Uninformed Search

Measuring problem-solving performance

Completeness: Is the algorithm guaranteed to find a solution when there is one?

Optimality: Does the strategy find the optimal solution, as defined on page 68?

Time complexity: How long does it take to find a solution?

Space complexity: How much memory is needed to perform the search?

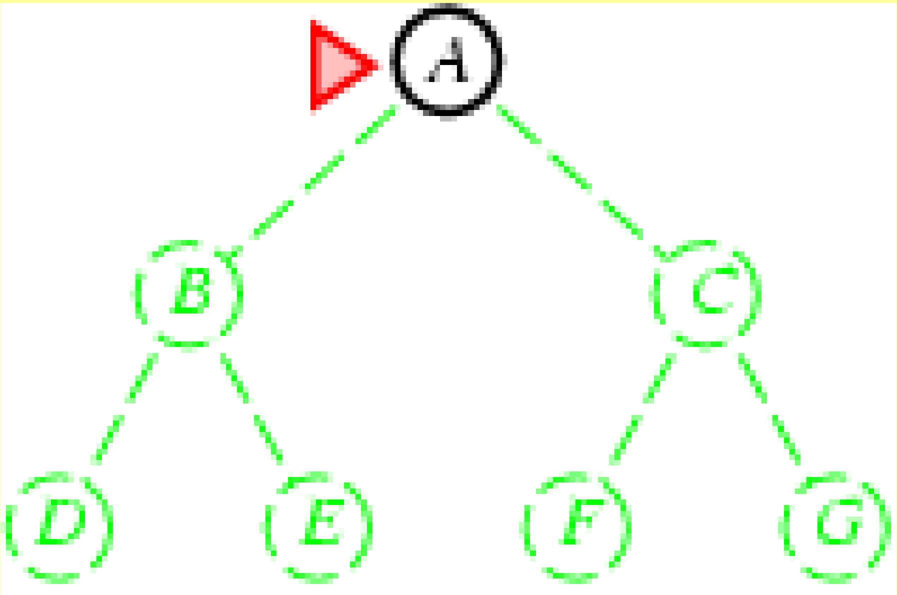
Uninformed search strategies

- Uninformed: While searching you have no clue whether one non-goal state is better than any other. Your search is blind.
- Various blind strategies:
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search

Breadth-first search

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

Is A a goal state?

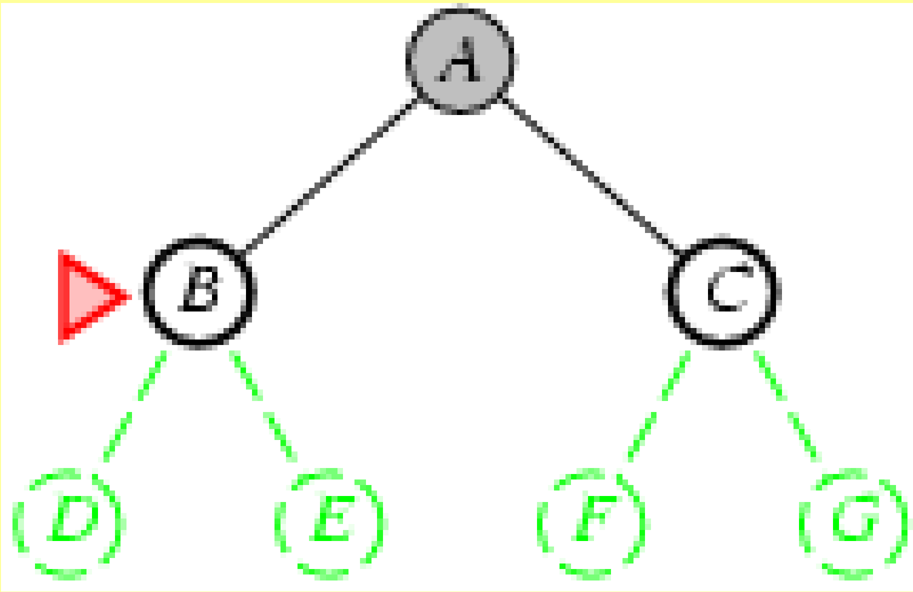


Breadth-first search

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

Expand:
fringe = [B,C]

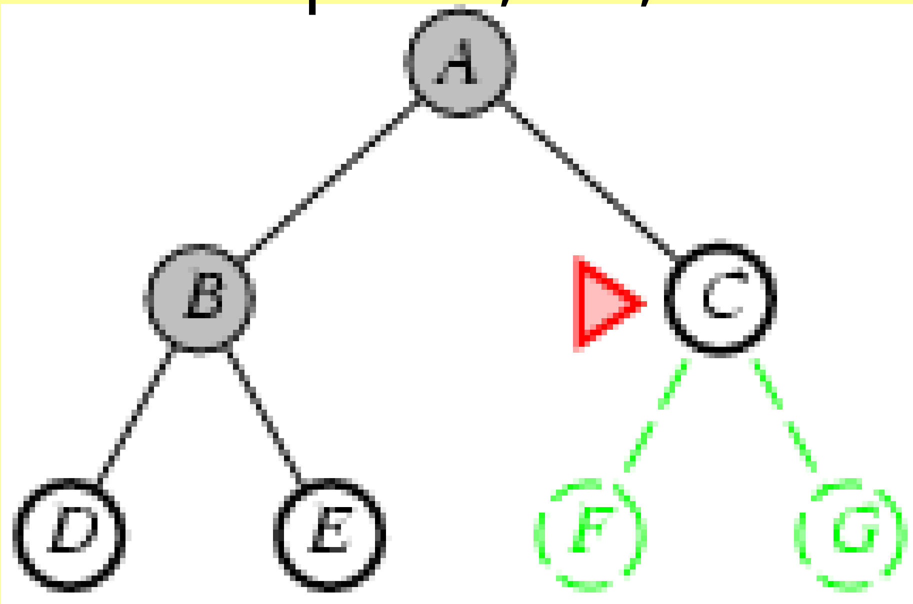
Is B a goal state?



Breadth-first search

Expand shallowest unexpanded node

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



Expand:
fringe=[C,D,E]

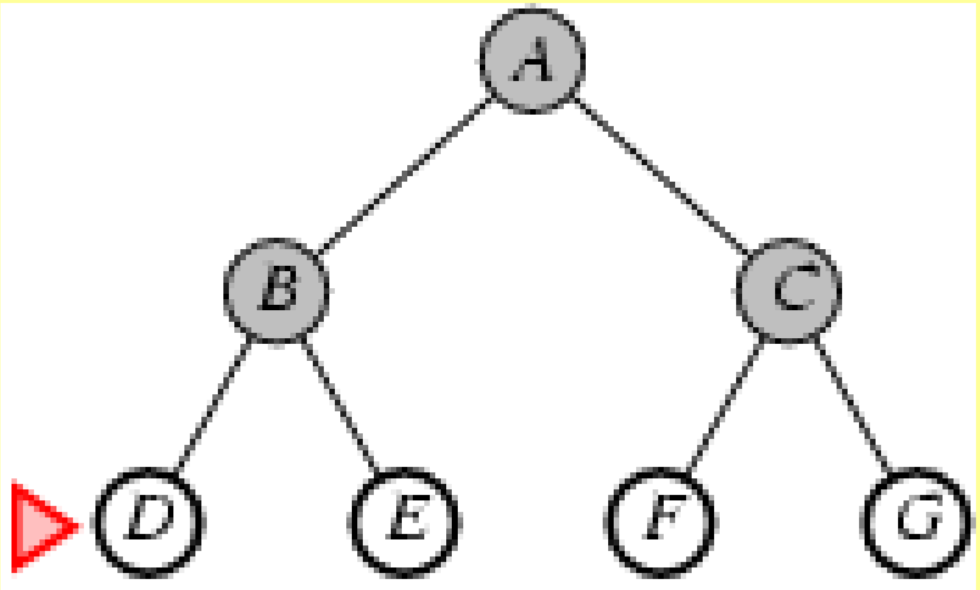
Is C a goal state?

