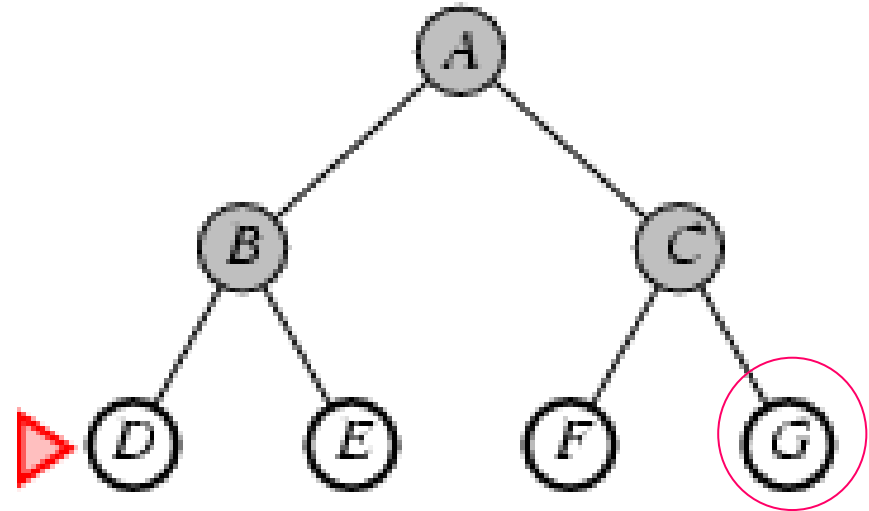
**Fringe:** F, G

**Expanded:** A, B, C, D, E

**Fringe:** G

**Expanded:** A, B, C, D, E, F

**G is a goal node => stop**



**Order of expansion:** ABCDEFG (the goal node is also included)

**Goal node found:** G

**Solution path found:** ACG (requires tracing back)

# BFS for Romania Example

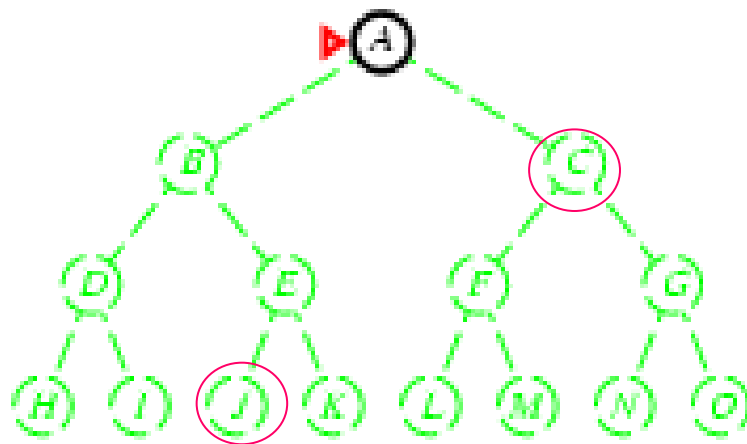
- **There will be loops (repeated states)**
- **Needs a repeated state checking mechanism**



# Properties of BFS

- **Complete?** Yes (if branching factor  $b$  is finite but we assume this)
- **Optimal?**
  - **Optimal solution = least cost path**
  - **It doesn't consider the step cost + we need to know the step costs**
  - **If all step costs are the same, e.g. =1, is BFS optimal? Yes**
  - **If the step costs are different, is BFS optimal? No**

The shallowest goal node is not necessarily the optimal one!



Suppose C and J were goal nodes and J is a better solution than C; BFS will find C – not optimal

# Properties of BFS (2)

- **Complete?** Yes
- **Optimal?** Not optimal in general; Yes, if step cost is the same, e.g. =1
- **Time?** generated nodes =  $1+b+b^2+b^3+b^4 + \dots + b^d = O(b^d)$ , exponential
- **Space?**  $O(b^d)$  (keeps every node in memory)
- Both time and space are problems as they grow exponentially with depth but space is the bigger problem!

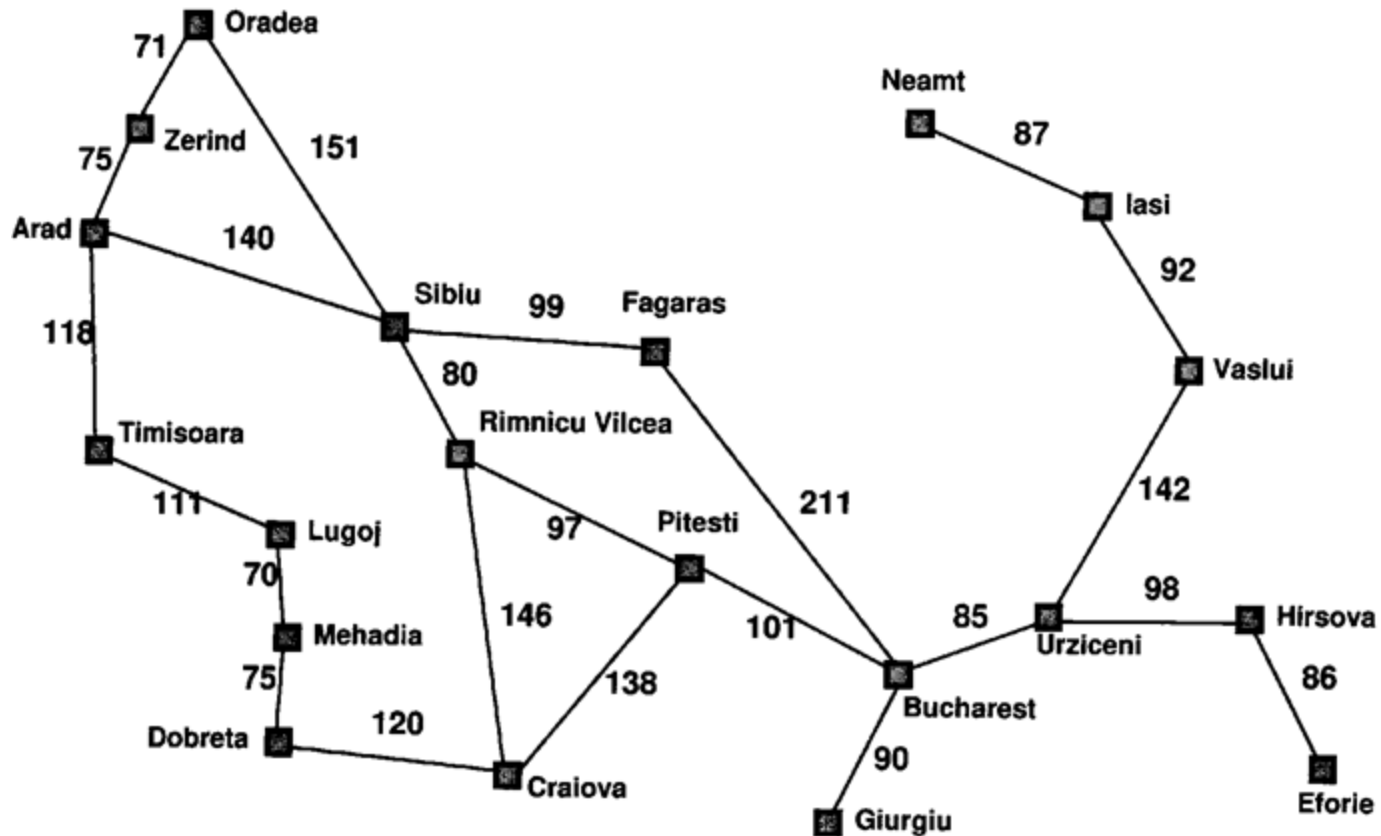
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	$10^6$	1.1 seconds	1 gigabyte
8	$10^8$	2 minutes	103 gigabytes
10	$10^{10}$	3 hours	10 terabytes
12	$10^{12}$	13 days	1 petabyte
14	$10^{14}$	3.5 years	99 petabytes
16	$10^{16}$	350 years	10 exabytes

**Figure 3.13** Time and memory requirements for breadth-first search. The numbers shown assume branching factor  $b = 10$ ; 1 million nodes/second; 1000 bytes/node.

Search problems with exponential complexity can NOT be solved by uninformed search except for small depth & small branching factor.

**We may wait 13 days for the solution to an important problem but there is still no personal computer with 1 petabyte of memory!**

# Romania with Step Coast in km



# Uniform Cost Search (UCS)

- We saw that:
  - BFS does not consider the step cost at all; it simply systematically expands the nodes level by level, from left to right
  - BFS finds the shallowest goal node but this may not be always the optimal solution (least cost solution = shortest path)
- UCS considers the step cost. It expands the least-cost unexpanded node - the node  $n$  with the *lowest path cost  $g(n)$  from the initial state*
- Implementation: insert nodes in the fringe in order of increasing path cost from the root

