CMSC 471
Artificial Intelligence

Search

KMA Solaiman – ksolaima@umbc.edu

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Many slides courtesy Tim Finin, Dan Klein, and Pieter Abbeel

Formalizing state space search

- A state space is a **graph** (V, E) where V is a set of **nodes** and E is a set of **arcs**, and each arc is directed from a node to another node
- **Nodes:** data structures with state description and other info, e.g., node’s parent, name of action that generated it from parent, etc.
- **Arcs:** instances of actions, head is a state, tail is the state that results from action

Formalizing search in a state space

- Each arc has fixed, positive **cost** associated with it corresponding to the action cost
  - Simple case: all costs are 1
- Each node has a set of **successor nodes** corresponding to all legal actions that can be applied at node’s state
  - **Expanding** a node = generating its successor nodes and adding them and their associated arcs to the graph
- One or more nodes are marked as **start nodes**
- A **goal test** predicate is applied to a state to determine if its associated node is a goal node

Formalizing search

- **Solution:** sequence of actions associated with a path from a start node to a goal node
- **Solution cost:** sum of the arc costs on the solution path
  - If all arcs have same (unit) cost, then solution cost is length of solution (number of steps)
  - Algorithms generally require that arc costs cannot be negative (why?)

Formalizing search

- **State-space search:** searching through state space for solution by **making explicit** a portion of an **implicit** state-space graph to find a goal node
  - Can’t materialize whole space for large problems
  - Initially V={S}, where S is the start node, E={}
  - On expanding S, its *successor nodes* are generated and added to V and associated *arcs added to E*
  - Process continues until a goal node is found
- Nodes represent a *partial solution* path (+ cost of partial solution path) from S to the node
  - From a node there may be many possible paths (and thus solutions) with this partial path as a prefix

A General Searching Algorithm

Core ideas:

1. Maintain a list of frontier (fringe) nodes
    1. Nodes coming *into* the frontier have been explored
    2. Nodes going out of the frontier have not been explored
2. Iteratively select nodes from the frontier and explore unexplored nodes from the frontier
3. Stop when you reach your **goal**

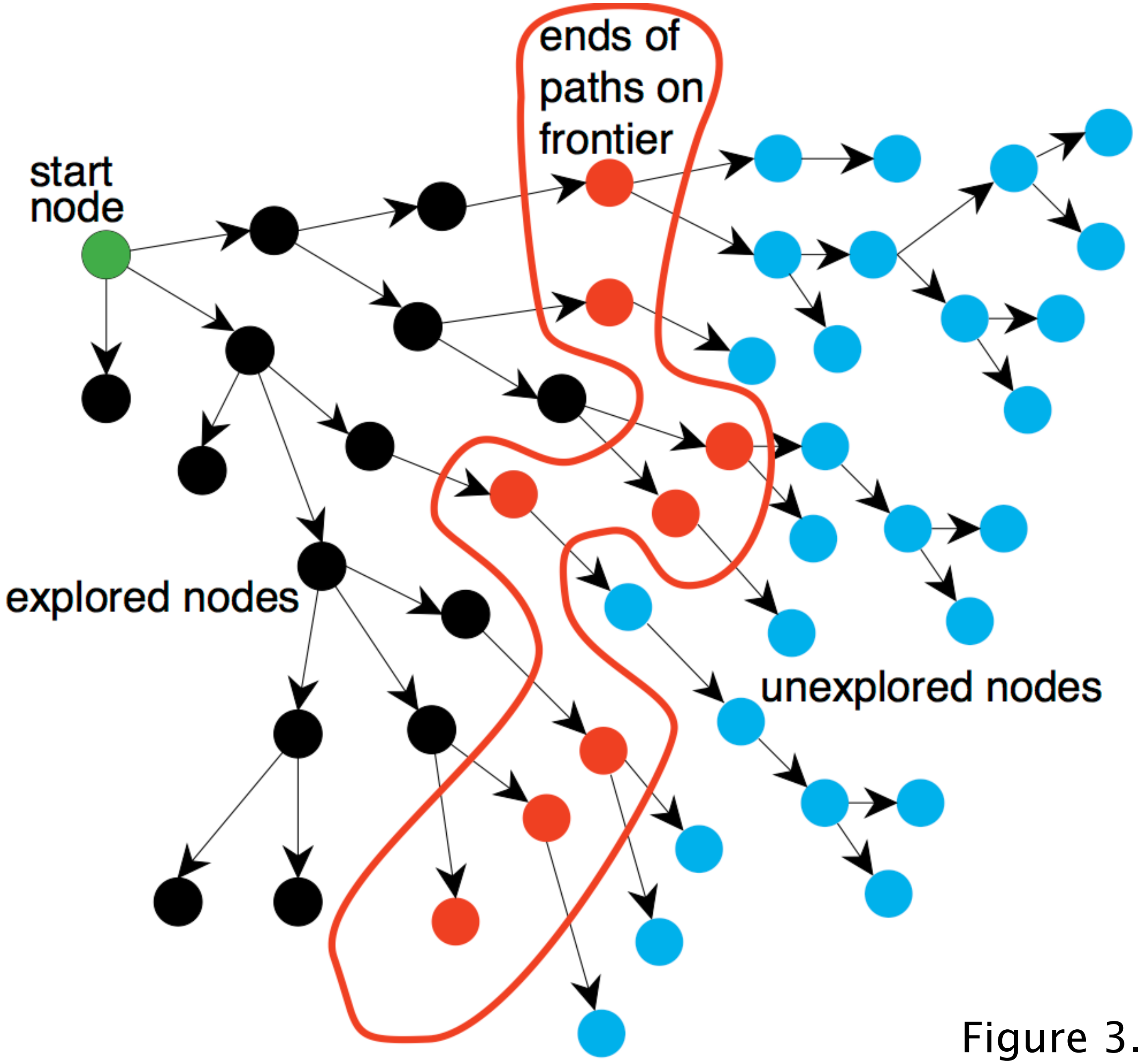


Figure 3.3

State-space search algorithm

;; problem describes the start state, operators, goal test, and operator costs

;; queueing-function is a comparator function that ranks two states

;; general-search returns either a goal node or failure

```
function general-search (problem, QUEUEING-FUNCTION)
  nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
     if EMPTY(nodes) then return "failure"
     node = REMOVE-FRONT(nodes)
     if problem.GOAL-TEST(node.STATE) succeeds
        then return node
     nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
             problem.OPERATORS))
 end
```

;; Note: The goal test is NOT done when nodes are generated

;; Note: This algorithm does not detect loops

Key procedures to be defined

- EXPAND
  - Generate a node’s successor nodes, adding them to the graph if not already there
- GOAL-TEST
  - Test if state satisfies all goal conditions
- QUEUEING-FUNCTION
  - Maintain ranked list of nodes that are candidates for expansion
  - Changing definition of the QUEUEING-FUNCTION leads to different search strategies

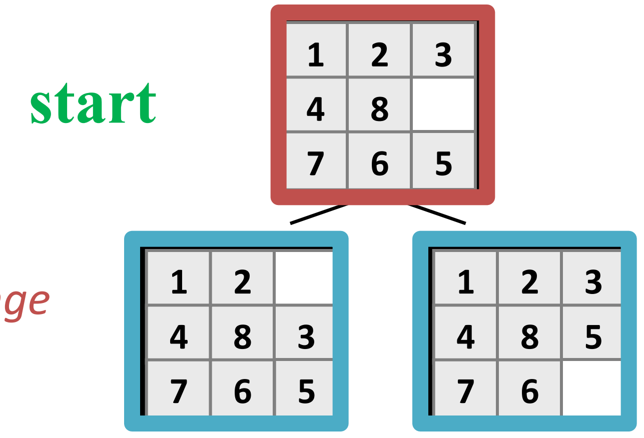
What does “search” look like for a particular problem?