Breadth-first search

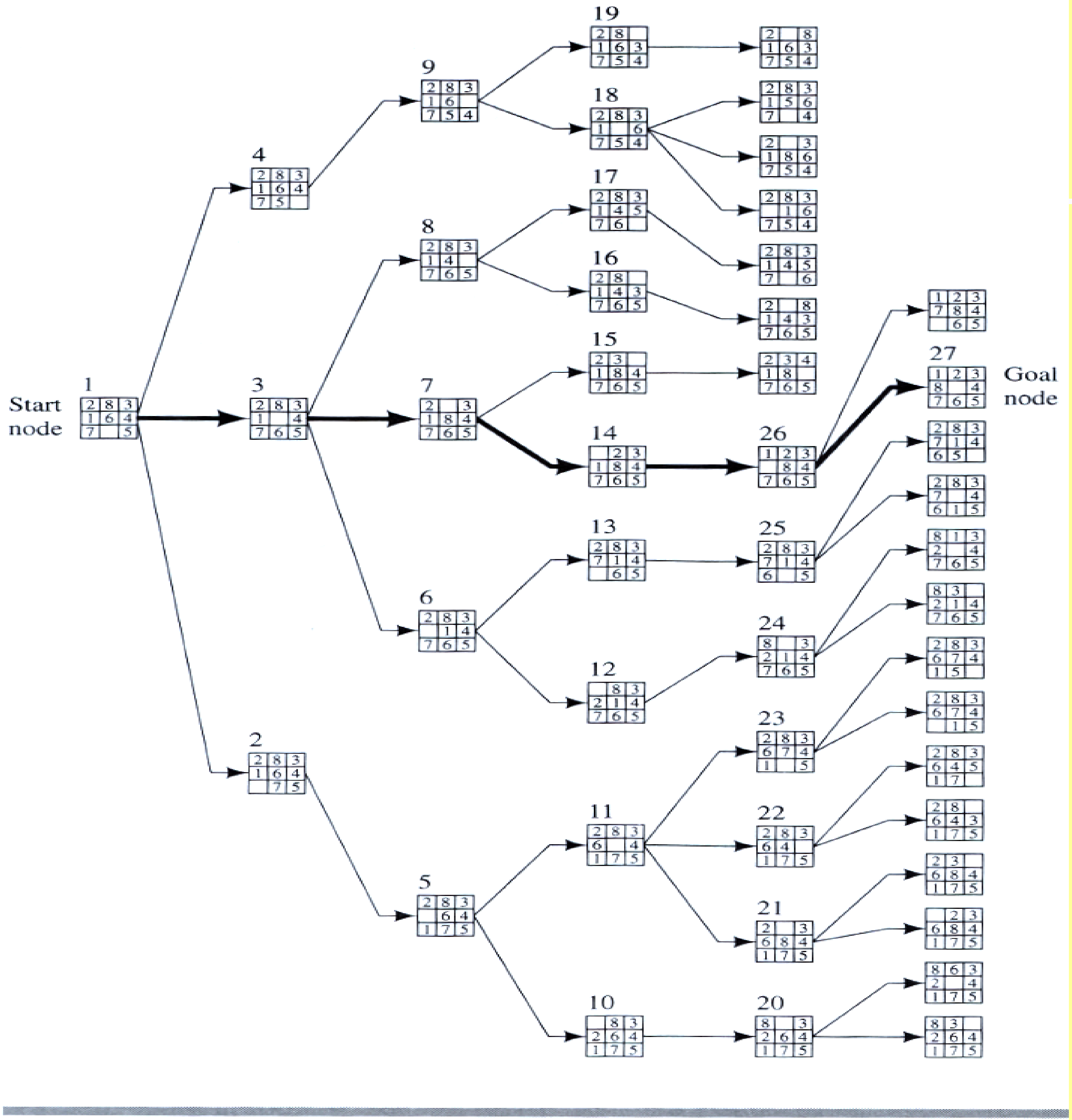
- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[D,E,F,G]

Is D a goal state?



Example BFS


Start node
Goal node
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

Properties of breadth-first search

- Complete? Yes it always reaches goal (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + (b^{d+1}-b)) = O(b^{d+1})$

  (this is the number of nodes we generate)
- Space? $O(b^{d+1})$ (keeps every node in memory,

  either in fringe or on a path to fringe).
- Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

- Space is the bigger problem (more than time)

Uniform-cost search

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant). Can we guarantee optimality for any step cost?

Uniform-cost Search: Expand node with smallest path cost g(n).

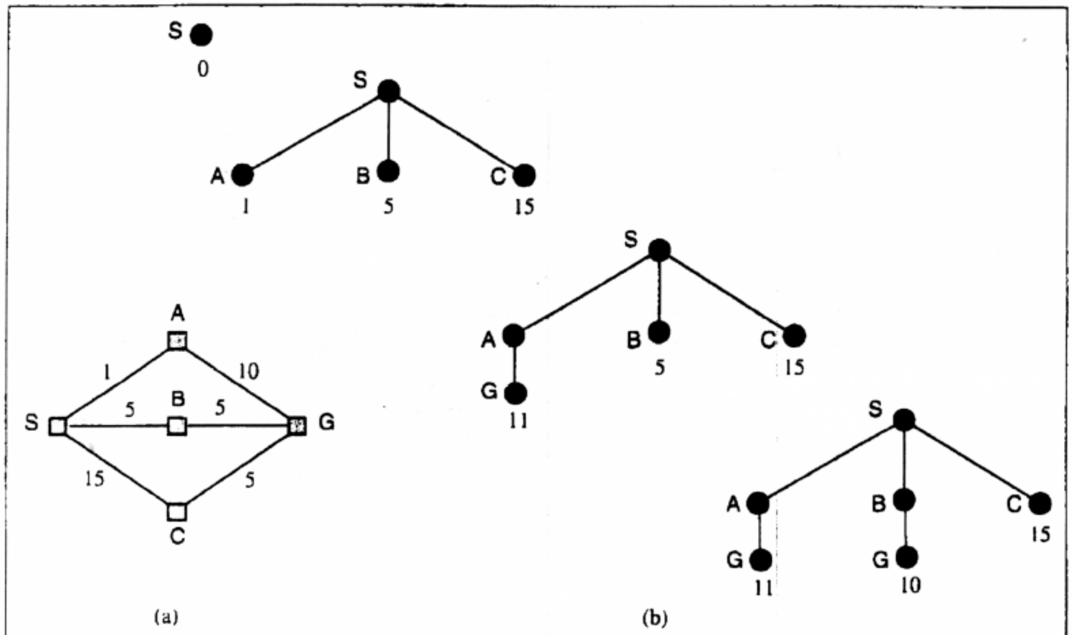


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost $\geqslant \varepsilon$
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* $\leqslant$ cost of optimal solution.

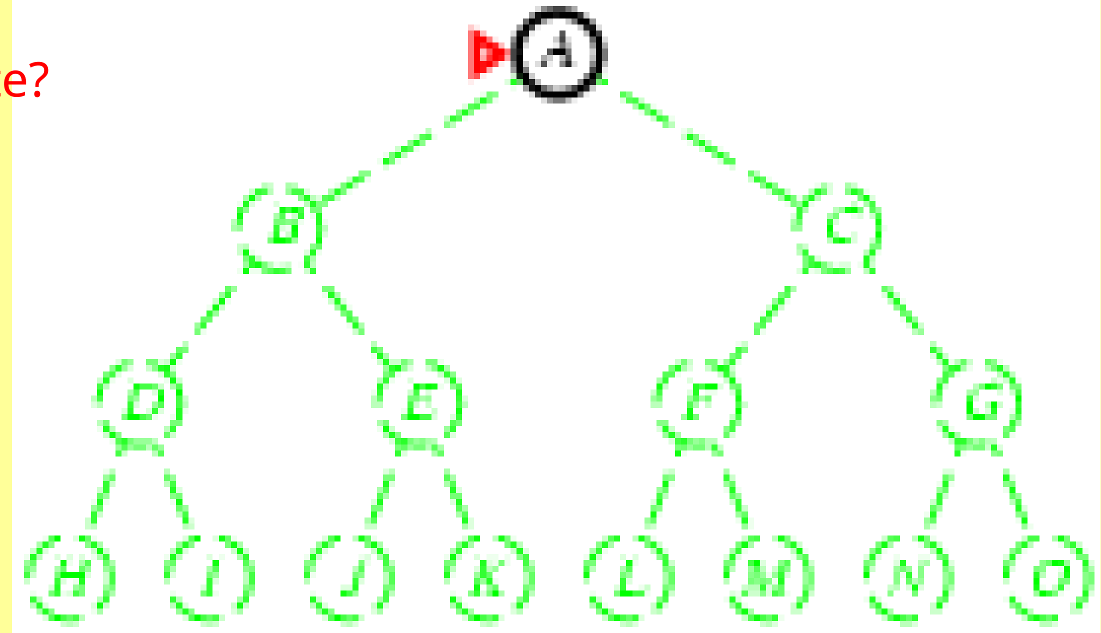
Space? # of nodes on paths with path cost $\leqslant$ cost of optimal solution.

Optimal? Yes, for any step cost.

Depth-first search

- Expand *deepest* unexpanded node
- Implementation:
  - *fringe* = Last In First Out (LIPO) queue, i.e., put successors at front

Is A a goal state?



