Uninformed (also called blind) search algorithms)

This Lecture

Chapter 3.1-3.4

Next Lecture

Chapter 3.5-3.7

(Please read lecture topic material before and after each lecture on that topic)

Outline

- Overview of uninformed search methods
- Search strategy evaluation
  - Complete? Time? Space? Optimal?
  - Max branching (b), Solution depth (d), Max depth (m)
- Search Strategy Components and Considerations
  - Queue? Goal Test when? Tree search vs. Graph search?
- Various blind strategies:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Iterative deepening search (generally preferred)
  - Bidirectional search (preferred if applicable)

Uninformed search strategies

- Uninformed (blind):
  - You have no clue whether one non-goal state is better than any other. Your search is blind. You don’t know if your current exploration is likely to be fruitful.
- Various blind strategies:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Iterative deepening search (generally preferred)
  - Bidirectional search (preferred if applicable)

Search strategy evaluation

- A search strategy is defined by the order of node expansion

- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?

- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

Uninformed search strategies

- Queue for Frontier:
  - FIFO? LIFO? Priority?
- Goal-Test:
  - When inserted into *Frontier*? When removed?
- Tree Search or Graph Search:
  - Forget *Explored* nodes? Remember them?

Queue for Frontier

- FIFO (First In, First Out)
  - Results in Breadth-First Search
- LIFO (Last In, First Out)
  - Results in Depth-First Search
- Priority Queue sorted by path cost so far
  - Results in Uniform Cost Search
- Iterative Deepening Search uses Depth-First
- Bidirectional Search can use either Breadth-First or Uniform Cost Search

When to do Goal-Test? When generated? When popped?

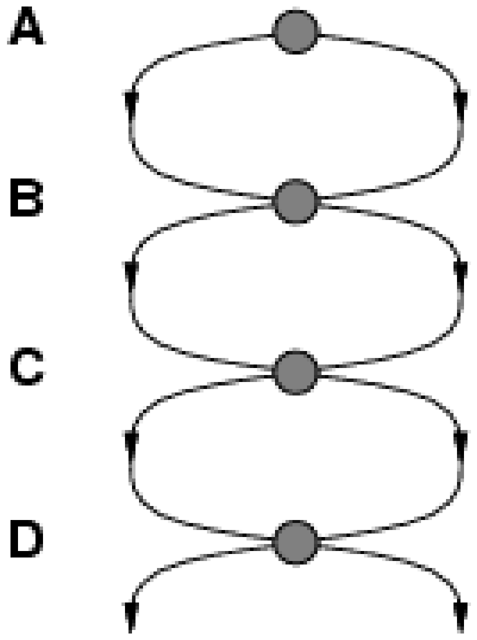
- Do Goal-Test when node is popped from queue

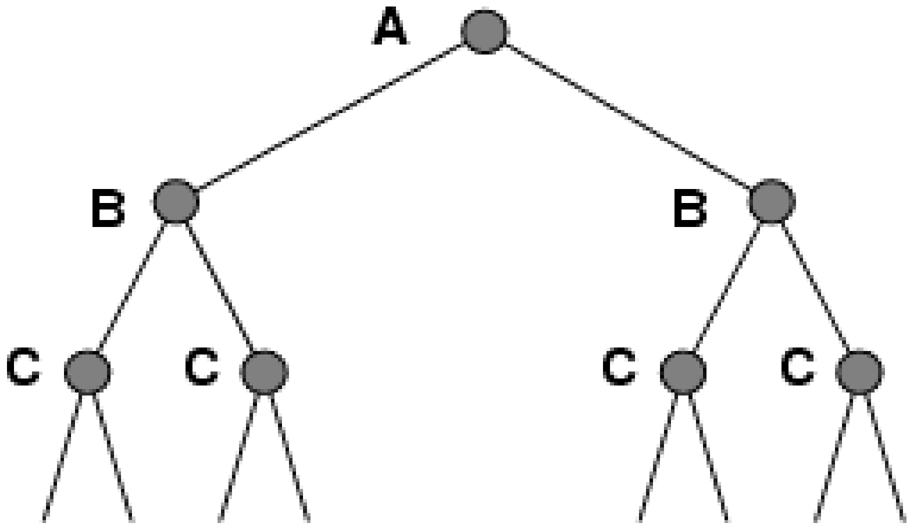
  IF you care about finding the optimal path

  AND your search space may have both short expensive and long cheap paths to a goal.
  - Guard against a short expensive goal.
  - E.g., Uniform Cost search with variable step costs.
- Otherwise, do Goal-Test when is node inserted.
  - E.g., Breadth-first Search, Depth-first Search, or Uniform Cost search when cost is a non-decreasing function of depth only (which is equivalent to Breadth-first Search).
- REASON ABOUT your search space & problem.
  - How could I possibly find a non-optimal goal?

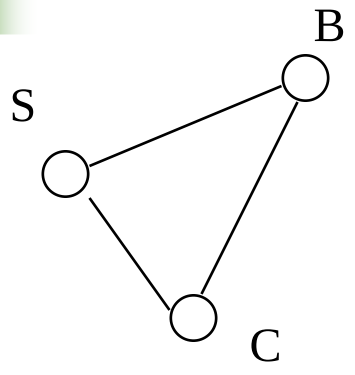
Repeated states

- Failure to detect repeated states can turn a linear problem into an exponential one!
- Test is often implemented as a hash table.

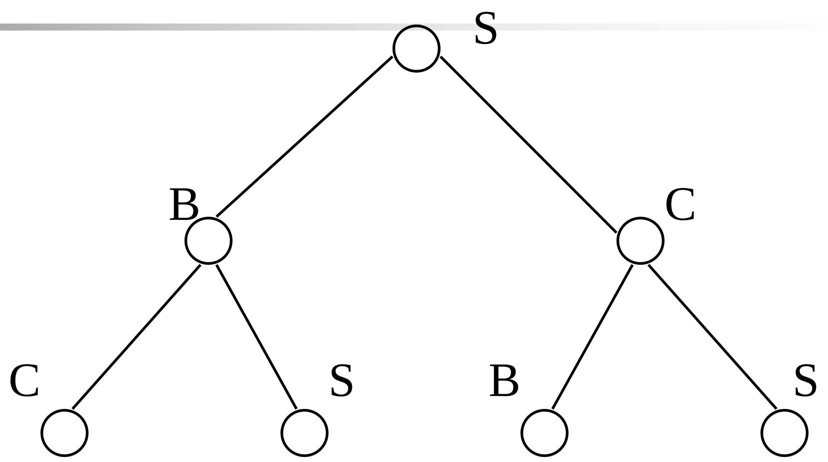




Solutions to Repeated States



State Space



Example of a Search Tree

Graph search ← faster but memory inefficient

- Never explore a state explored before
  - Must keep track of all possible states (a lot of memory)
    - E.g., 8-puzzle problem, we have 9! = 362,880 states
  - Memory-efficient approximation for DFS/DLS
    - Avoid states on path to root: avoid looping paths.
- Graph search optimality/completeness
  - Same as Tree search; just a space-time trade-off

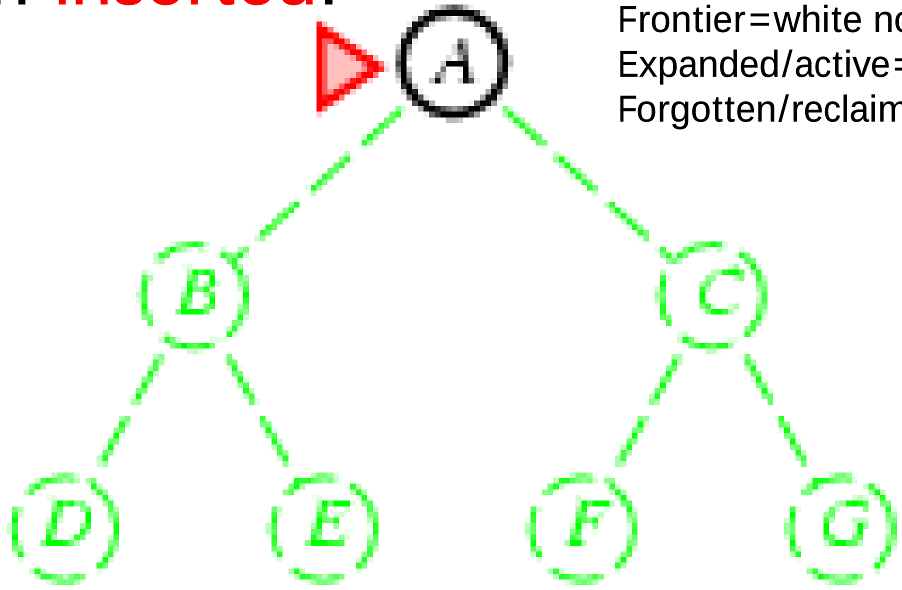
Breadth-first search

- Expand shallowest unexpanded node
- *Frontier* (or fringe): nodes in queue to be explored
- *Frontier* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.
- *Goal-Test* when inserted.

