CS 461
Artificial Intelligence

Dr. Hashim Yasin

Terminologies

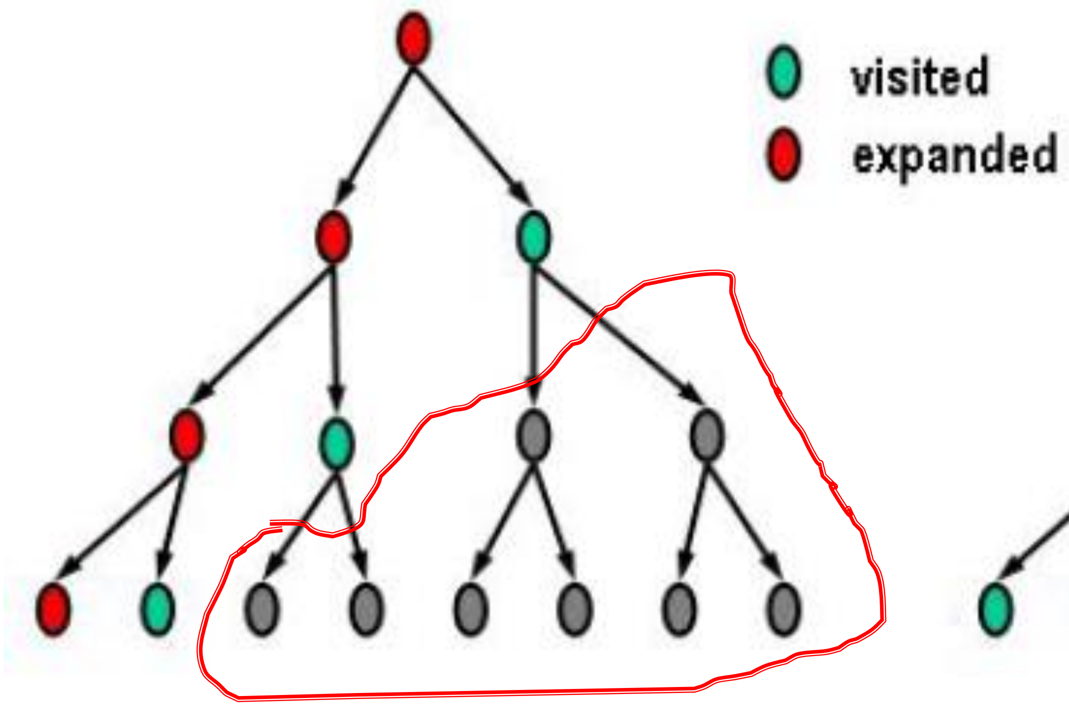
Visited (Open) List:

- The *set of all leaf nodes available for expansion* at any given point is called the open list, (may be referred as **frontier**).
- In general, a state is said to be visited if it has ever shown up in search a node.
- The intuition is that ***we have visited*** them, but ***we have not generated its descendants***.

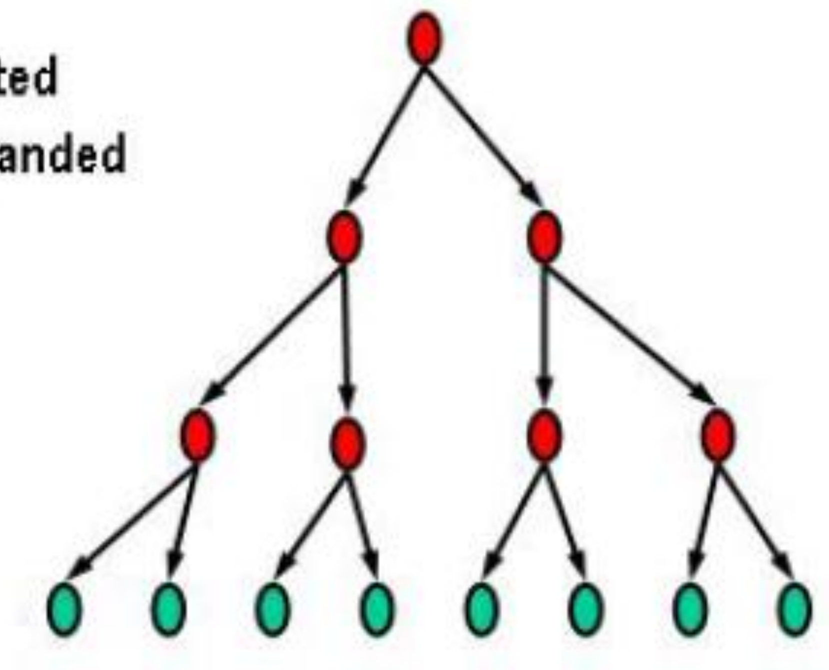
Expanded (Closed, Explored) List:

- *Algorithms that forget their history are doomed to repeat it.*
- The way to avoid exploring redundant paths is to remember where one has been.
- To do this, we design **explored set** (also known as the **closed list**), which remembers every expanded node.