Depth-first search

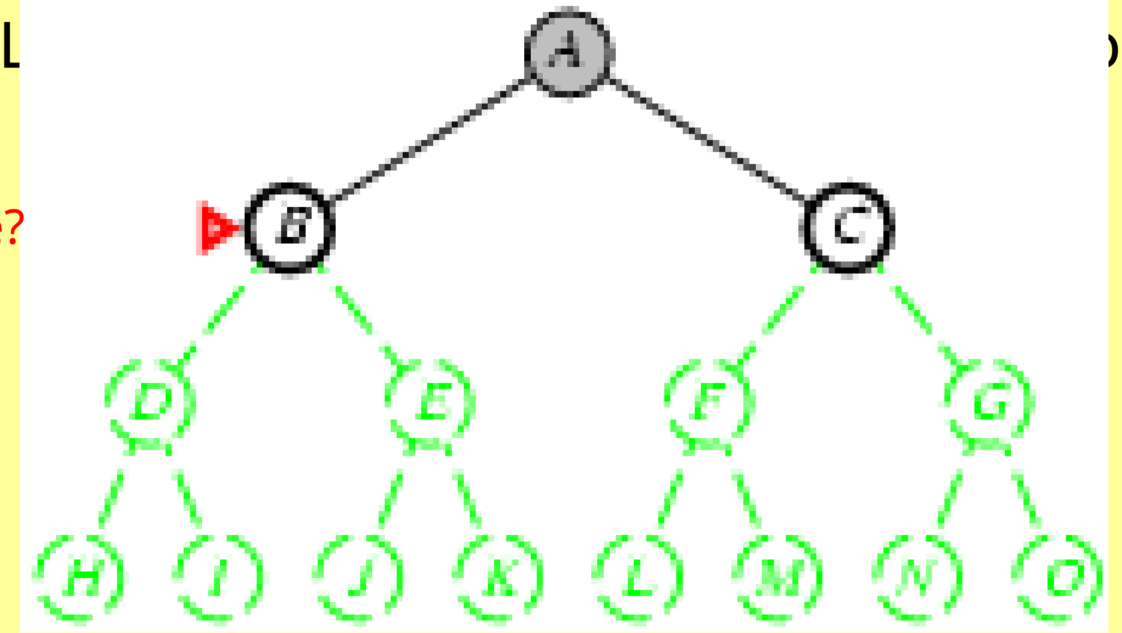
Expand deepest unexpanded node

-

- Implementation:
- *fringe* = L ont

queue=[B,C]

Is B a goal state?



Depth-first search

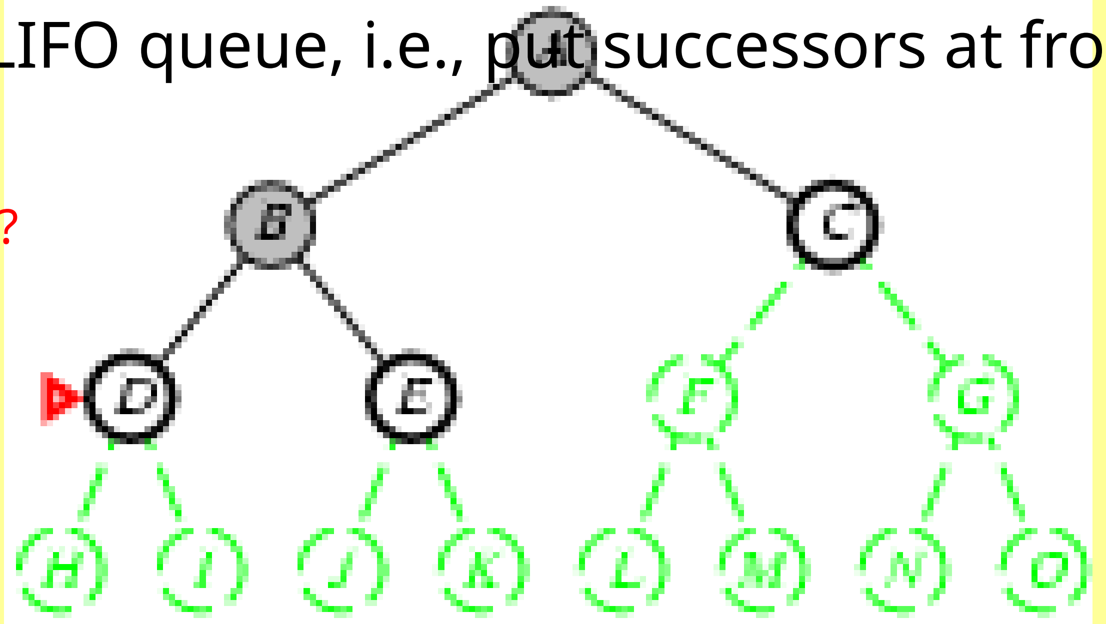
Expand deepest unexpanded node

-

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

queue=[D,E,C]

Is D = goal state?



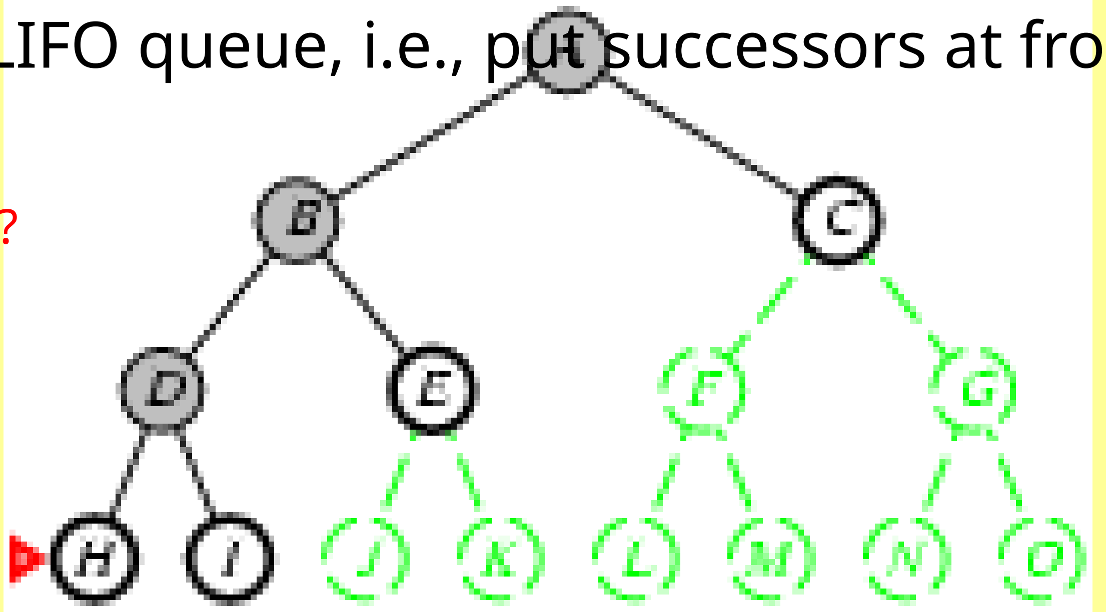
Depth-first search

Expand deepest unexpanded node

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

queue=[H,I,E,C]

Is H = goal state?



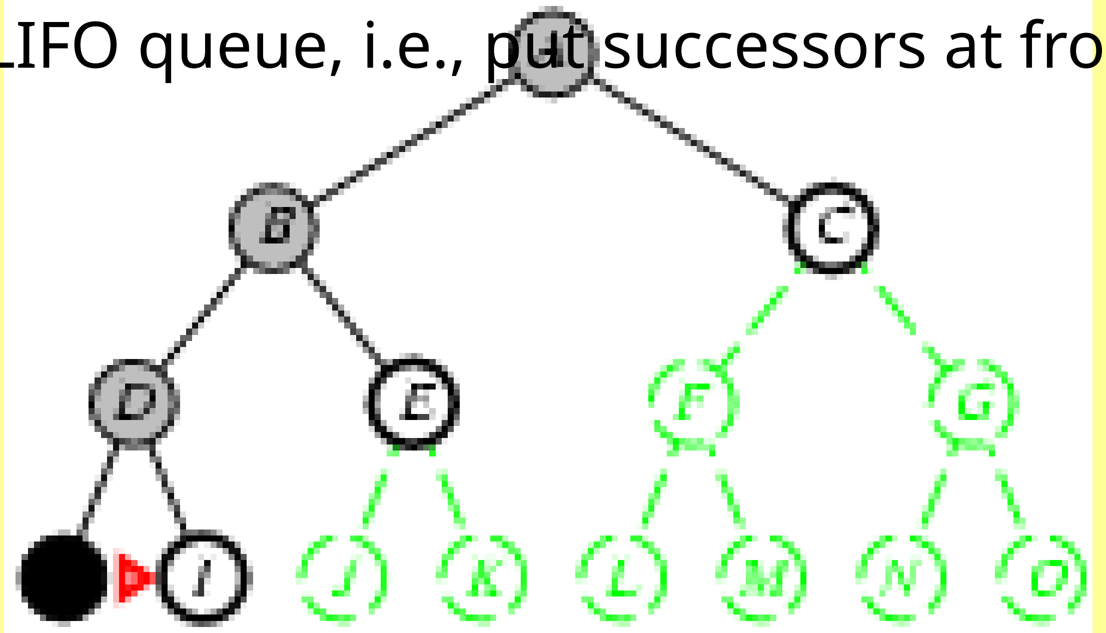
Depth-first search

Expand deepest unexpanded node

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

queue=[I,E,C]

Is I = goal state?



