Basic Search Strategies

Lecture 1

Ning Xiong

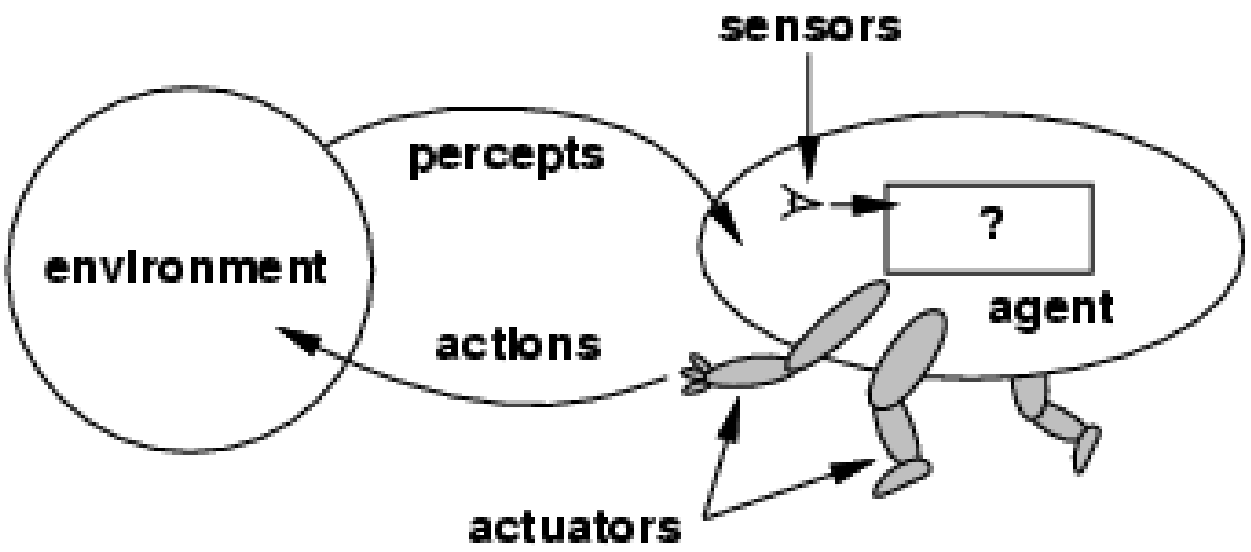
Mälardalen University

Outline

- Motivation (problems requiring search)
- How to represent and formulate search problems
- Basic search strategies on tree structure
  - breadth-first
  - depth-first
  - uniform cost

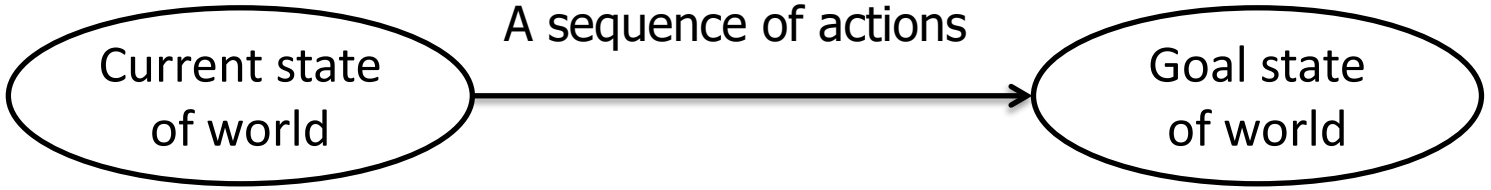
- depth limited
- Iterative deepening

Agents and Environments



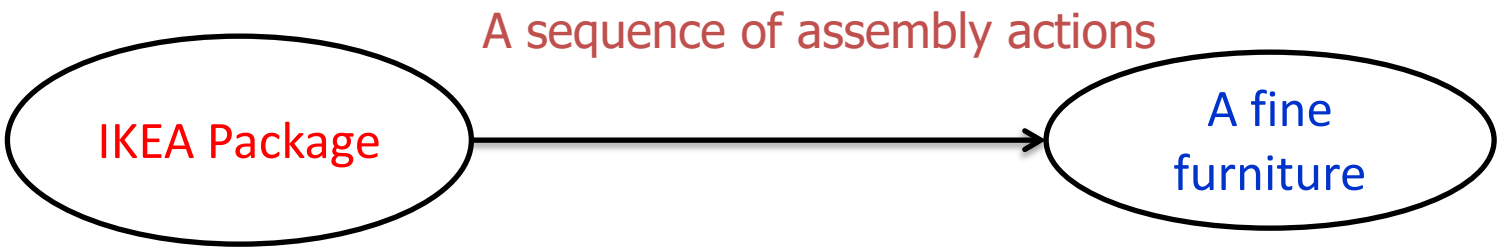
- The agent can perceive the "state" of the environment via sensors
- The "state" can reflect the situation of the agent itself and other entities in the environment
- Agent takes an action to change the "state" of the environment

The Need of Search by Agents

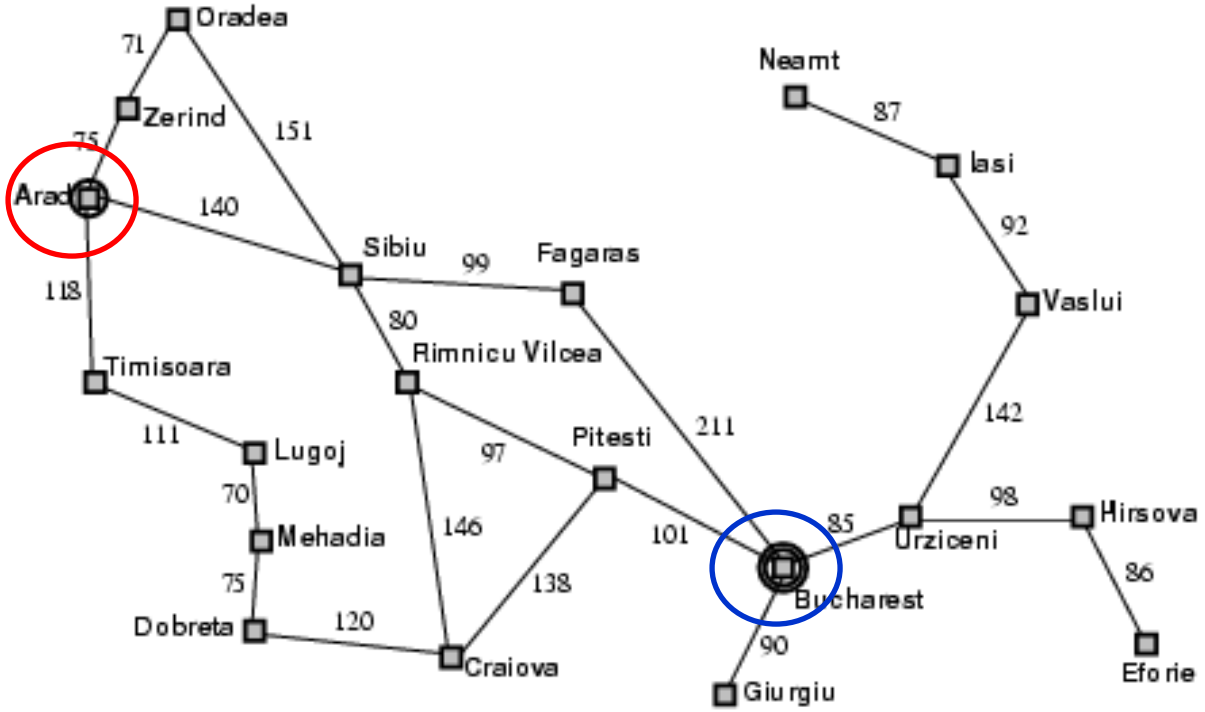


The agent needs to search for a plan for actions to transform the world from the current state to the goal state through of a series of intermediate states

Example 1:



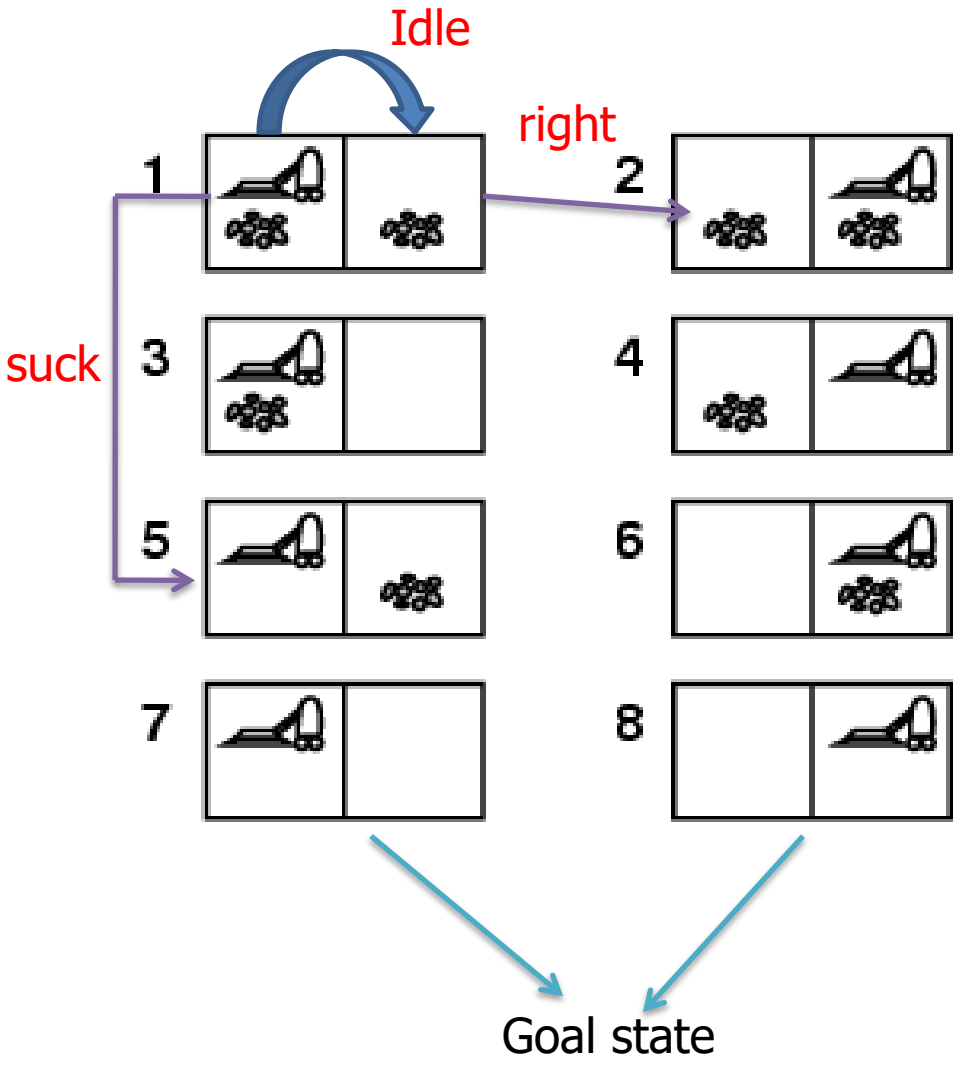
Example 2: Travel in Romania



Suppose you are in vacation in Romania. Currently you are situated in Arad, how can you plan a route to drive to destination Bucharest?

Example 3: vacuum world

Suppose the vacuum is situated in an environment depicted in the picture, how should the vacuum perform actions to remove dirt in both locations ?