start

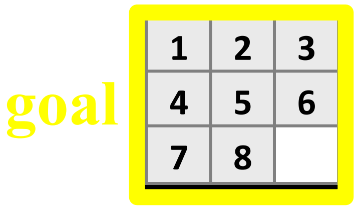
1	2	3
4	8	
7	6	5

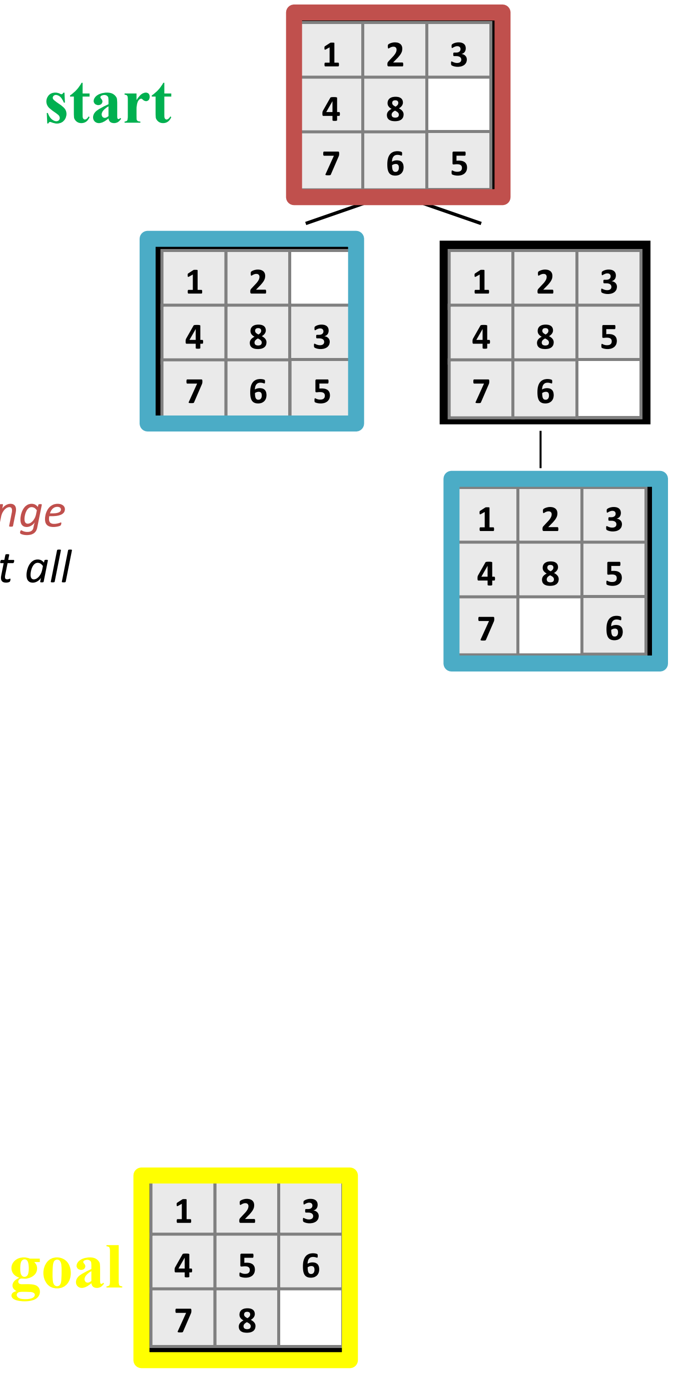
goal

1	2	3
4	5	6
7	8	

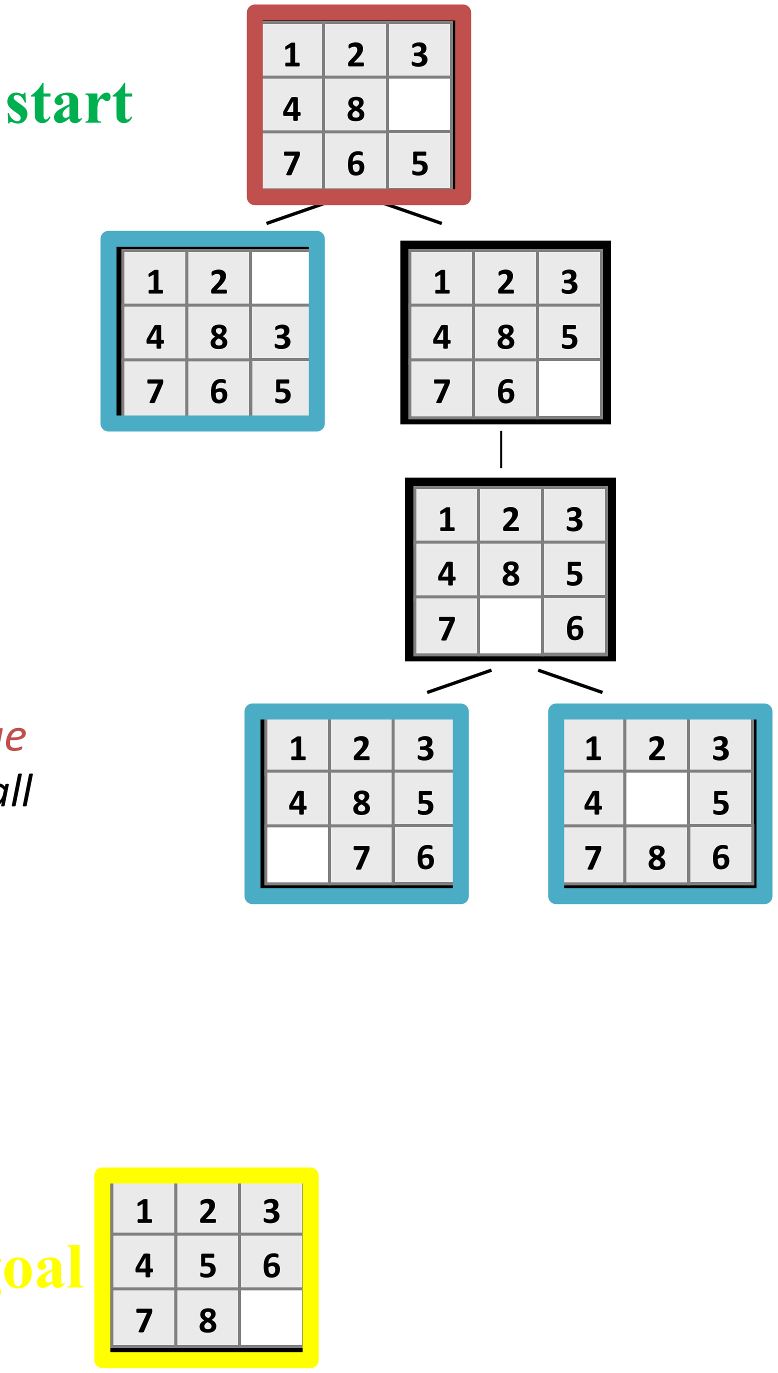


Expanding a node on the fringe
(taking a certain action)