# UCS for the Romania Example (1)

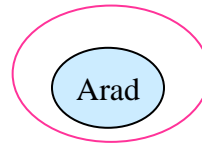
**Fringe: Arad**

**Expanded: none**

**Is Arad a goal state? No**

**-Move it to Expanded**

**-Generate its children and  
put them in Fringe in  
order**

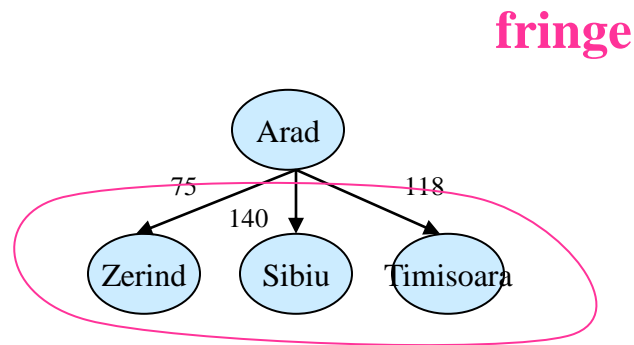


**fringe**

# UCS for the Romania Example

**Fringe: (Zerind,75), (Timisoara, 118), (Sibiu, 140)**

**Expanded: Arad**

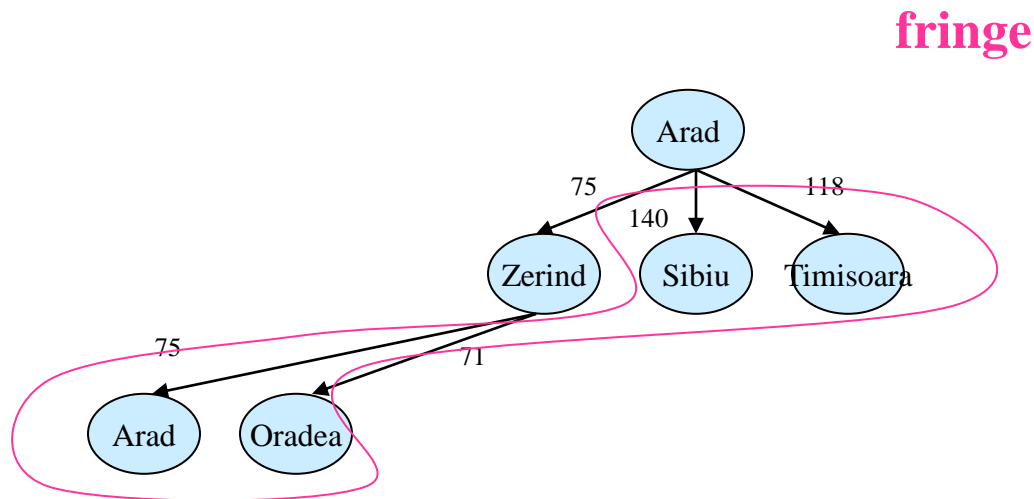


**The fringe is ordered based on the path cost from the root**

# UCS for the Romania Example

**Fringe:** (Timisoara, 118), (Sibiu, 140), (Oradea,146), (Arad, 150)

**Expanded:** Arad, (Zerind, 75)

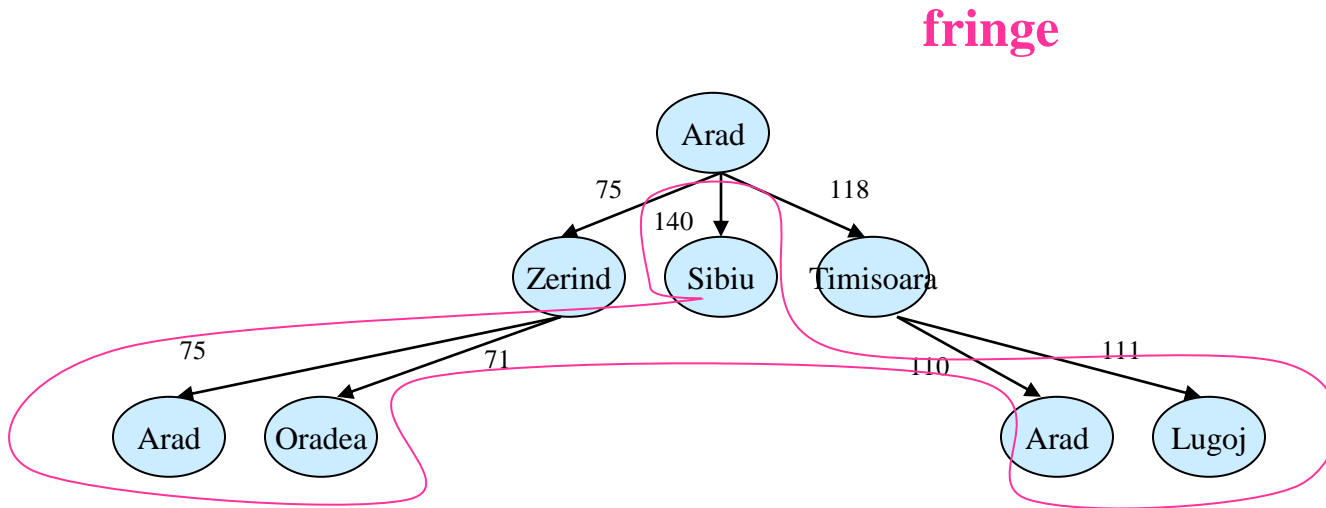


**The fringe is ordered based on the path cost from the root**

# UCS for the Romania Example

**Fringe:** (Sibiu, 140), (Oradea,146), (Arad, 150), (Arad,228), (Lugoj, 229)

**Expanded:** Arad, (Zerind, 75), (Timisoara, 118)



**The fringe is ordered based on the path cost from the root**

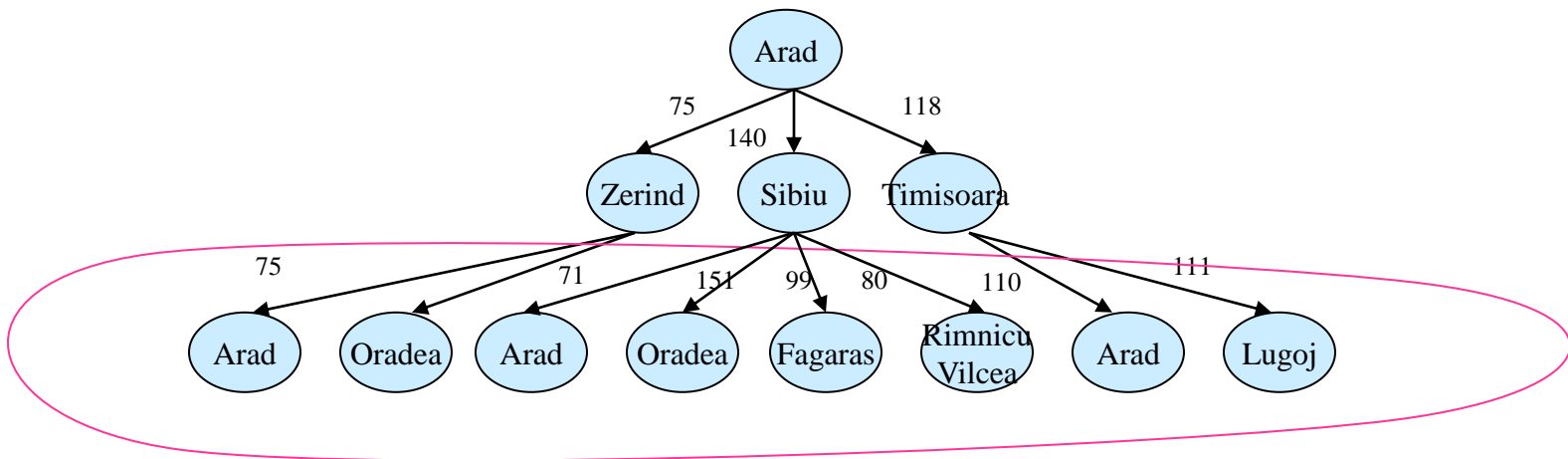


# UCS for the Romania Example

**Fringe:** (Oradea,146), (Arad, 150), (Rimnicu Vilcea, 220),  
(Arad,228), (Lugoj, 229), (Fagaras, 239), (Oradea , 291)

**Expanded:** Arad, (Zerind, 75), (Timisoara, 118), (Sibiu, 140)

fringe



**The fringe is ordered based on the path cost from the root**

etc. – not finished

# Properties of UCS

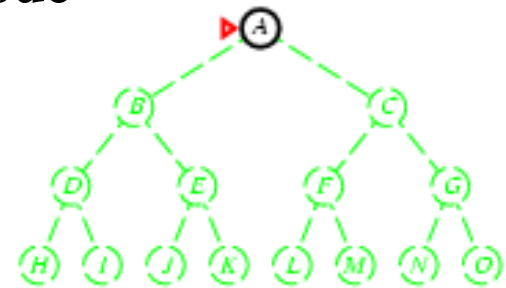
- Complete? Yes ( if step cost > 0 )
  - Optimal? Yes
  - Time? Depend on path costs not depths => difficult
  - Space? to characterize them in terms of b and d
- 
- Time?  $O(b^{1+\lceil C^*/\epsilon \rceil})$ , typically  $> O(b^d)$  -> because UCS may explore long paths of small steps before exploring paths with large steps which might be more useful
  - Space?  $O(b^{1+\lceil C^*/\epsilon \rceil})$ 
    - $C^*$  - cost of optimal solution
    - $\epsilon$  - the smallest step cost

# UCS and BFS

- When is UCS equivalent to BFS? What should be the step cost and the  $g(n)$  cost? Assume that among the nodes with the same priority the left most is expanded first. Reminder:
  - $g(n)$  is the cost from the root node to a given node
  - step cost is the cost between 2 nodes

Answer – in all of these cases:

1. step cost is 1
2. step cost is the same (a constant) - generalization of case 1
3.  $g(n) = \text{depth}(n)$
4.  $g(n)$  is the same at each level and increasing as the level increases, e.g. 5 for level 1, 7 for level 2, etc.



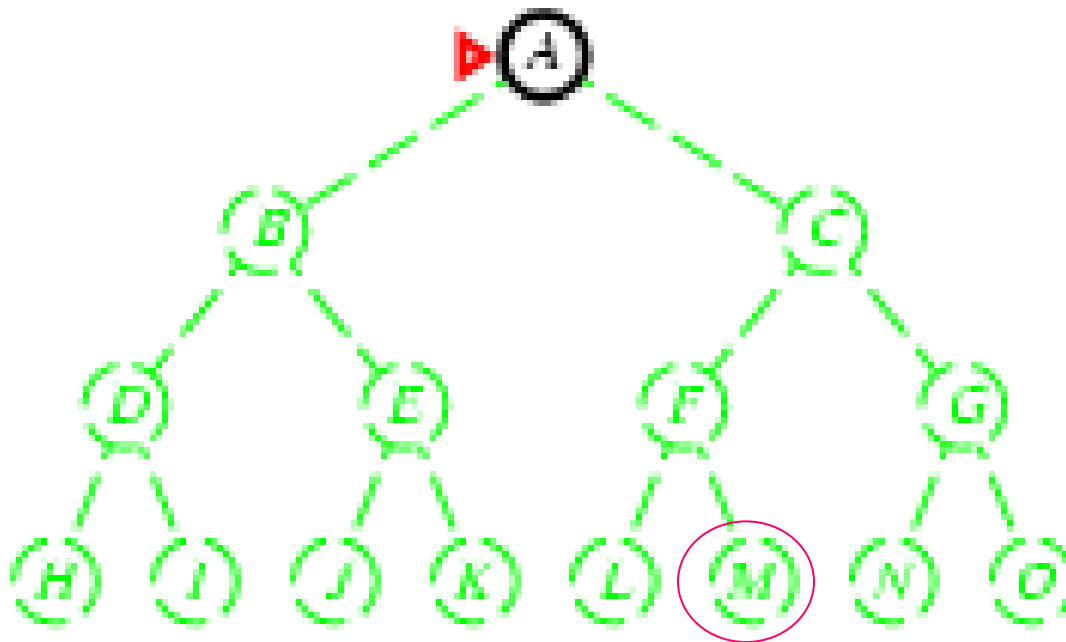
- Draw a tree for each of these cases and run UCS and BFS!

# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert children at the front of the fringe