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Initial state = A
Is A a goal state?

Put A at end of queue.
frontier = [A]

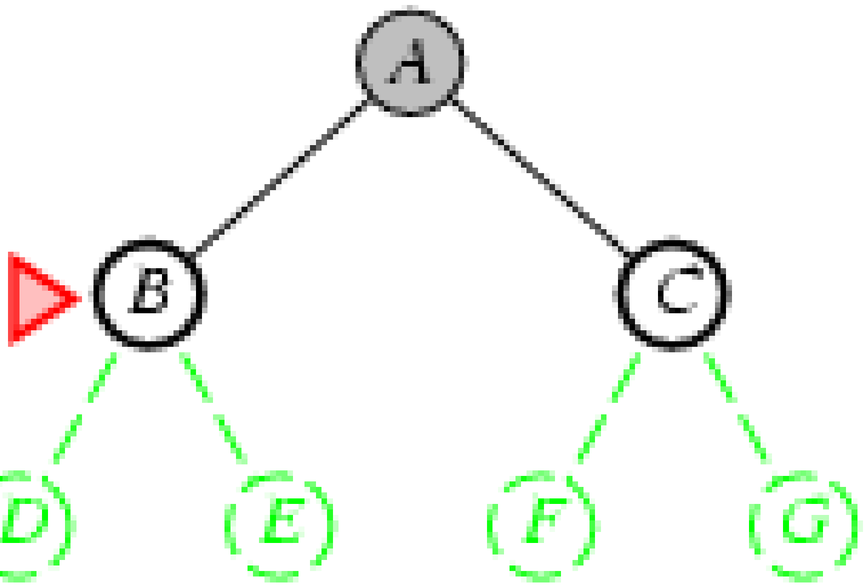


Breadth-first search

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

Expand A to B, C.
Is B or C a goal state?

Put B, C at end of queue.
frontier = [B,C]

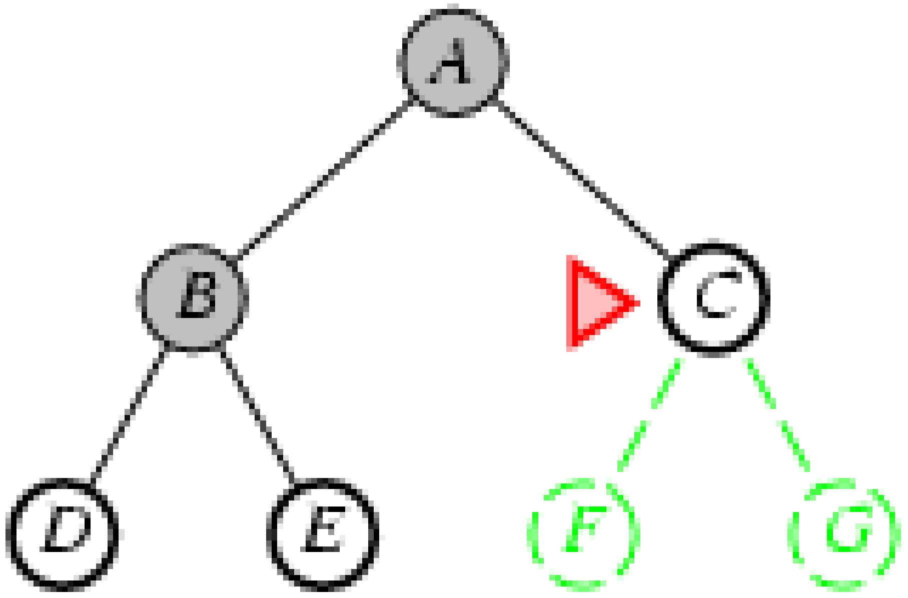


Breadth-first search

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

Expand B to D, E
Is D or E a goal state?

Put D, E at end of queue
frontier=[C,D,E]

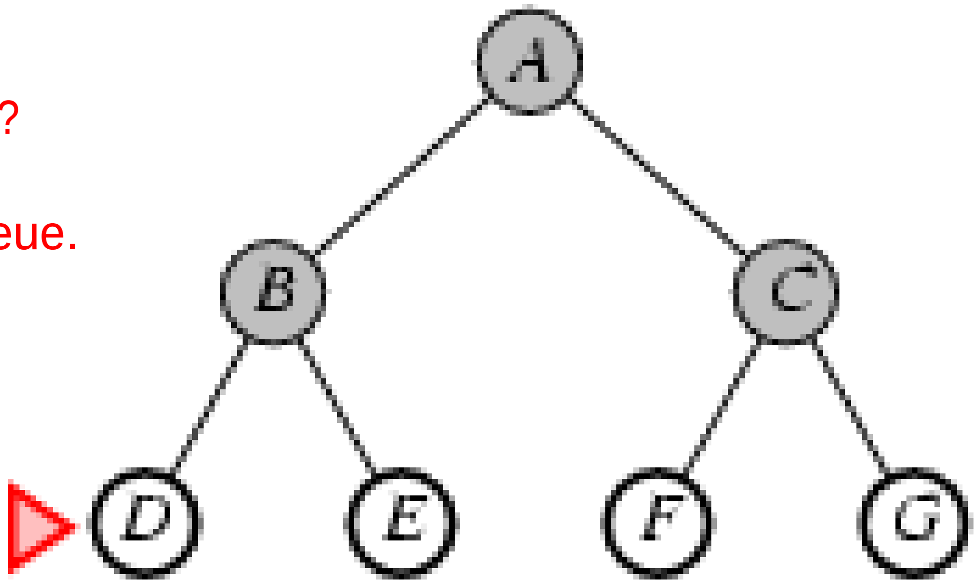


Breadth-first search

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

Expand C to F, G.
Is F or G a goal state?

Put F, G at end of queue.
frontier = [D,E,F,G]

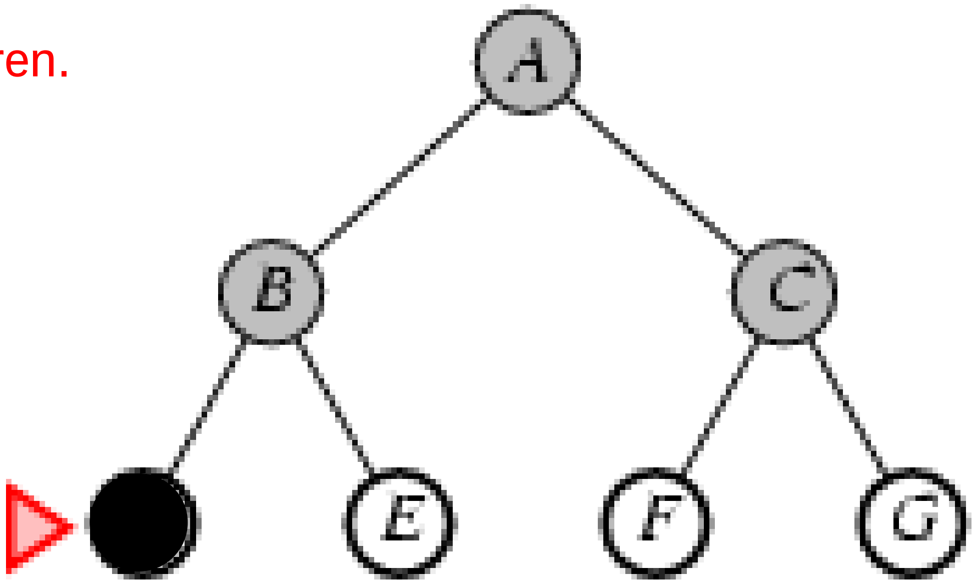


Breadth-first search

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

Expand D to no children.
Forget D.