Depth-first search

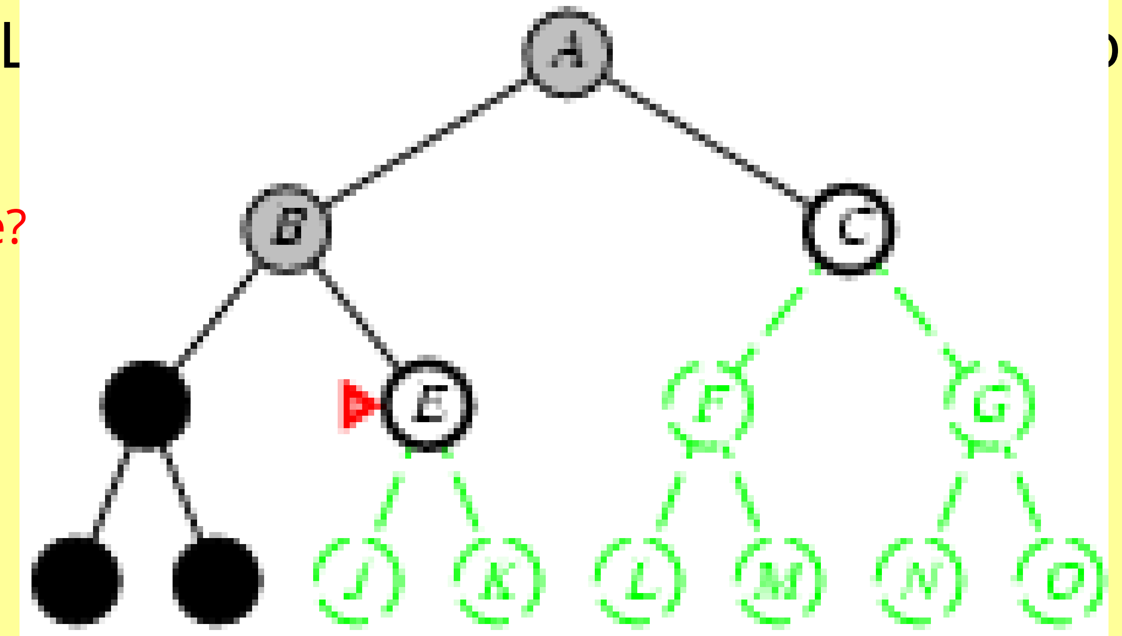
Expand deepest unexpanded node

-

- Implementation:
- *fringe* = L   ont

queue=[E,C]

Is E = goal state?



Depth-first search

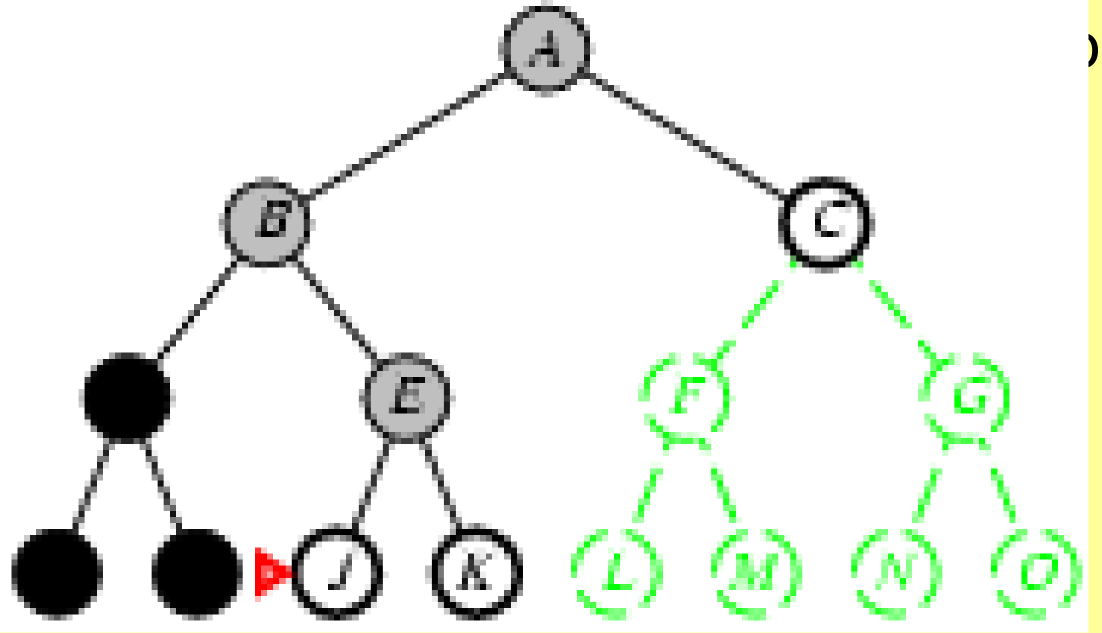
Expand deepest unexpanded node

-

- Implementation:
- *fringe* = L ... ont

queue=[J,K,C]

Is J = goal state?



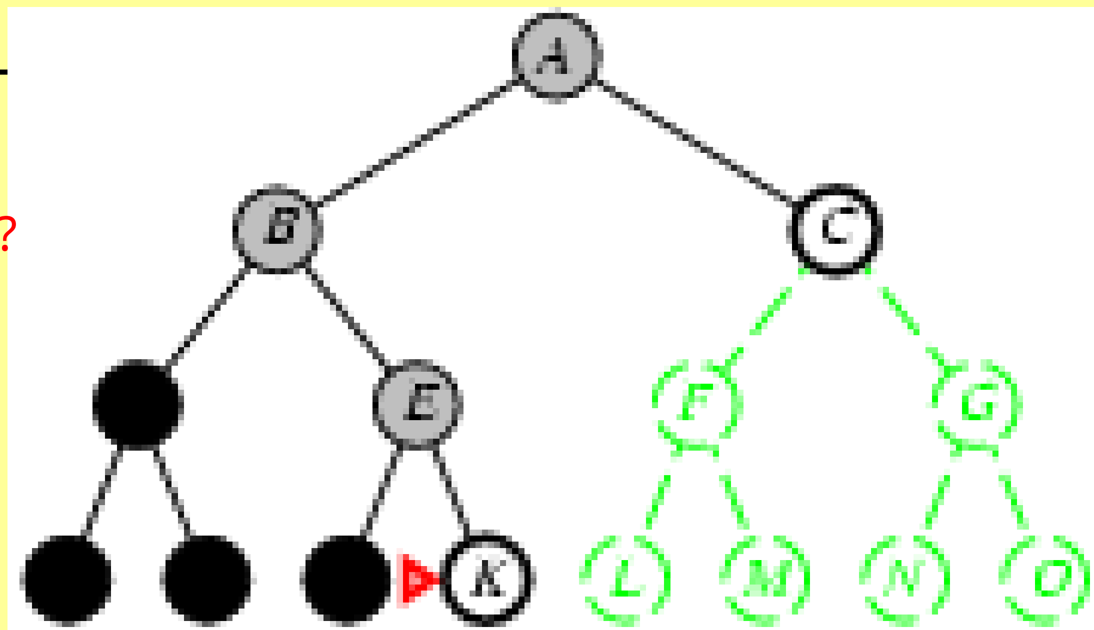
Depth-first search

Expand deepest unexpanded node

-
- Implementation:
- *fringe* = L … ont

queue=[K,C]

Is K = goal state?



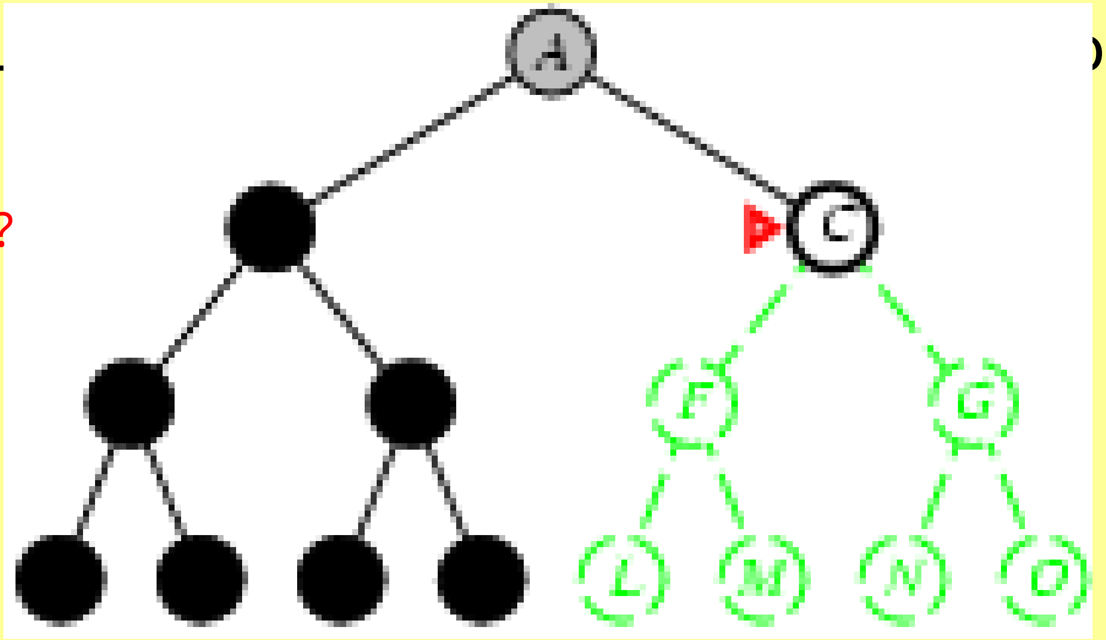
Depth-first search

Expand deepest unexpanded node

- Implementation:
- *fringe* = L ... ont

queue=[C]

Is C = goal state?