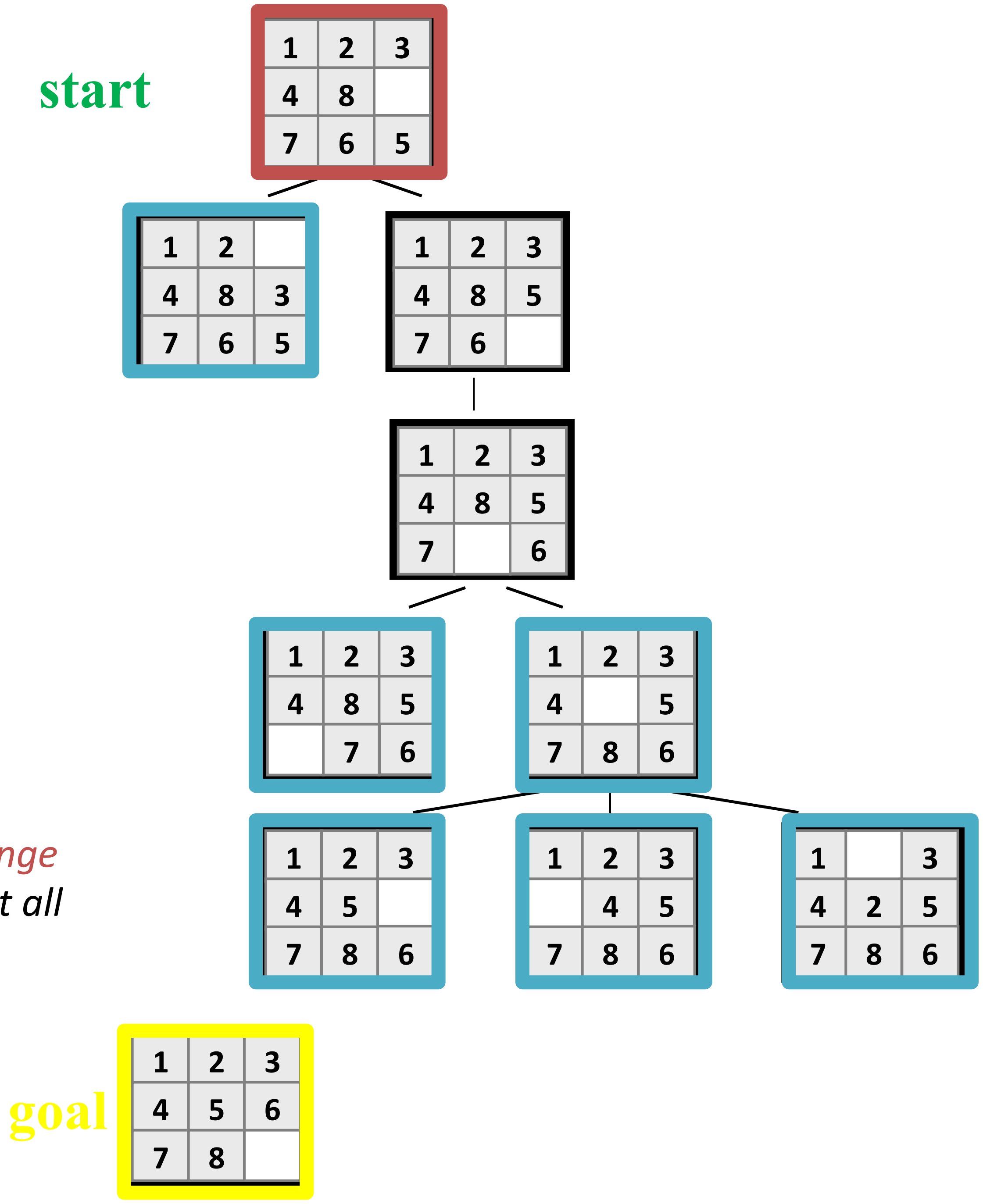




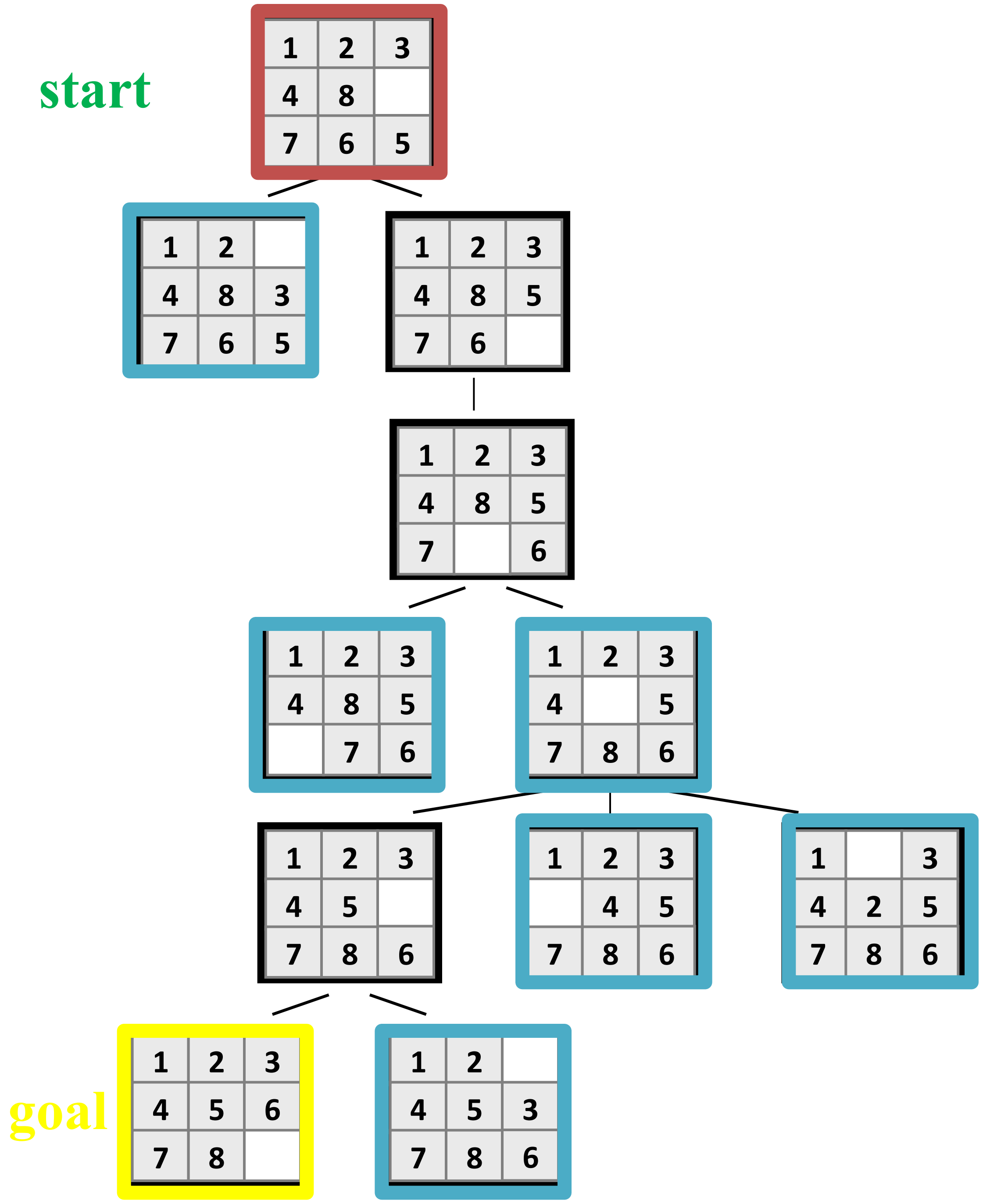
Expanding a node on the fringe (taking a certain action). Not all actions shown.



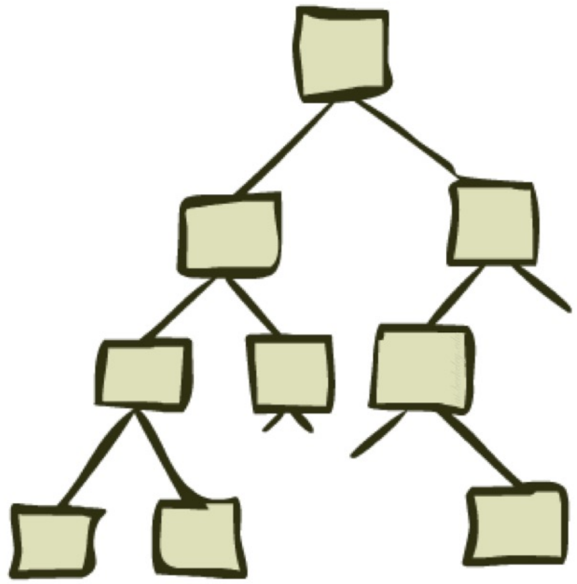
Expanding a node on the fringe (taking a certain action). Not all actions shown.



Expanding a node on the fringe (taking a certain action). Not all actions shown.

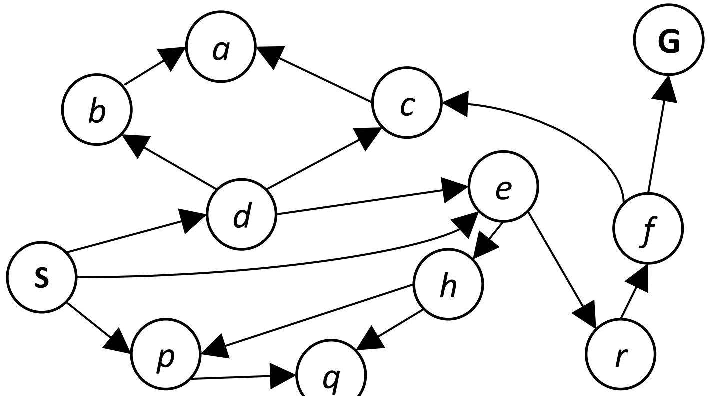

start
goal

State Space Graphs and Search Trees



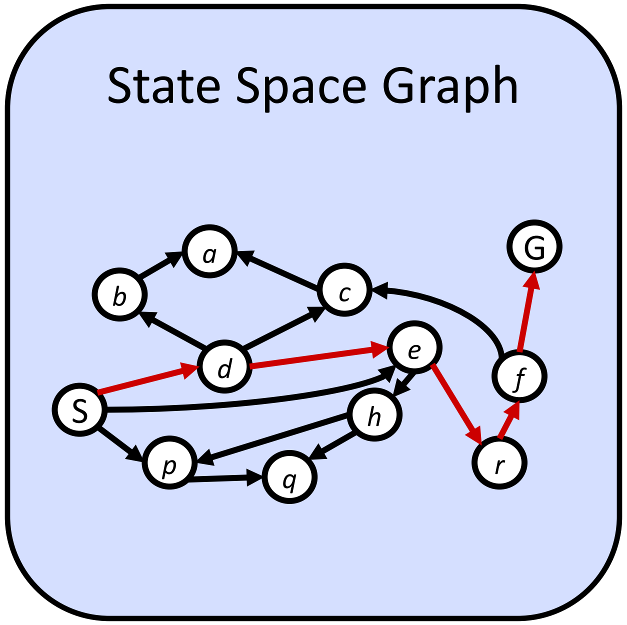
State Space Graphs

- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions/ successors (action results)
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it’s too big), but it’s a useful idea