frontier = [E,F,G]

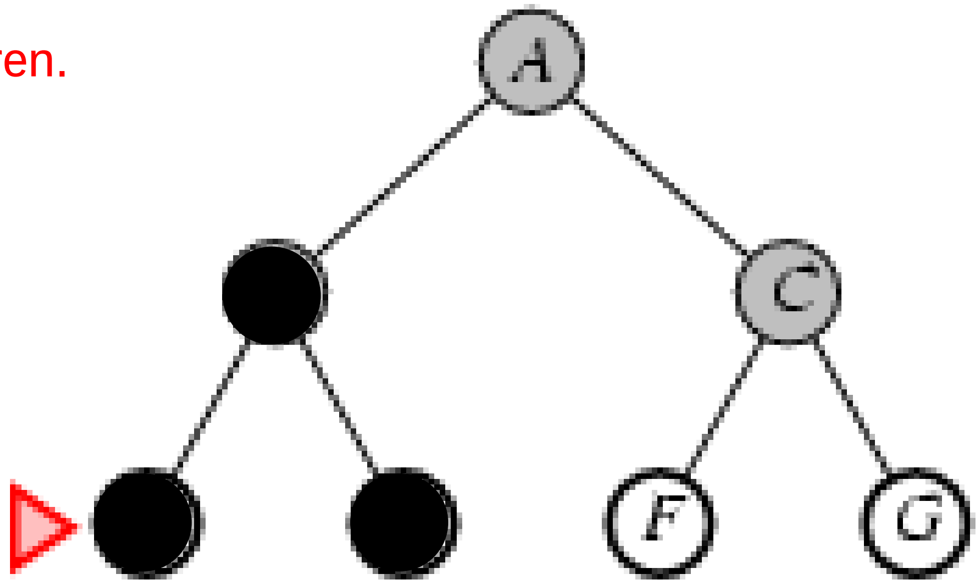


Breadth-first search

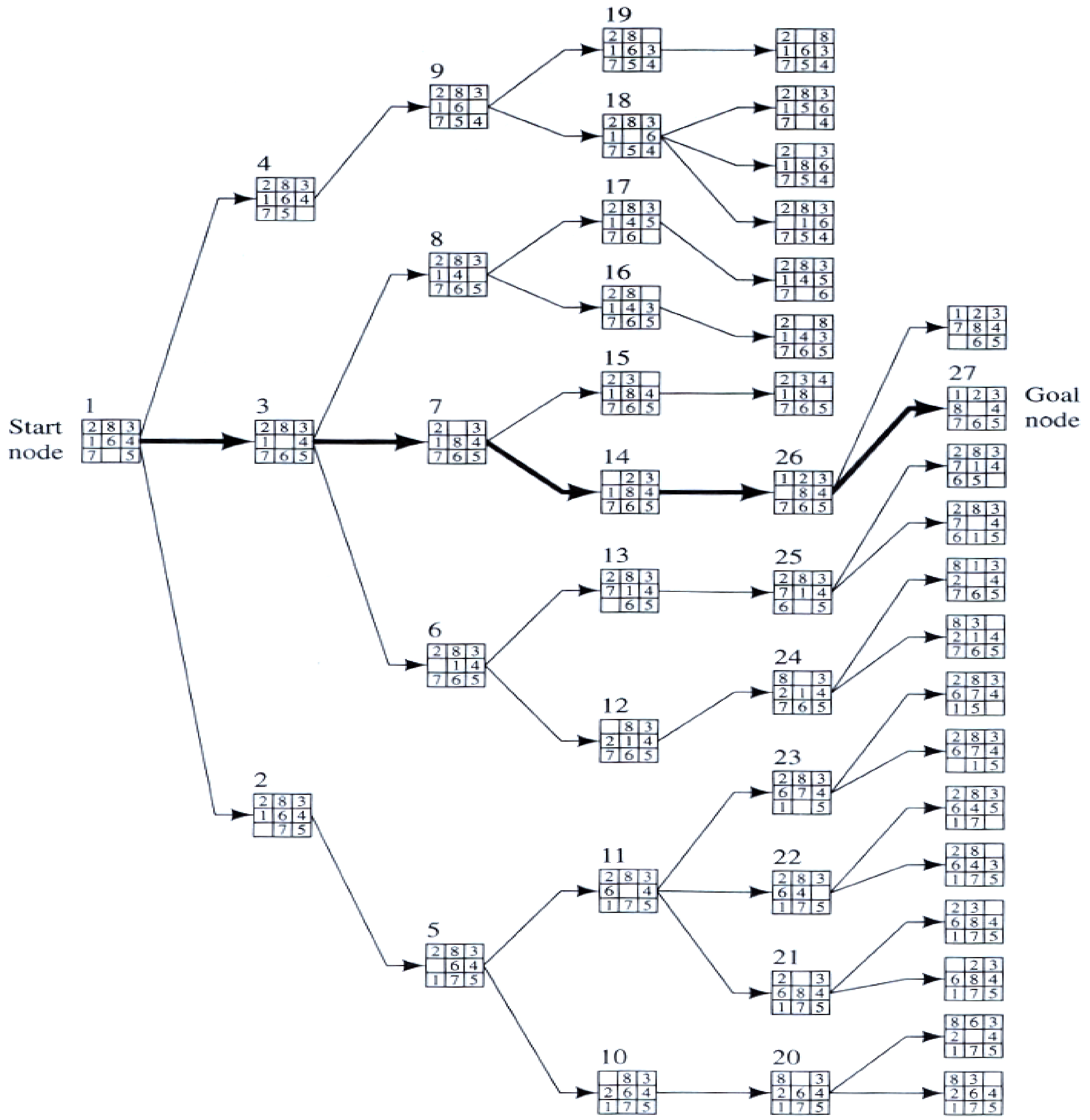
- Expand shallowest unexpanded node
- *Frontier* is a FIFO queue, i.e., new successors go at end

Expand E to no children.
Forget B,E.

frontier = [F,G]



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 a goal (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d = O(b^d)$
  (this is the number of nodes we generate)
- Space? $O(b^d)$ (keeps every node in memory, either in fringe or on a path to fringe).
- Optimal? No, for general cost functions.
  Yes, if cost is a non-decreasing function only of depth.
  - With $f(d) \geq f(d-1)$, e.g., step-cost = constant:
    - All optimal goal nodes occur on the same level
    - Optimal goal nodes are always shallower than non-optimal goals
    - An optimal goal will be found before any non-optimal goal
- Space is the bigger problem (more than time)

Uniform-cost search

Breadth-first is only optimal if path cost is a non-decreasing function of depth, i.e., $f(d) \geq f(d-1)$; e.g., constant step cost, as in the 8-puzzle.

Can we guarantee optimality for variable positive step costs $\geq\varepsilon$?

(Why $\geq\varepsilon$? To avoid infinite paths w/ step costs 1, ½, ¼, ...)

Uniform-cost Search:

Expand node with smallest path cost g(n).

- *Frontier* is a priority queue, i.e., new successors are merged into the queue sorted by g(n).
  - Can remove successors already on queue w/higher g(n).
    - Saves memory, costs time; another space-time trade-off.
- *Goal-Test* when node is popped off queue.

Uniform-cost search

Uniform-cost Search:

Expand node with smallest path cost g(n).