Depth-first search

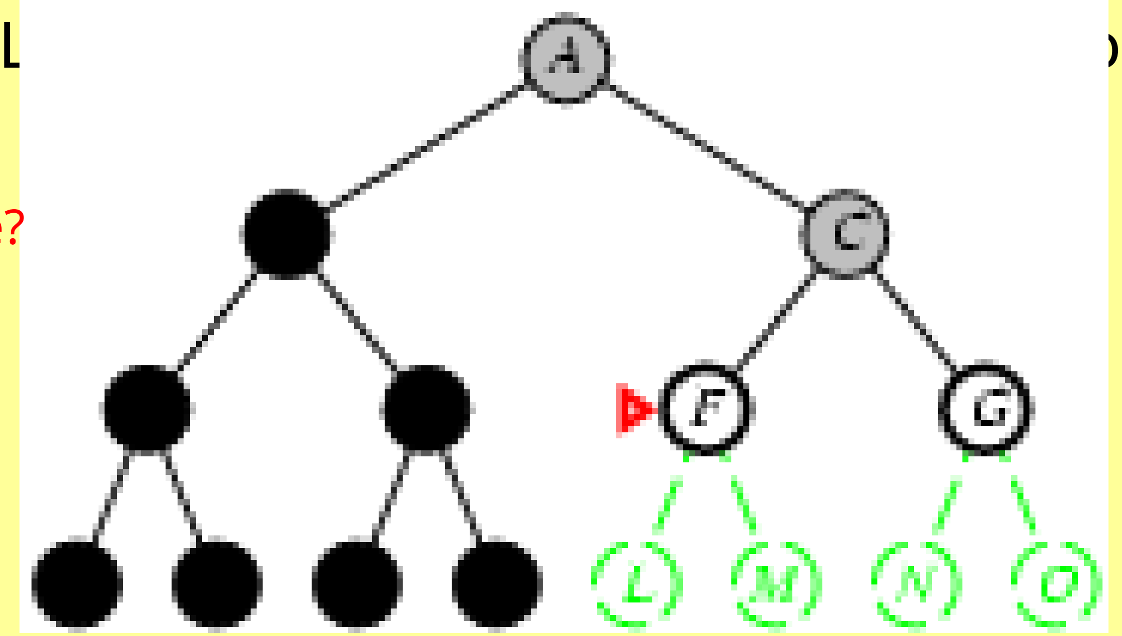
Expand deepest unexpanded node

-

- Implementation:
- *fringe* = L

queue=[F,G]

Is F = goal state?



Depth-first search

Expand deepest unexpanded node

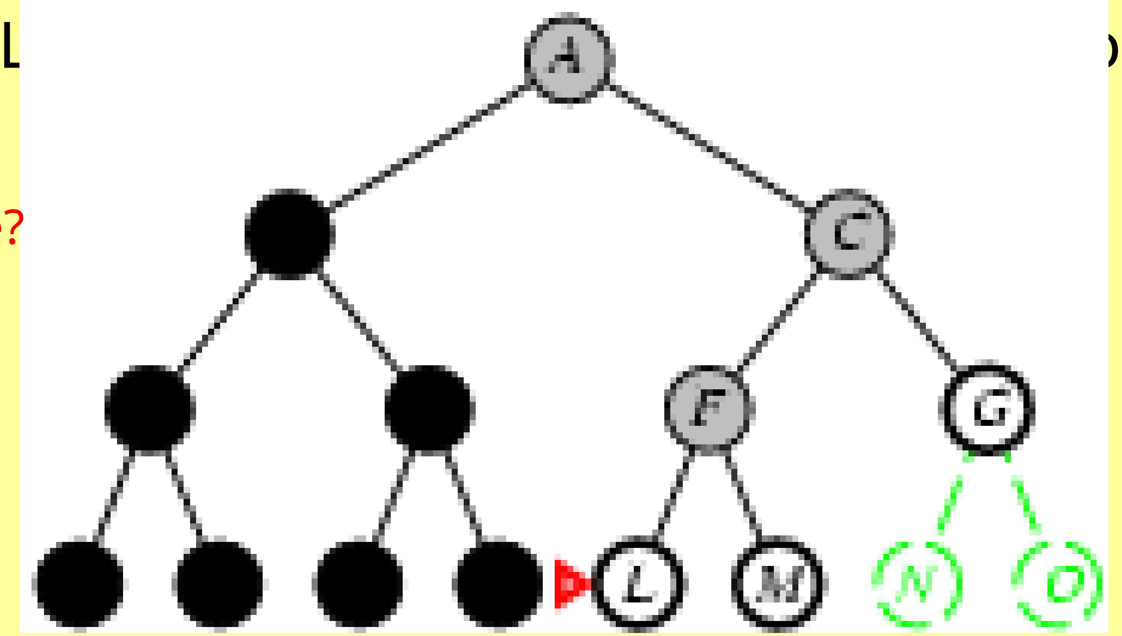
-

- Implementation:

- *fringe* = L ... ont

queue=[L,M,G]

Is L = goal state?