Fringe: A

Expanded: none



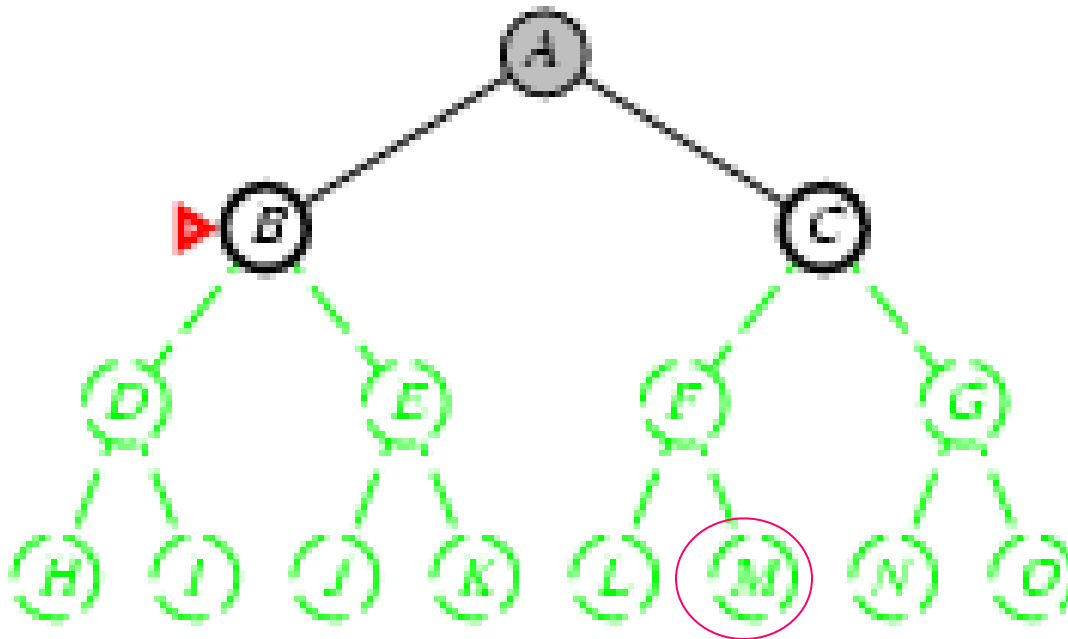
Goal node: M

# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: B, C

Expanded: A

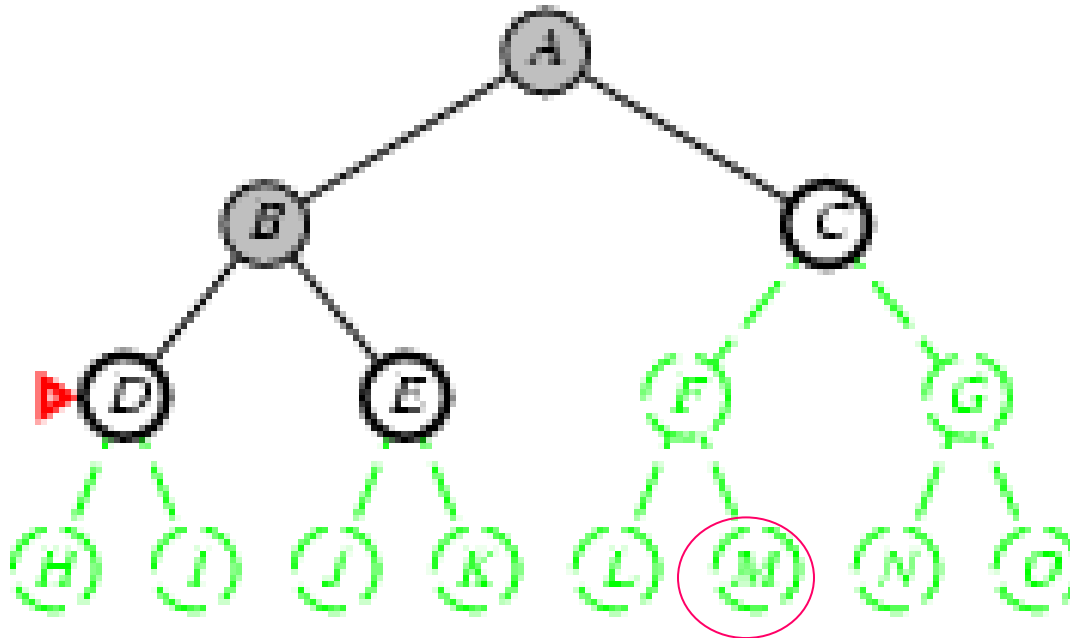


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: D, E, C

Expanded: A, B

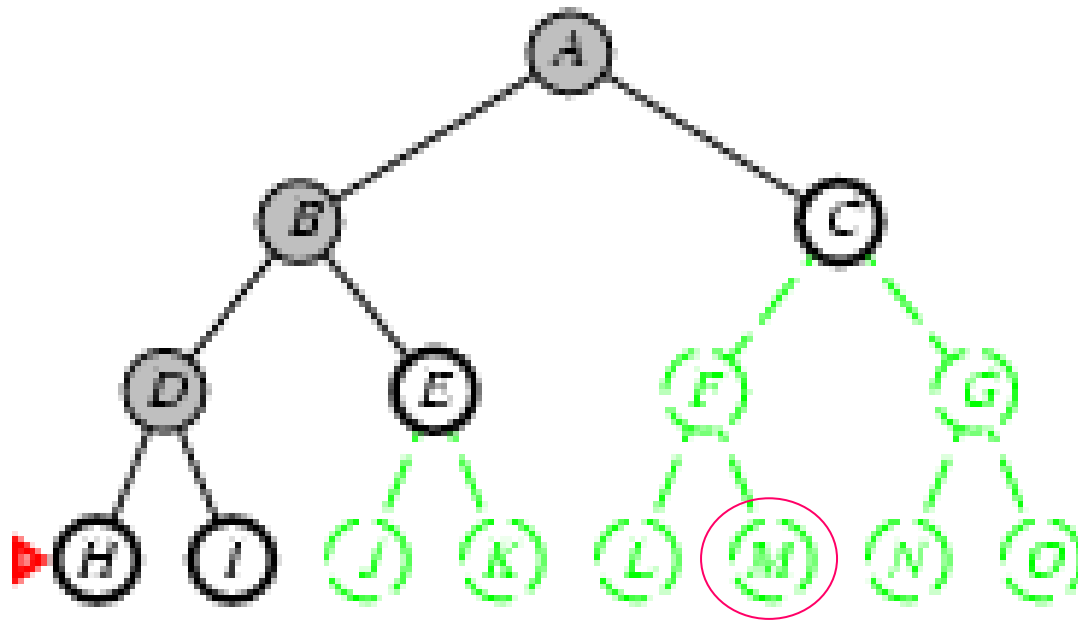


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: H, I, E, C

Expanded: A, B, D

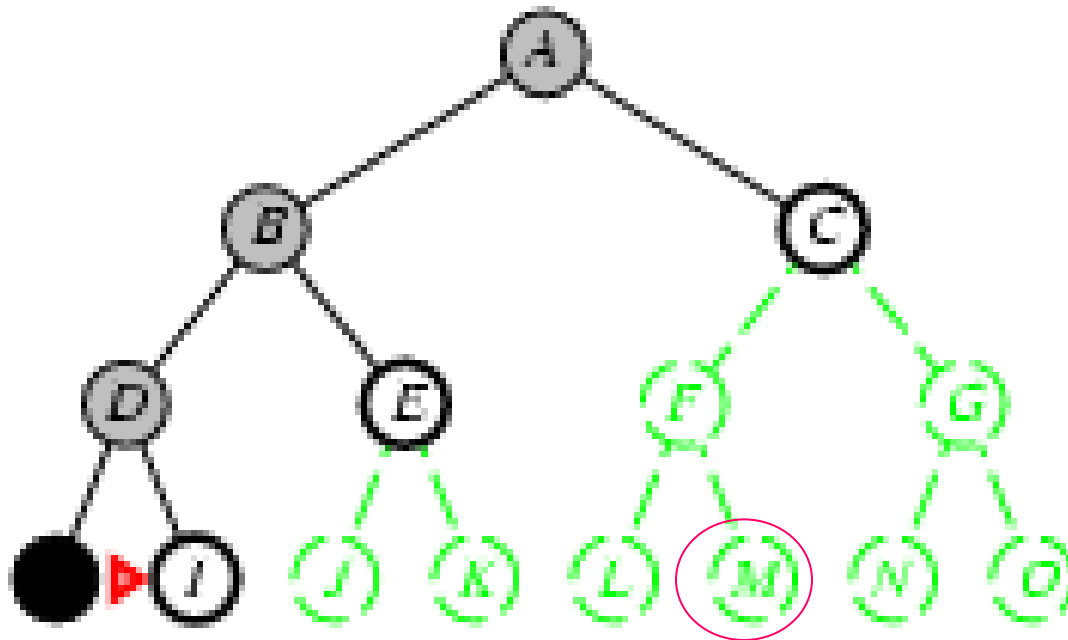


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: I, E, C

Expanded: A, B, D, H



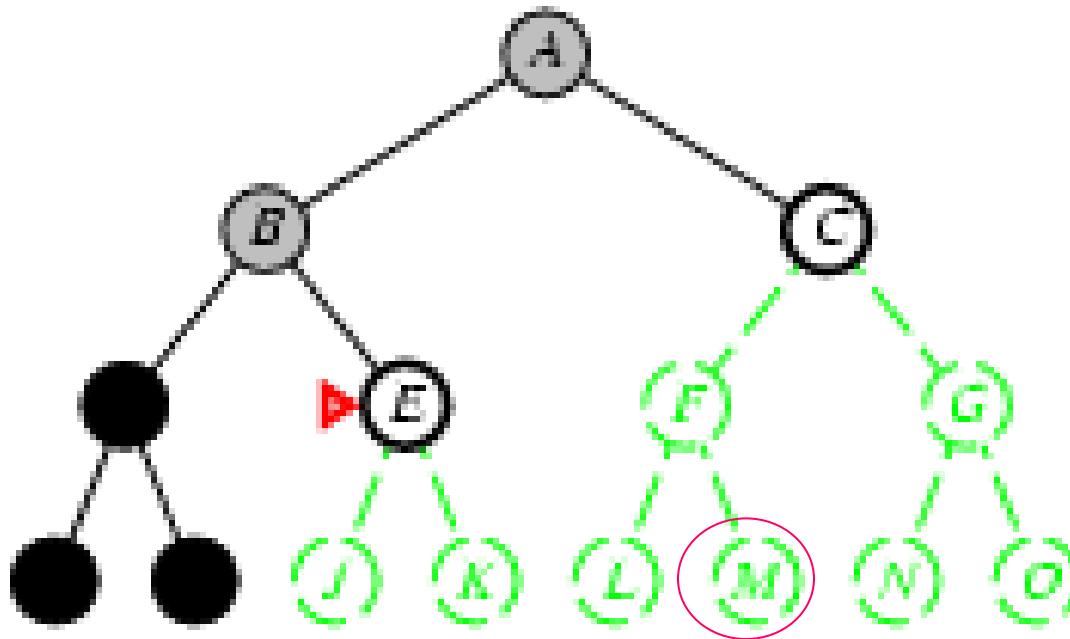


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: E, C

Expanded: A, B, D, H, I

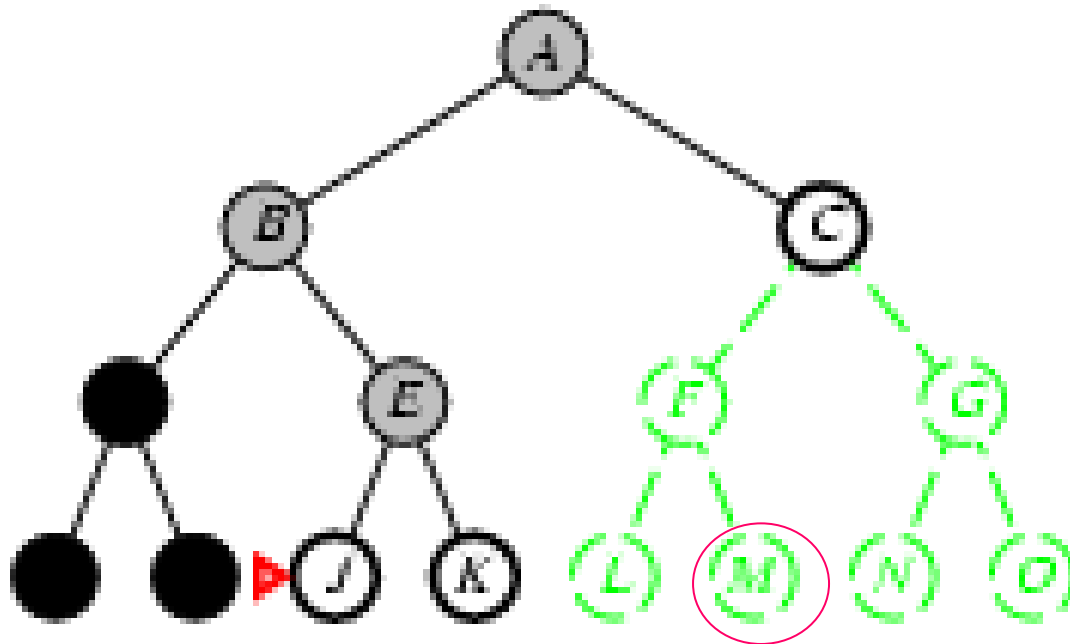


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: J, K, C

Expanded: A, B, D, H, I, E

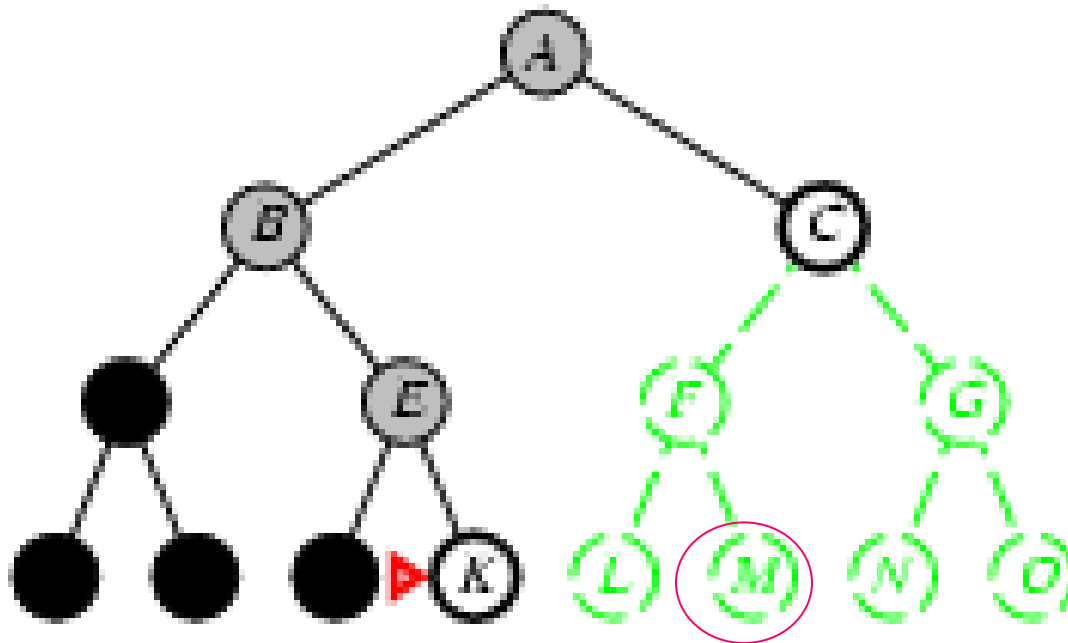


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: K, C

Expanded: A, B, D, H, I, E, J

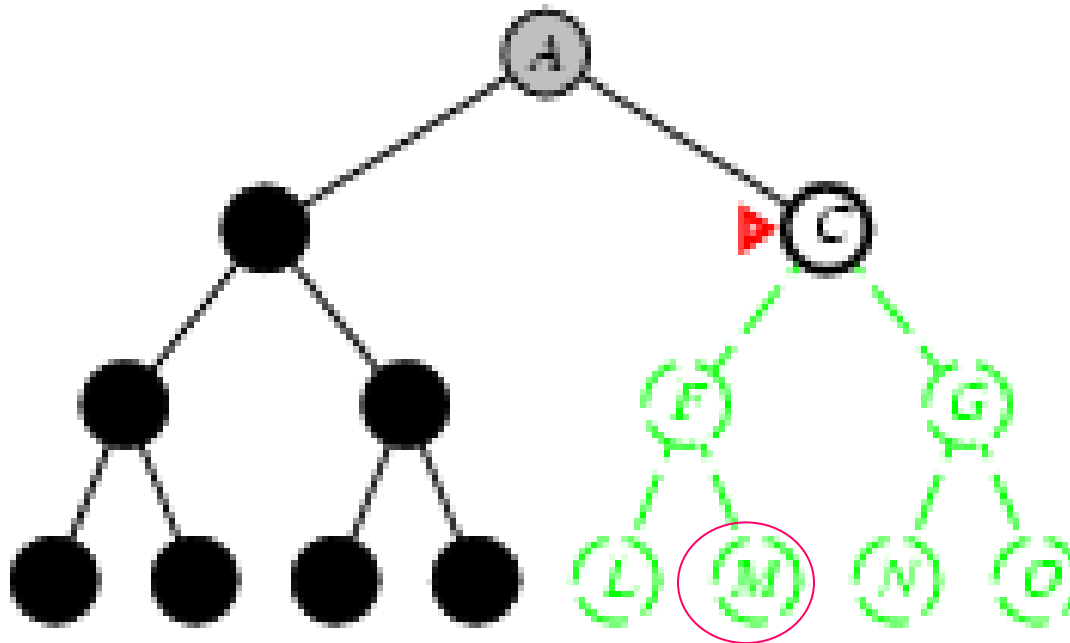


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: C

Expanded: A, B, D, H, I, E, J, K

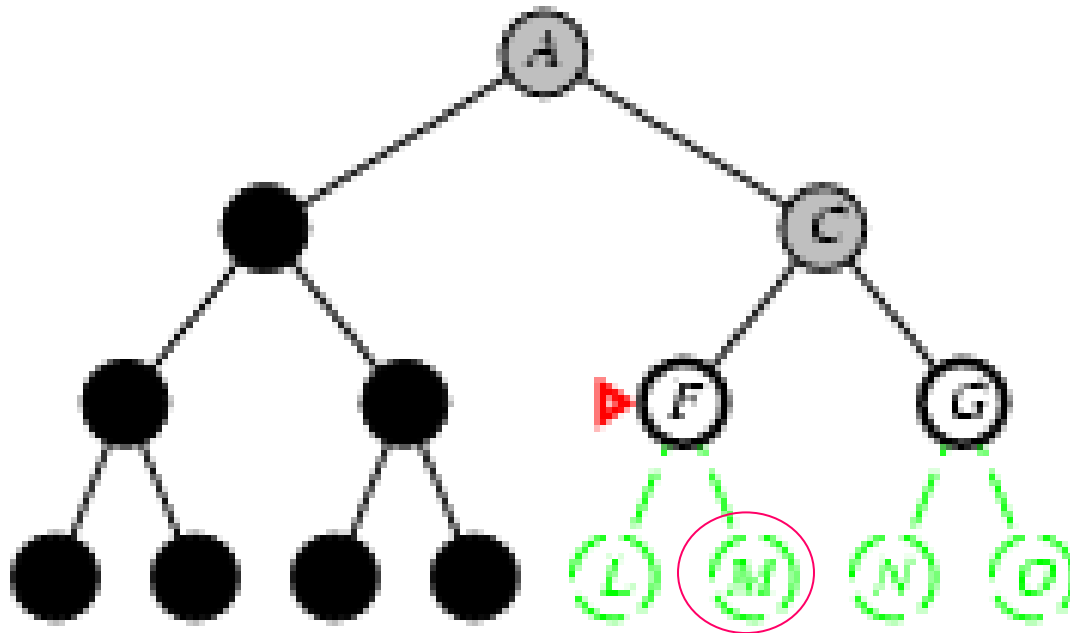


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: F, G

Expanded: A, B, D, H, I, E, J, K, C

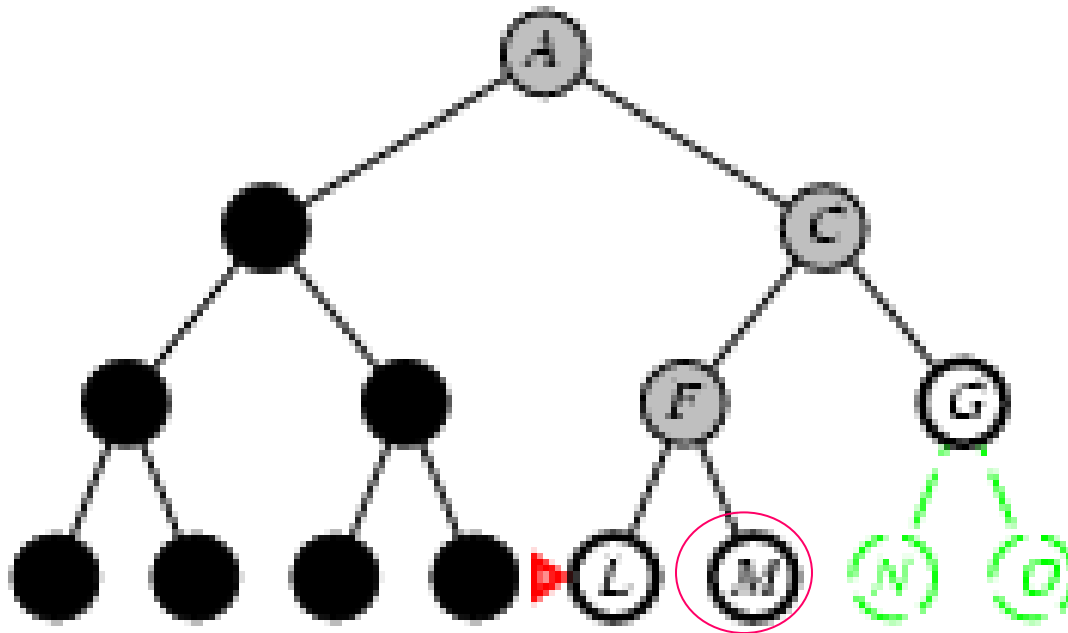


# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert successors at the front of the fringe