Environment: two rooms

Actions: left, right, suck, idle

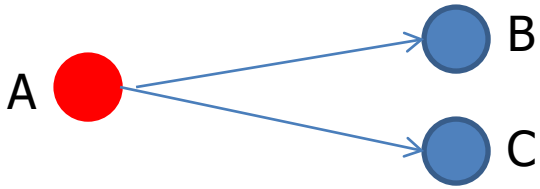
Initial state: both rooms with dust, vacuum in the left room

Goal state: no dust in any room

Search Problem Formulation

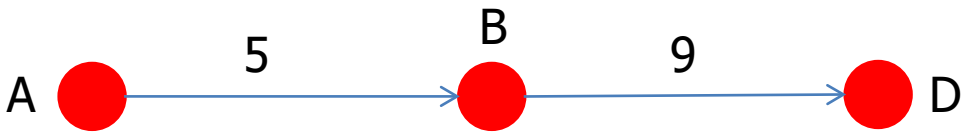
A (concrete) search problem consists of information in the following aspects:

1. Initial state that agent starts
2. Legal actions for every state and consequent states



3. Goal specification: what can be a goal state

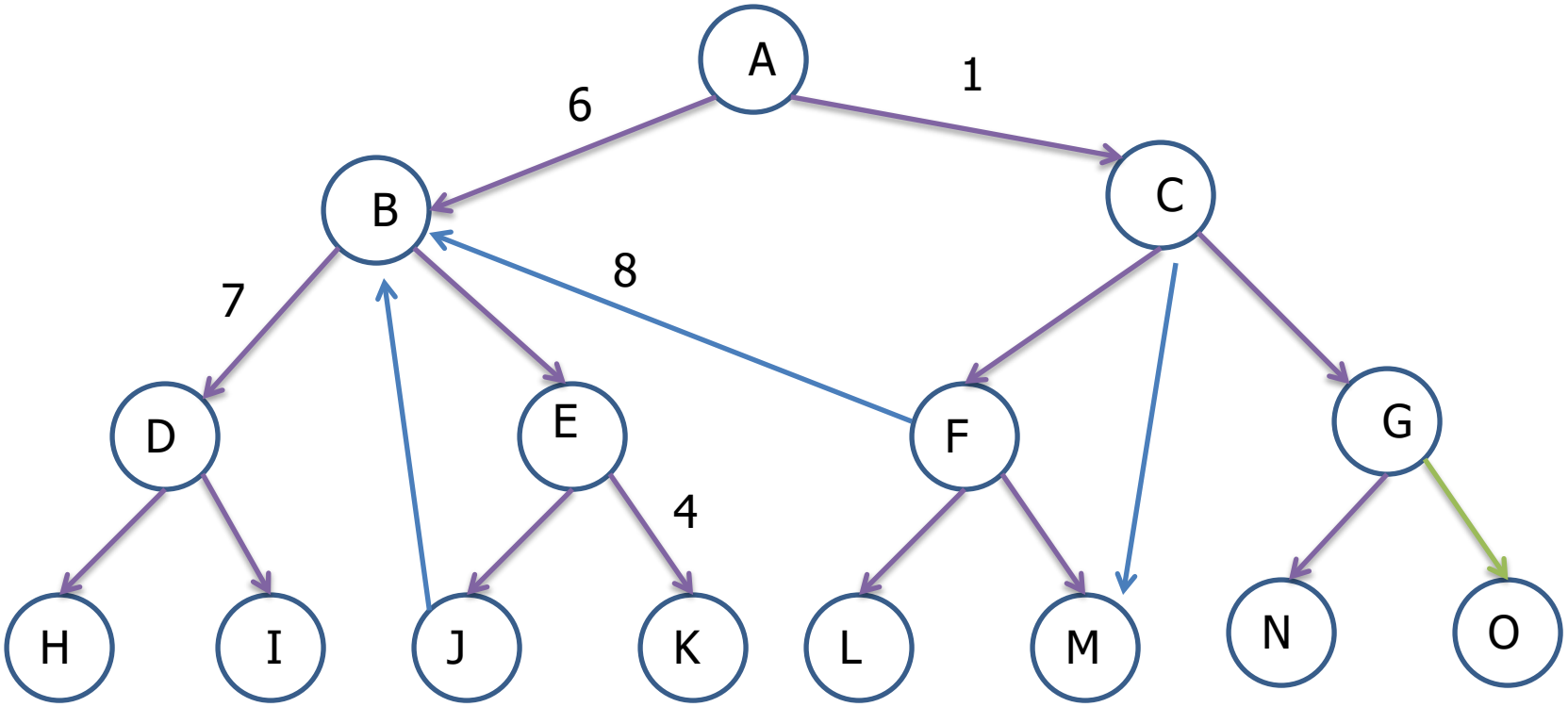
4. Cost of all legal actions or transitions
   Path cost: sum of costs of actions along a path,



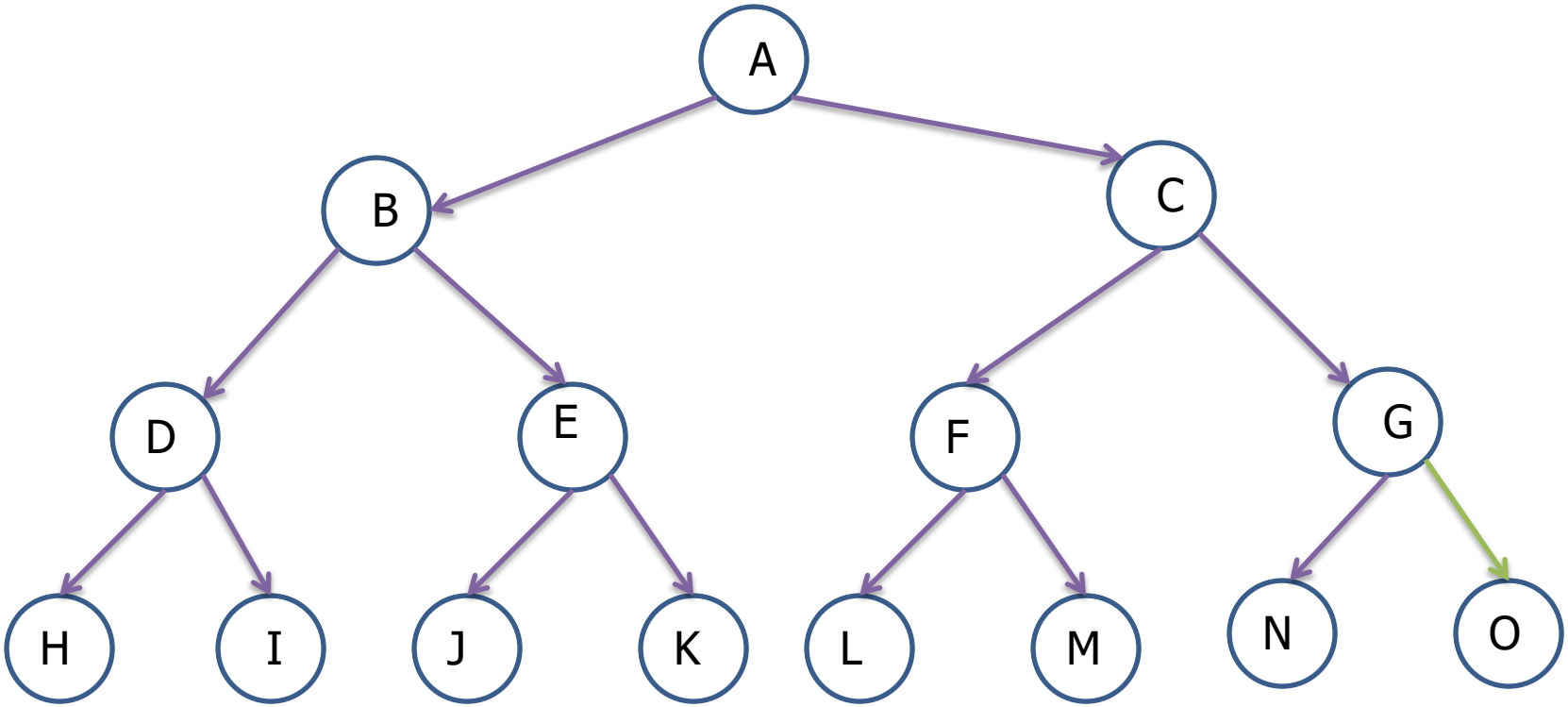
5+9=path cost

Graph Representation

- A state is represented as a node
- The transition between two states caused by an action is described by an arc between corresponding nodes
- An arc can have an associated value meaning the cost of the corresponding action



Special Case: Tree



- In the tree every node has only one entry. There is **at most one path** from one node to another
- **No loop or cycle** in the tree

General about tree search

- Basic idea:

1. Somehow **Choose a leaf node** for examination

2. Check whether the selected leaf node meets the goal.

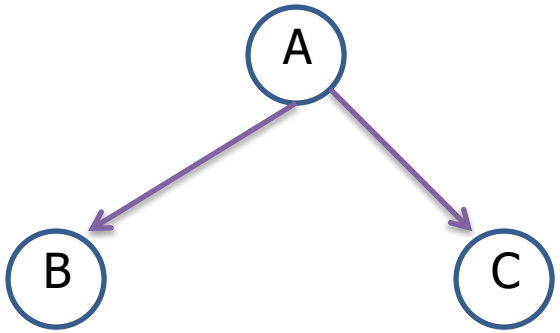
3. If yes, terminate and return the solution, otherwise expand the node and add the child nodes to the search tree

- The choice of leaf node for examination/expansion is decided by search algorithm

Tree search example

Start from A, this is the only node to expand
(A is the leaf node)

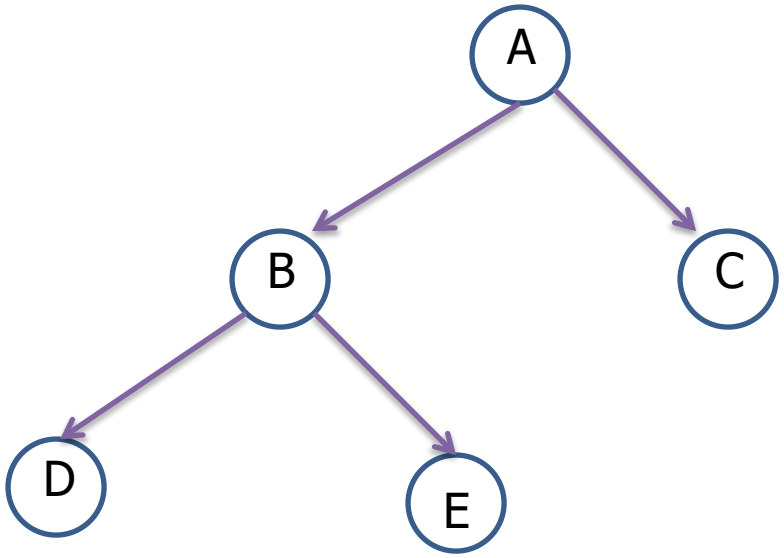
Tree search example



After expansion of "A", two successors B and C are added to the tree.

Now we choose "B" for examination

Tree search example



B is not goal, we expand B with its successors D and E.

Now we have three leaf nodes : D, E, C as candidates for next examination.