Terminologies



Depth First Search

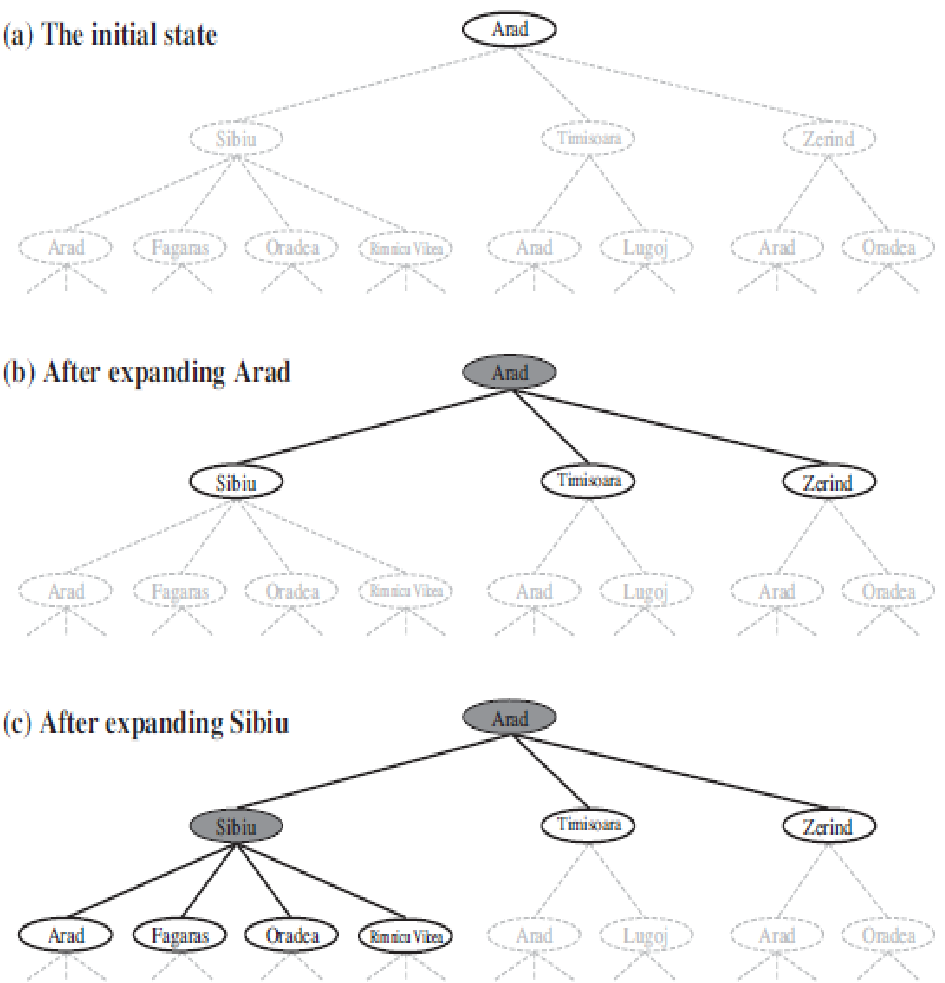


Breadth First Search

Terminologies

Partial search trees for finding a route from Arad to Bucharest.

- ❑ Nodes that have been **visited but not yet expanded** are outlined in **bold**;
- ❑ Nodes that have been **expanded are shaded**;
- ❑ Nodes that have **not been visited** are shown in faint dashed lines.



Uninformed Search

Uninformed Search

- **Uninformed search** also called **blind search**
- The strategies have **no additional information** about the states beyond that provided in the problem definition.
- ***Use the information only provided*** in the problem definition.
    - Breadth-first search
    - Depth-first search
        - Depth-limited search
        - Iterative deepening search
    - Uniform cost search
    - Bidirectional search

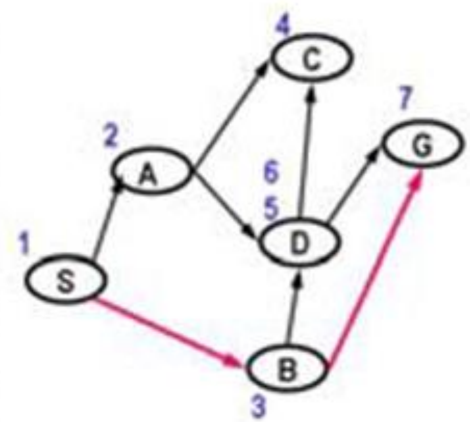
Breadth-First Search

Breadth-First Search

Breadth-First (without Visited list)

Pick first element of Q; Add path extensions to end of Q

	Q
1	(S)
2	(A S) (B S)
3	(B S) (C A S) (D A S)
4	(C A S) (D A S) (D B S) (G B S)*
5	(D A S) (D B S) (G B S)
6	(D B S) (G B S) (C D A S) (G D A S)
7	(G B S) (C D A S) (G D A S) (C D B S) (G D B S)



Added paths in blue

We show the paths in reversed order; the node's state is the first entry.

* We could have stopped here, when the first path to the goal was visited

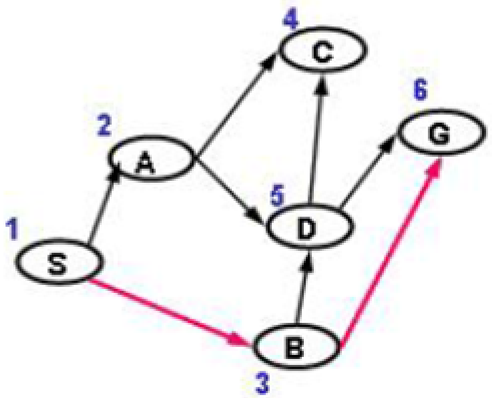
When we need the final path, we have to continue till the goal state is expanded.

Breadth-First Search

Breadth-First

Pick first element of Q; Add path extensions to end of Q

	Q	Visited	Expanded
1	(S)	S	S
2	(A S) (B S)	A,B,S	A,S
3	(B S) (C A S) (D A S)	C,D,B,A,S	B,A,S
4	(C A S) (D A S) (G B S)*	G,C,D,B,A,S	C,B,A,S
5	(D A S) (G B S)	G,C,D,B,A,S	D,C,B,A,S
6	(G B S)	G,C,D,B,A,S	G,D,C,B,A,S



Added paths in blue

We show the paths in reversed order; the node's state is the first entry.

* We could have stopped here, when the first path to the goal was visited

When we need the final path, we have to continue till the goal state is expanded.

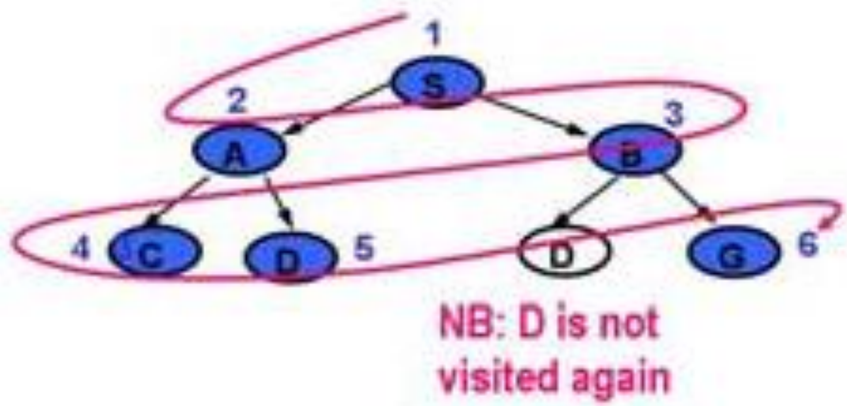
Tree Traversal = S, A, B, C, D, G

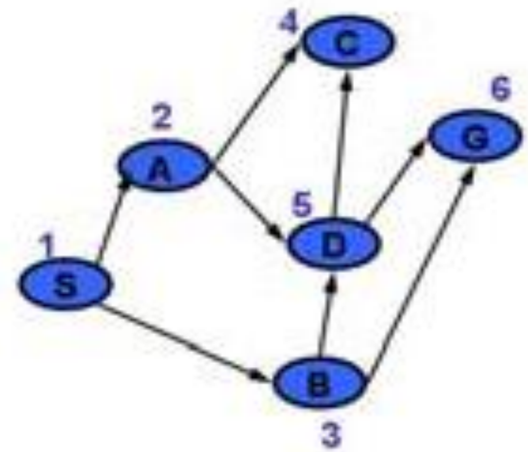
The Final path = S, B, G

Breadth-First Search

Breadth-First

Another (easier?) way to see it





Numbers indicate order pulled off of Q (expanded)

Dark blue fill = Visited & Expanded
Light gray fill = Visited

Breadth-First Search

Completeness: Yes, (if b is finite)

Time complexity: Imagine ***searching a uniform tree*** where every state has b successors.