Fringe: L, M, G

Expanded: A, B, D, H, I, E, J, K, C, F

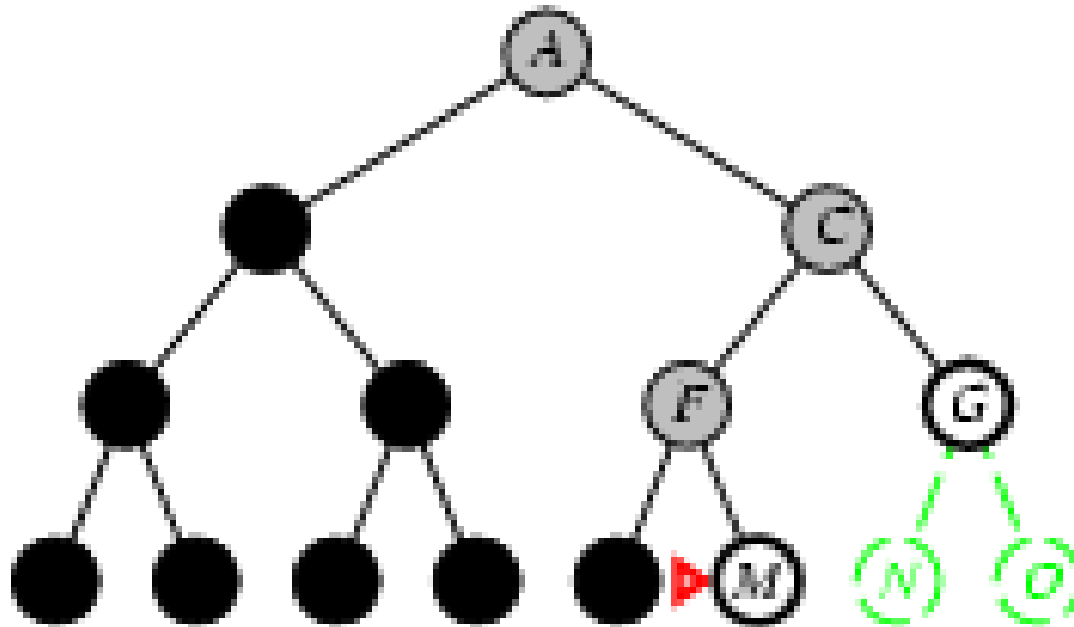


# Depth-First Search (DFS)

- **Expands the deepest unexpanded node**
- **Implementation: insert successors at the front of the fringe**

## Fringe: M, G

**Expanded: A, B, D, H, I, E, J, K, C, F, L**



# Depth-First Search (DFS)

- Expands the deepest unexpanded node
- Implementation: insert children at the front of the fringe

Fringe: M, G

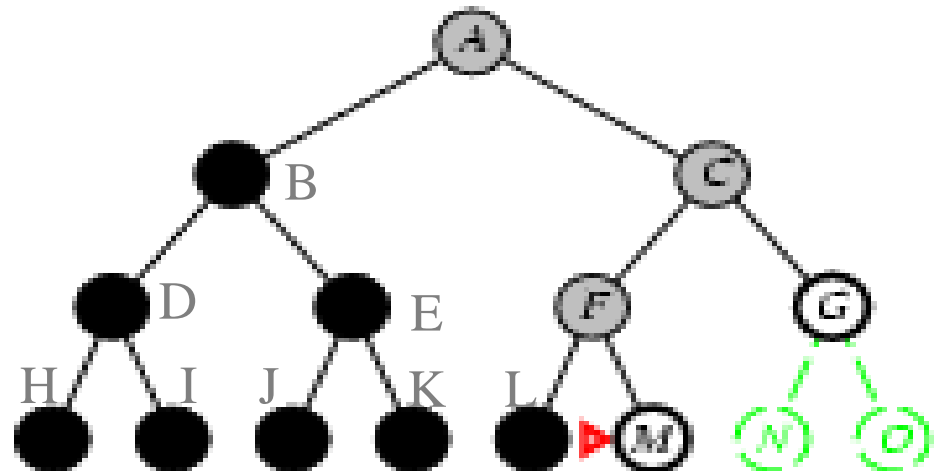
Expanded: A, B, D, H, I, E, J, K, C, F, L

M is a goal node => stop

order of expansion: ABDHIEJKCFLM

solution node found: M

solution path found: ACFM





# DFS for Romania Example

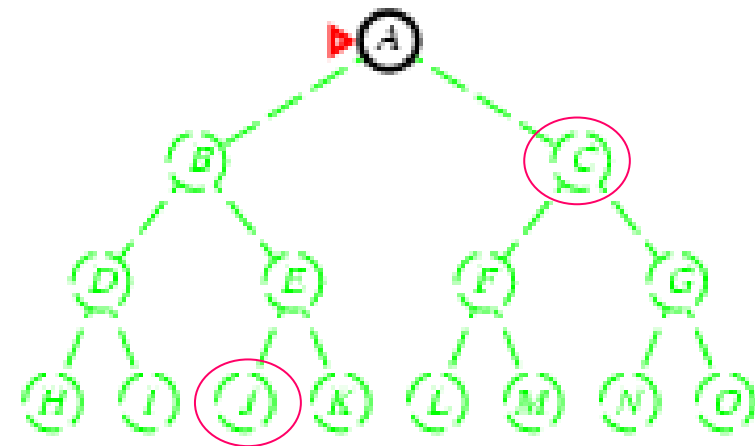
- **DFS can perform infinite cyclic explorations => needs a finite, non-cyclic search space or a repeated state checking mechanism**

# Properties of DFS

- Complete?
  - No, fails in infinite-depth spaces (i.e.  $m = \infty$ )
  - Yes, in finite spaces (i.e. if  $m$  is finite)
- Optimal? No
  - May find a solution longer than the optimal (even if step cost=1)
- Time?  $1+b+b^2+b^3+b^4 + \dots + b^m = O(b^m)$  similar to BFS
  - higher than BFS as  $m \gg d$  ( $m$ =max depth,  $d$ =least cost solution depth)
- Space?  $O(bm)$ , linear; excellent; Why?