Implementation: general tree search

- Fringe is a list of generated but not yet examined nodes (leaf nodes)
- In other AI books it is called Open list
- Every time **the first node** in the list is selected for examination and expansion. So the key issue remaining is how to order the nodes in the fringe list

Function TREE-SEARCH (problem, fringe)
  fringe = [initial state]
  loop do
    If fringe is empty **then return** failure
    Remove the first node from fringe to s
    If goal-test on s succeeds **then return** solution(s)
    Expand s by creating its successor nodes
    Insert the successors of s into fringe and order nodes according to the search strategy

Search strategies

- A search strategy is defined by setting the order of nodes in the Fringe list (which is first)
- Basic search strategies use only the information in the problem definition rather than any domain kowledge. They are called uninformed or blind search

  - **Breadth-first search**
  - **Uniform-cost search**
  - **Depth-first search**
  - Limited depth search
  - Iterative deepening

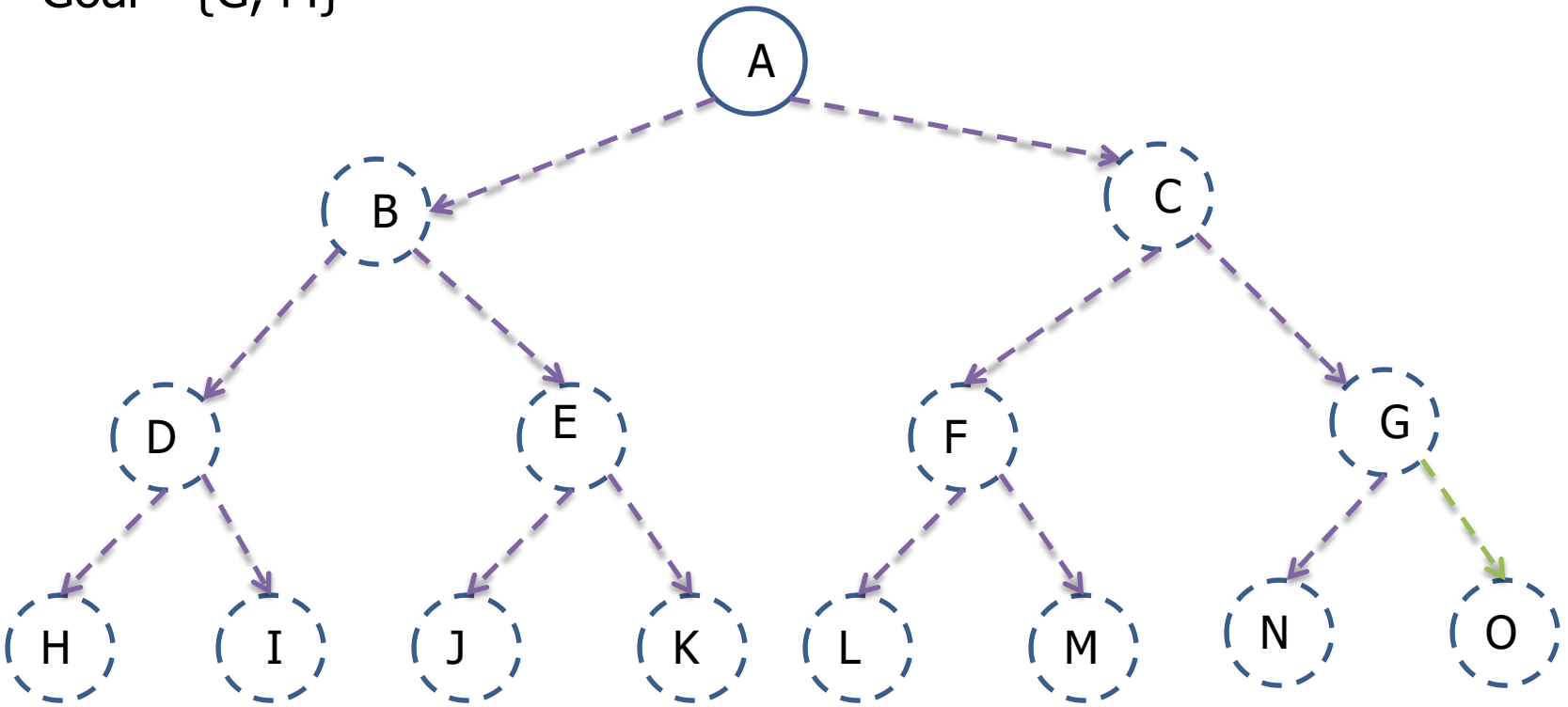
Breadth-first search

- Expand nodes level by level, starting from shallowest level
- Implementation:
  - *fringe* is a first in and first out (FIFO) queue, i.e., new successors enter at right end of the queue

Breadth First Search Example

Initial state = A
Goal ={G, M}

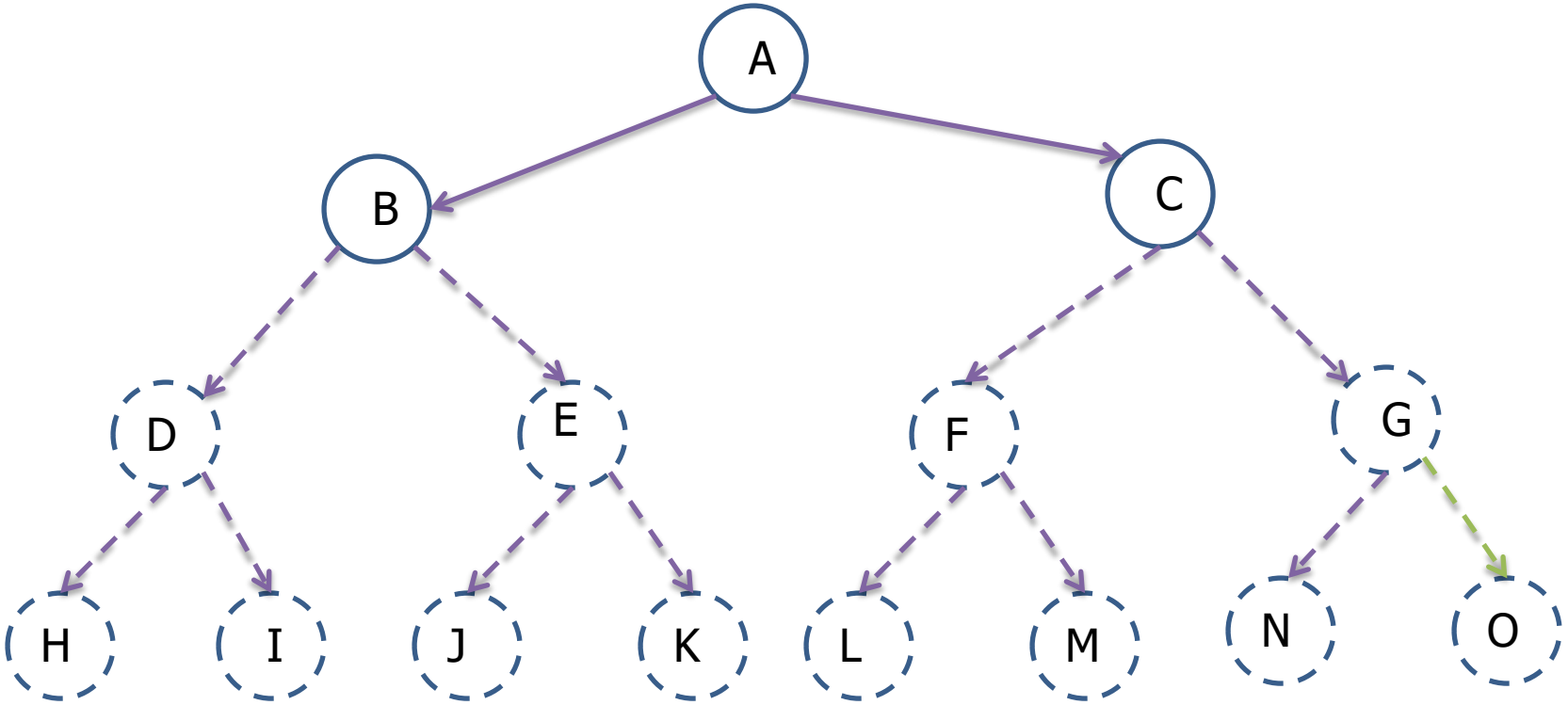
Fringe=(A)



Breadth First Search Example

A is expanded with its successors B and C

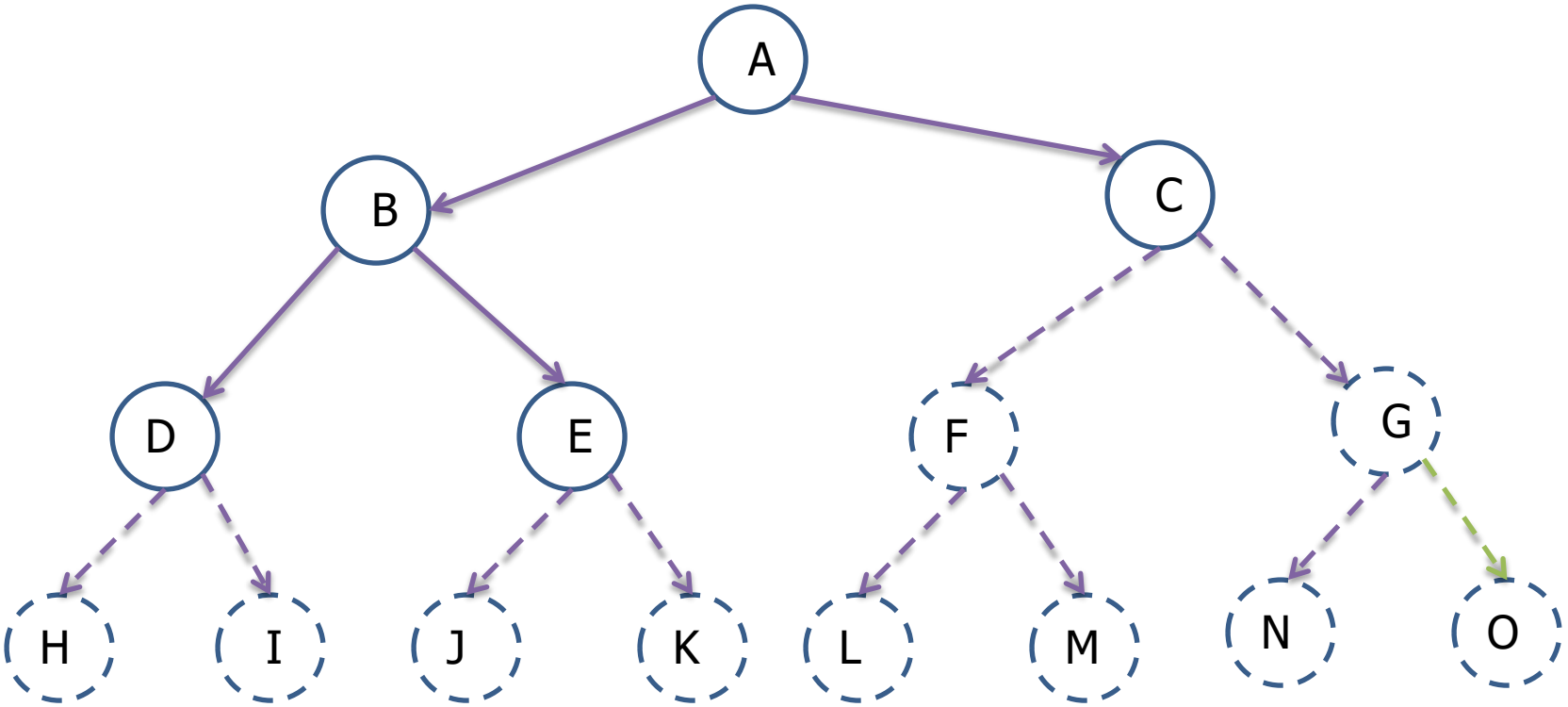
Fringe=(B, C)