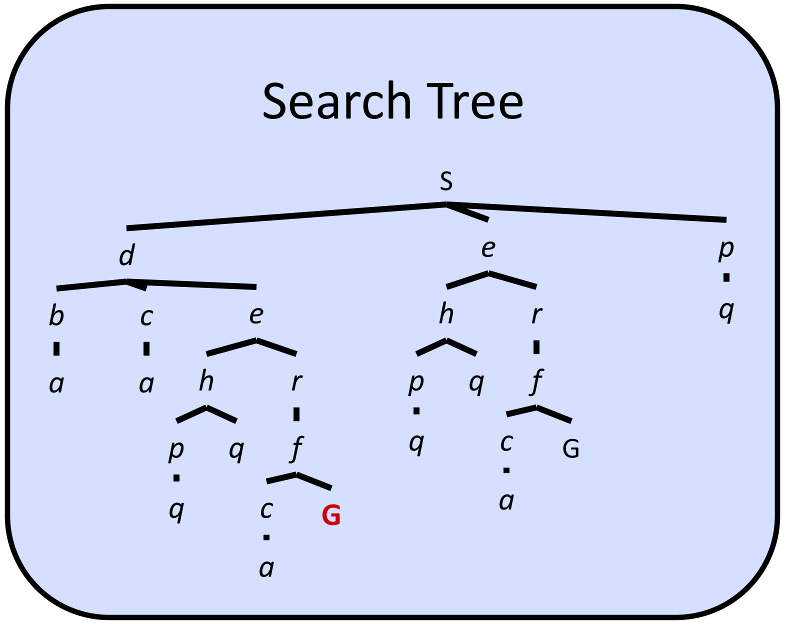
Tiny state space graph for a tiny search problem

State Space Graphs vs. Search Trees



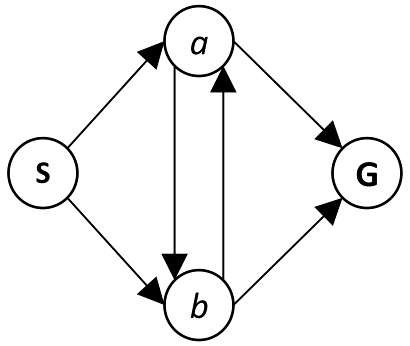
Each NODE in in the search tree is an entire PATH in the state space graph.

We construct the tree on demand – and we construct as little as possible.



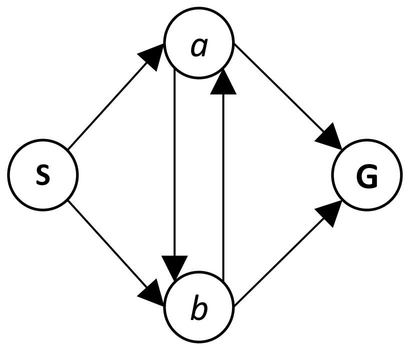
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



Quiz: State Space Graphs vs. Search Trees

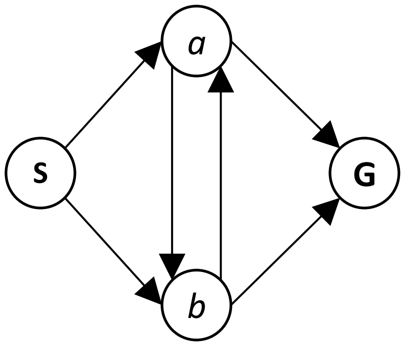
Consider this 4-state graph:



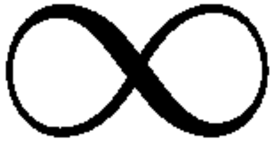
How big is its search tree (from S)?

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

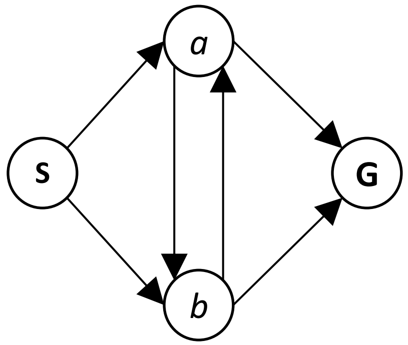


How big is its search tree (from S)?



Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

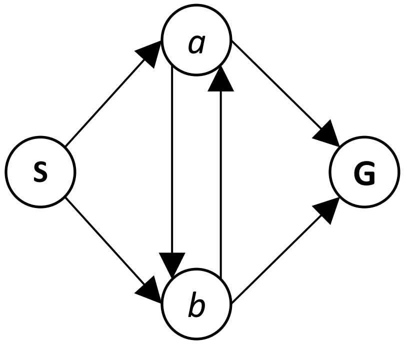


How big is its search tree (from S)?

∞

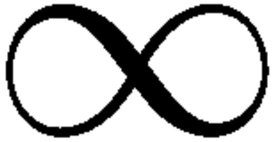
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



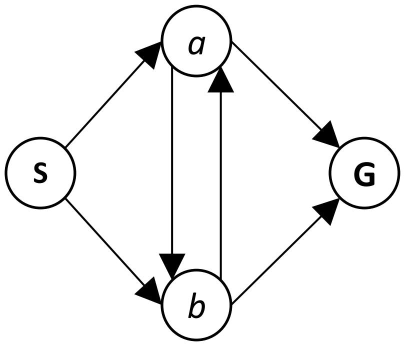
How big is its search tree (from S)?

S

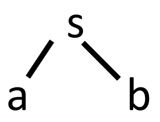


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



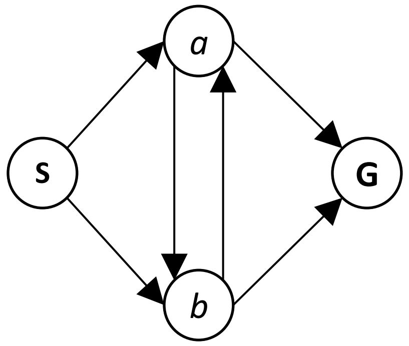
How big is its search tree (from S)?



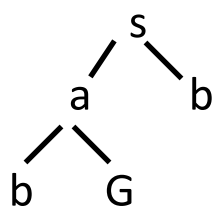
$$\infty$$

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



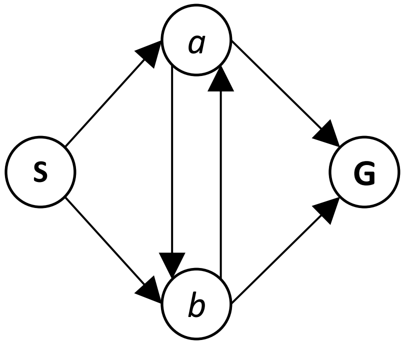
How big is its search tree (from S)?



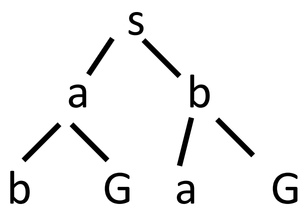
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?

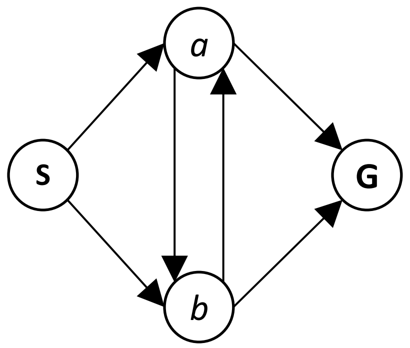


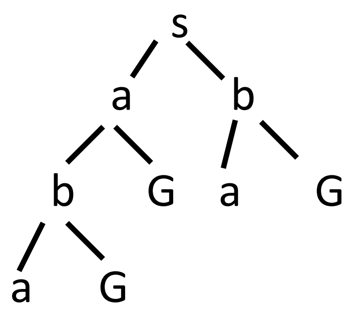
∞

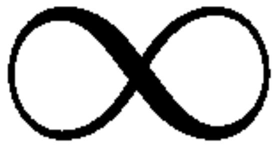
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:

How big is its search tree (from S)?