- The root of the search tree generates b nodes at the first level, each of which generates b more nodes,
- for a total of b^2 at the second level, yielding b^3 nodes at the third level, and so on. Now suppose that the solution is at depth d

$$b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

- If the algorithm were to apply the **goal test** to nodes when selected for **expansion**, rather than when visited,
  - the whole *layer of nodes at depth d would be expanded before the goal was detected* and
  - the time complexity would be $\boldsymbol{O(b^{d+1})}$.

Breadth-First Search

Space complexity: For breadth-first graph search in particular, every node generated remains in memory.

- There will be $\boldsymbol{O(b^{d-1})}$ nodes in the explored set
- $\boldsymbol{O(b^{d})}$ nodes in the frontier,

Optimality: Yes, if the cost = 1 per step

The ***memory requirement is*** *a bigger problem for breadth-first search than the execution time.*

Breadth-First Search

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

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

Breadth-First Search

- The root node is expanded by **first-in-first-out (FIFO), Queue data structure.**
- **Complete**: find the solution eventually
- **Optimal**: if the step cost is 1

Disadvantages:

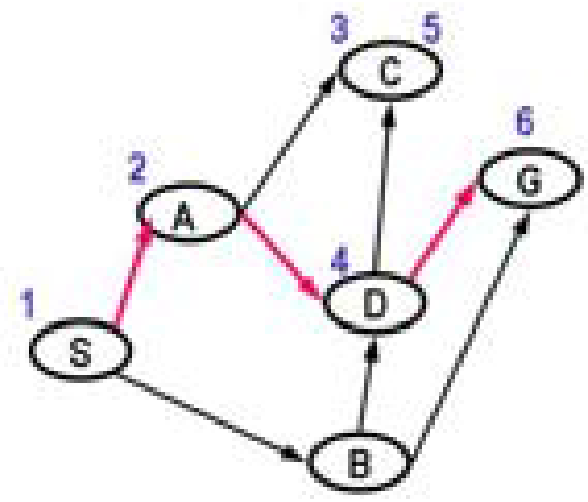
- The branching factor of a node is large,
- The **space complexity and time complexity** are enormous for even small instances (e.g., chess)

Depth-First Search

Depth-First Search (without-visited)

- ❑ Pick first element of Q
- ❑ Add path extensions **in front** to Q

	Q
1	(S)
2	**(A S) (B S)**
3	**(C A S) (D A S)** (B S)
4	(D A S) (B S)
5	**(C D A S) (G D A S)** (B S)
6	**(G D A S)** (B S)

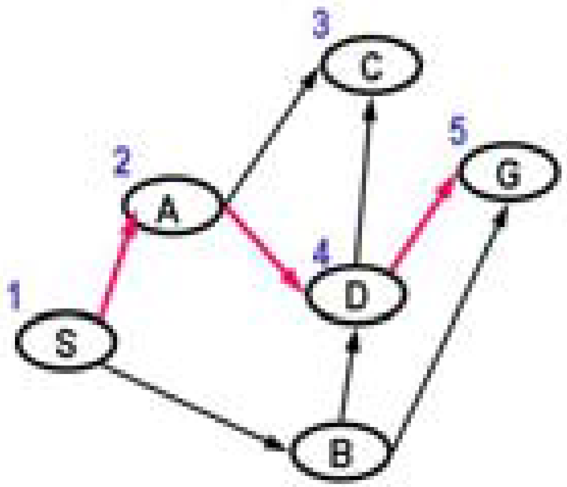


- ❑ **Blue Color represents added paths**
- ❑ Path has been shown in **reversed order**, node state is the first entry.
  - ❑ **Tree Traversal = S, A, C, D, C, G**
  - ❑ **The Final path = S, A, D, G**

Depth-First Search

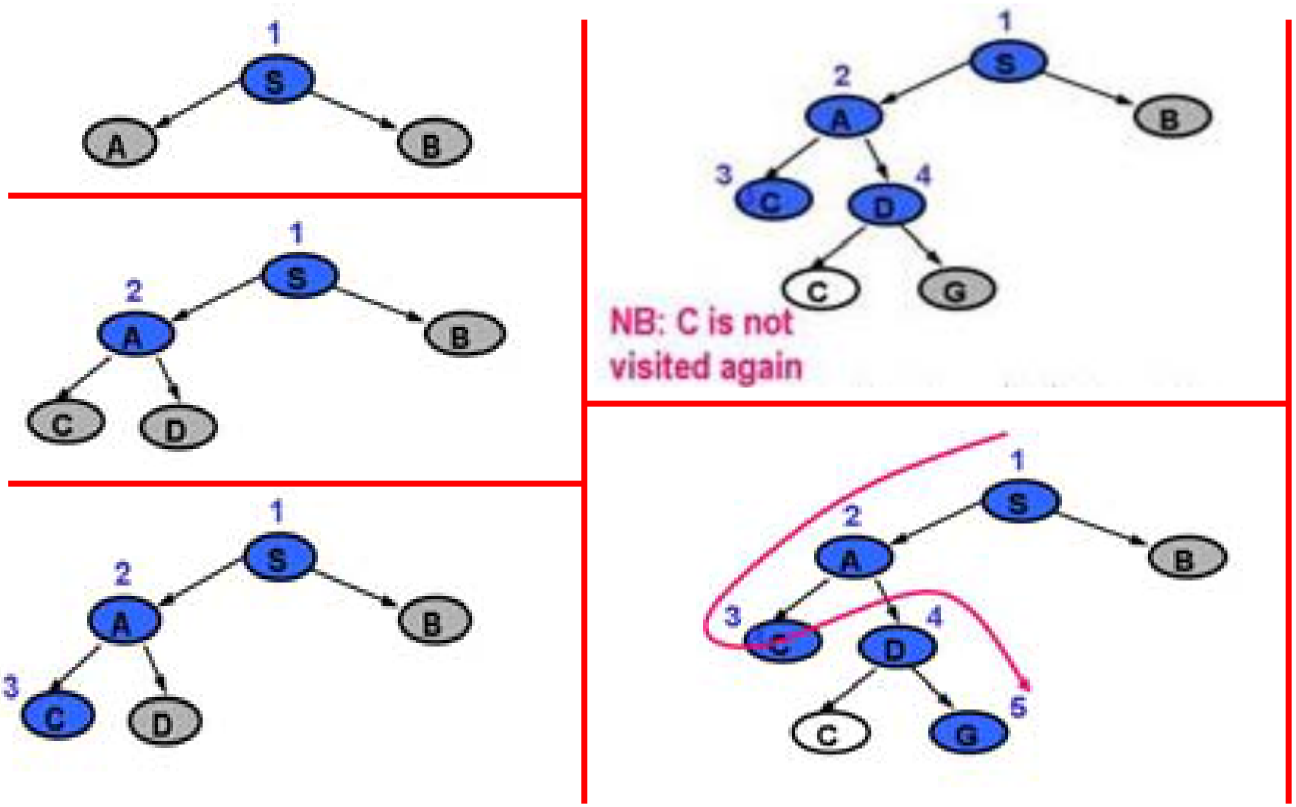
- ❑ Pick first element of Q
- ❑ Add path extensions **in front** to Q