Breadth First Example

B is expanded with its successors D and E

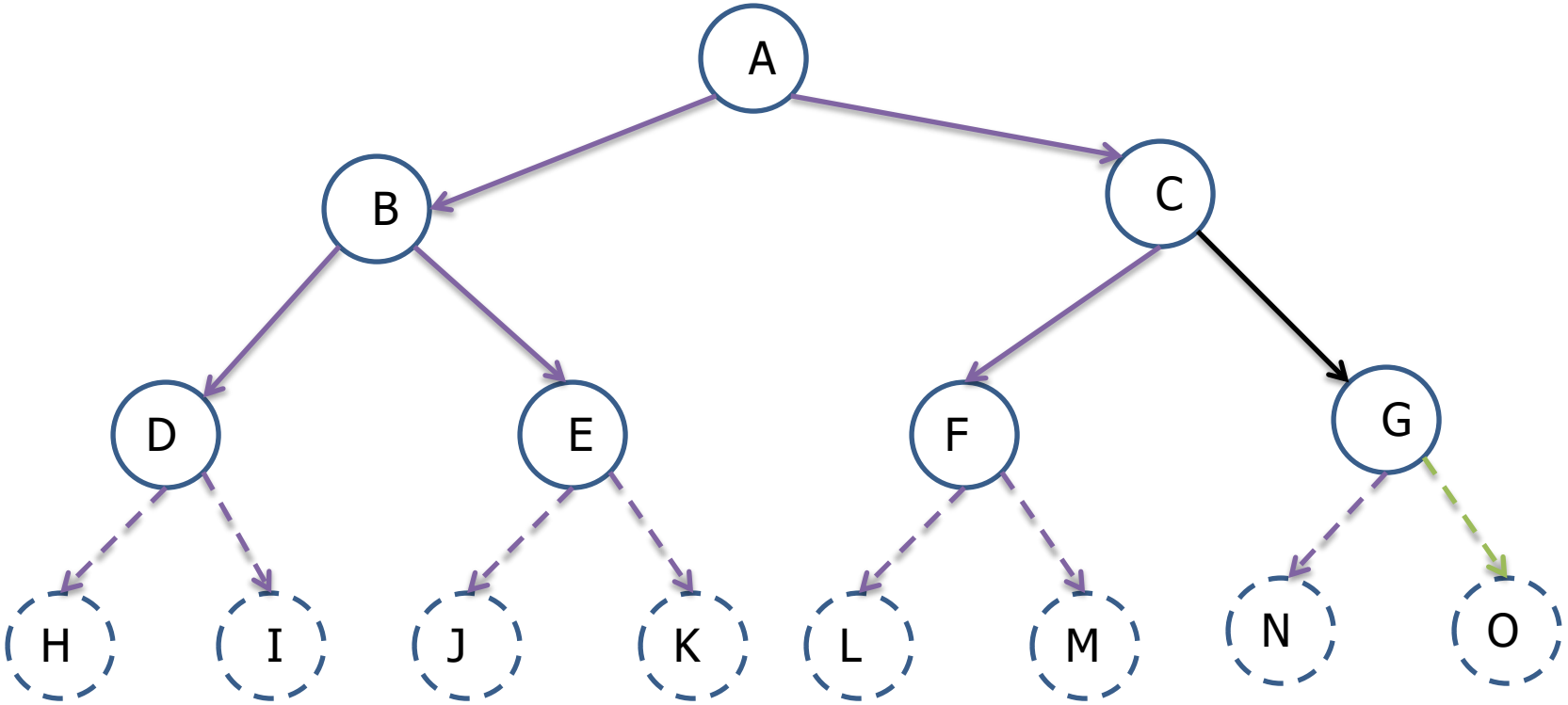
Fringe=(C, D, E)



Breadth First Search Example

C is expanded with its successors F and G

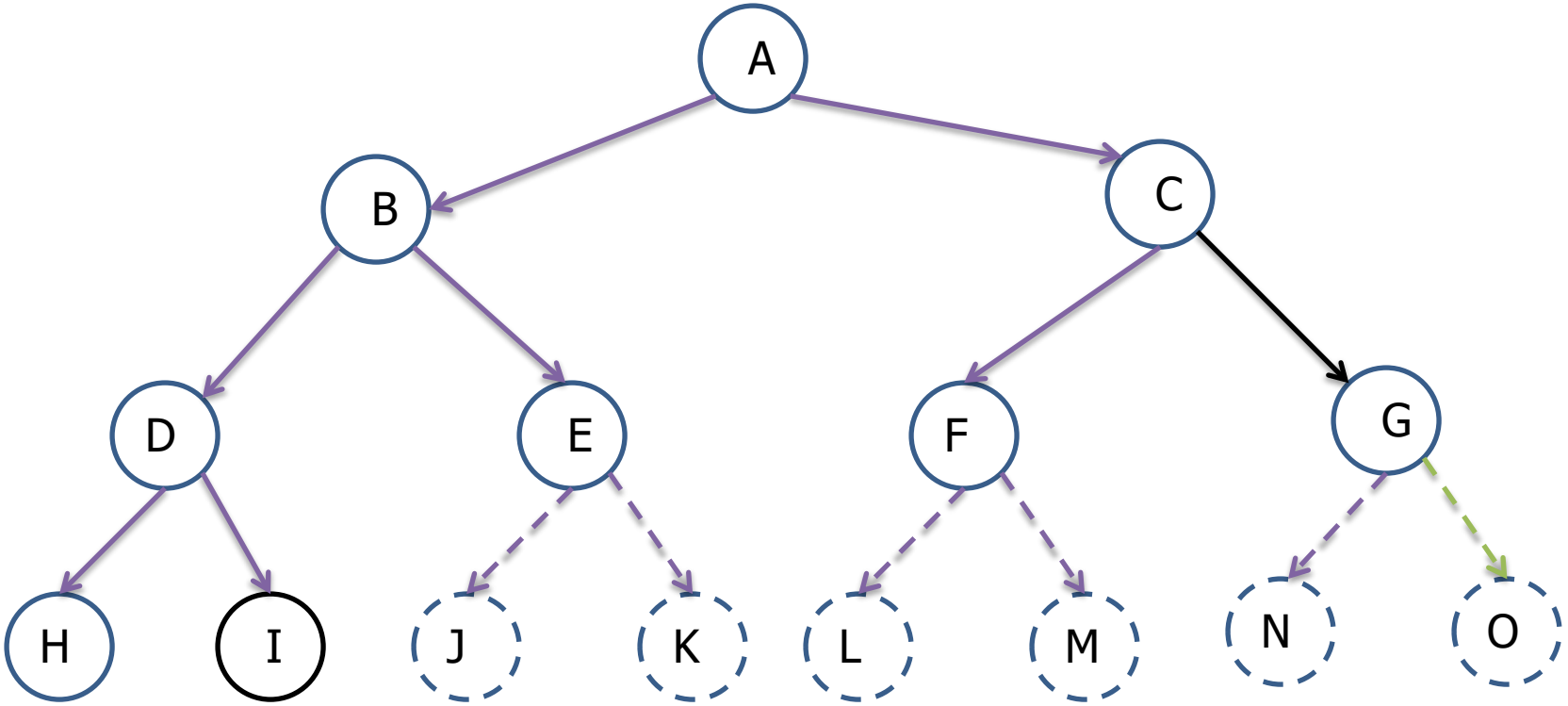
Fringe=(D,E,F,G)



Breadth First Search Example

D is expanded with its successors H and I

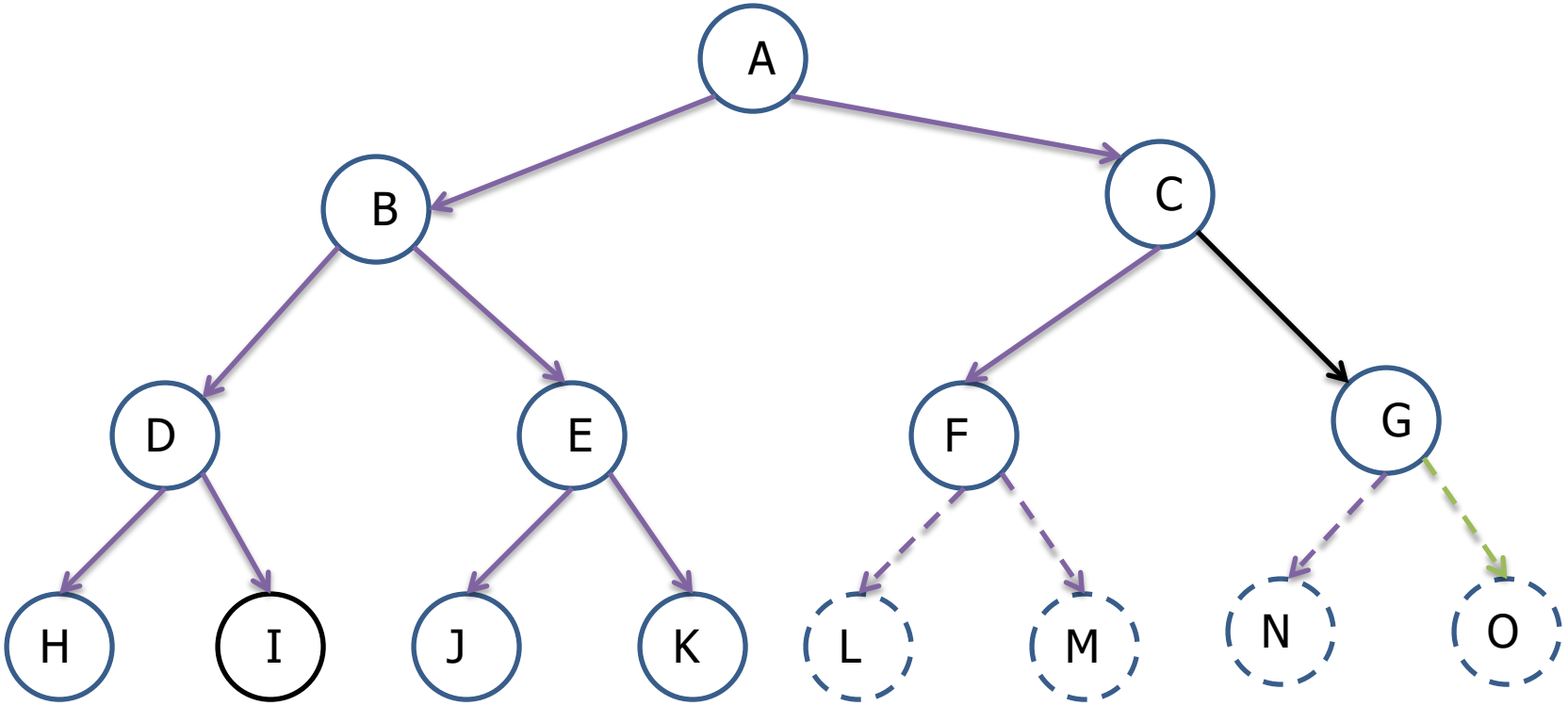
Fringe=(E,F,G,H,I)



Breadth First Search Example

E is expanded with its successors J and K

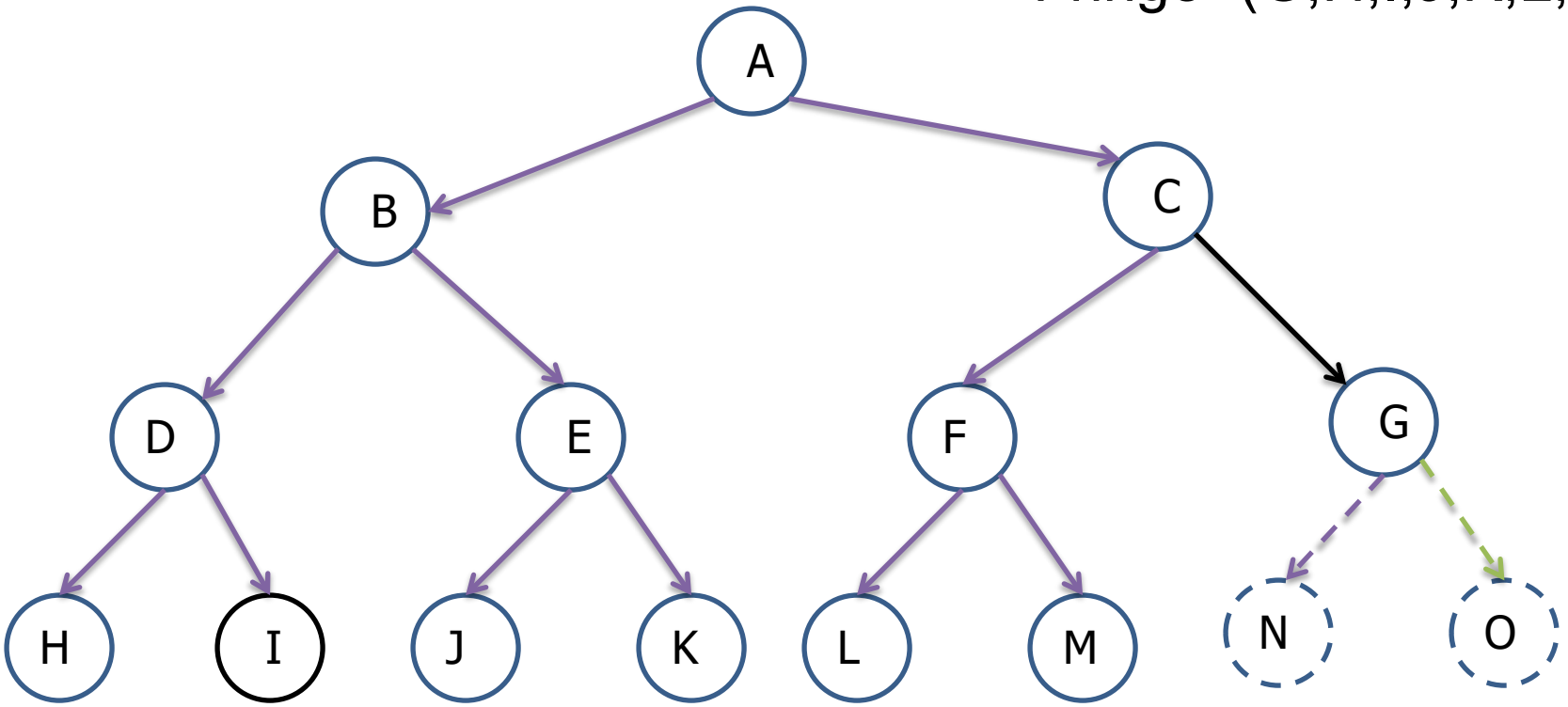
Fringe=(F,G,H,I,J,K)