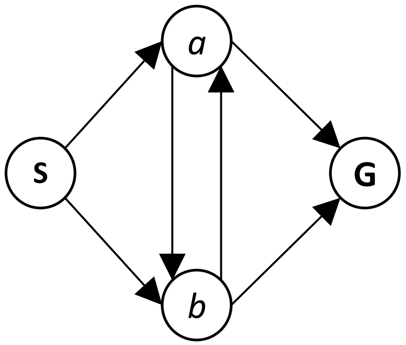




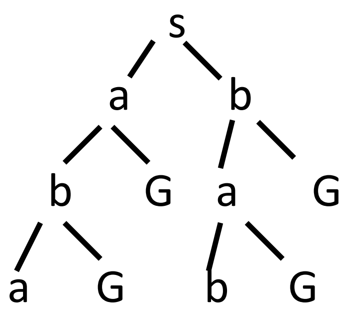


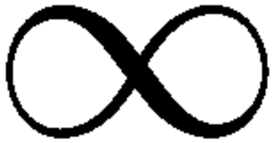
Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



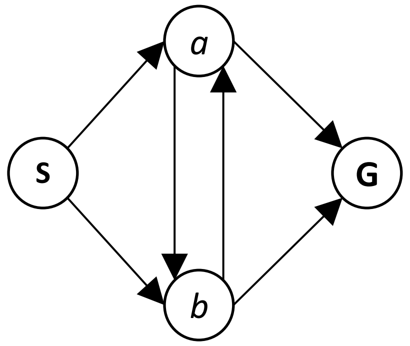
How big is its search tree (from S)?



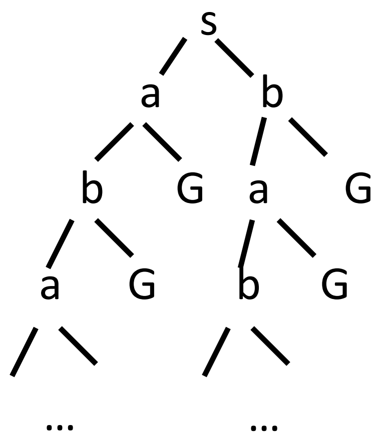


Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



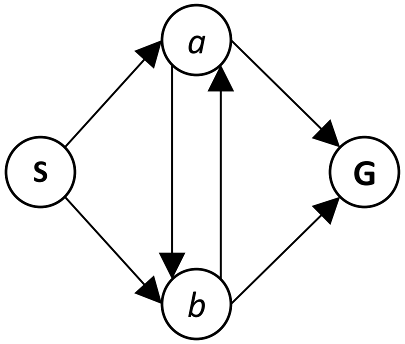
How big is its search tree (from S)?



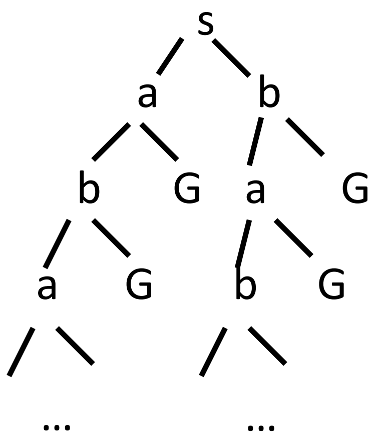
∞

Quiz: State Space Graphs vs. Search Trees

Consider this 4-state graph:



How big is its search tree (from S)?



Important: Those who don't know history are doomed to repeat it!

Informed vs. uninformed search

Uninformed search strategies (blind search)

- Use no information about likely *direction* of a goal
- Methods: breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional

Informed search strategies (heuristic search)

- Use information about domain to (try to) (usually) head in the general direction of goal node(s)
- Methods: hill climbing, best-first, greedy search, beam search, algorithm A, algorithm A*

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
- **Space complexity**
- **Optimality/Admissibility**

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
- **Optimality/Admissibility**

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

Search Algorithm Properties

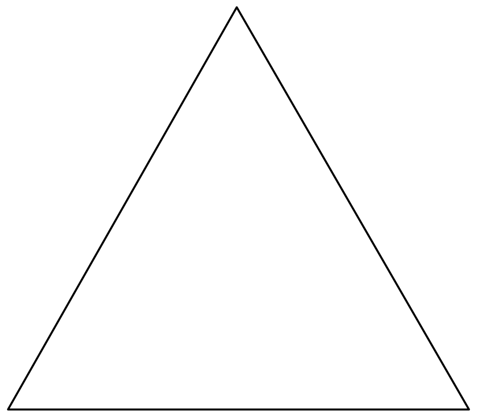
- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:

Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

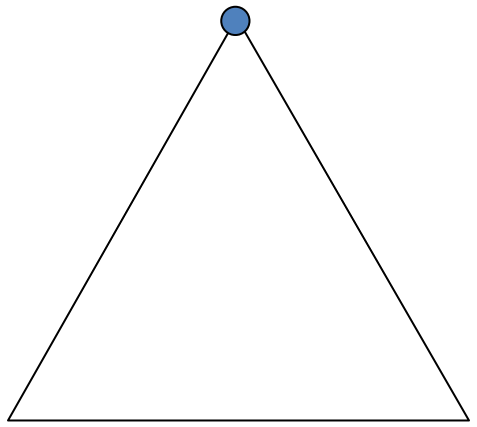
- Cartoon of search tree:



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

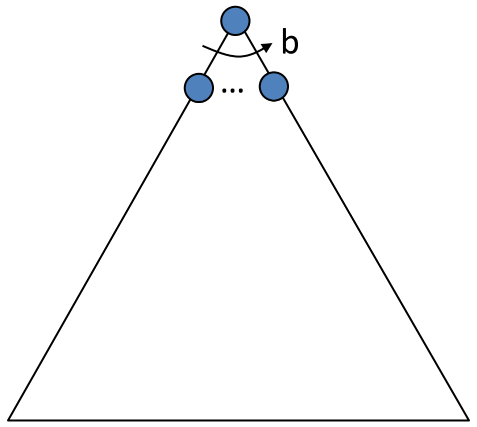
- Cartoon of search tree:



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

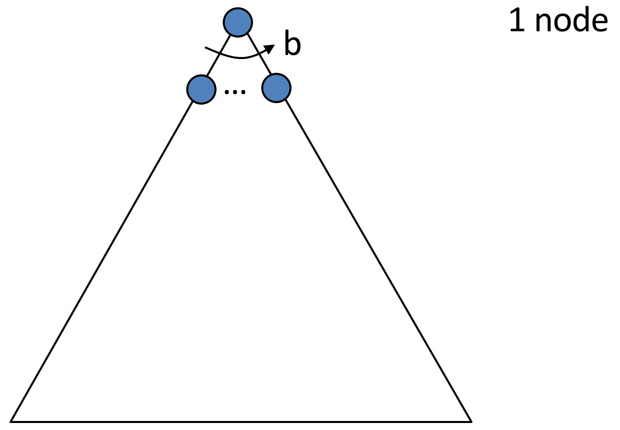
- Cartoon of search tree:
  - b is the branching factor



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

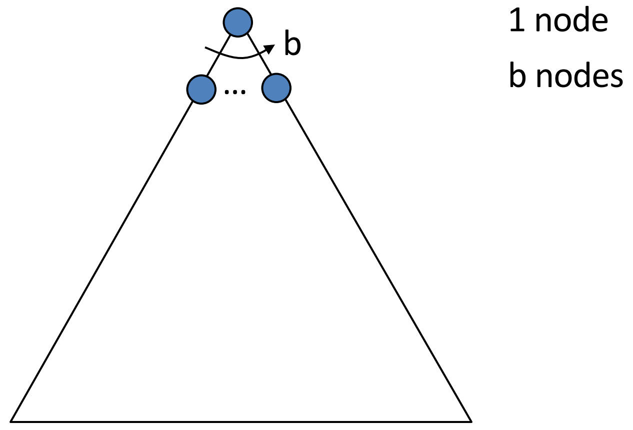
- Cartoon of search tree:
  - b is the branching factor