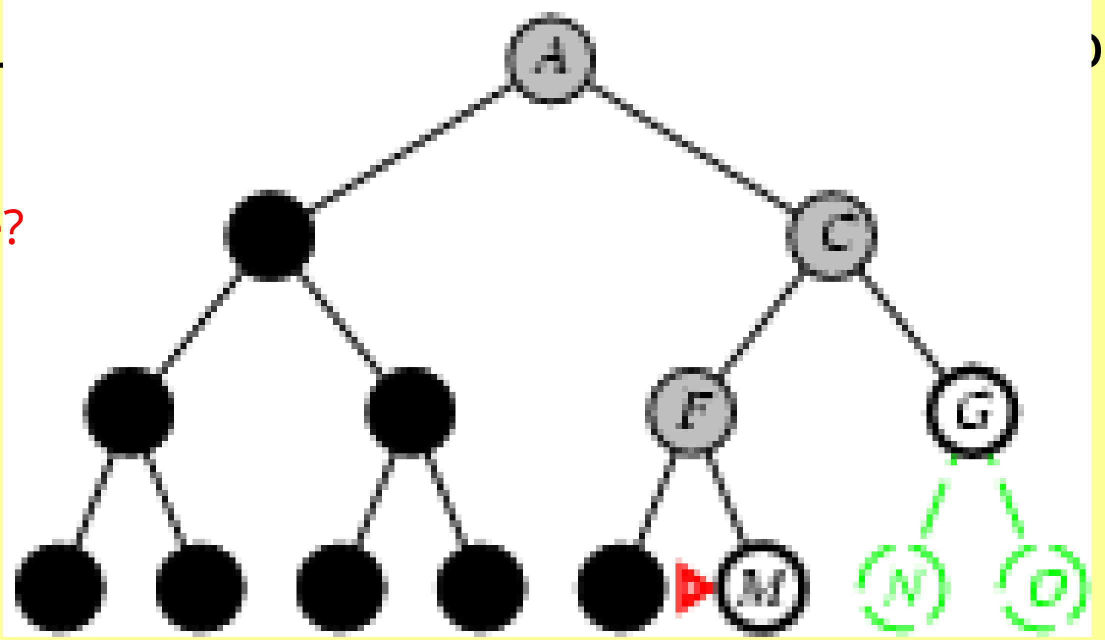
Depth-first search

Expand deepest unexpanded node

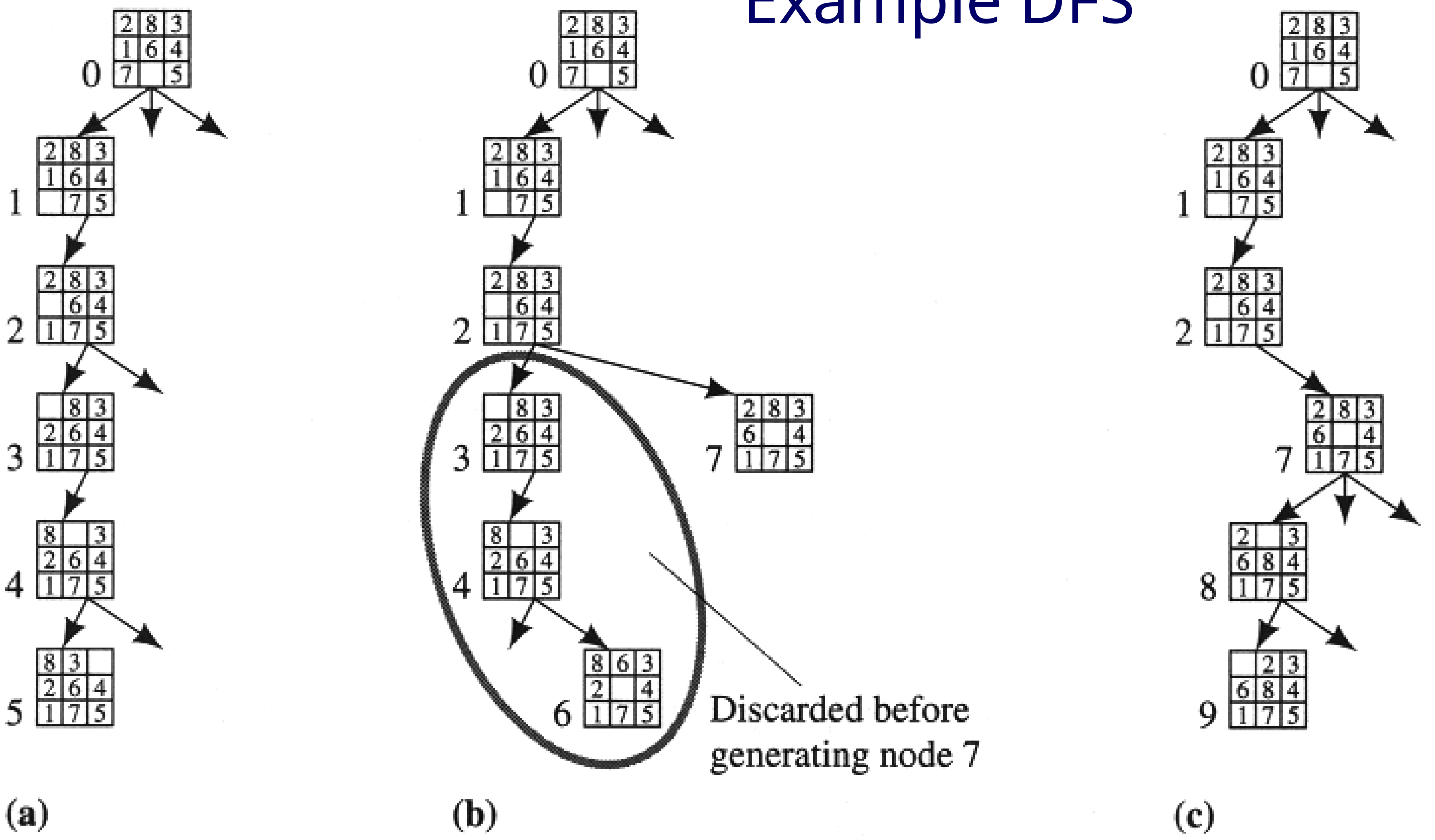
-
- Implementation:
- *fringe* = L …ont

queue=[M,G]

Is M = goal state?



Example DFS



Generation of the First Few Nodes in a Depth-First Search

Properties of depth-first search

- Complete? No: fails in infinite-depth spaces

  Can modify to avoid repeated states along path
- Time? $O(b^m)$ with m=maximum depth
- terrible if *m* is much larger than *d*
  - but if solutions are dense, may be much faster than

    breadth-first
- Space? *O(bm),* i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)
- Optimal? No (It may find a non-optimal goal first)

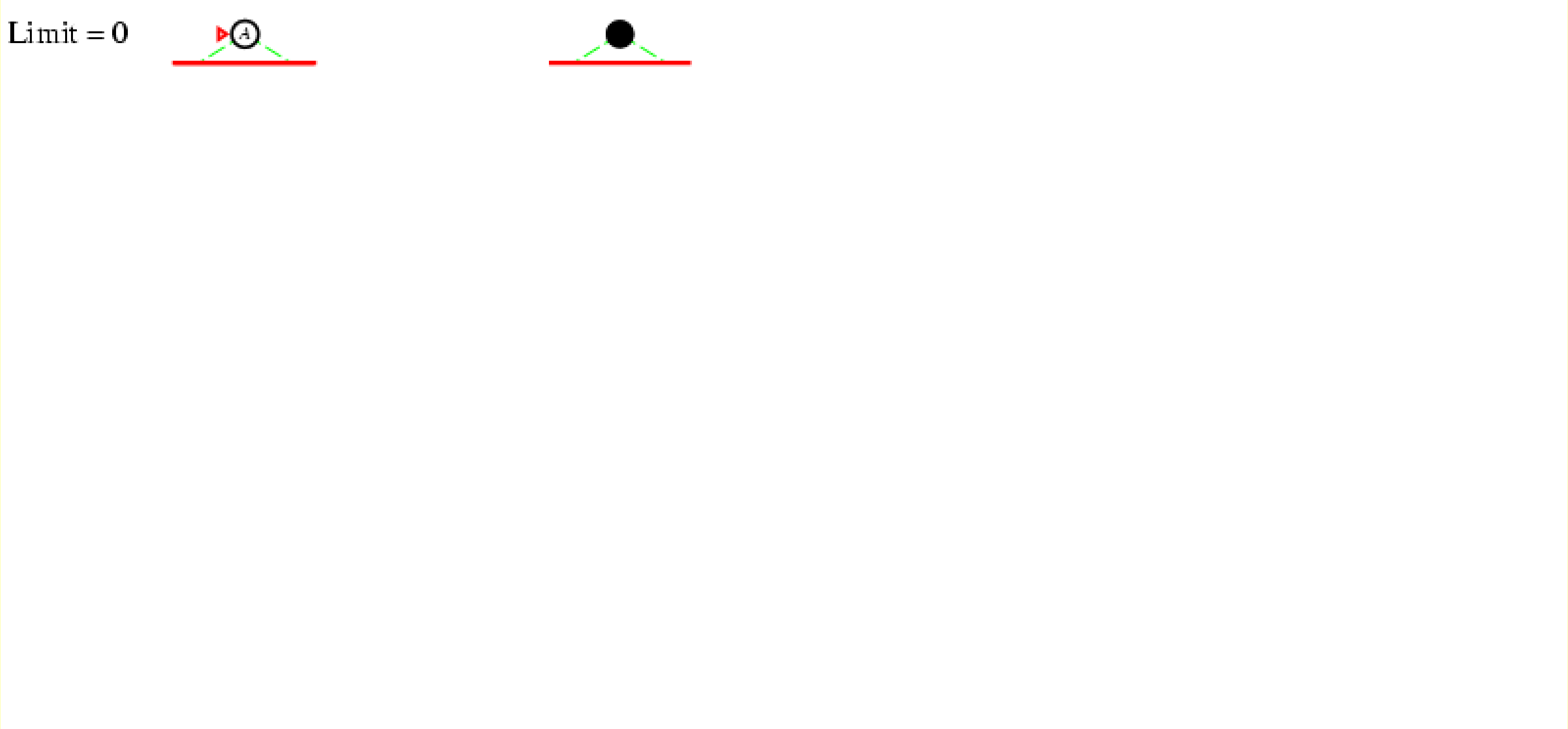
Iterative deepening search

- To avoid the infinite depth problem of DFS, we can decide to only search until depth L, i.e. we don’t expand beyond depth L.
  ☾ Depth-Limited Search

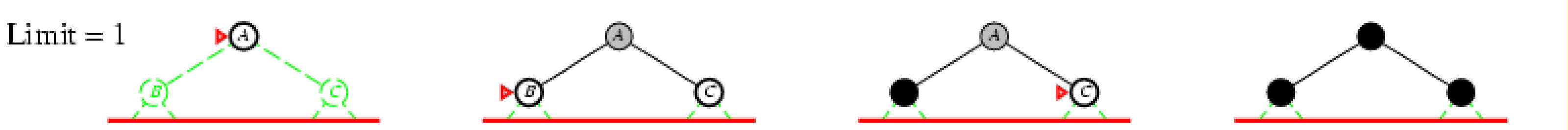
- What of solution is deeper than L? ☾ Increase L iteratively.
  ☾ Iterative Deepening Search

- As we shall see: this inherits the memory advantage of Depth-First search.