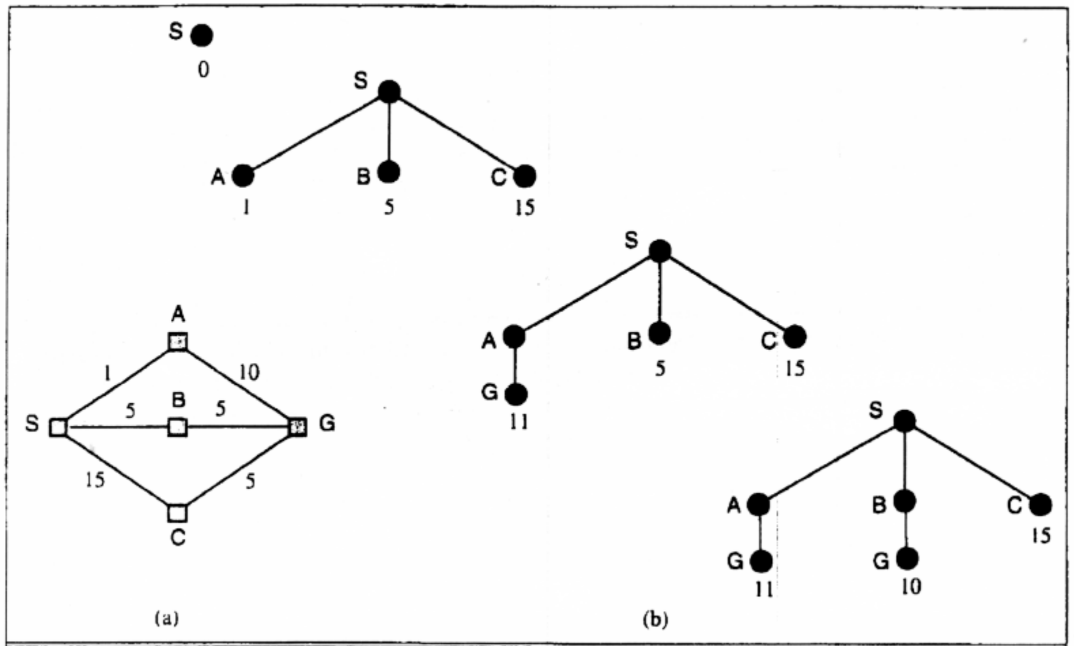


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Proof of Completeness:

Given that every step will cost more than 0, and assuming a finite branching factor, there is a finite number of expansions required before the total path cost is equal to the path cost of the goal state. Hence, we will reach it.

Proof of optimality given completeness:

Assume UCS is not optimal.
Then there must be an (optimal) goal state with path cost smaller than the found (suboptimal) goal state (invoking completeness).
However, this is impossible because UCS would have expanded that node first by definition.
Contradiction.

Uniform-cost search

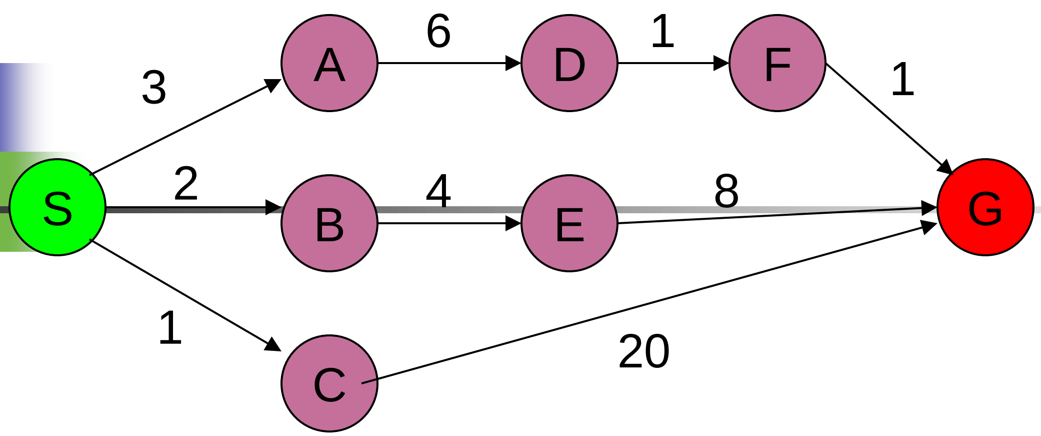
Implementation: *Frontier* = queue ordered by path cost.
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if b is finite and step cost $\geq \varepsilon > 0$.
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* $\leq$ cost of optimal solution.
$O(b^{\lfloor 1+C^*/\varepsilon \rfloor}) \approx O(b^{d+1})$

Space? # of nodes with path cost $\leq$ cost of optimal solution.
$O(b^{\lfloor 1+C^*/\varepsilon \rfloor}) \approx O(b^{d+1})$

Optimal? Yes, for any step cost $\geq \varepsilon > 0$.



The graph above shows the step-costs for different paths going from the start (S) to the goal (G).

Use uniform cost search to find the optimal path to the goal.

Exercise for at home

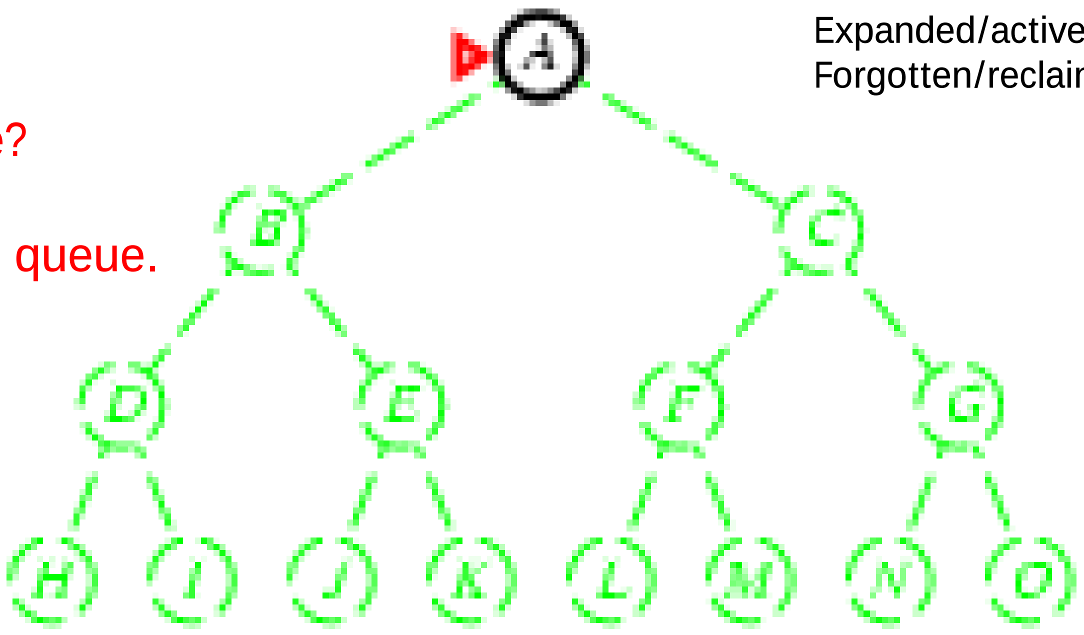
Depth-first search

- Expand *deepest* unexpanded node
- *Frontier* = Last In First Out (LIFO) queue, i.e., new successors go at the front of the queue.
- *Goal-Test* when inserted.

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Initial state = A
Is A a goal state?