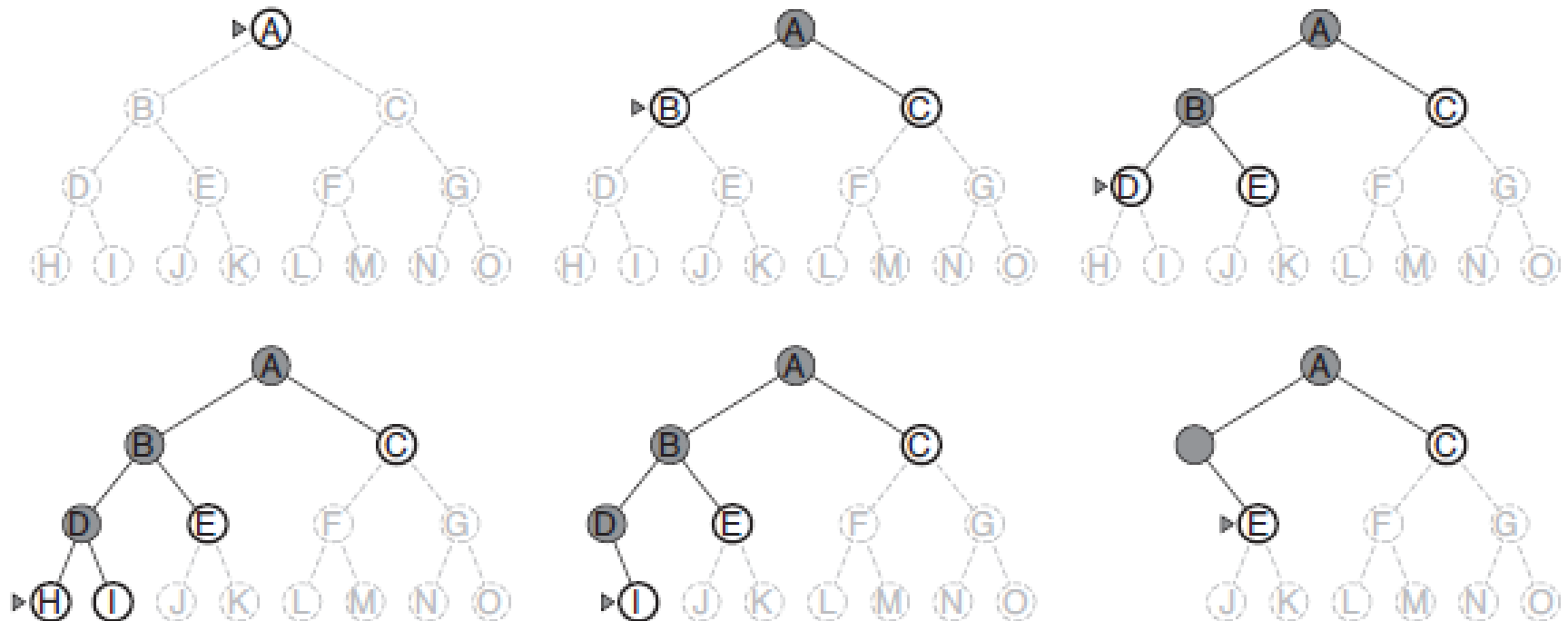
Can get stuck going down a very long (or even infinite) path e.g. suppose the left subtree was unbounded and did not contain a goal node – DFS would never terminate

Suppose J and C were goal nodes and C is a better solution than J; DFS will find J – not optimal



# Space Complexity of DFS

- Space?  $O(bm)$ , linear; excellent – why?
- Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored



# Depth-Limited Search

- Depth-limited search = DFS with depth limit  $l$ 
  - i.e. it imposes a cutoff on the maximum depth
- Properties – similar to DFS:
  - Complete? Yes (as the search depth is always finite)
  - Optimal? No
  - Time?  $1+b^2+b^3+b^4 + \dots + b^l = O(b^l)$
  - Space?  $O(bl)$
- Problem - how to select a good limit  $l$ ? Based on domain knowledge.
  - There are 20 cities on the map of Romania => if there is a solution it must be of length 19 at most =>  $l=19$
  - We can even see that any city can be reached from any other city in at most 9 steps; this is called *diameter of a state space* =>  $l=9$
  - However, for most problems the diameter of the state space is not known in advance , i.e. before we have solved the problem

# Iterative Deepening Search (IDS)

- Sidesteps the issue of choosing the best depth limit by trying all possible depth limits in turn (0, 1, 2, etc.) and applying DFS
- for depth=0 to  $\infty$  do *depth-limited search(depth)*:

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or fail-  
ure  
  inputs: problem, a problem  
  for depth  $\leftarrow$  0 to  $\infty$  do  
    result  $\leftarrow$  DEPTH-LIMITED-SEARCH(problem, depth)  
    if result  $\neq$  cutoff then return result
```

- Tries to combine the benefits of DFS (low memory) and BFS (completeness if  $b$  is finite and optimality if step costs are the same) by **repeated DFS searches with increased depth**
- The nodes close to the root will be expanded multiple times but the number of these nodes is small – the majority of the nodes are close to the leaves, not the root  $\Rightarrow$  the overhead is small

# IDS, $l=0$

cutoff limit  $l=0$

A tree till level  $l=0$ ; do DFS

Goal found?

Yes  $\rightarrow$  stop

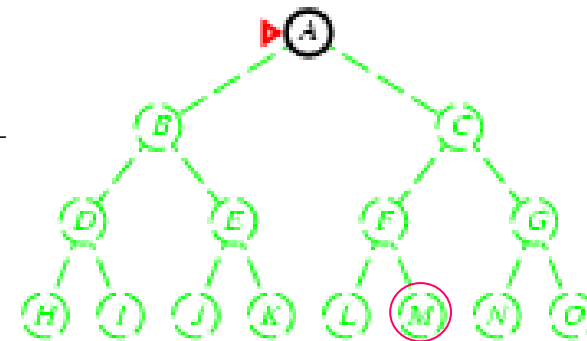
No  $\rightarrow$  increment  $l$  and repeat

Limit = 0



**Expanded: A**

The tree we are searching –  
M is the goal node



# IDS, $l=1$

cutoff limit  $l=1$

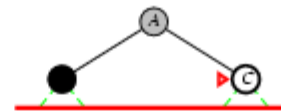
A tree till level  $l=1$ ; do DFS

Goal found?

Yes  $\rightarrow$  stop

No  $\rightarrow$  increment  $l$  and repeat

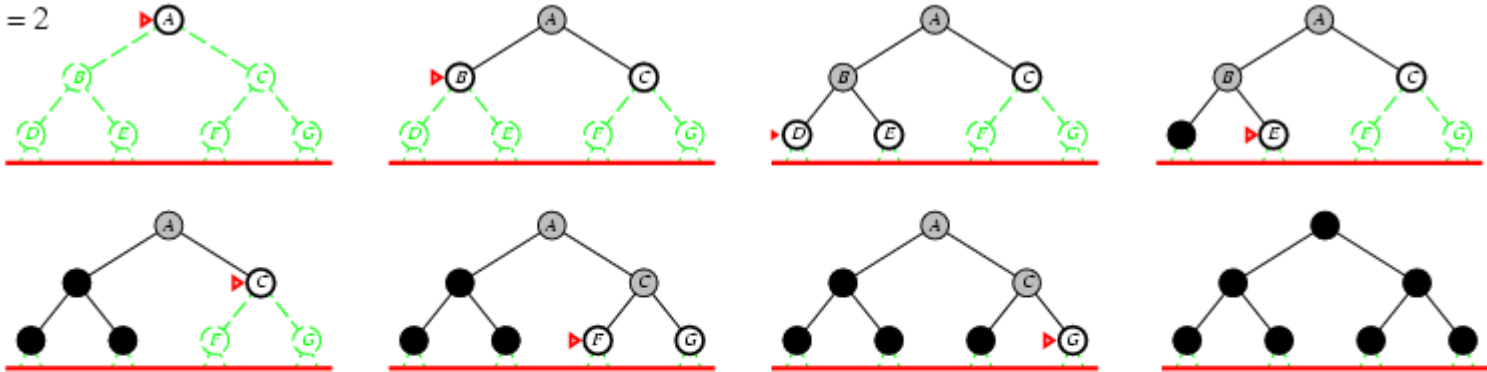
Limit = 1



**Expanded: A, B, C**

# IDS, l=2

Limit = 2

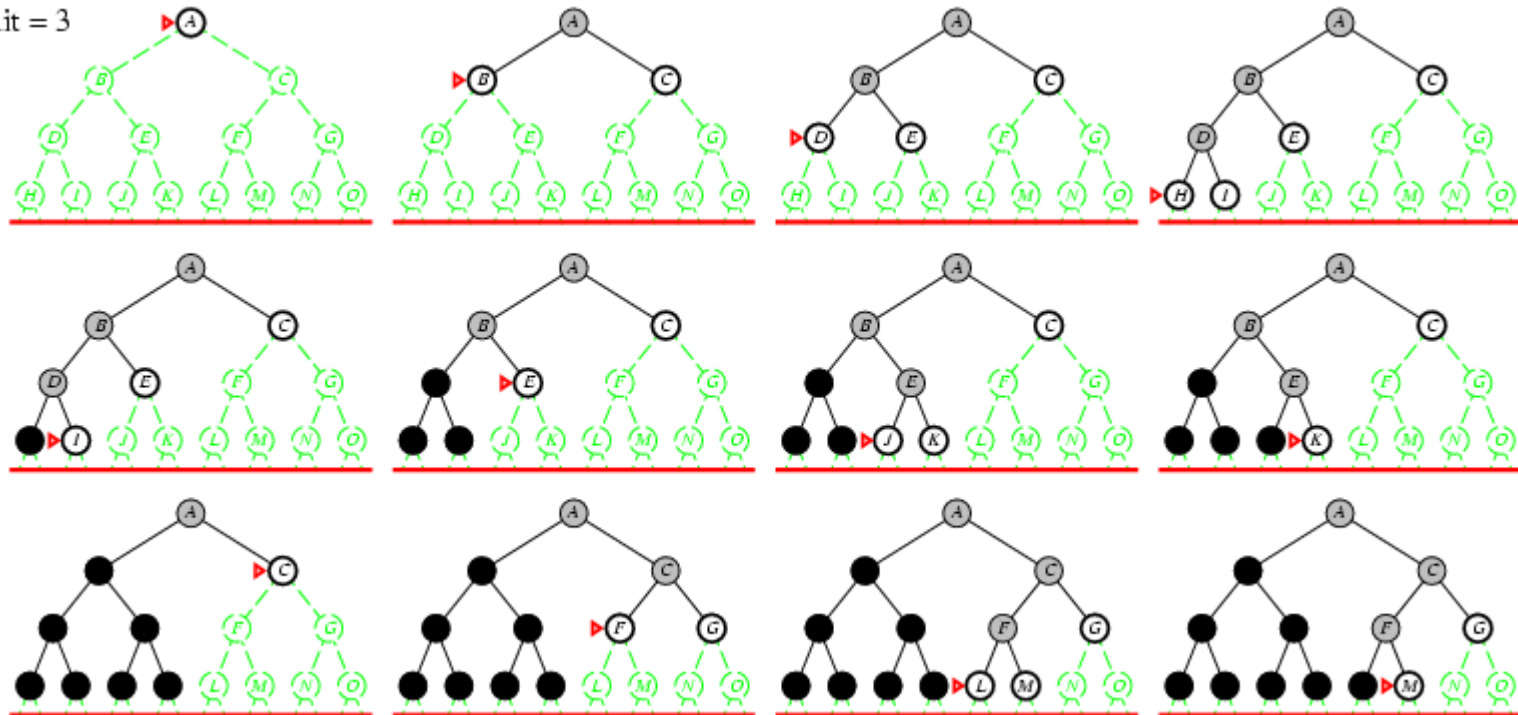


**Expanded: A, B, D, E, C, F, G**



# IDS, l=3

Limit = 3



**Expanded: A, B, D, H, I, E, J, K, C, F, L, M (goal)**

# IDS – Overhead of Multiple Expansion

- May seem wasteful as many nodes are expanded multiple times
- But for most problems the overhead of this multiple expansion is small!
  - BFS: # expansions:  $1+b+b^2+\dots+b^d$ , i.e. 111 111 for  $b=10$ ,  $d=5$
  - IDS: #expansions:  $(d+1)1+(d)b+(d-1)b^2+\dots+3b^{d-2}+2b^{d-1}+1b^d$ , i.e. 123 456 for  $b=10$ ,  $d=5 \Rightarrow$  only 11% more
- IDS is the preferred method when there is a large search space and the depth of the solution is unknown

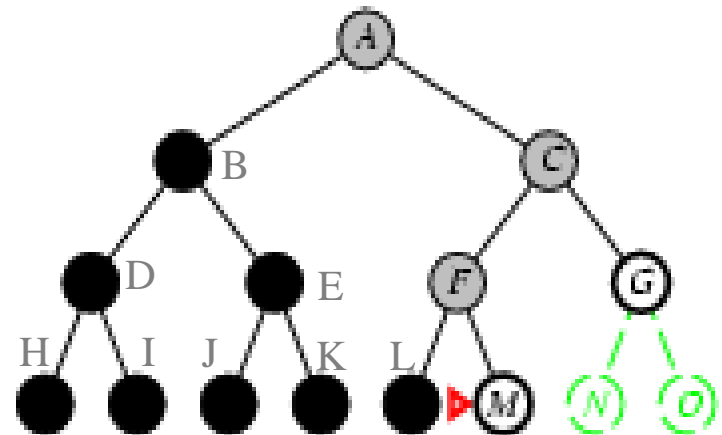
Expanded:

A

A, B, C

A, B, D, E, C, F, G

A, B, D, H, I, E, J, K, C, F, L, M (goal)



# Properties of IDS

$b$  – branching factor

$d$  - depth of least cost solution

$m$  – max depth

- Combines the benefits of DFS and BFS

- Complete? As BFS:

Yes

[DFS: yes, if  $m$  is finite; no otherwise]\*

- Optimal? As BFS:

No in general; Yes if step cost=1

[DFS: not optimal, even if step cost=1] \*

- Time? As BFS:

$$(d+1)b^0 + db^1 + (d-1)b^2 + \dots + b^d = O(b^d) \quad [\text{DFS: } O(b^m)] *$$

- Space? As DFS:

$O(bd)$ , linear

- Where are the **improvements of IDS in comparison to DFS?**

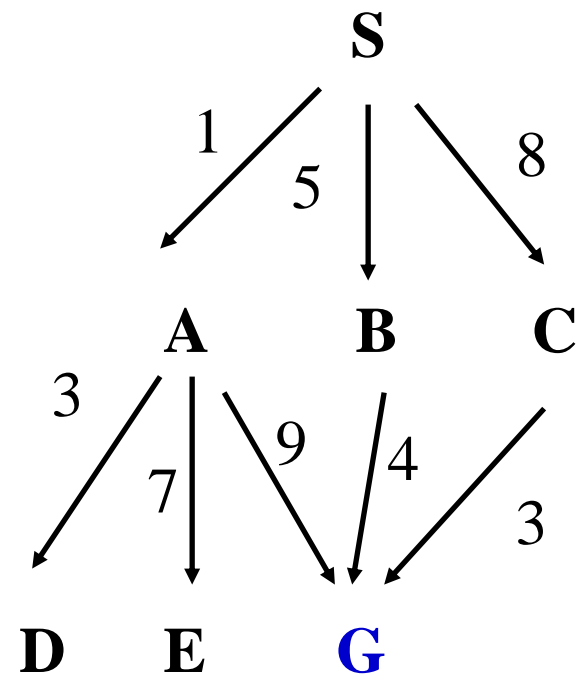
- in completeness, optimality and time (shown with \*)

- Can be modified to explore uniform-cost tree

# Summary

## Example

- **G is the goal.**  
Give the list of expanded nodes and the solution path.



- **DFS?**
  - SADEG, solution found: SAG (non optimal)
- **BFS?**
  - SABCDEG, solution found: SAG (non-optimal solution; will find optimal solution only if step cost=1)
- **UCS?**