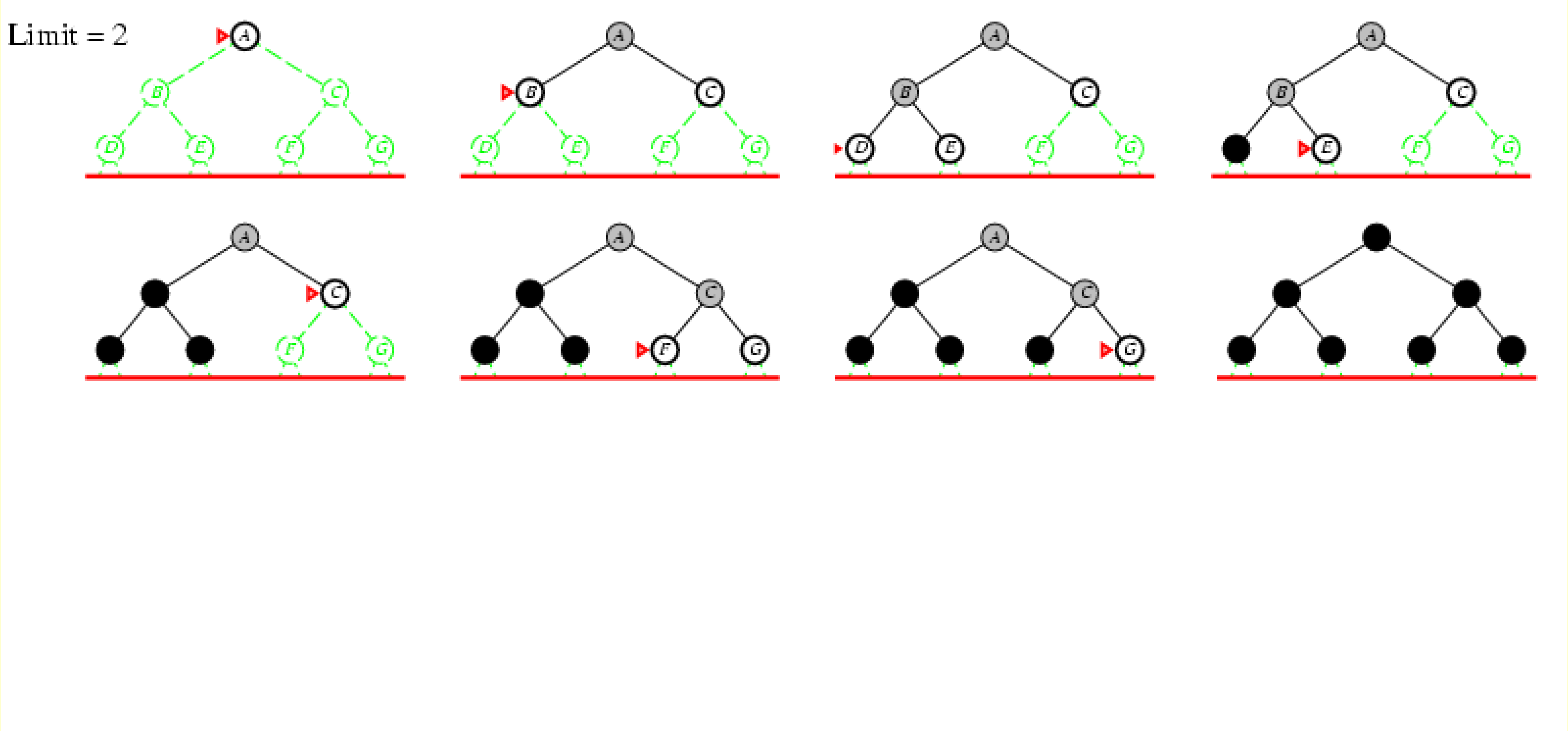
Iterative deepening search L=0



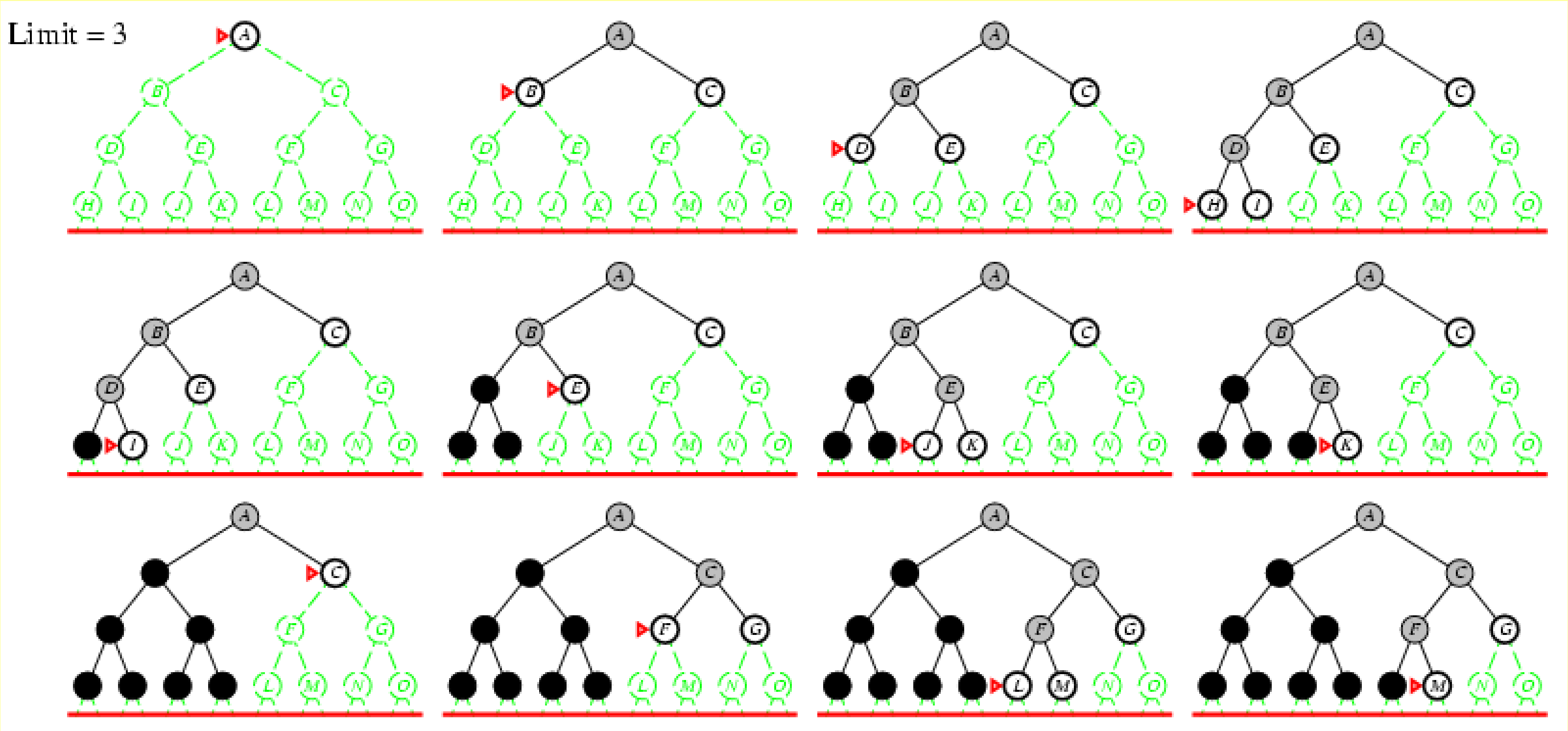
Iterative deepening search *L*=1



Iterative deepening search *l*=2



Iterative deepening search *l* =3



Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + 3b^{d-2} + 2b^{d-1} + 1b^d =$$