Put A at front of queue.
frontier = [A]



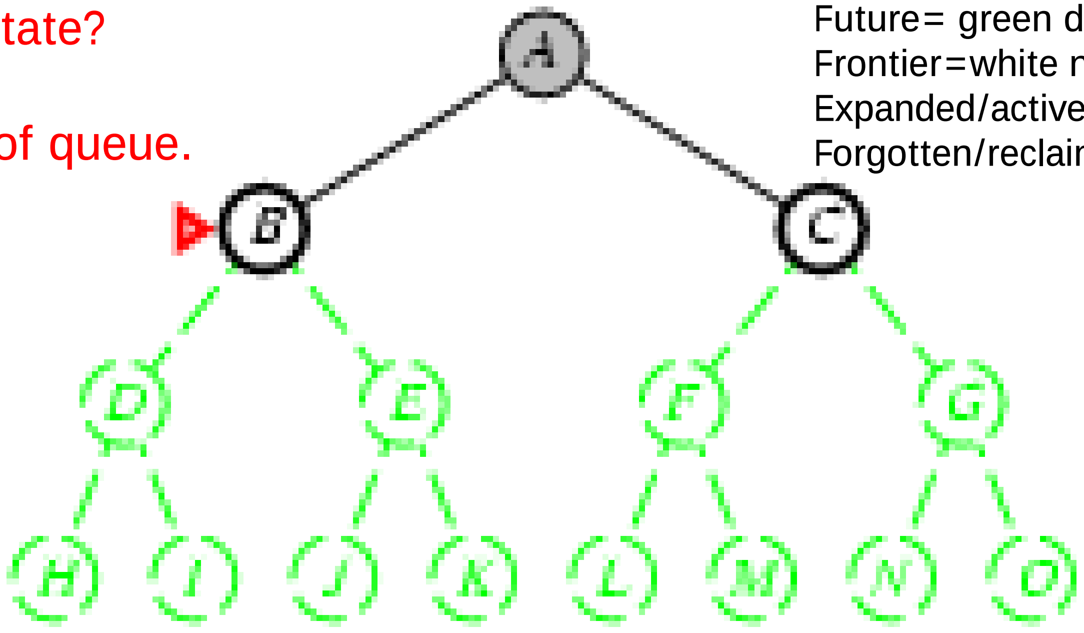
Depth-first search

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

Expand A to B, C.
Is B or C a goal state?

Put B, C at front of queue.
frontier = [B,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes



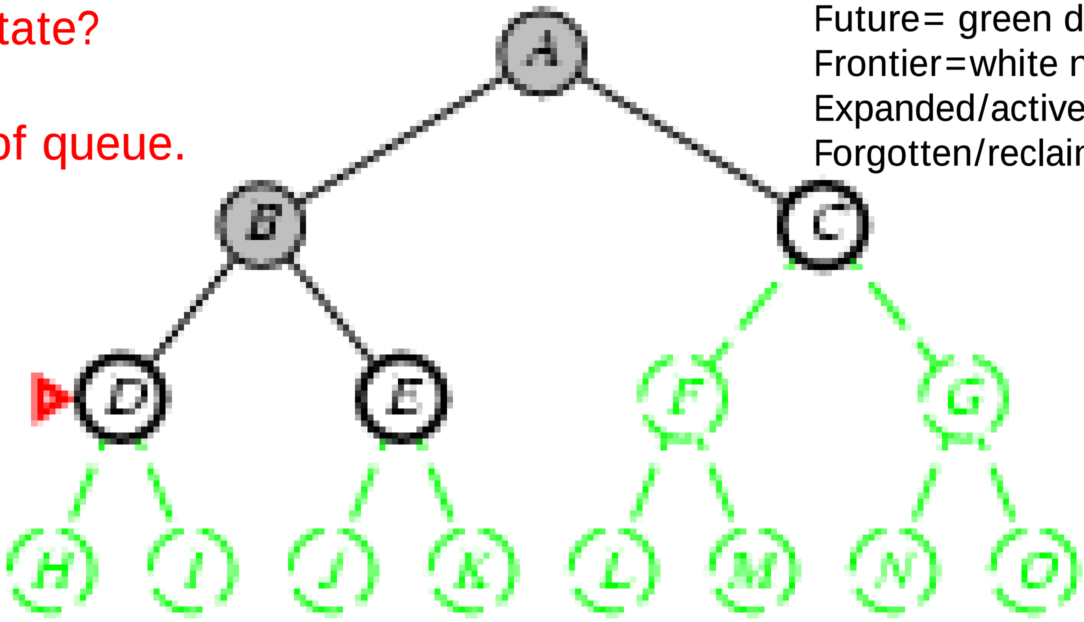
Note: Can save a space factor of *b* by generating successors one at a time.
See **backtracking search** in your book, p. 87 and Chapter 6.

Depth-first search

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

Expand B to D, E.
Is D or E a goal state?

Put D, E at front of queue.
frontier = [D,E,C]



Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

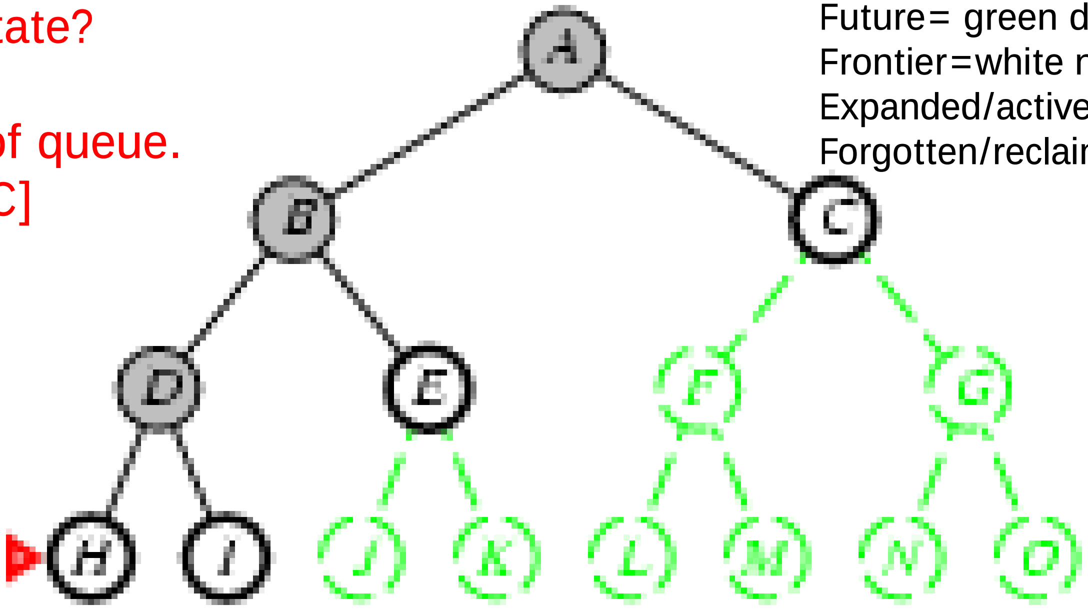
Depth-first search

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

Expand D to H, I.
Is H or I a goal state?

Put H, I at front of queue.
frontier = [H,I,E,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes



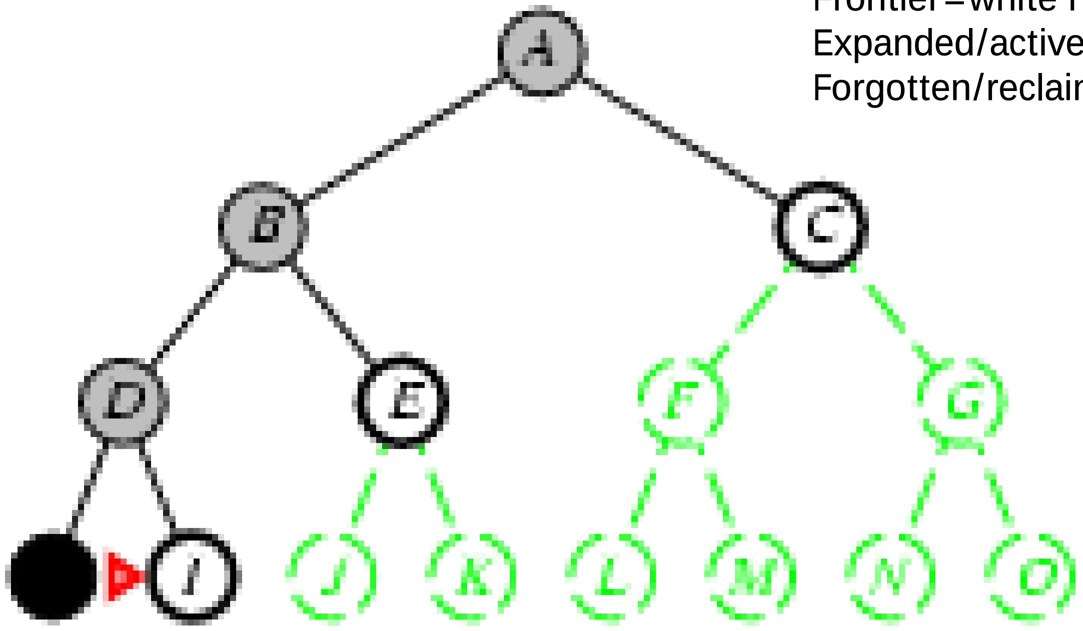
Depth-first search

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

Expand H to no children.
Forget H.