  - **Resolving ties: Among nodes with the same priority, expand the node that was added first to the fringe**
  - SADBCEG, solution found: SBG (optimal)  
Among nodes with the same priority (E and C, g=8), the first added to the fringe (C) was chosen to be expanded first
- **IDS (l=2)**
  - S SABC SADEG, solution found: SAG (non-optimal solution; will find optimal solution only if step cost=1)

# Real-World Path Finding Problems

- **Planning trips from A to B - Google maps, planning public transport trips**
  - **Example - web sites for planning air travel:**
    - **Initial and goal states:** specified by the user
    - **Actions:** take any flight from the initial location, leaving after the current time with enough time for within-airport transfer
    - **Path cost:** monetary cost, flight time, waiting time, time of the day (too early/too late), frequent-flyer millage awards, customs and immigration procedures
- **VLSI layout - position millions of components and connections on a very small chip (to minimize area and circuits delays)**
- **Assembling objects by a robot - find the correct order for the parts; wrong order will require undoing some of the work already done**
- **Protein design - find a sequence of amino acids that will fold into a 3-dimensional protein with useful properties (i.e. curing a disease)**

# Informed (Heuristic) Search Strategies

# Informed vs Uninformed Search

- A search strategy defines the order of node expansion
- *Uninformed search* strategies do not use problem specific knowledge beyond the definition of the problem, i.e. they **do not use heuristic knowledge**. They:
  - expand nodes systematically, e.g. BFS: level by level, from left to right
  - know if a given node is a goal or non-goal node
  - cannot compare 2 non-goal nodes – they do not know if one non-goal node is better than another one based on heuristic knowledge
    - UCS does this but it is not based on heuristic knowledge but path cost so far, so it is classified as uninformed
  - are typically inefficient in finding the solution (time and space)
- *Informed search* strategies use problem-specific **heuristic** knowledge to select the order of node expansion. They:
  - can compare non-goal nodes – they know if one non-goal node is better than another one
  - are typically more efficient

# Best-first Search

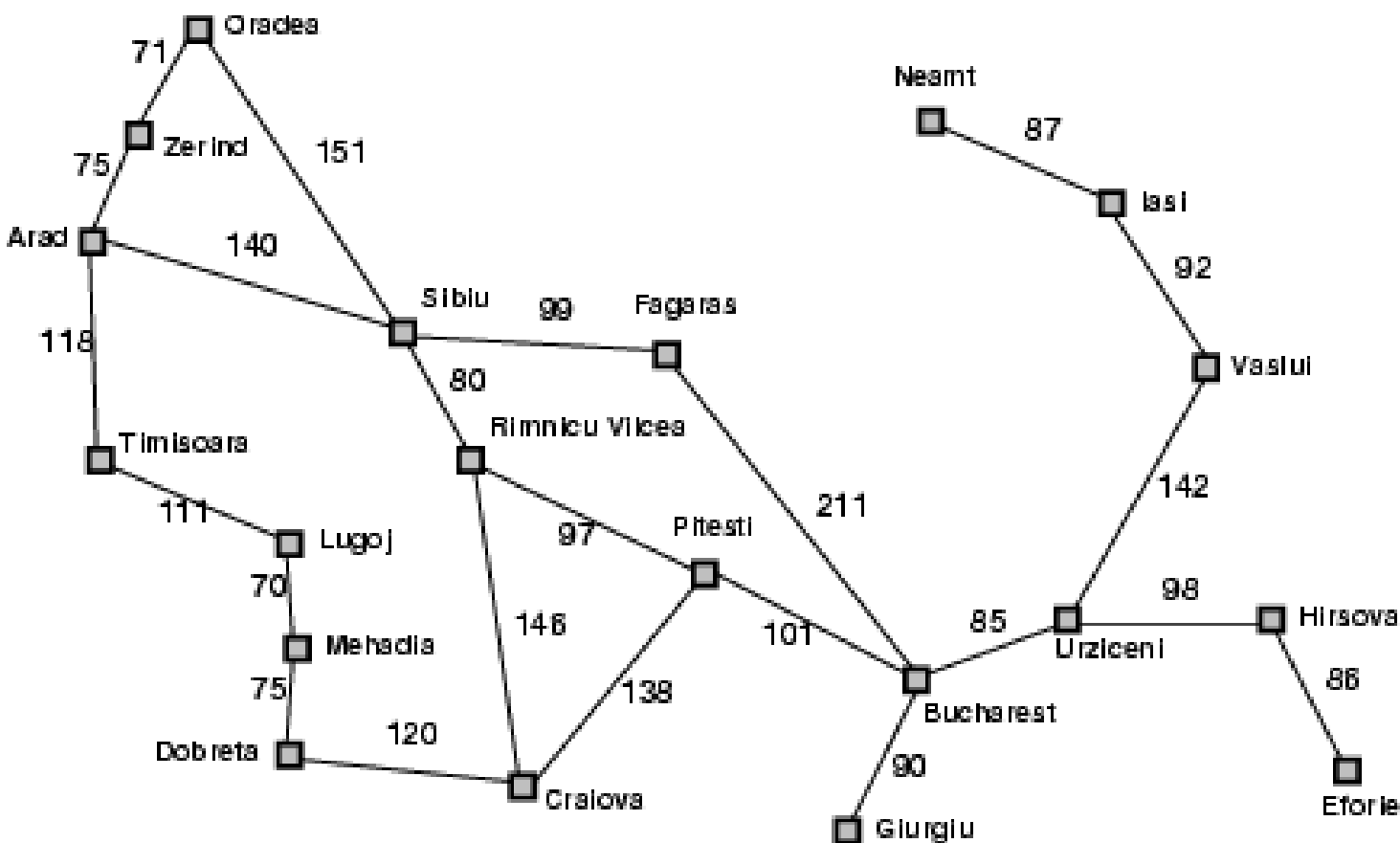
- How can informed strategies compare non-goal nodes?
- By using domain specific knowledge to devise an *evaluation function* which estimates how good each node is
- The evaluation function assigns a *value* to each node
- At each step, the best node is expanded (the one with the best value)
- This is called *best-first search*
  - Note that we don't *really* know which is the best node as we use an estimate based on the evaluation function. So best-first search expands the node that *appears* to be the best.
- Fringe: insert children in decreasing order of desirability
- We will study 2 **best-first** search algorithms: *greedy* and  $A^*$