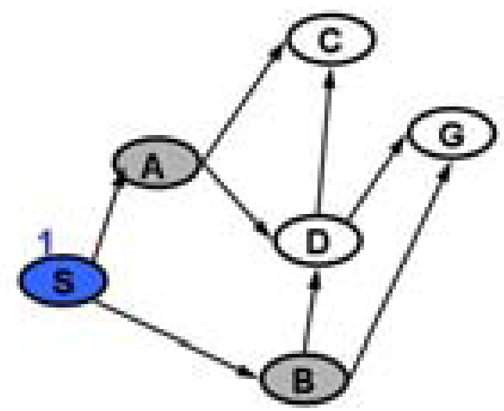
	Q	Visited	Expanded
1	(S)	S	S
2	**(A S) (B S)**	A, B, S	A, S
3	**(C A S) (D A S)** (B S)	C, D, B, A, S	C, A, S
4	(D A S) (B S)	C, D, B, A, S	D, C, A, S
5	**(G D A S)** (B S)	G, C, D, B, A, S	G, D, C, A, S



- ❑ **Blue Color represents added paths**
- ❑ Path has been shown in **reversed order**, node state is the first entry.
- ❑ **Tree Traversal = S, A, C, D, G**
- ❑ **The Final path = S, A, D, G**

Depth-First Search



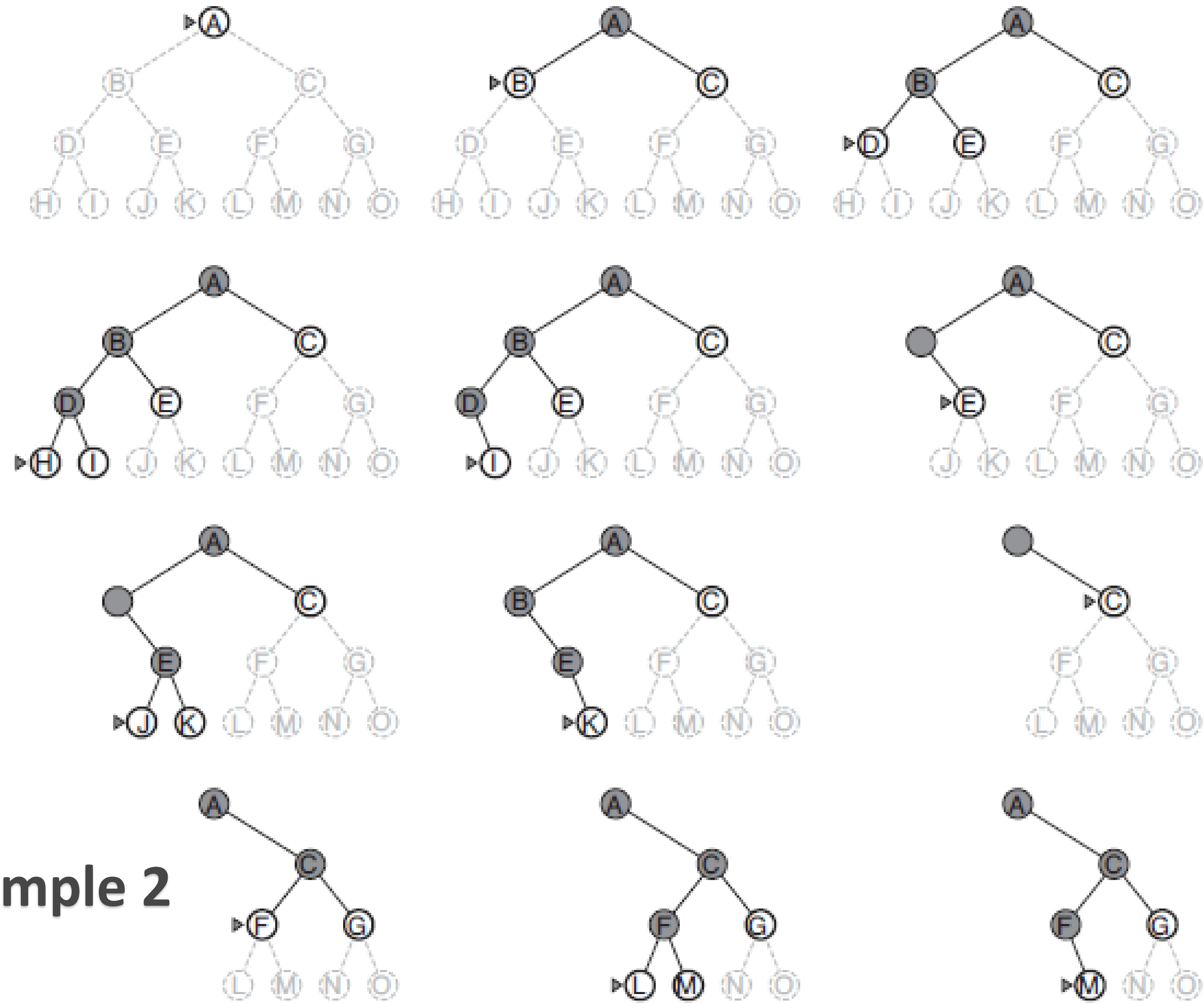


Numbers indicate order pulled off of Q (expanded)

Dark blue fill = Visited and Expanded

Light gray fill = Visited

DFS Example 2



Depth-First Search

- Depth-first search always expands the ***deepest node***
- Implemented with a **last-in-first-out (LIFO)** strategy, also known as a ***stack***.
- As an alternative to the ***Graph-Search-style*** implementation, depth-first search is implemented with a **recursive function** that calls itself on each of its children in turn.