frontier = [I,E,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes



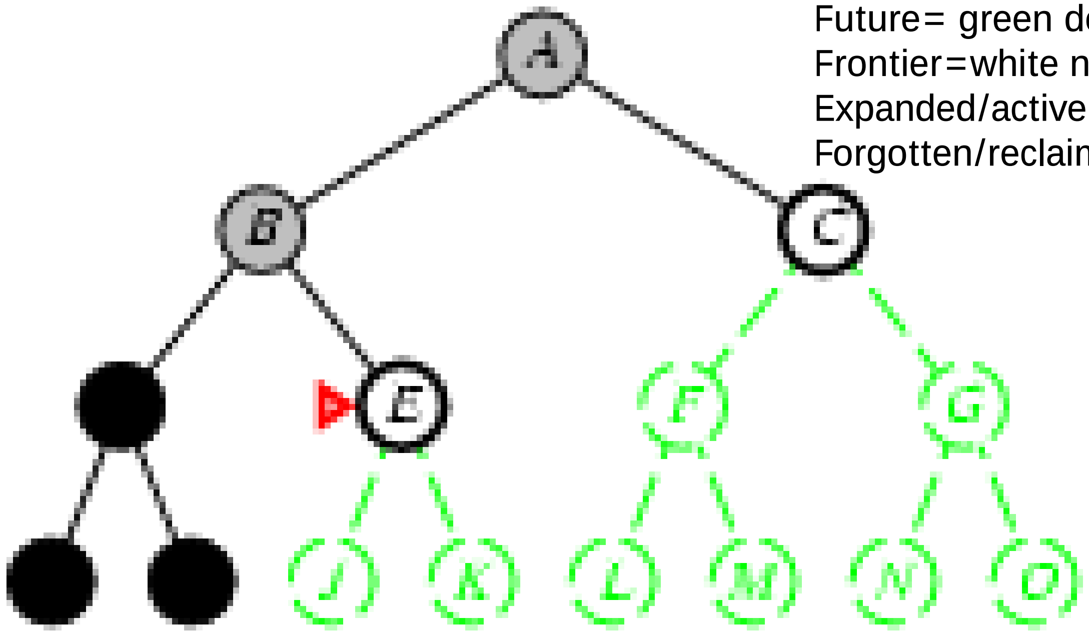
Depth-first search

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

Expand I to no children.
Forget D, I.

frontier = [E,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

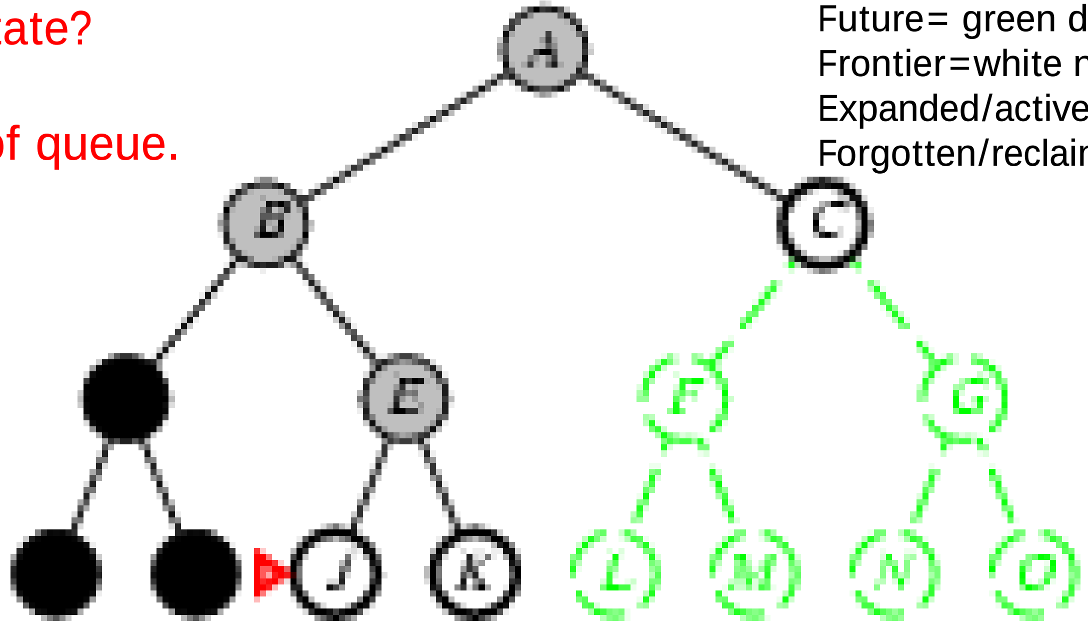


Depth-first search

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

Expand E to J, K.
Is J or K a goal state?

Put J, K at front of queue.
frontier = [J,K,C]



Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

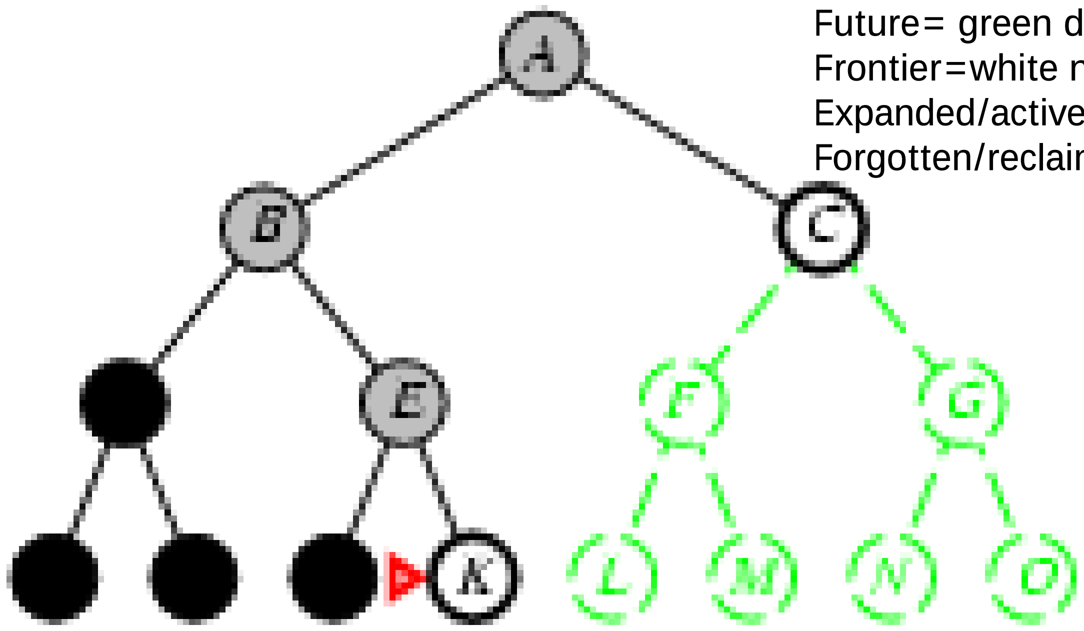
Depth-first search

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

Expand I to no children.
Forget D, I.

frontier = [E,C]