# Greedy Search

# Romania with Step Coast and Straight-line Distance to Bucharest

Problem-specific knowledge



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

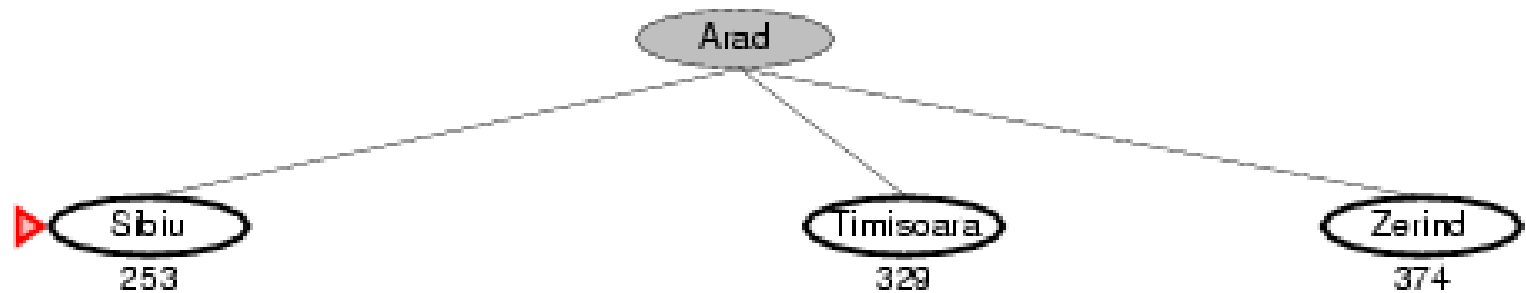
# Greedy Search (GS)

- It uses the so called  $h$  value as an evaluation function ( $h$  from *heuristic*)
- The  $h(n)$  for a node  $n$  is the estimated cost from  $n$  to a goal node
- E.g. for the Romania example we can use  $h(n) = SLD(n, Bucharest)$  = straight-line distance from  $n$  to Bucharest
- Note that the  $h$  value of a goal node is 0, i.e.  $h(goal) = 0$
- GS expands the node with the smallest  $h$ , i.e.
  - The node that appears to be closest to a goal
- Thus: GS minimizes  $h$ , the estimated cost to a goal

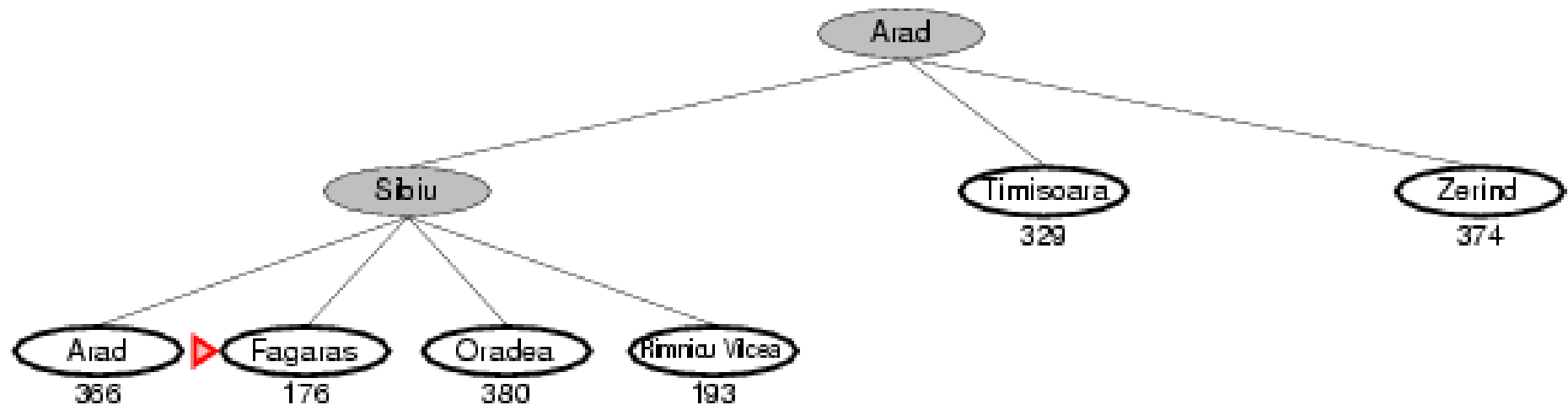
# Greedy Search for Romania Example



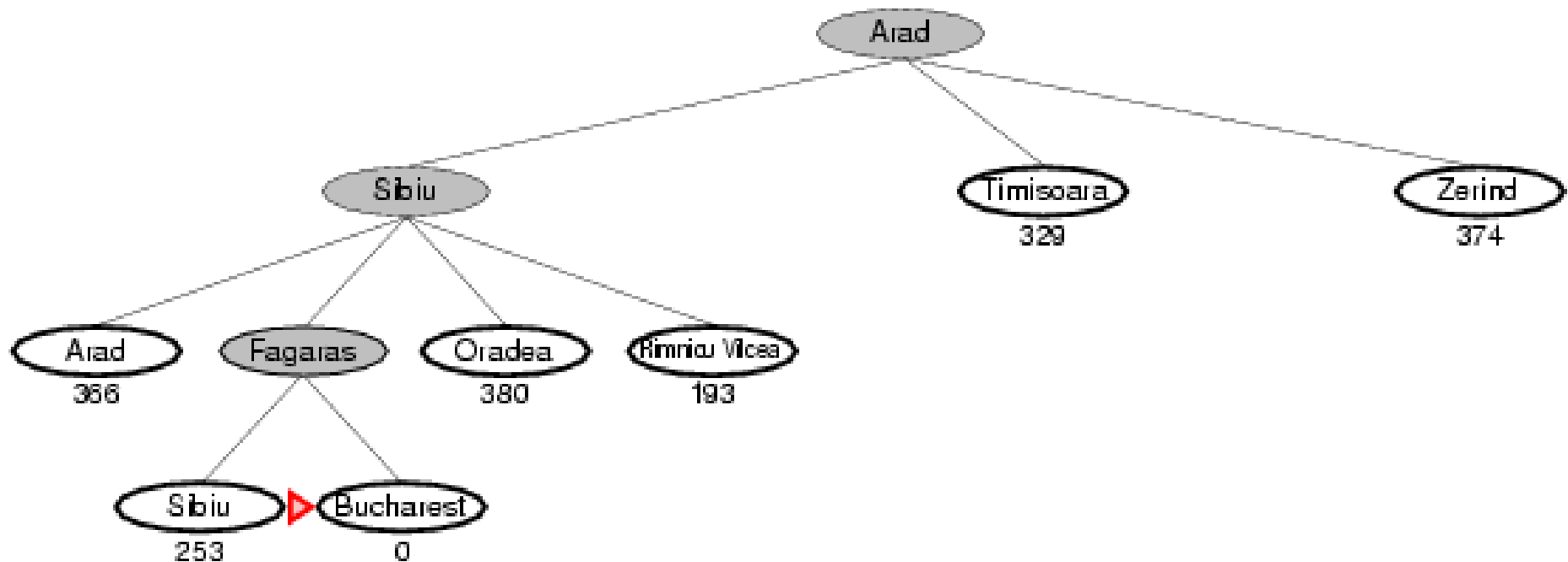
# Greedy Search for Romania Example



# Greedy Search for Romania Example



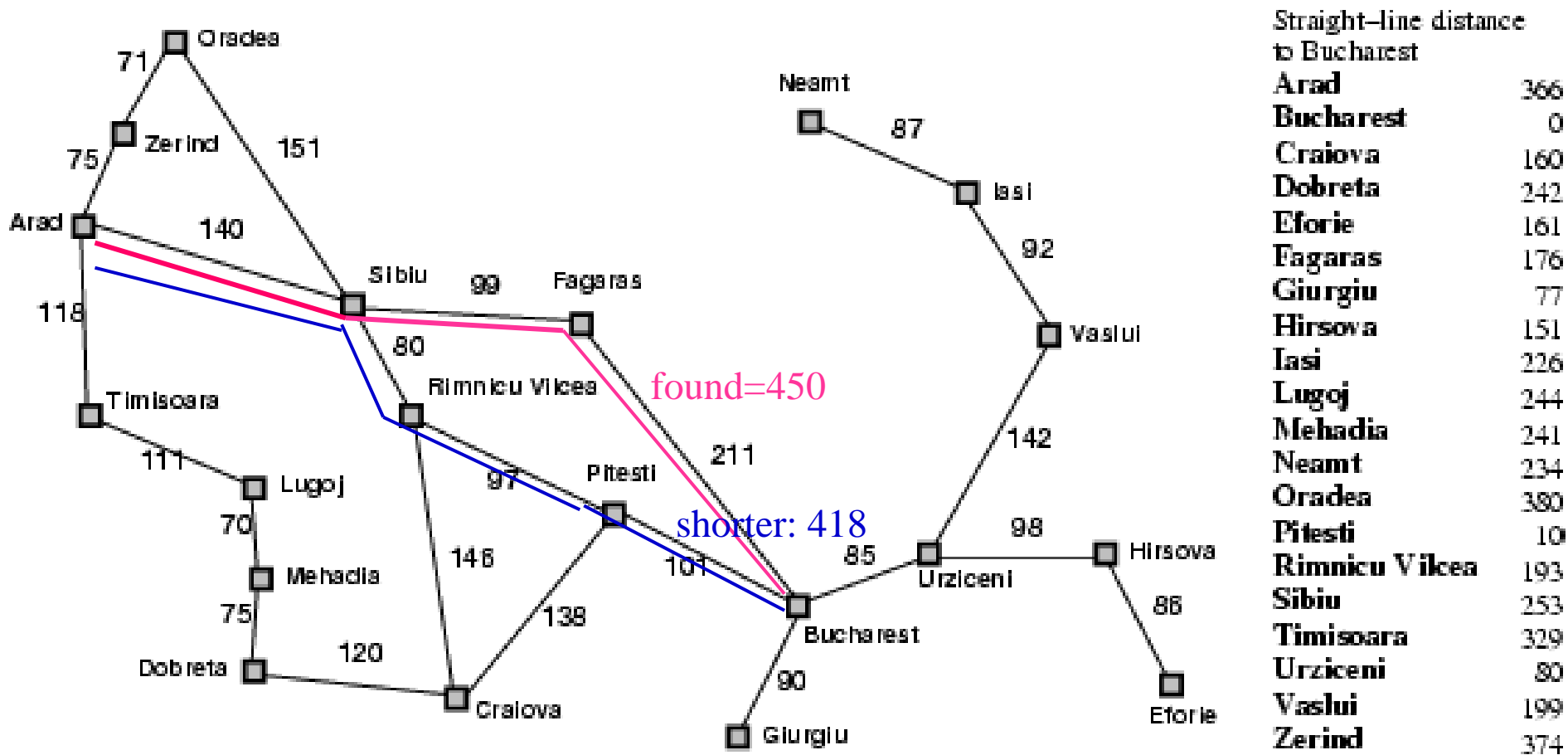
# Greedy Search for Romania Example



**Solution path: Arad-Sibiu-Fagaras-Bucharest, cost=450**

**Optimal =?**

# Greedy Search for Romania Example



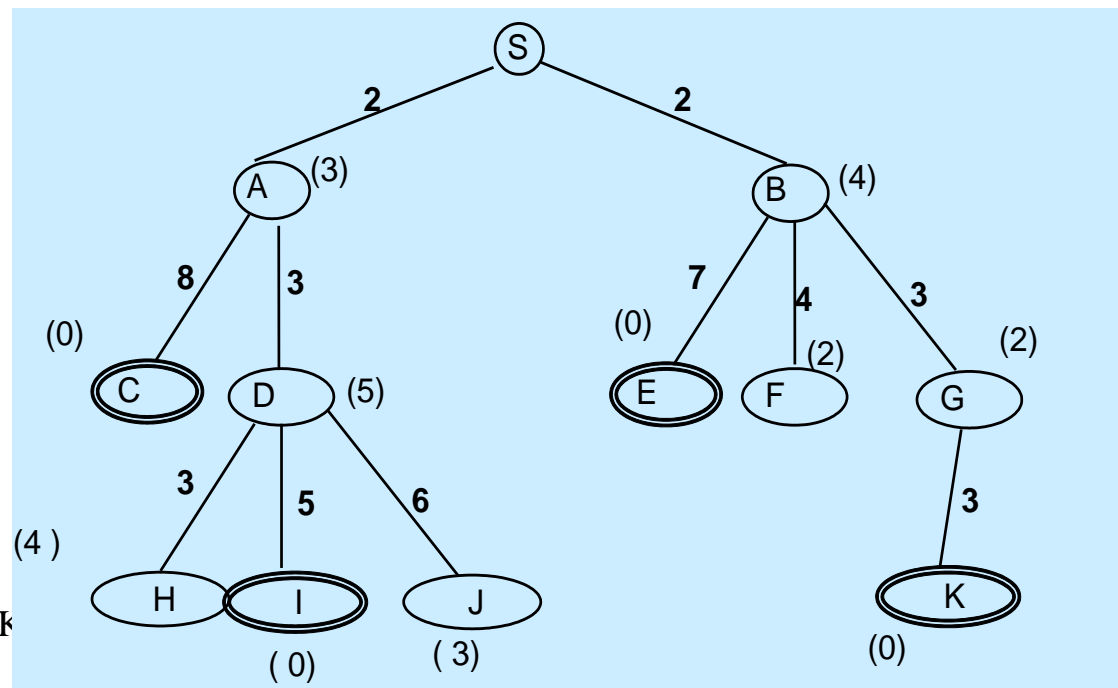
**Solution path: Arad-Sibiu-Fagaras-Bucharest, cost=450**

**Optimal =?**



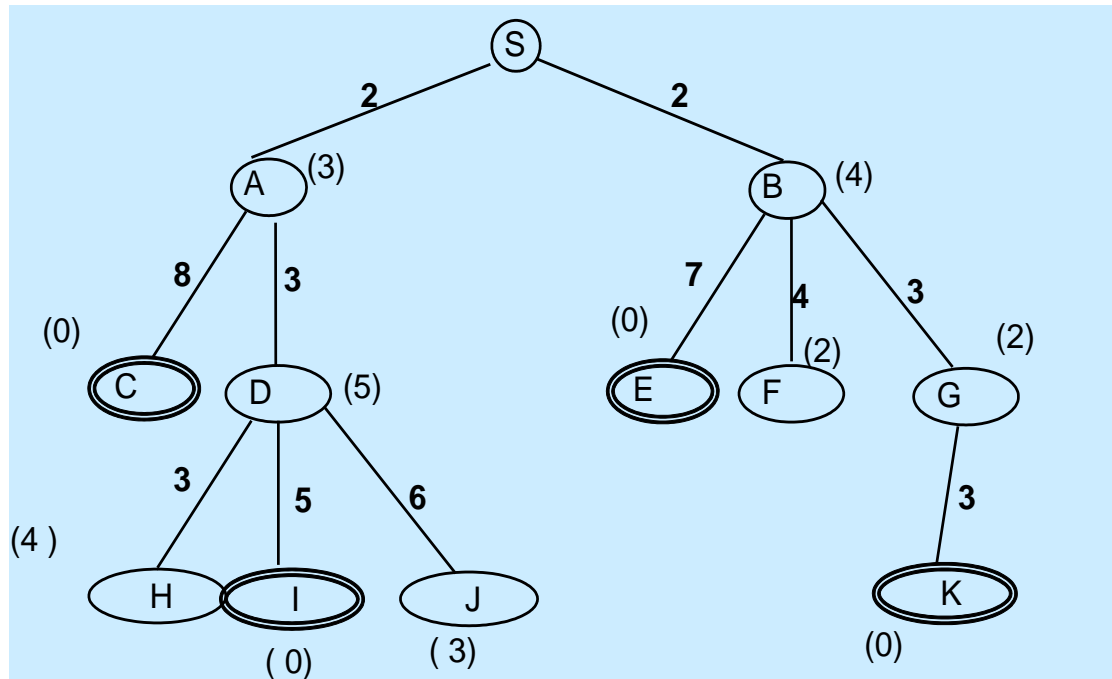
# Greedy Search – Another Example

- **Given:**
  - **Goal nodes: C, I, E and K**
  - **Step path cost: along the links**
  - **h value of each node: in ()**
- **Run GS**
  - **List of expanded nodes =?**
  - **Solution path =? Cost of the solution path=?**



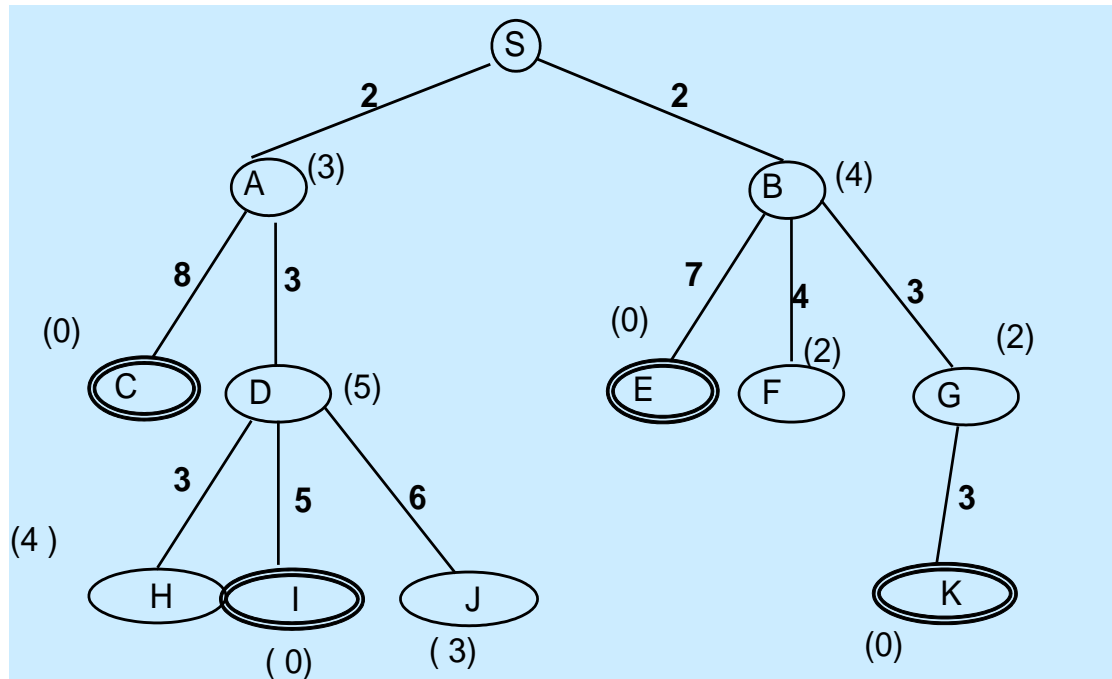
# Solution

- **Fringe:** S
- **Expanded:** nil



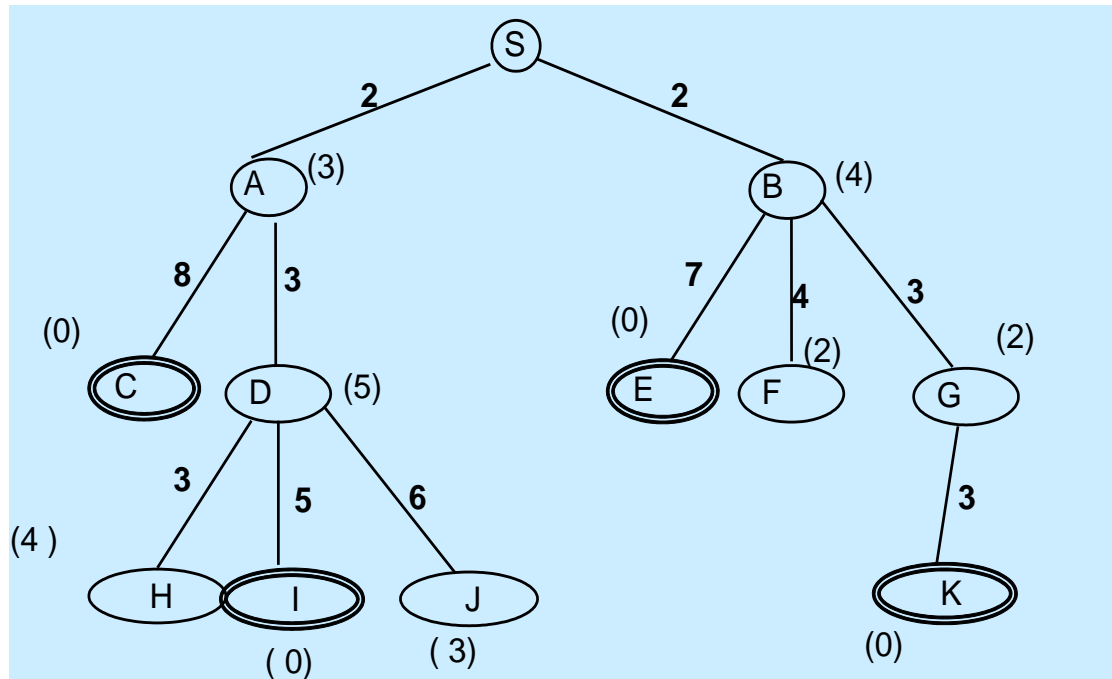
# Solution

- **Fringe:** (A,3), (B,4)
- **Expanded:** S



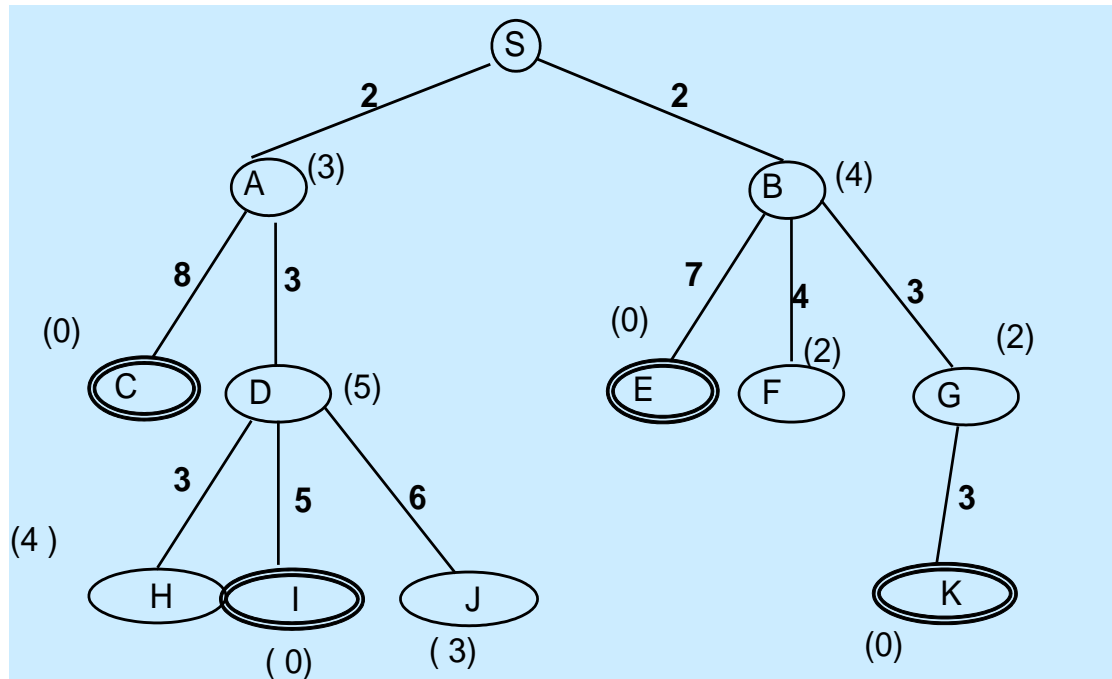
# Solution

- **Fringe:** (C,0), (B,4), (D, 5) //C and D are added and the fringe is ordered
- **Expanded:** S, (A,3)



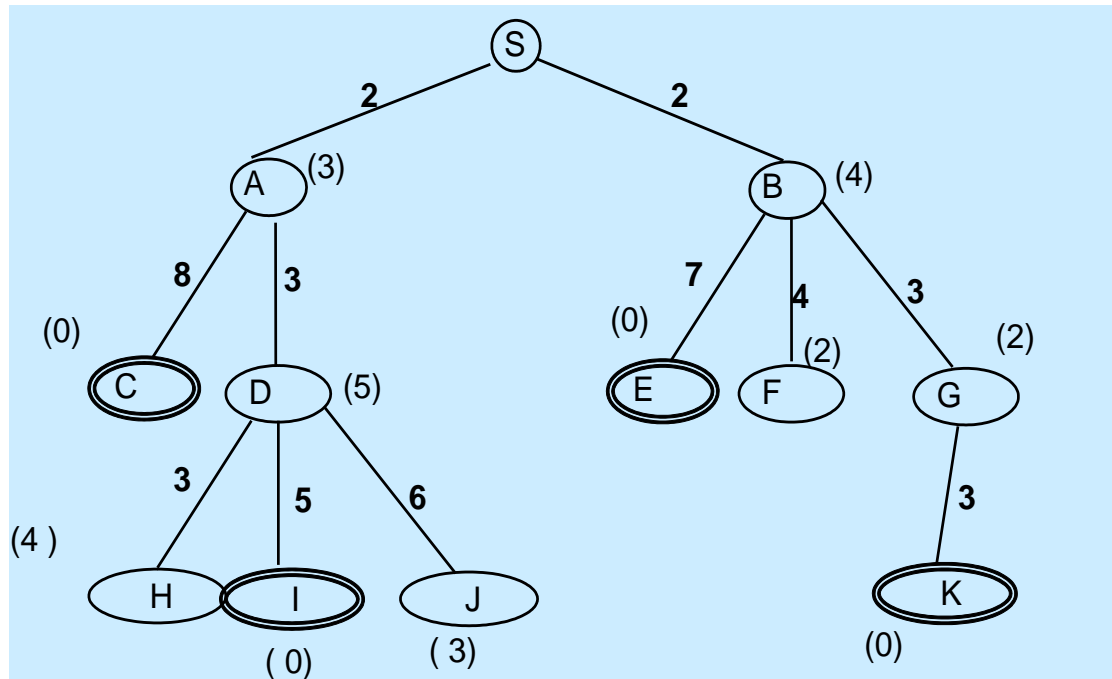
# Solution

- **C is selected for expanding; is it a goal node? Yes ->stop**
- **Expanded: S, A, C** //note that the goal node is also considered to be  
// expanded and is included in the list of expanded
- **Solution path: SAC, path cost:  $2+8=10$**



# Solution

- Is SAC the optimal solution (i.e. shortest path to a goal)?



# Properties of Greedy Search

- **Complete?** As DFS
  - Yes, in finite space (if  $m$  is finite)
  - No – fails in infinite spaces (and can get stuck in loops, e.g. Iasi->Neamt->Iasi->Neamt)
- **Optimal?** No (e.g. Arad->Sibiu->R.Vincea->Pitesti->Buharest is shorter)
- **Time?**  $O(b^m)$ , but a good heuristic can give dramatic improvement
- **Space?**  $O(b^m)$  , keeps every node in memory

# Question

- $g(n)$  = cost so far
- $h(n)$  = estimated cost to the goal
- Greedy search uses  $h(n)$

Suppose that we run a greedy search algorithm with  $h(n) = g(n)$ . This greedy search will be equivalent to what kind of uninformed search?

In other words, which algorithm uses  $g(n)$ ?

UCS