Breadth First Search Example

F is expanded with its successors L and M

Fringe=(G,H,I,J,K,L,M)



Now we find the goal G and search algorithm terminates

Properties of breadth-first search

- Complete? Yes (can always find the goal if existing)

- Optimal? Yes (the path with minimal steps)

Suppose every state has b successors and the nearest goal solution is at the depth d

- Time (number of generated nodes)? $b+b^2+b^3+\ldots+b^d+b^{d+1}-b=O(b^{d+1})$

- Space (number of memorized nodes)? $O(b^{d+1})$ (keeps every node in memory)

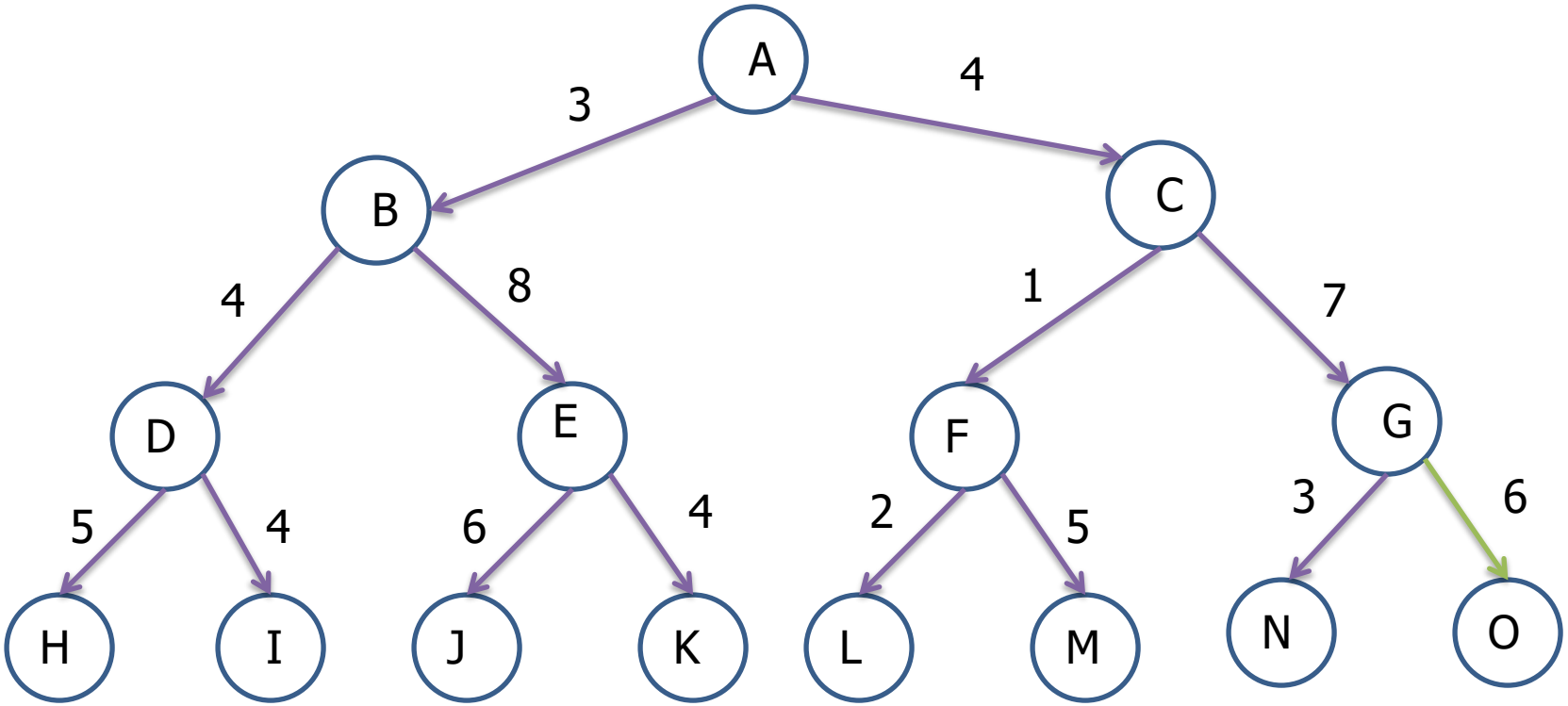
Space is the bigger problem (more than time)

Uniform-Cost Search

- Search for a solution with minimum total cost
- *fringe* ordered by costs of paths from initial state to nodes (node with small path cost appears first)
- Expand the leaf node with the minimum path cost
- Equivalent to breadth-first search if step costs are all equal (the nodes in shallow levels will always appear first in the Fringe list)

Uniform-Cost Search Example

Finding a path from A to {M or G} with total minimum cost

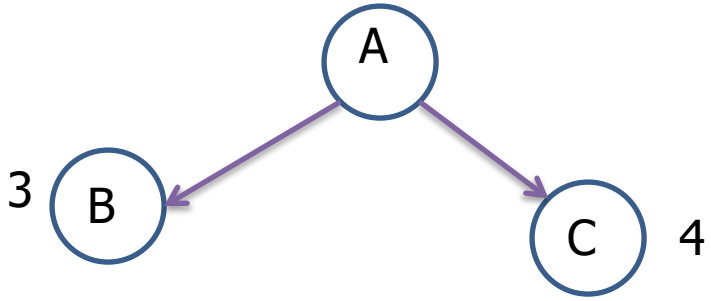


Uniform-Cost Search Example

Fringe=(A)

A is root node, put A in the Fringe list

Uniform-Cost Search Example



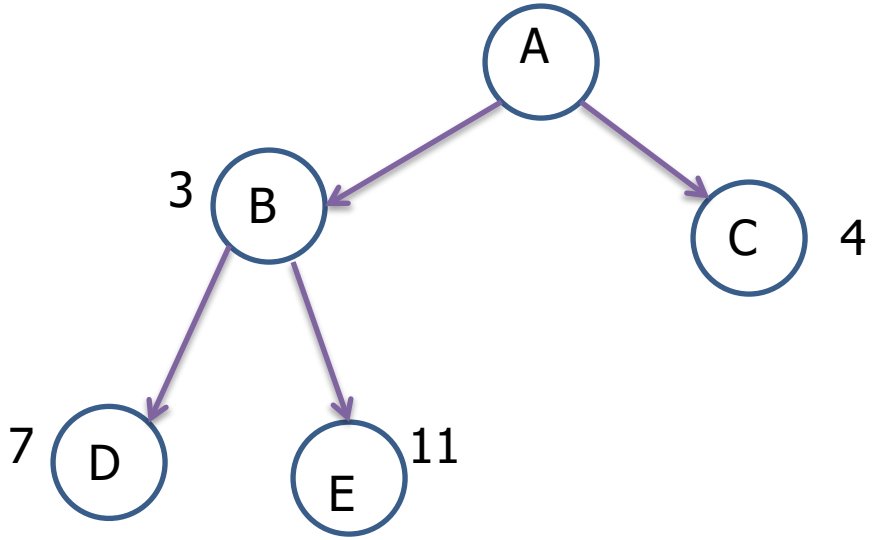
Fringe=(B,C)

A is expanded with its successors B and C

The path cost for B is 3

The Path cost for C is 4

Uniform-Cost Search Example



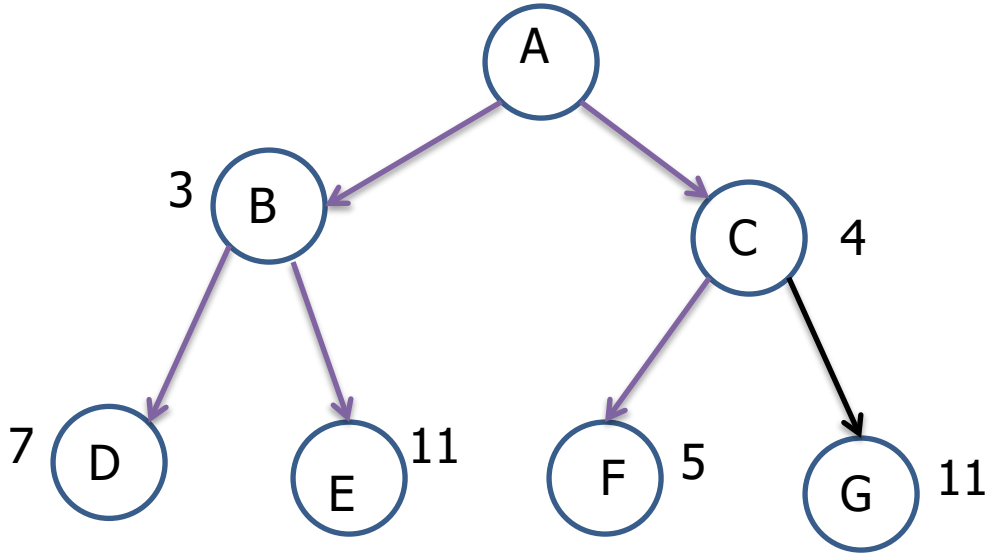
Fringe=(C,D,E)

B is expanded with its successors D and E

The path cost for D is 7

The Path cost for E is 11

Uniform-Cost Search Example



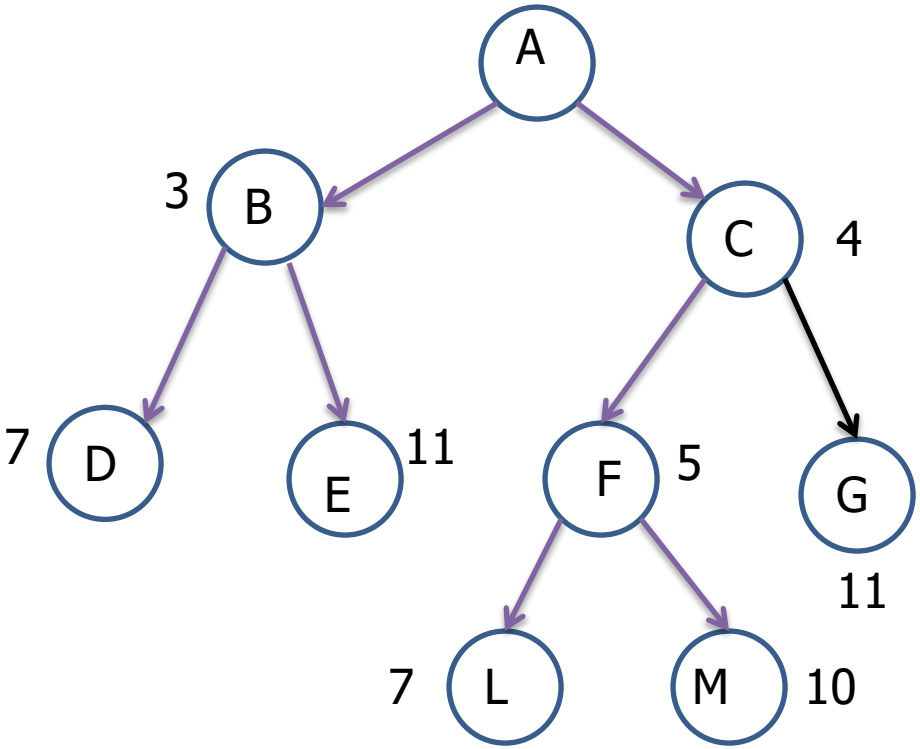
Fringe=(F,D,E,G)