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

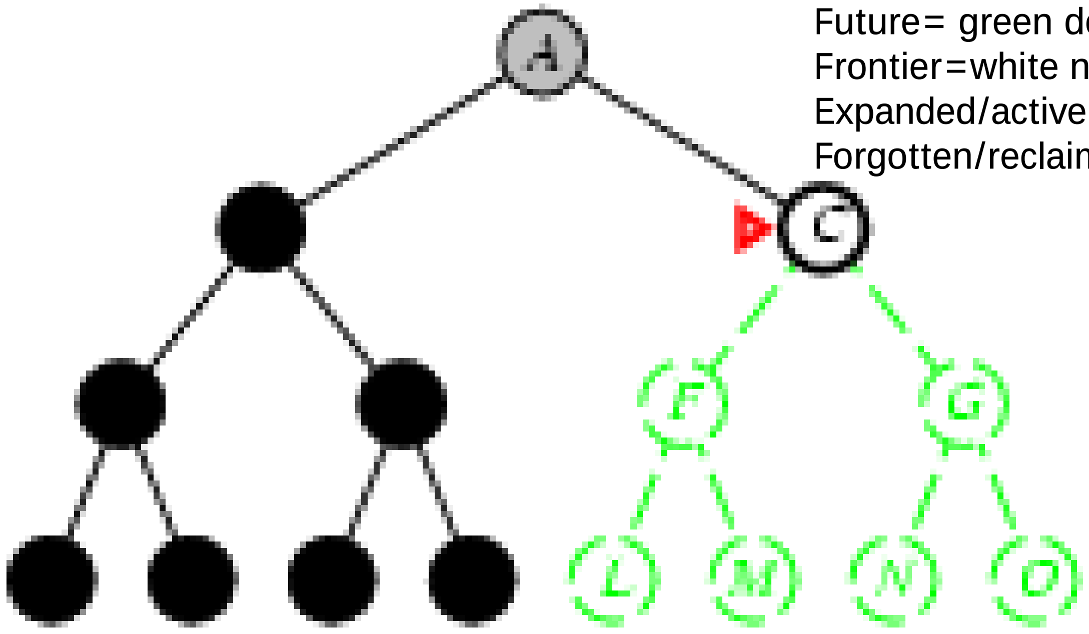


Depth-first search

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

Expand K to no children.
Forget B, E, K.

frontier = [C]



Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

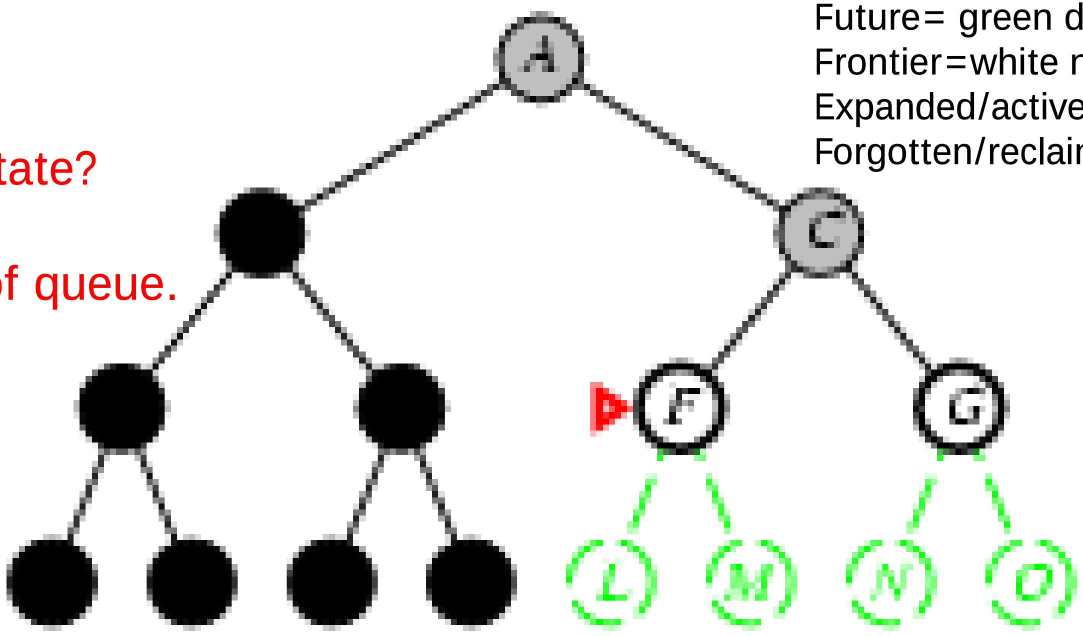
Depth-first search

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

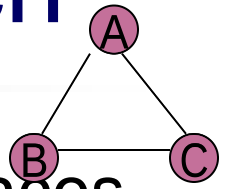
Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes

Expand C to F, G.
Is F or G a goal state?

Put F, G at front of queue.
frontier = [F,G]



Properties of depth-first search



- Complete? No: fails in loops/infinite-depth spaces
  - Can modify to avoid loops/repeated states along path
    - check if current nodes occurred before on path to root
  - Can use graph search (remember all nodes ever seen)
    - problem with graph search: space is exponential, not linear
  - Still fails in infinite-depth spaces (may miss goal entirely)
- Time? $O(b^m)$ with m =maximum depth of space
  - Terrible if m is much larger than d
  - If solutions are dense, may be much faster than BFS
- Space? $O(bm)$, i.e., linear space!
  - 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,
  only search until depth L,
  i.e., we don't expand nodes beyond depth L.
  → Depth-Limited Search

- What if solution is deeper than L? → Increase L iteratively.
  → Iterative Deepening Search

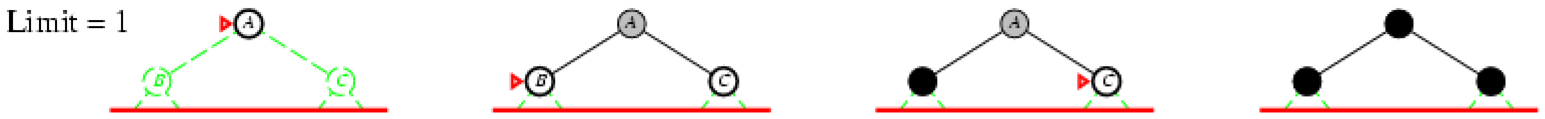
- This inherits the memory advantage of Depth-first search

- Better in terms of space complexity than Breadth-first search.