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

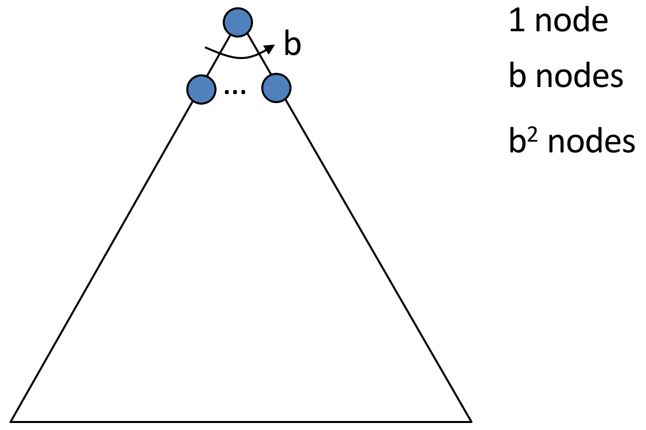
- Cartoon of search tree:
  - b is the branching factor



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

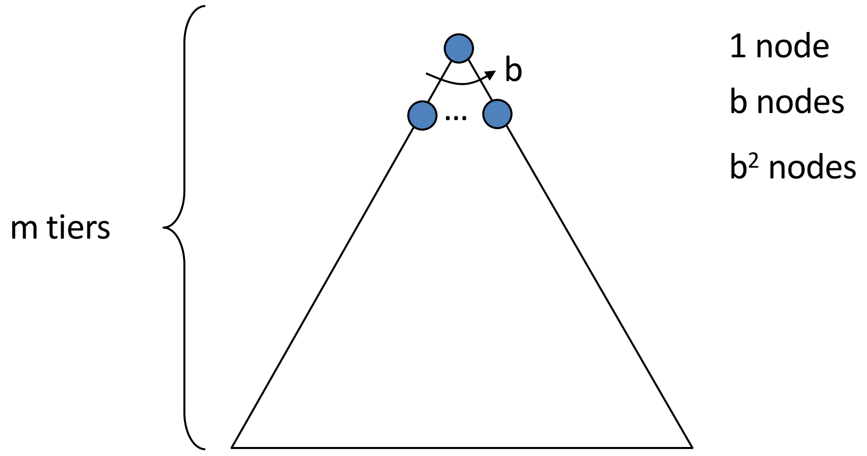
- Cartoon of search tree:
  - b is the branching factor



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

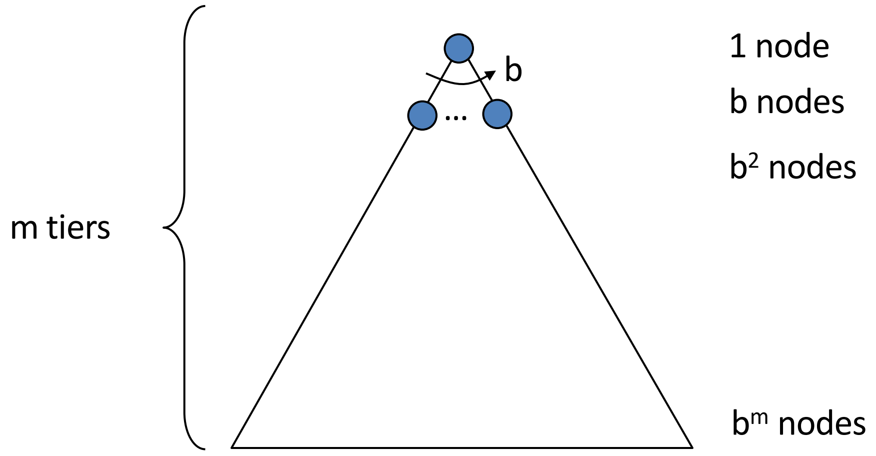
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

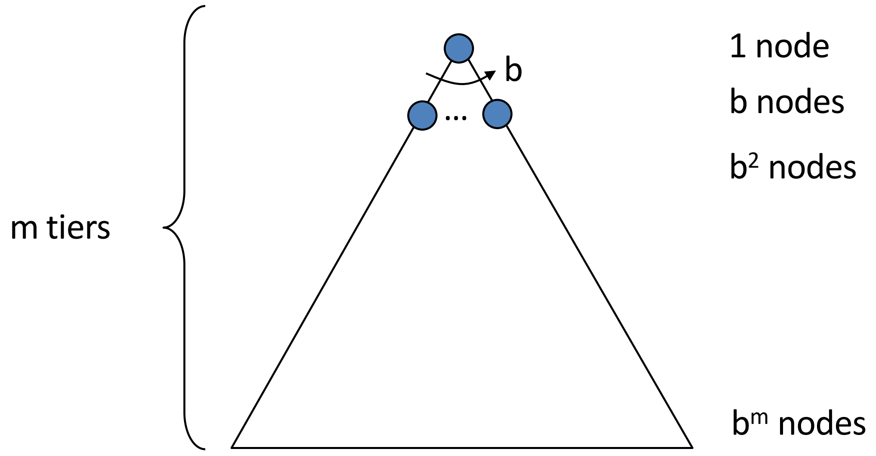
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

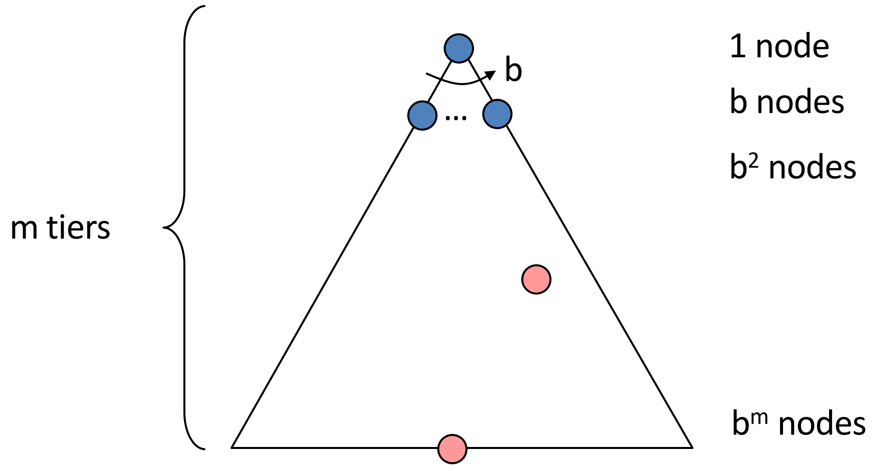
- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths



Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

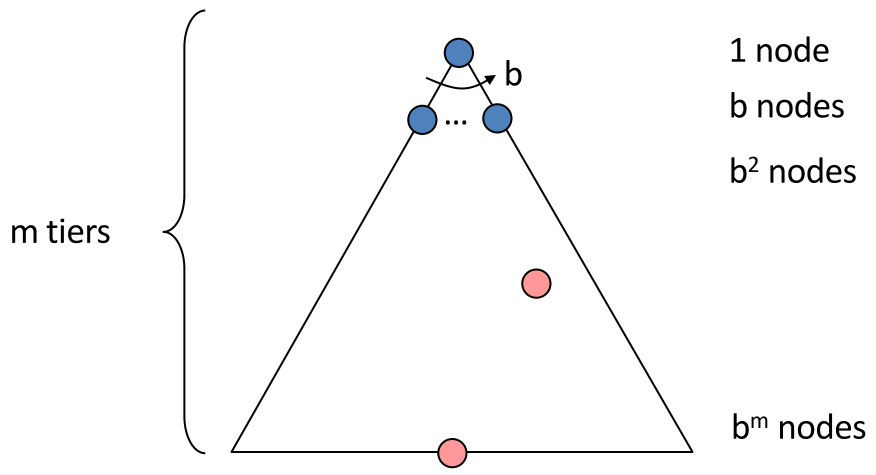


Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

- Number of nodes in entire tree?