C is expanded with its successors F and G

The path cost for F is 5

The path cost for G is 11

Uniform-Cost Search Example



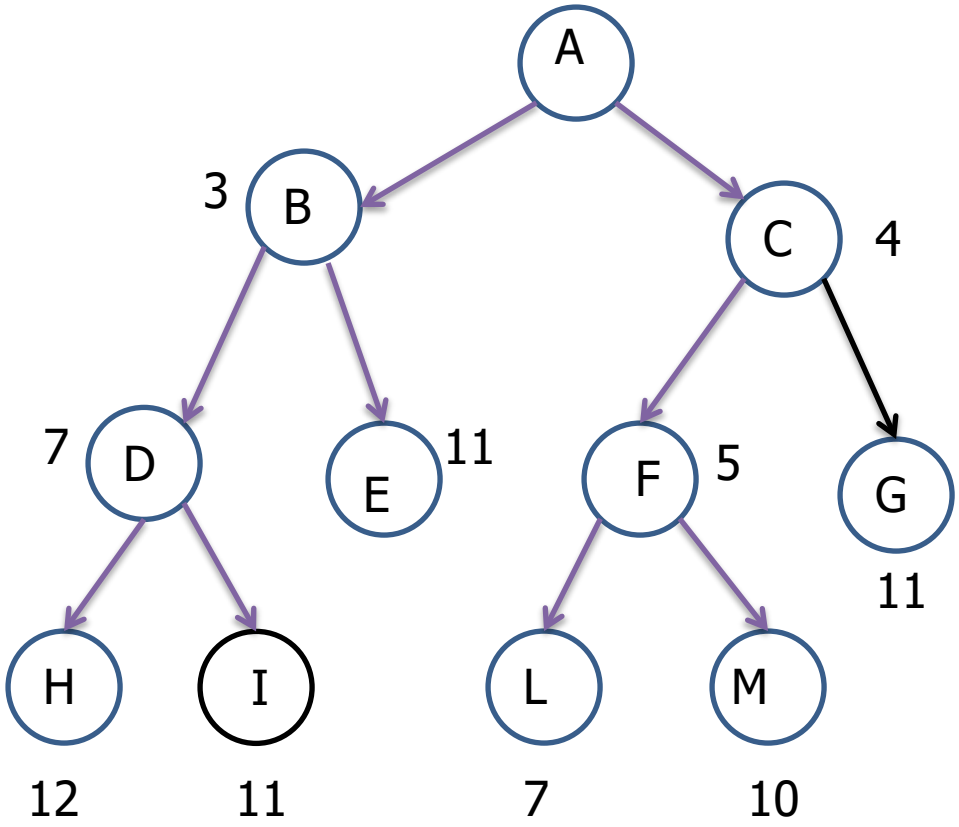
Fringe=(D,L,M,E,G)

F is expanded with its successors L and M

The path cost for L is 7

The path cost for M is 10

Uniform-Cost Search Example



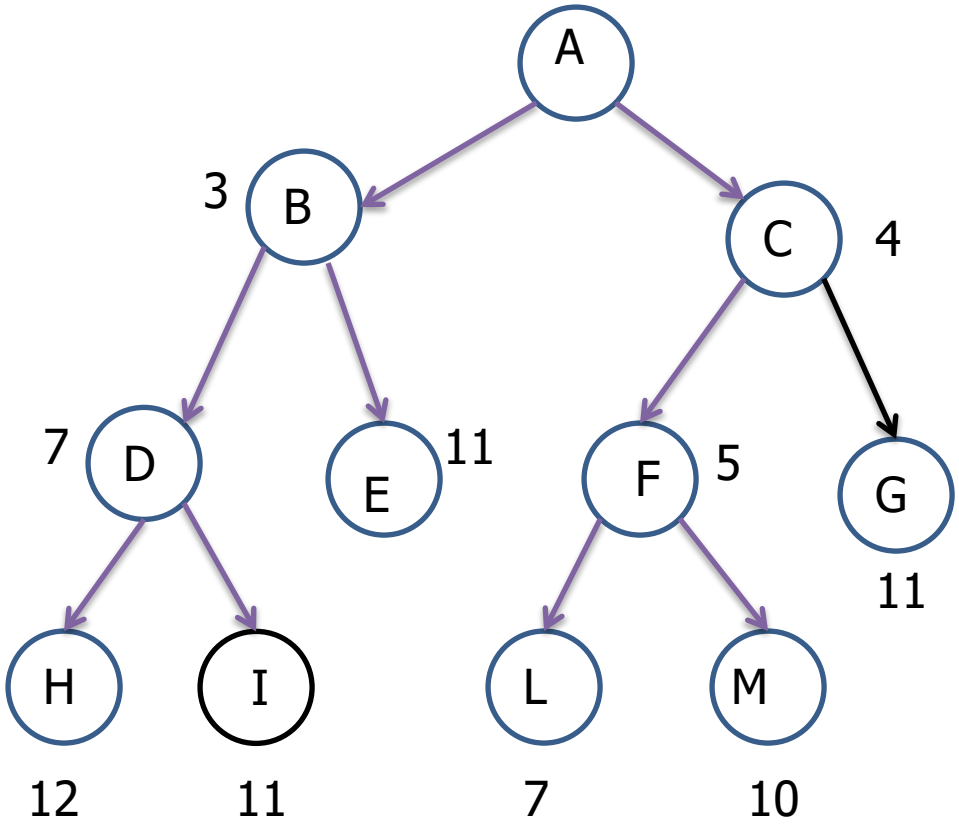
Fringe=(L,M,E,G,I,H)

D is expanded with its successors H and I

The path cost for H is 12

The path cost for I is 11

Uniform-Cost Search Example



Fringe=(M,E,G,I,H)

L can not be expanded

M is the goal

Search is terminated with the optimal total cost 10

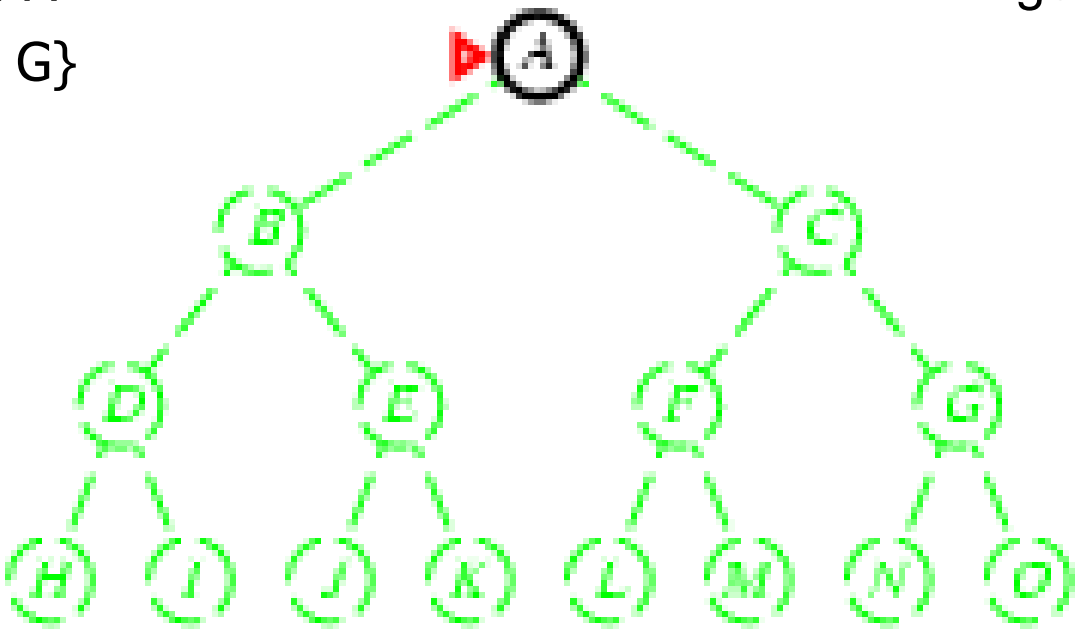
Depth-first search

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

Initial state: A

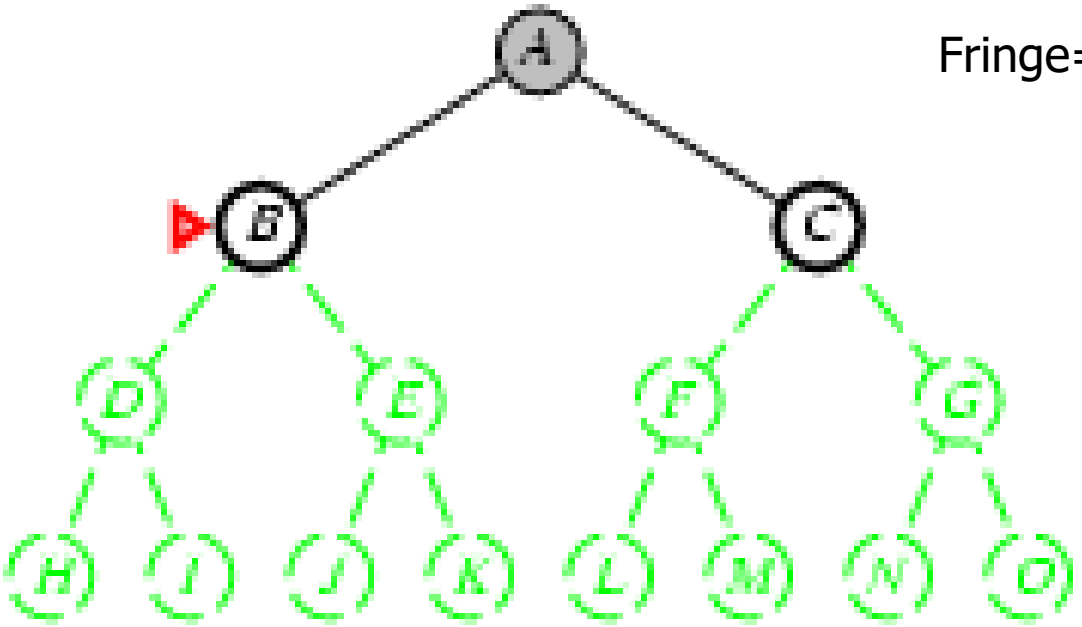
Goal={M or G}

Fringe=(A)



Depth-first search

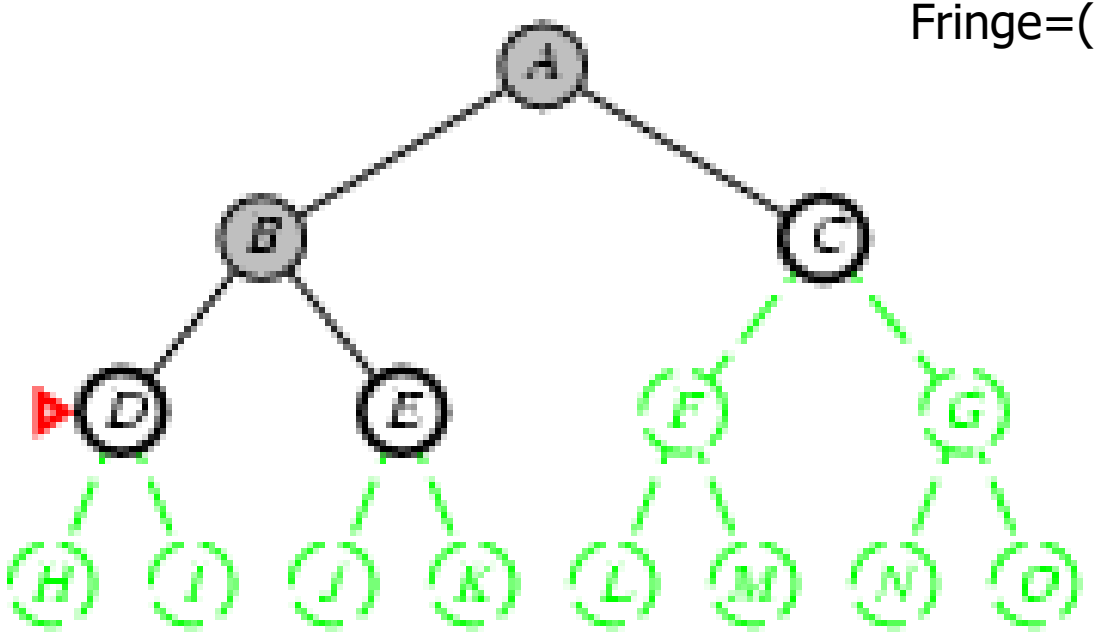
A is expanded with its successors B and C