$$O(b^d) \neq O(b^{d+1})$$

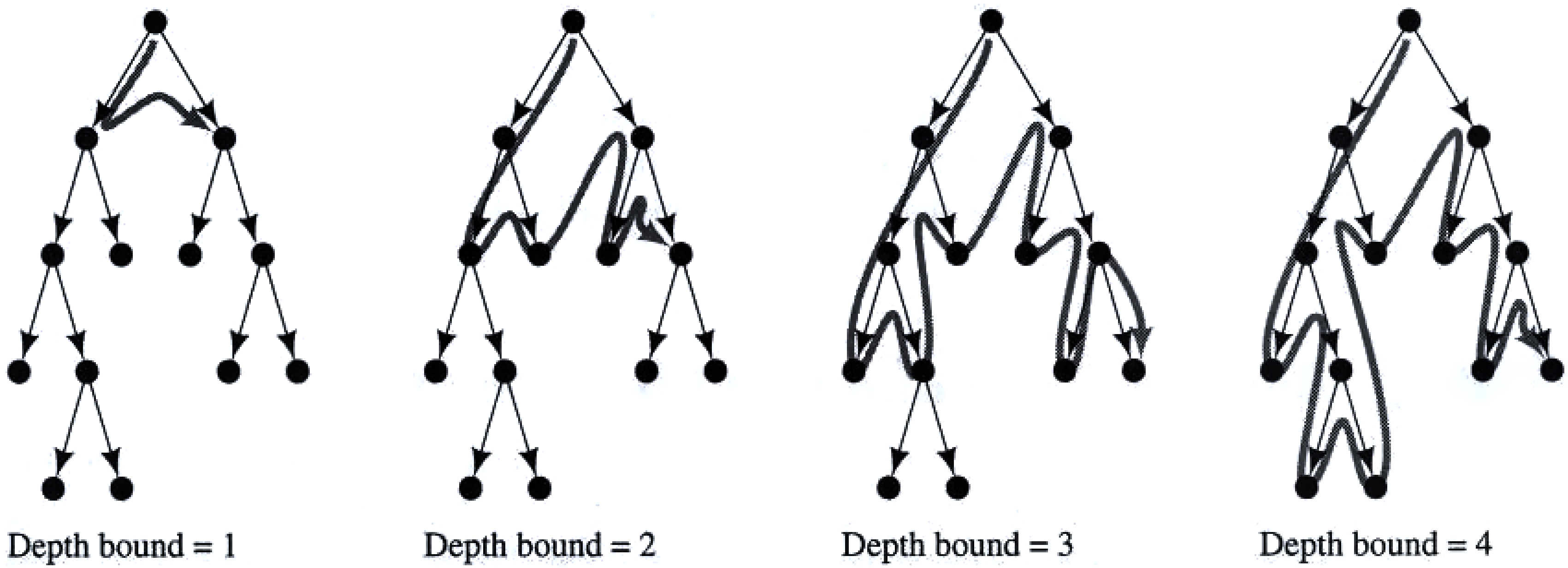
BFS

- For $b = 10$, $d = 5$,
  - N_{DLS} = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - N_{IDS} = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450
  - N_{BFS} = ... = 1,111,100

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

Example IDS



Stages in Iterative-Deepening Search

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both “meet in the middle”
  - need to keep track of the intersection of 2 open sets of nodes
- What does searching backwards from G mean
  - need a way to specify the predecessors of G
    - this can be difficult,
    - e.g., predecessors of checkmate in chess?
  - what if there are multiple goal states?
  - what if there is only a goal test, no explicit list?

Bi-Directional Search

Complexity: time and space complexity are: $O(b^{d/2})$

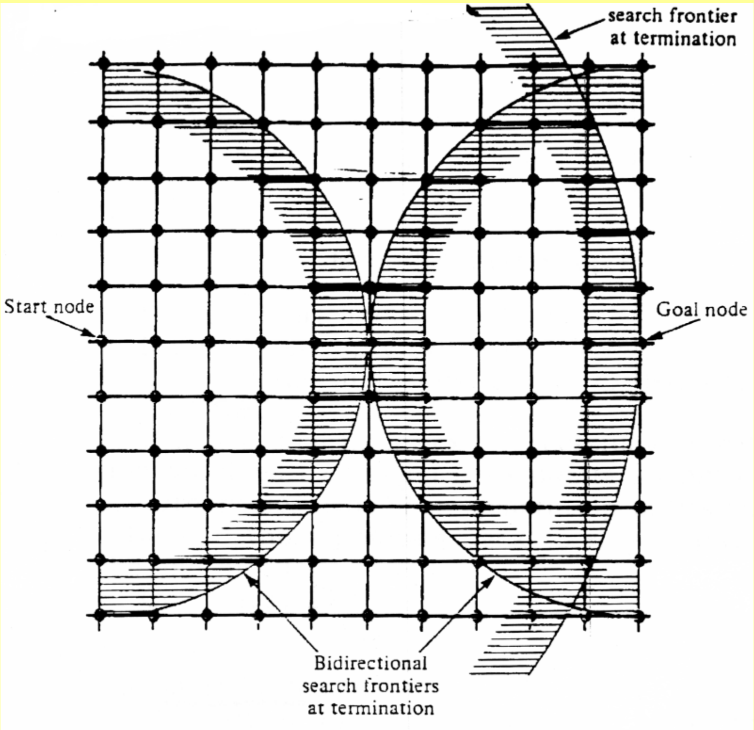


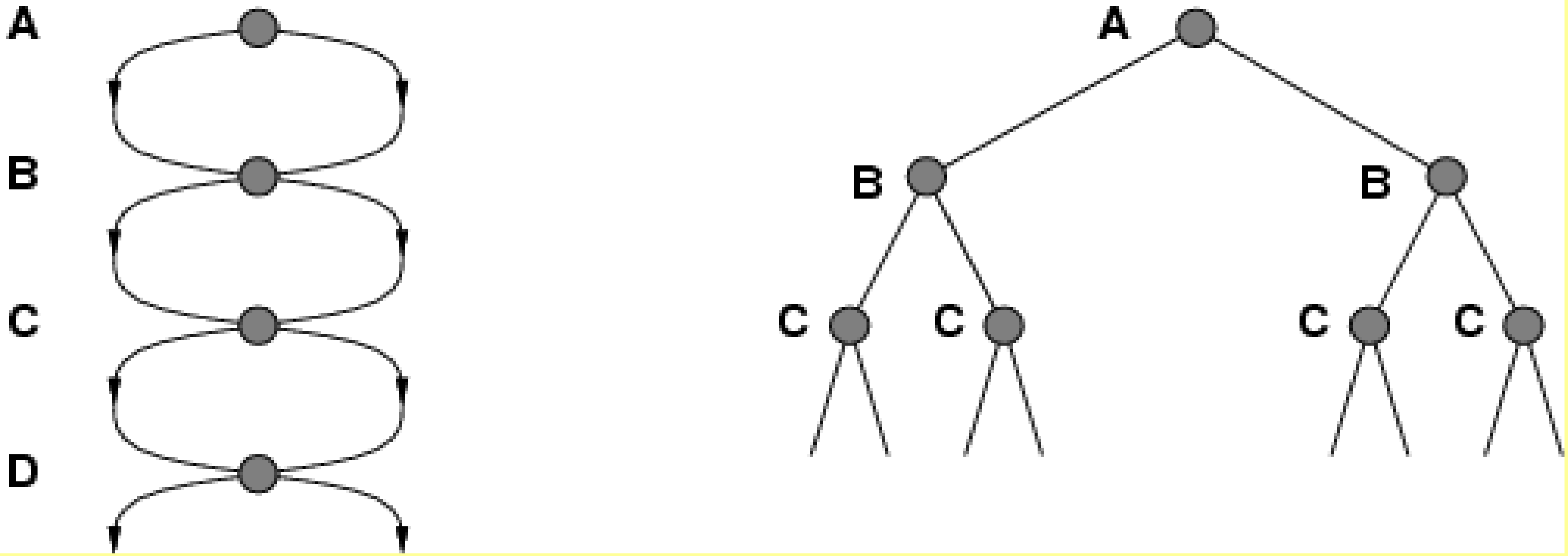
Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Summary of algorithms

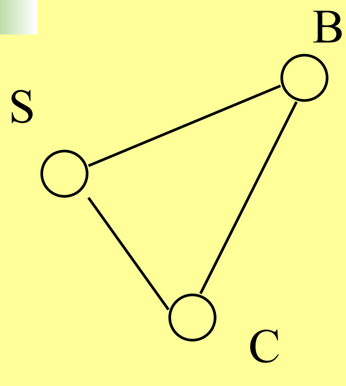
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Repeated states

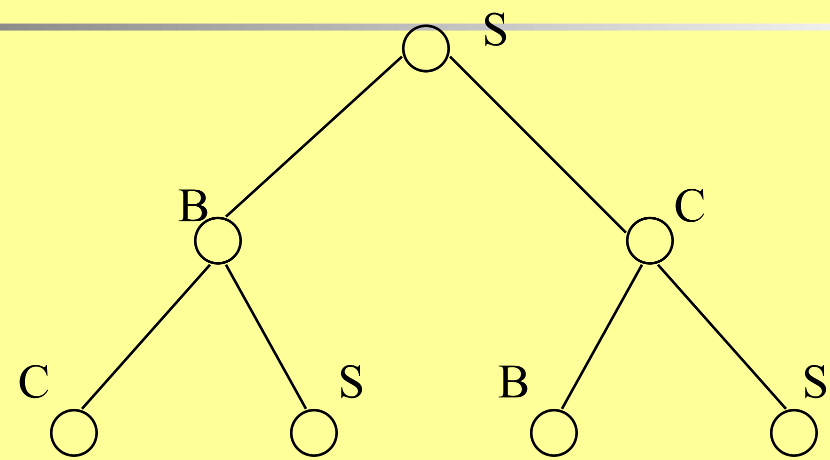
- Failure to detect repeated states can turn a linear problem into an exponential one!



Solutions to Repeated States



State Space



Example of a Search Tree

- Method 1 ← suboptimal but practical
  - do not create paths containing cycles (loops)
- Method 2 ← optimal but memory inefficient
  - never generate a state generated before
    - must keep track of all possible states (uses a lot of memory)
    - e.g., 8-puzzle problem, we have 9! = 362,880 states

Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

http://www.cs.rmit.edu.au/AI-Search/Product/

http://aima.cs.berkeley.edu/demos.html (for more demos)