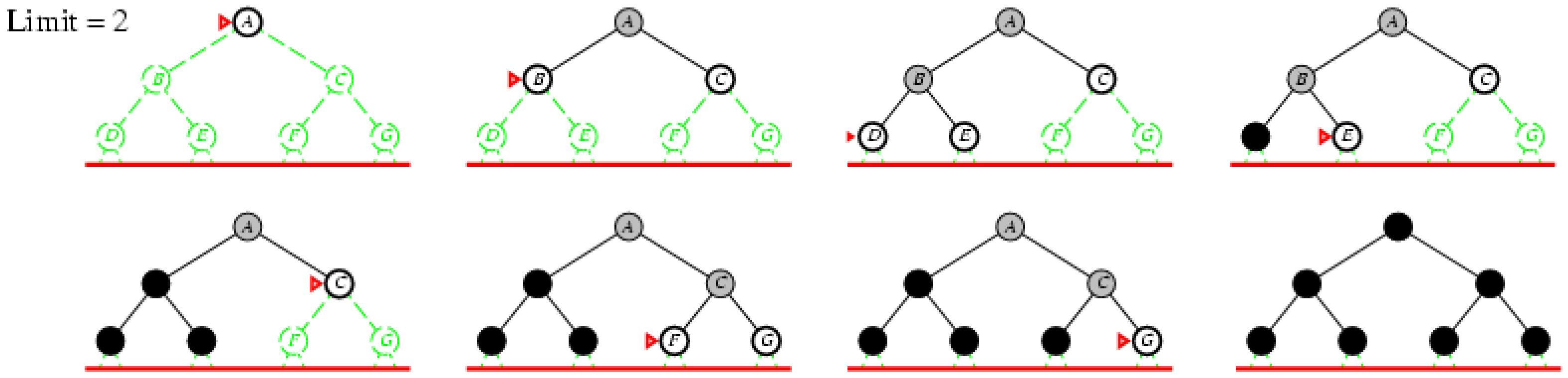
Iterative deepening search *l*=0

Limit = 0

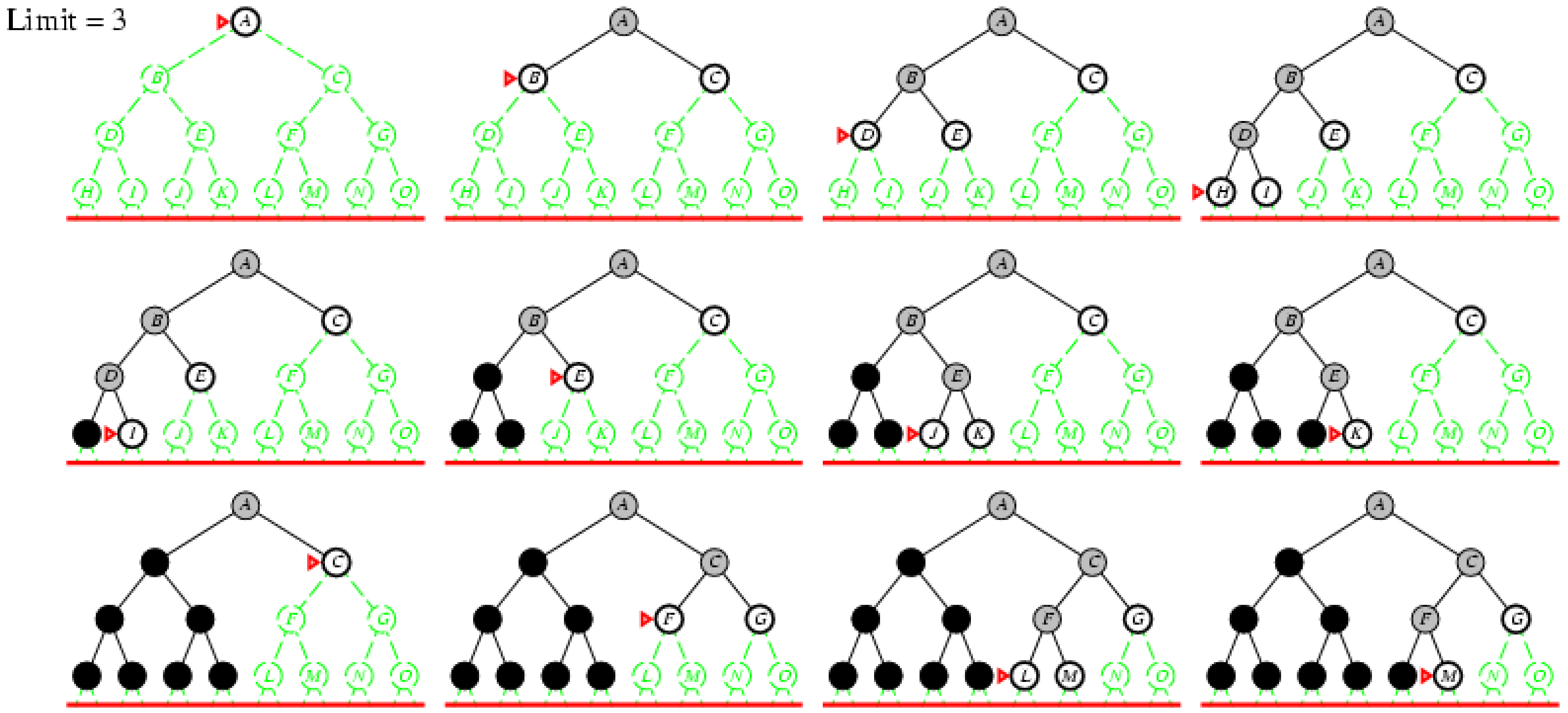
Iterative deepening search *l* =1



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*:

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

- 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

$O(b^d)$

Properties of iterative deepening search

- Complete? Yes
- Time? $O(b^d)$
- Space? $O(bd)$
- Optimal? No, for general cost functions.
  Yes, if cost is a non-decreasing function only of depth.

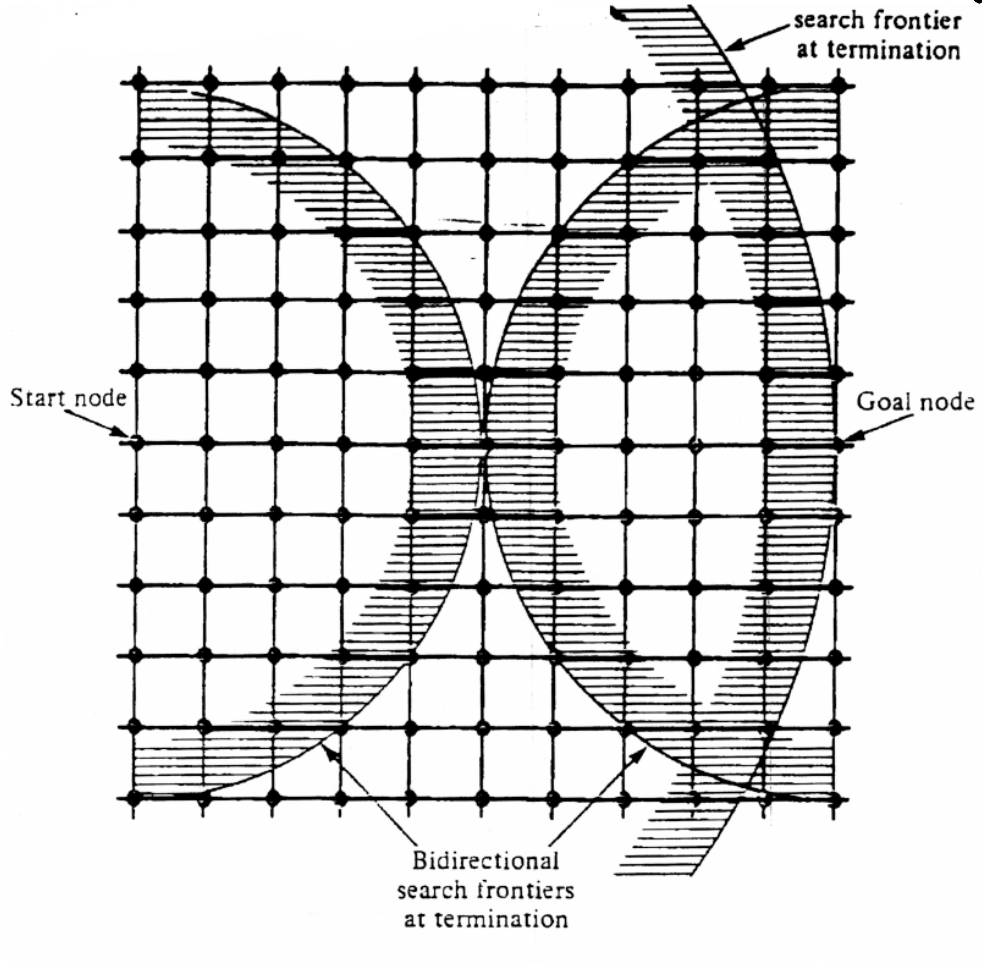
Generally the preferred uninformed search strategy.

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?
  - which to take if there are multiple goal states?
  - where to start if there is only a goal test, no explicit list?

Bi-Directional Search

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



search frontier at termination
Start node
Goal node
Bidirectional search frontiers at termination

Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening DLS	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{\lfloor 1+C^*/\varepsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{\lfloor 1+C^*/\varepsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

There are a number of footnotes, caveats, and assumptions.
See Fig. 3.21, p. 91.
[a] complete if b is finite
[b] complete if step costs $\geq \varepsilon > 0$
[c] optimal if step costs are all identical
(also if path cost non-decreasing function of depth only)
[d] if both directions use breadth-first search
(also if both directions use uniform-cost search with step costs $\geq \varepsilon > 0$)

Generally the preferred uninformed search strategy

Note that $d \leq \lfloor 1+C^*/\varepsilon \rfloor$

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)