Fringe=(B,C)

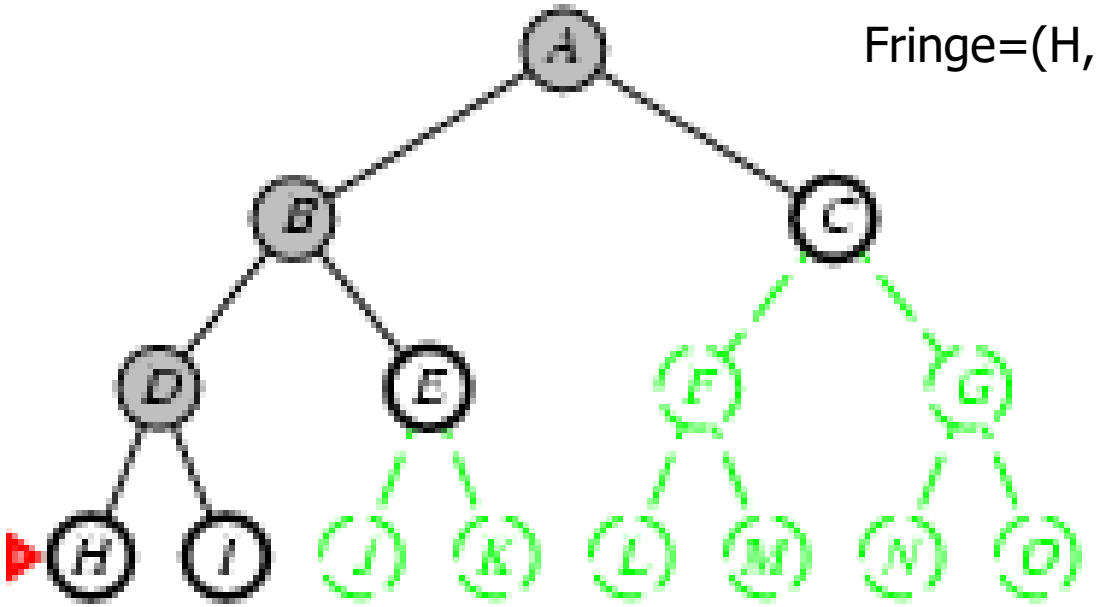
Depth-first search

Fringe=(D, E, C)



B is expanded with its successors D and E

Depth-first search example

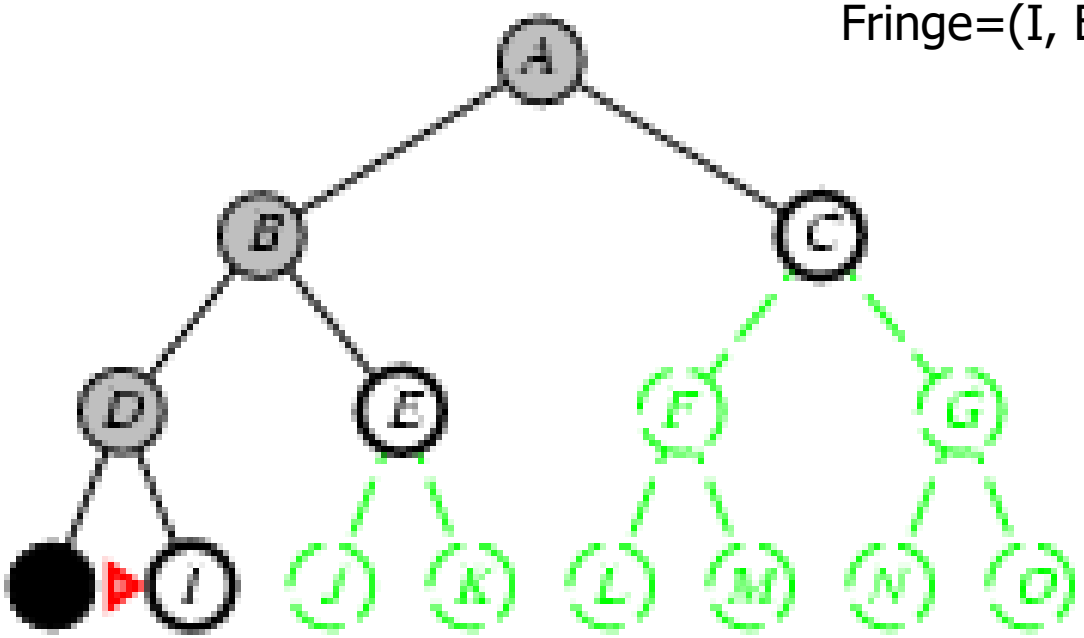


Fringe=(H, I, E, C)

D is expanded with its successors H and I

Depth-first search

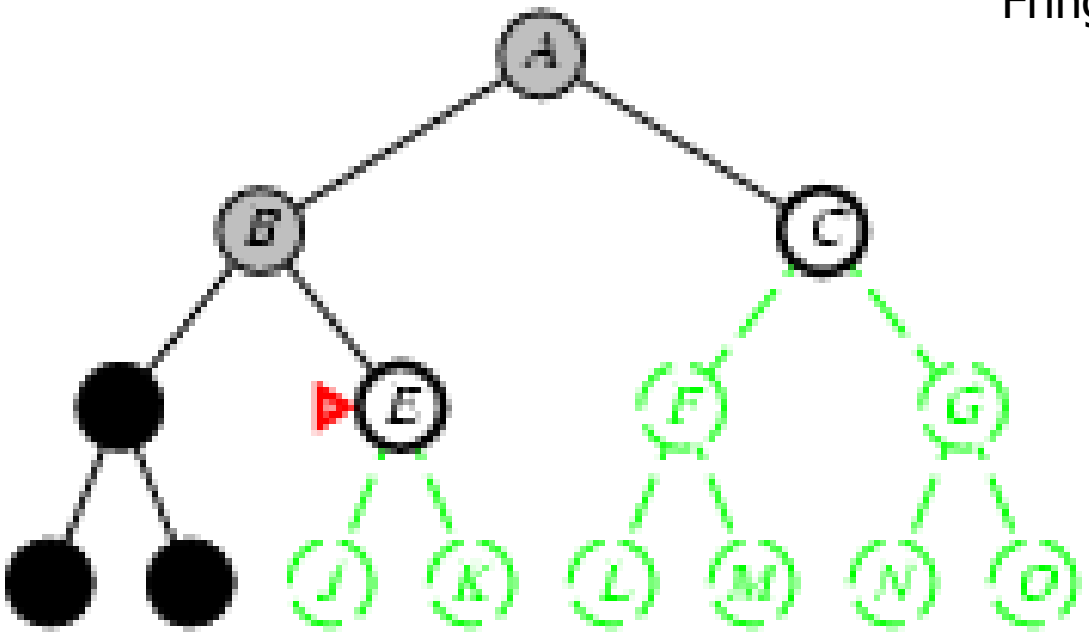
Fringe=(I, E, C)



H can not be expanded, only removed from Fringe

Depth-first search

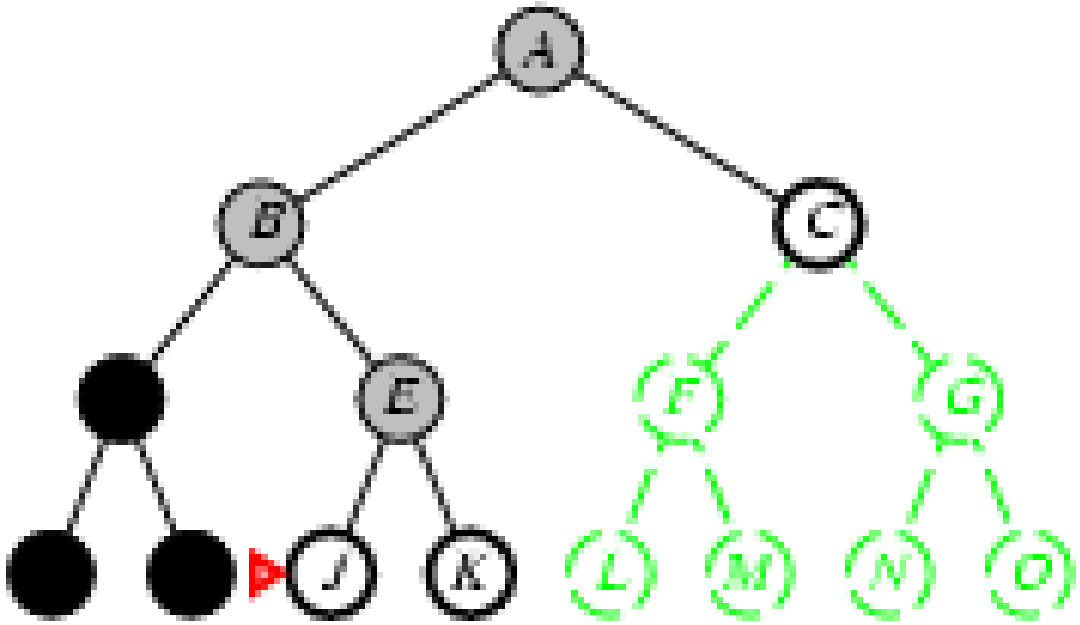
Fringe=(E,C)



The node "I" can not be expanded, only removed from Fringe

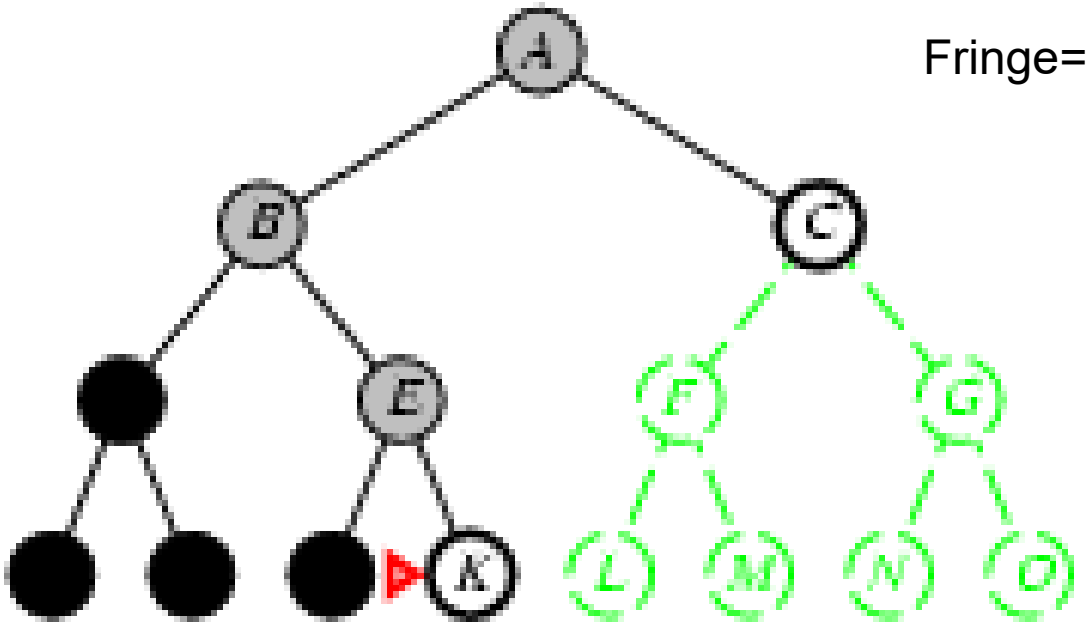
Depth-first search

Fringe=(J, K, C)