Depth-First Search

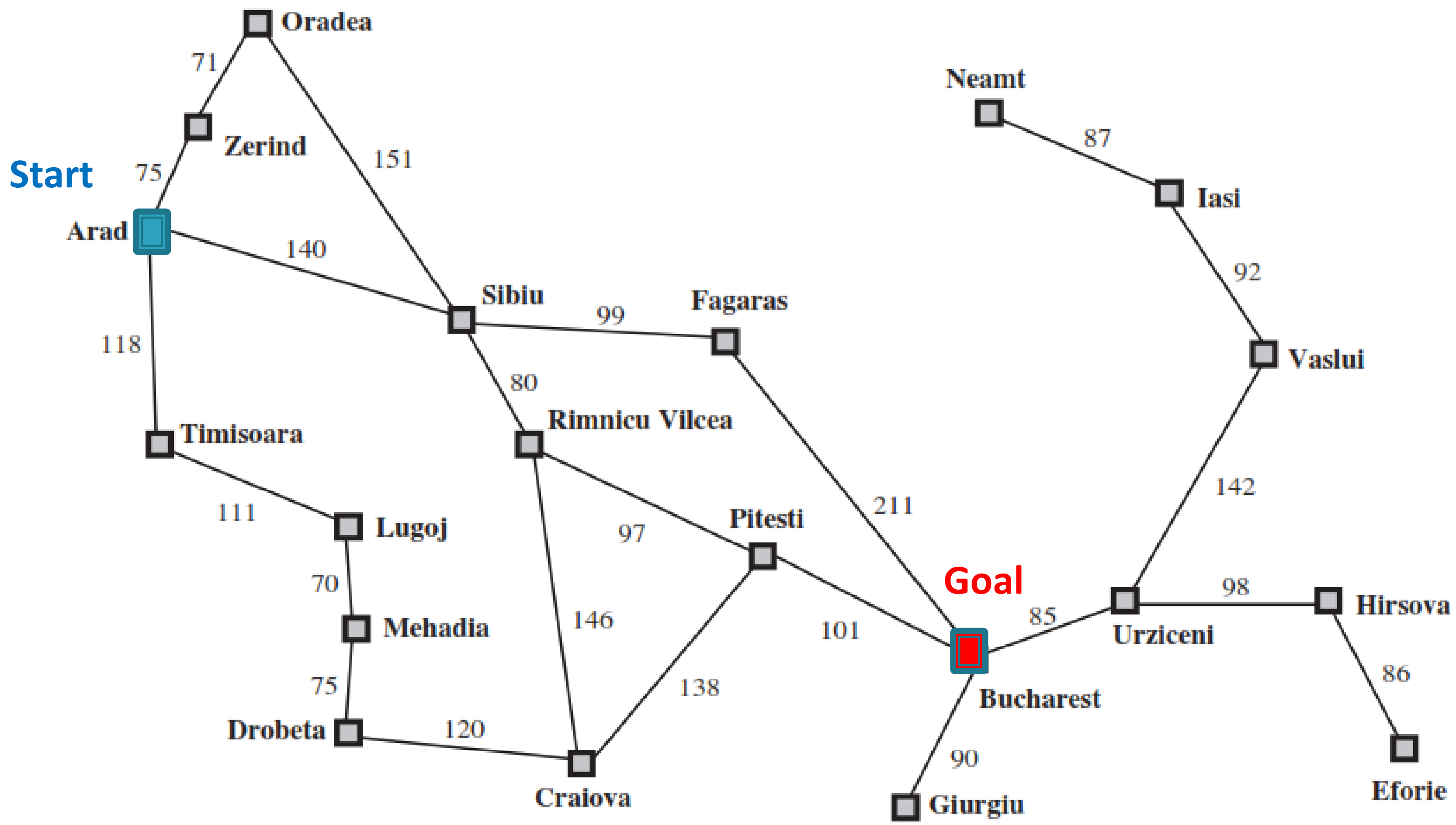
Completeness:

- The *graph-search version*, which *avoids repeated states and redundant paths*, is **complete** in finite state spaces because it will eventually expand every node.
- The *tree-search version*, on the other hand, it is ***not* complete**.