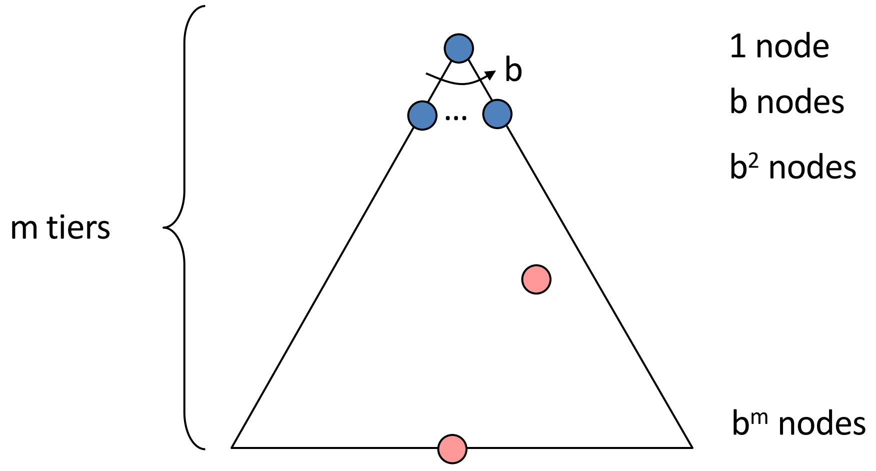


Search Algorithm Properties

- Complete: Guaranteed to find a solution if one exists?
- Optimal: Guaranteed to find the least cost path?
- Time complexity?
- Space complexity?

- Cartoon of search tree:
  - b is the branching factor
  - m is the maximum depth
  - solutions at various depths

- Number of nodes in entire tree?
  - $1 + b + b^2 + \ldots. b^m = O(b^m)$



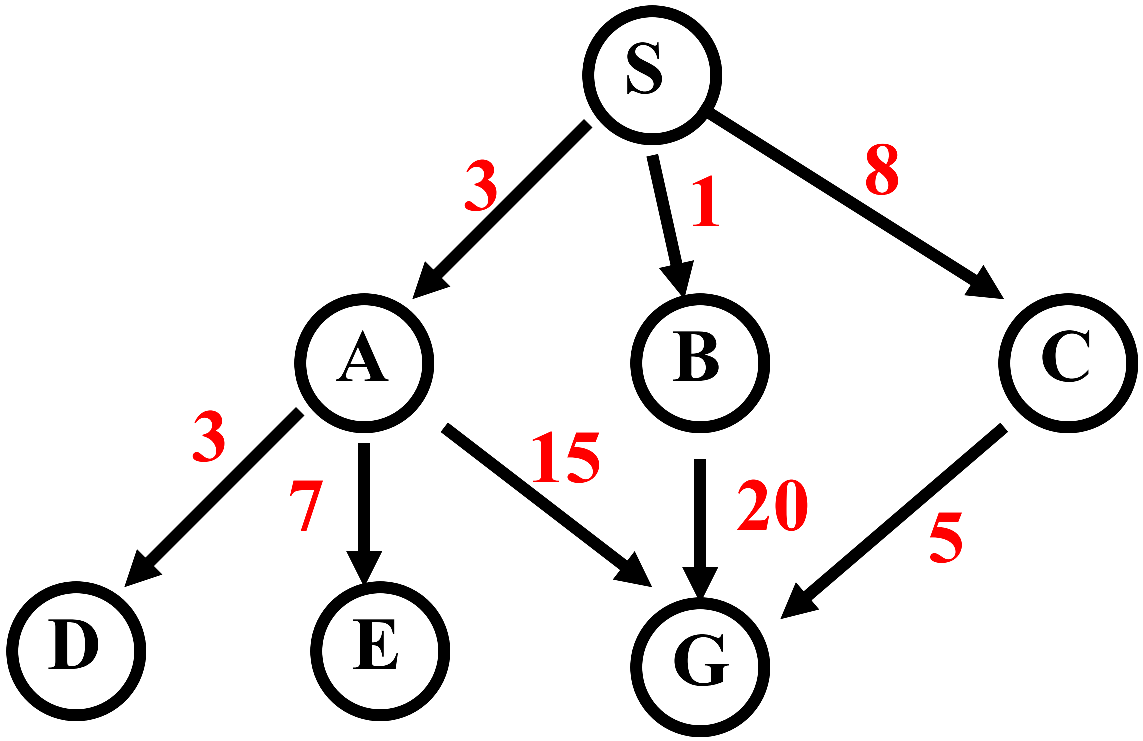
Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of graph/tree during the search
- **Optimality/Admissibility**

Evaluating search strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity** (worst or average case)
  - Usually measured by *number of nodes expanded*
- **Space complexity**
  - Usually measured by maximum size of graph/tree during the search
- **Optimality/Admissibility**
  - If a solution is found, is it **guaranteed** to be an optimal one, i.e., one with minimum cost

Example of uninformed search strategies



Consider this search space where S is the start node and G is the goal. Numbers are arc costs.

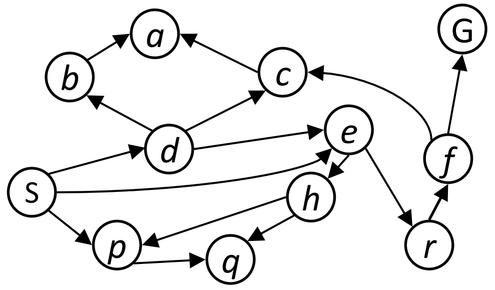
Classic uninformed search methods

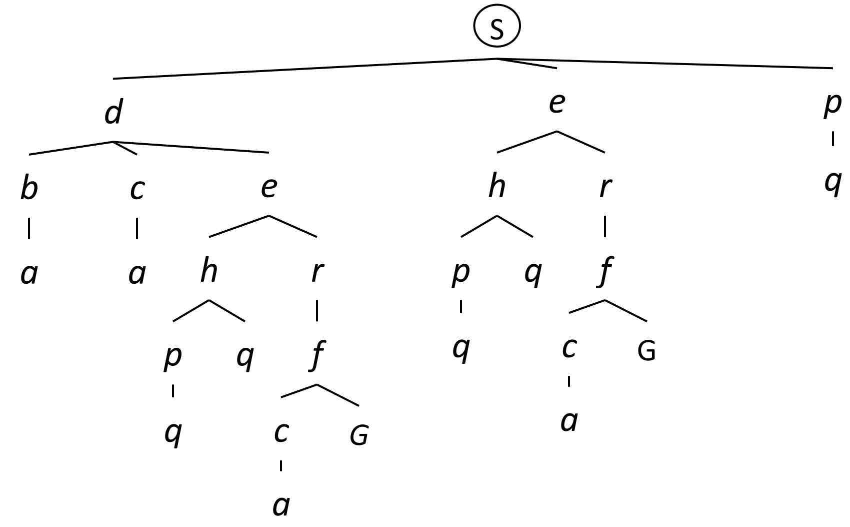
- The four classic uninformed search methods
  - Breadth first search (BFS)
  - Depth first search (DFS)
  - Uniform cost search *(generalization of BFS)*
  - Iterative deepening *(blend of DFS and BFS)*
- To which we can add another technique
  - Bi-directional search *(hack on BFS)*