E is expanded with its successors J and K

Depth-first search

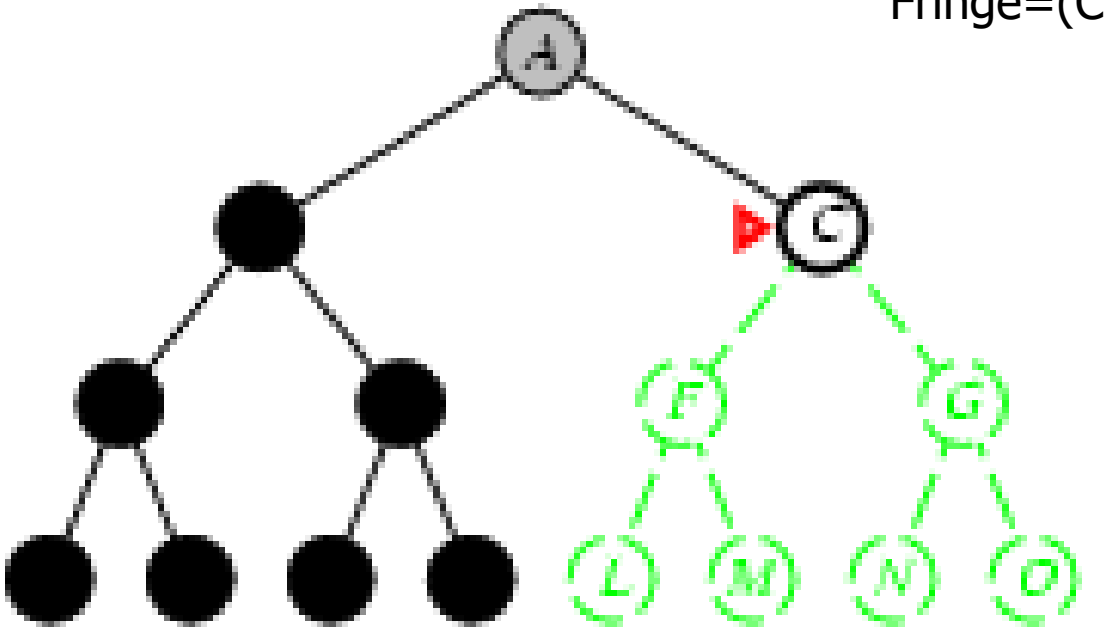


Fringe=(K, C)

J can not be expanded, only removed from Fringe

Depth-first search

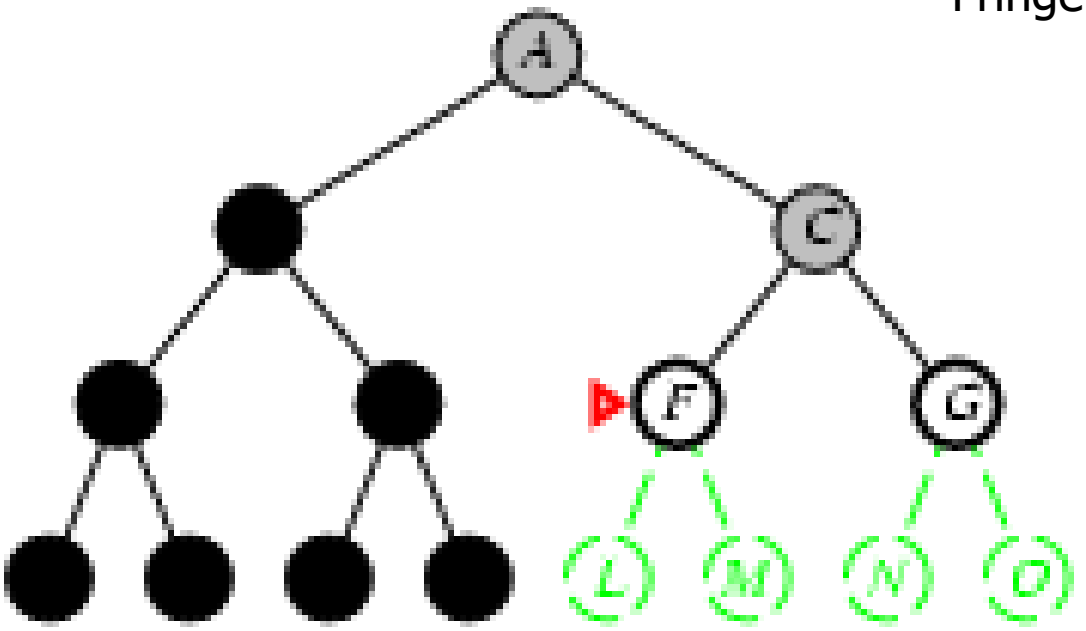
Fringe=(C)



K can not be expanded, only removed from Fringe

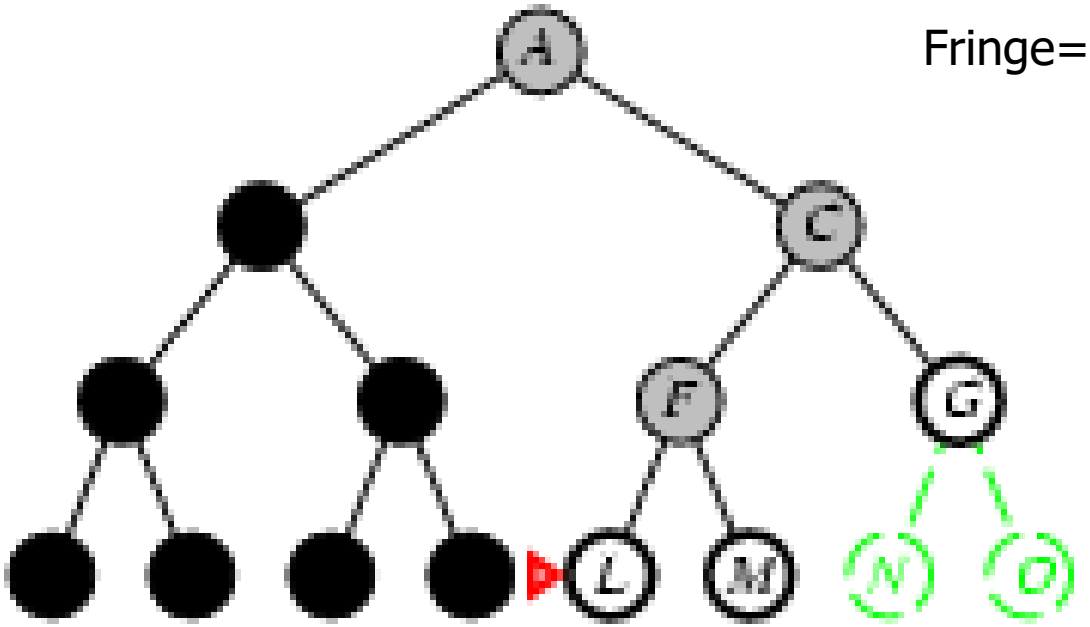
Depth-first search

Fringe=(F, G)



C is expanded with its successors F and G

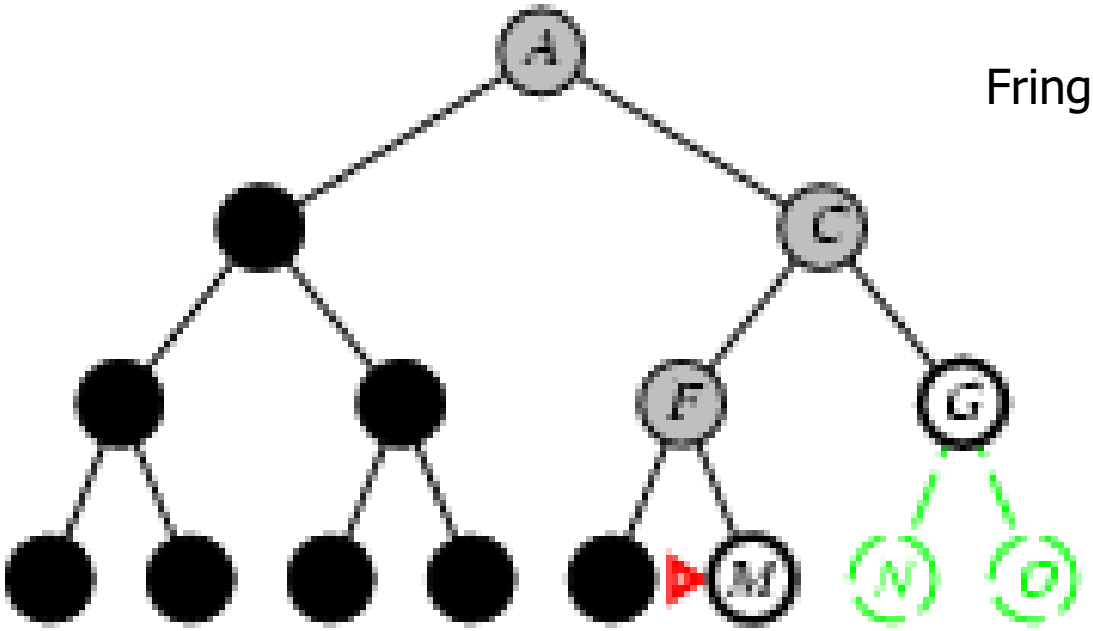
Depth-first search



Fringe=(L, M, G)

F is expanded with its successors L and M

Depth-first search



Fringe=(M, G)

L can not be expanded, only removed from Fringe

The search terminates with M found as the goal

Properties of depth-first search

- Complete? No: fails in infinite-depth spaces (tree is unbounded)

- Optimal? No, it can miss a goal more close to the root node

Suppose every state has b successors, and m is the maximum depth of the tree

- Time? In worst case it will produce all of the $O(b^m)$ *nodes,*

- Space? The number of nodes to be memorized is not more than bm, i.e., linear space!

  Note : the number of nodes to be memorized at every level is at most b

Depth-limited search

- To avoid unbounded search space or reduce the time complexity when m (natural maximum depth) is large
- Limited depth << natural maximum depth
- Nodes at the limited depth are treated as terminate nodes (no children)
- Depth-limited search = depth-first search but stopping expansion at depth limit

Iterative Deepening Depth First

- Gradually increase the depth limit for depth-first search

- Combining the benefits of both depth-first and breadth-first search

  羊modest memory requirement as depth-first
  羊complete and finding the shallowest goal as breadth first.

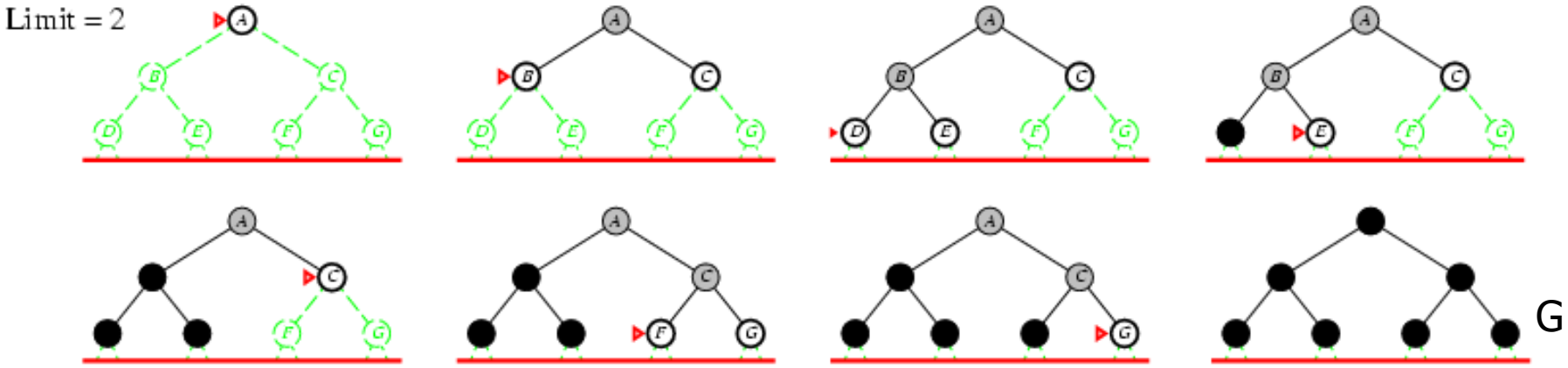
- Suitable to be used when the search space is large and the depth of goal is not known

Iterative Deepening (Limit=1)

Do the depth-first search with limit=1, fail

Iterative Deepening (Limit=2)

Do the depth-first search with limit=2, succeed



The search process terminates at G, getting the solution path by back tracking