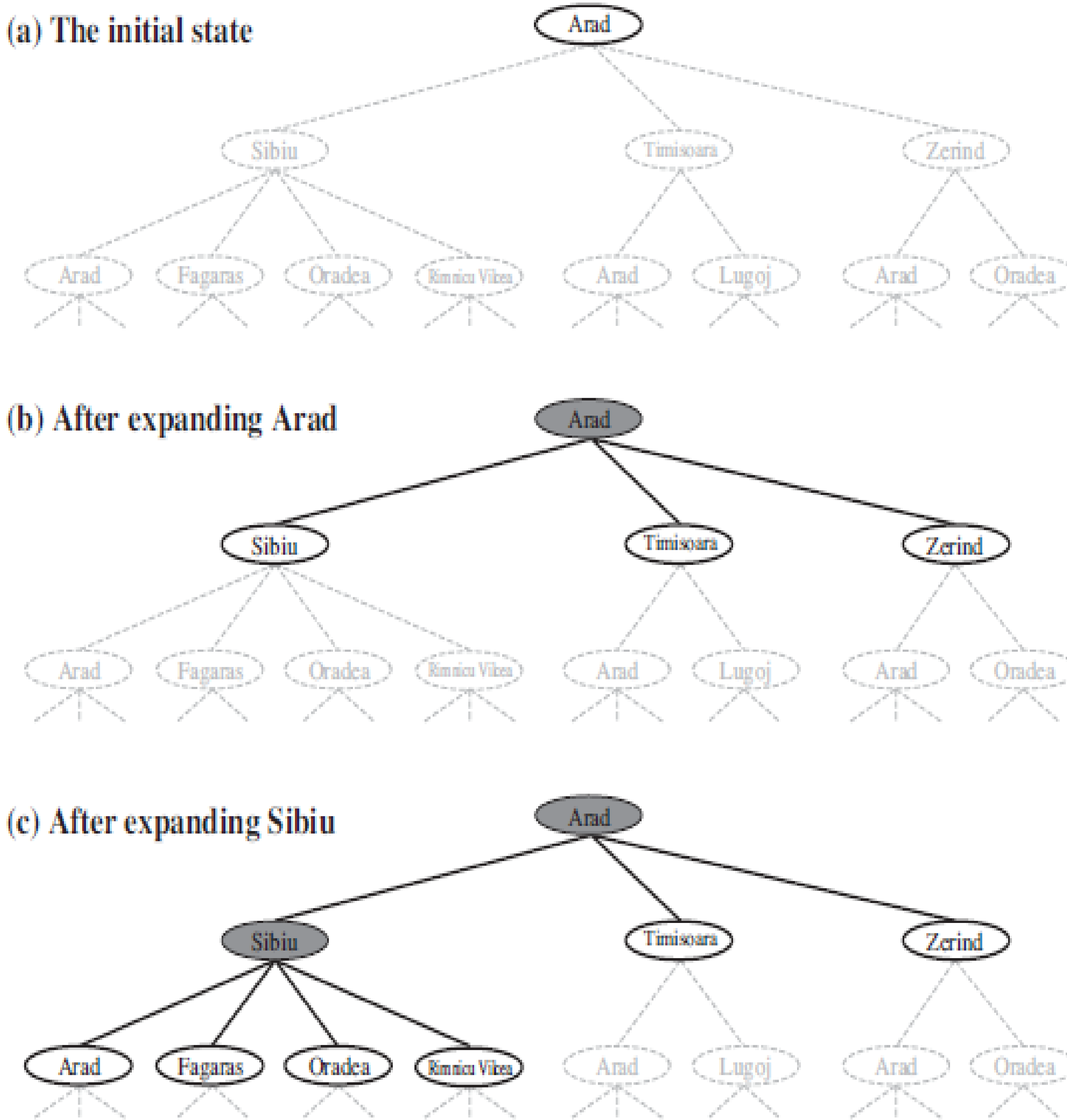
Optimality:

- It doesn't guarantee the best solution.

Depth-First Search



A simplified road map of part of Romania.


(a) The initial state
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(b) After expanding Arad
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
Zerind
Arad
Fagaras
Oradea
Rimnicu Vilcea
Arad
Lugoj
Arad
Oradea

Depth-First Search

It overcomes **time and space** complexities.

Time:

- $\boldsymbol{O(b^m)}$: terrible if $\boldsymbol{m}$ (maximum length of the depth) is much larger than the size of the state space.
- *If solutions are dense, may be much faster than breadth-first*.

Space:

- $\boldsymbol{O(bm)}$**,** i.e., linear space!
- A variant of depth-first search with **backtracking search** uses still less memory $\boldsymbol{O(m)}$**.**

Comparison with BFS

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

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

The depth-first search would require **156 kilobytes** instead of 10 Exabyte at depth $d = 16$, a factor of **7 trillion times less space**.

Depth-limited Search

Depth-limited Search

- The problem of depth first search can be alleviated by supplying depth-first search with a **pre-determined depth limit $\boldsymbol{l}$**.

$$\boldsymbol{l} < \boldsymbol{d}$$

- Nodes at depth $\boldsymbol{l}$ are treated as if they have no successors.

Completeness:

- It *may be incomplete* if we choose $\boldsymbol{l} < \boldsymbol{d}$, that is, the shallowest **goal is beyond the depth limit $\boldsymbol{l}$.**

Optimality:

- The depth-limited search *may be non-optimal* if we choose $\boldsymbol{l} > \boldsymbol{d}$.

Depth-limited Search

Time complexity: $O(b^l)$

Space complexity: $O(bl)$

- *Depth-first search can be viewed as a special case of depth-limited search with*

$$l = \infty$$

- Depth limited search can be based on knowledge of the problem.
  - For the most problems, however, **we will not know a good depth limit until** we have solved the problem.

Iterative Deepening Search

Iterative Deepening Search

- No selection of the best depth limit
- It tries all possible depth limits:
  - first 0, then 1, 2, and so on
- Combines the benefits of depth-first and breadth-first search,

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution
    inputs: *problem*, a problem

    for *depth* $\leftarrow$ 0 **to** ∞ **do**
        result $\leftarrow$ DEPTH-LIMITED-SEARCH(*problem*, *depth*)
        if *result* $\neq$ cutoff **then return** *result*
    end

The cutoff value indicates that there is no any solution within the depth limit.

Iterative Deepening Search

- $\ell = 0$:

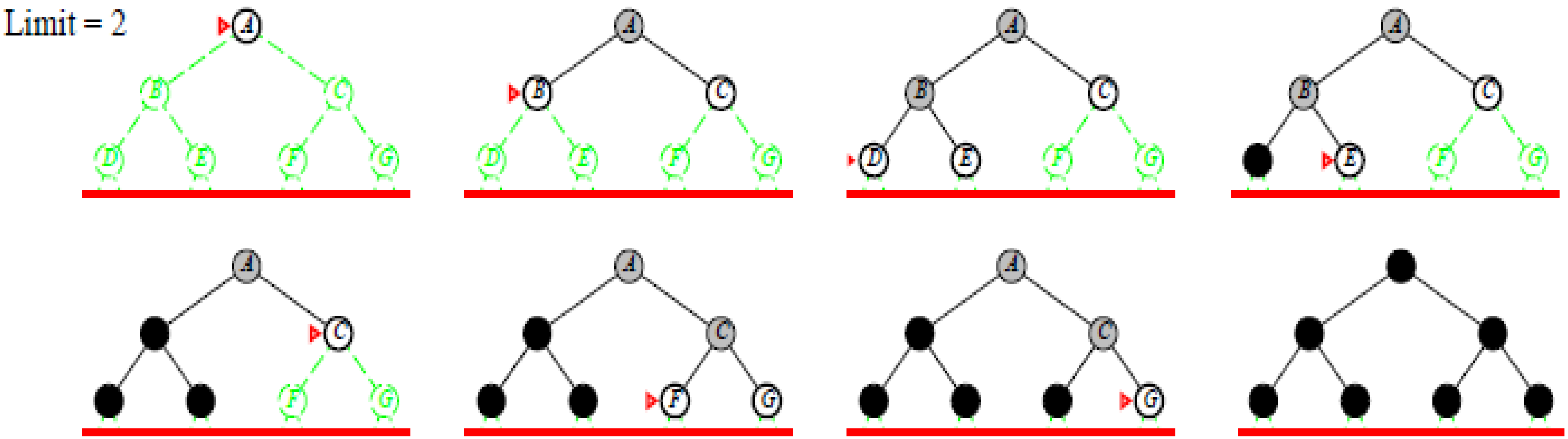
Limit = 0

- $\ell = 1$:

Limit = 1

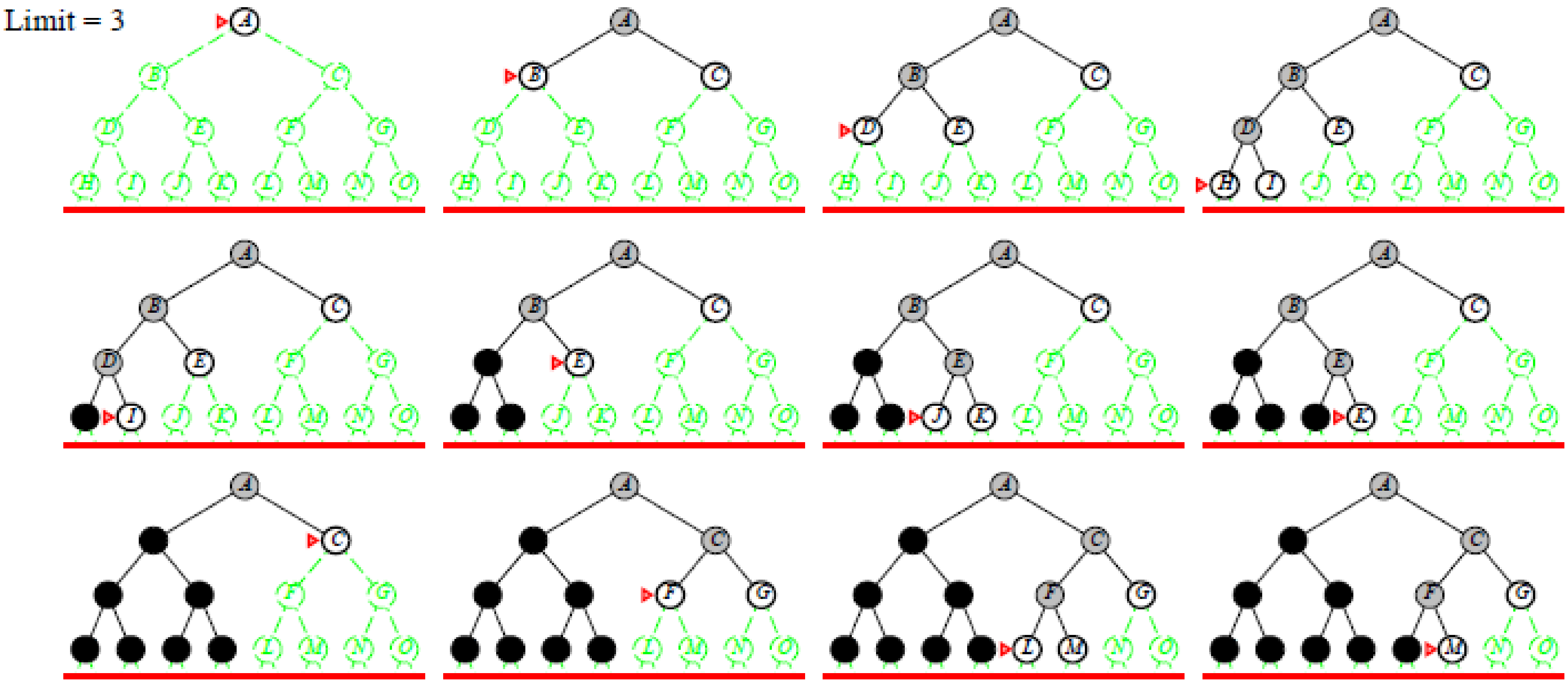
Iterative Deepening Search

- $\ell = 2$:



Iterative Deepening Search

- $\ell = 3$:



Iterative Deepening Search

Complete?? Yes if ***b*** is finite

Time?? $(d)b + (d-1)b^2 + \cdots + (1)b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes, if step cost = 1
Can be modified to explore uniform-cost tree

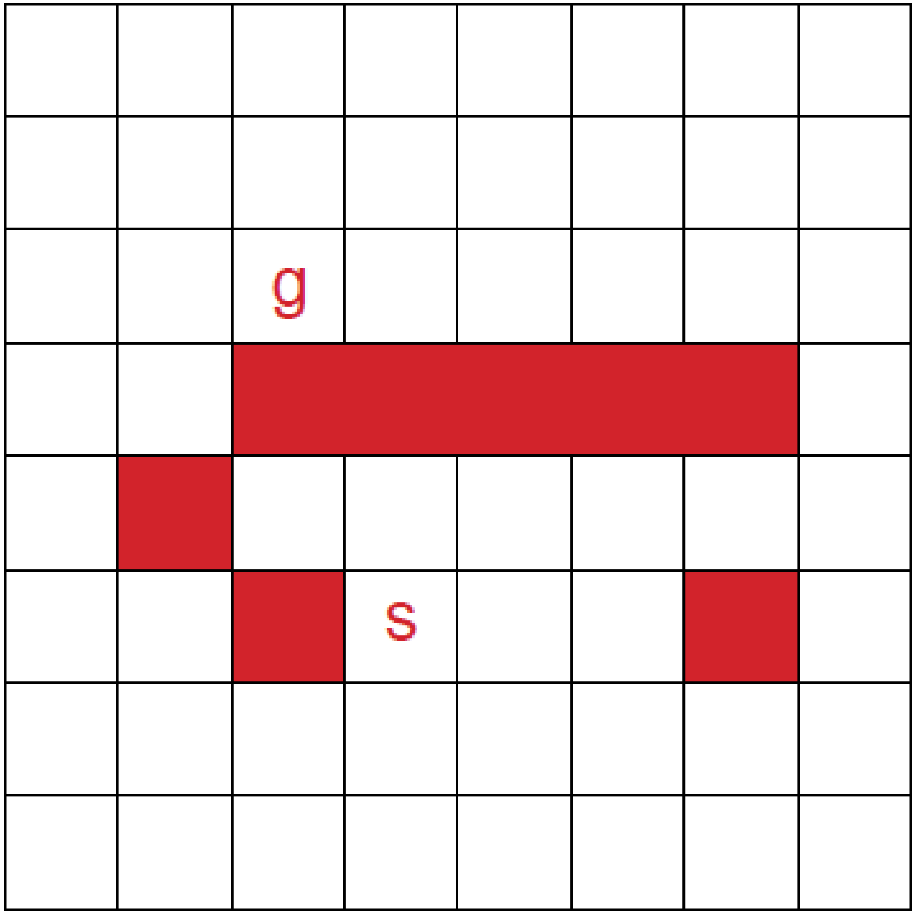
IDS suitable for the problem

- having a large search space
- and the *depth of the solution is not known*

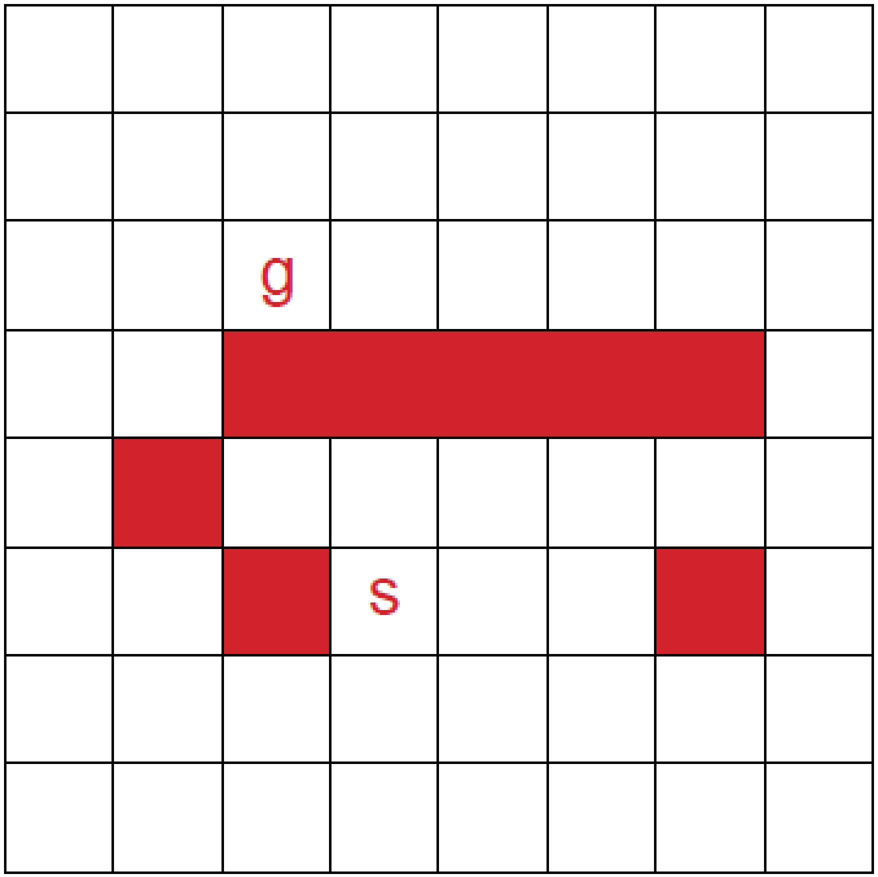
Real Life Example

Example … DFS

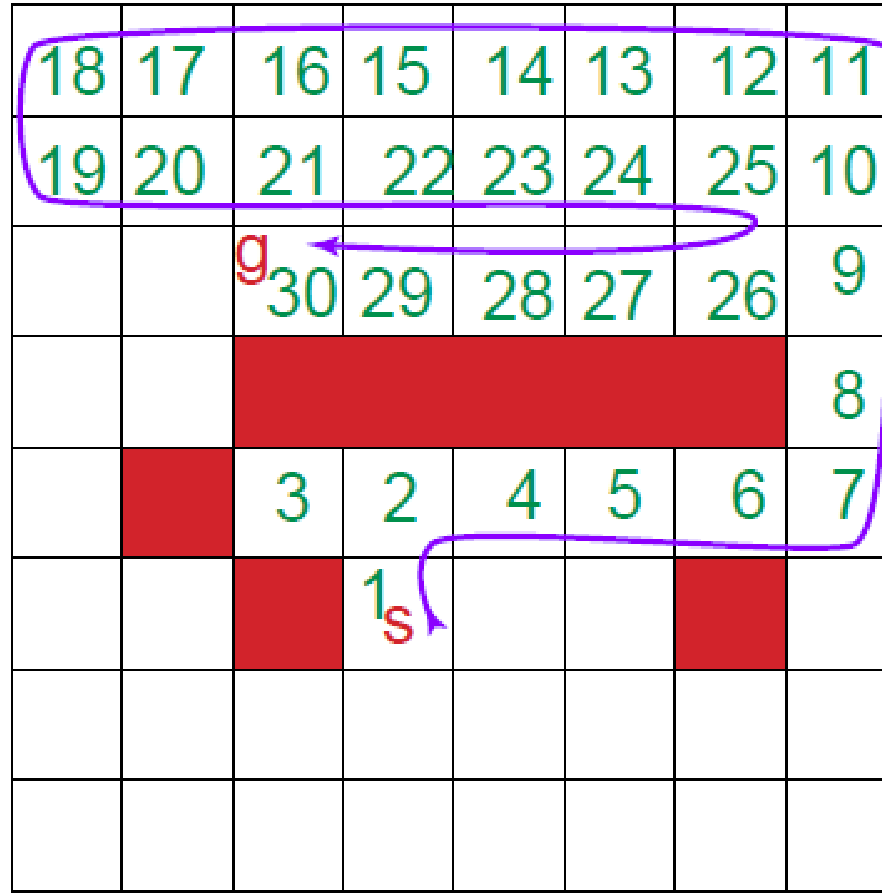
- Finding a path in the given grid using DFS.
- The order of the actions are **up**, **left**, **right**, then **down**.
- Maintain the visited list to avoid looping.
- Number the square according to the traversal.



Example



g
s



18 17 16 15 14 13 12 11
19 20 21 22 23 24 25 10
g 30 29 28 27 26 9
8
3 2 4 5 6 7
1 s

Reading Material

- **Artificial Intelligence,** A Modern Approach
  Stuart J. Russell and Peter Norvig
  - **Chapter 3.**