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

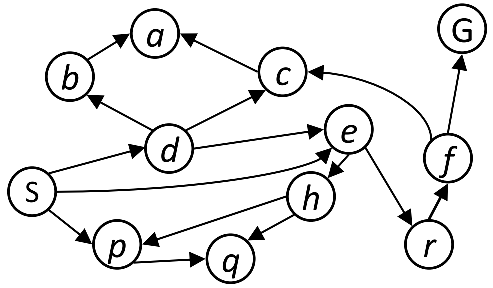


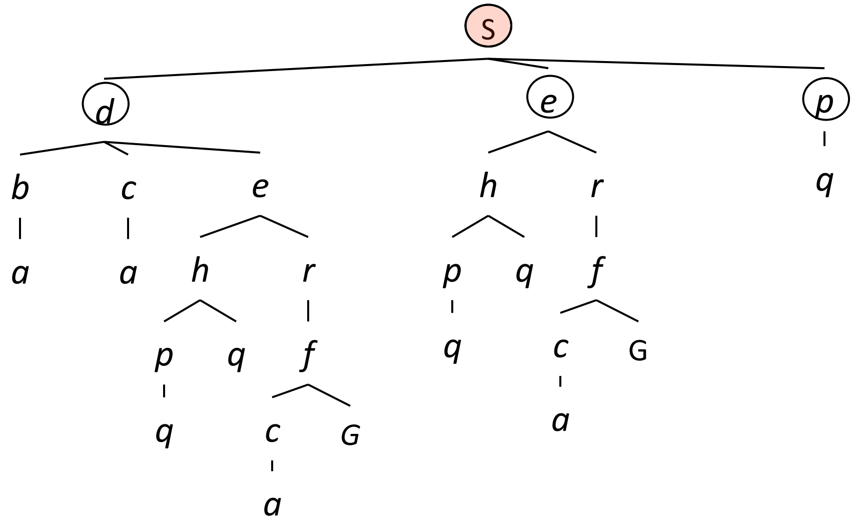


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

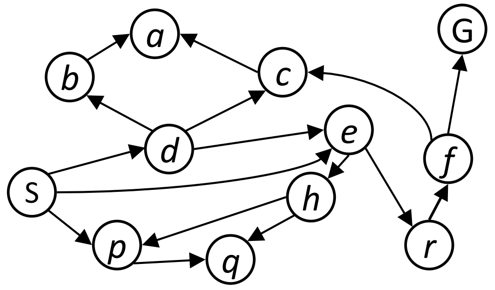


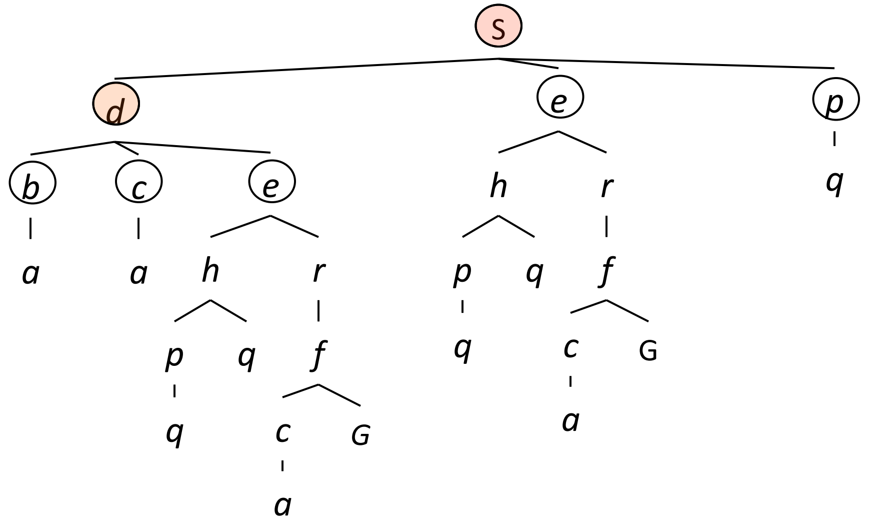


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

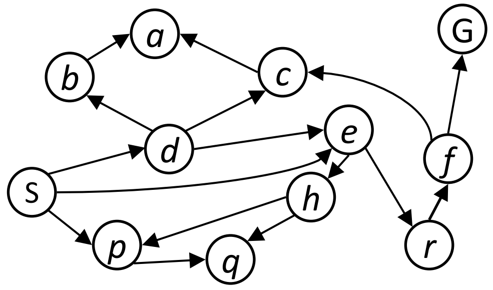


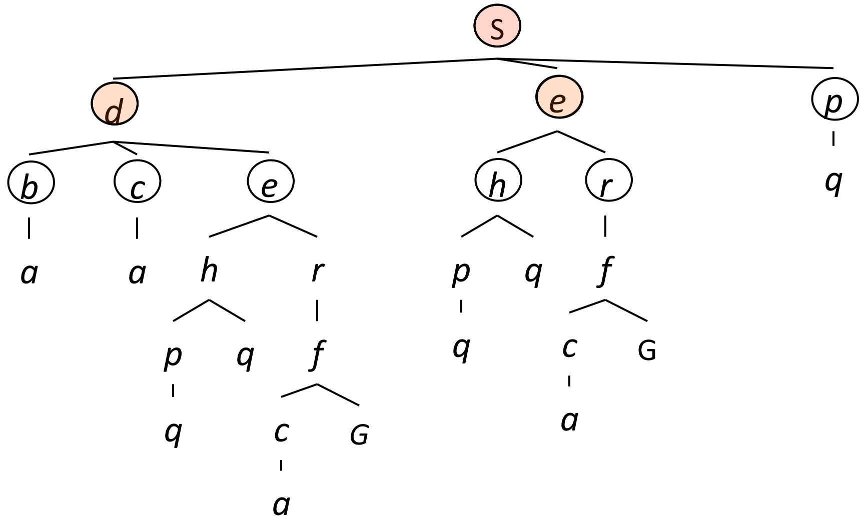


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

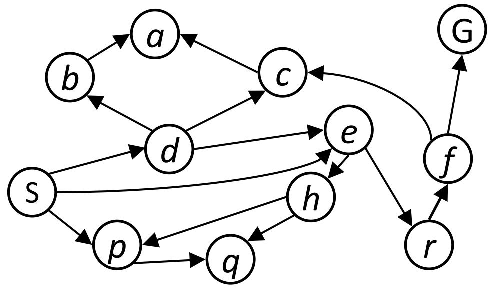


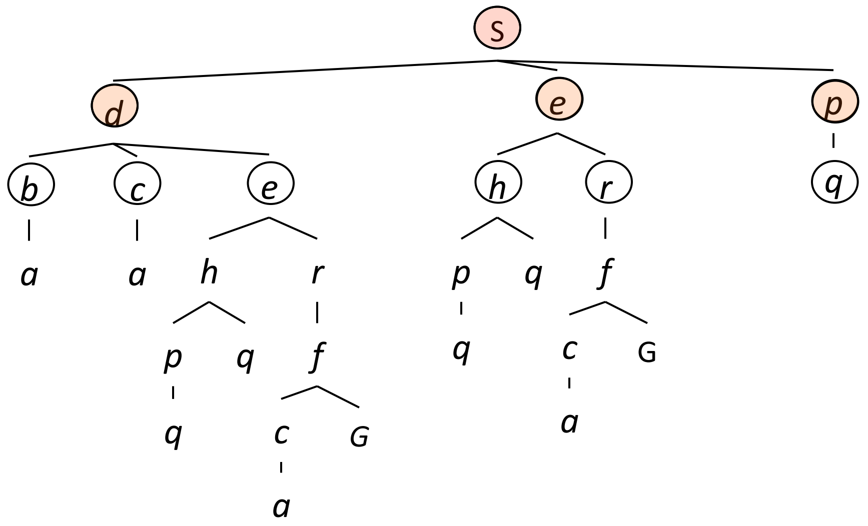


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

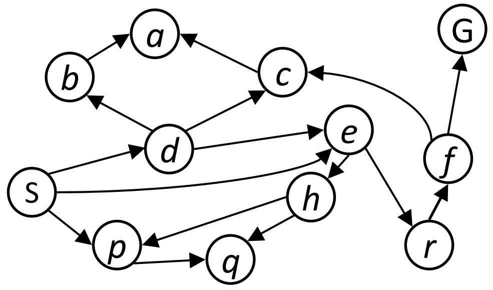


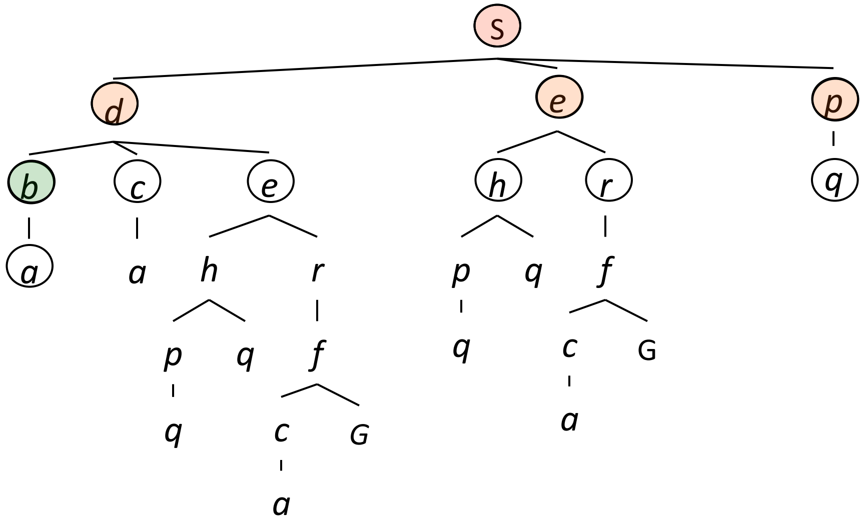


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Frontier is a FIFO queue

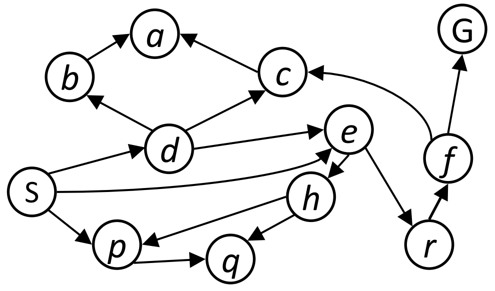


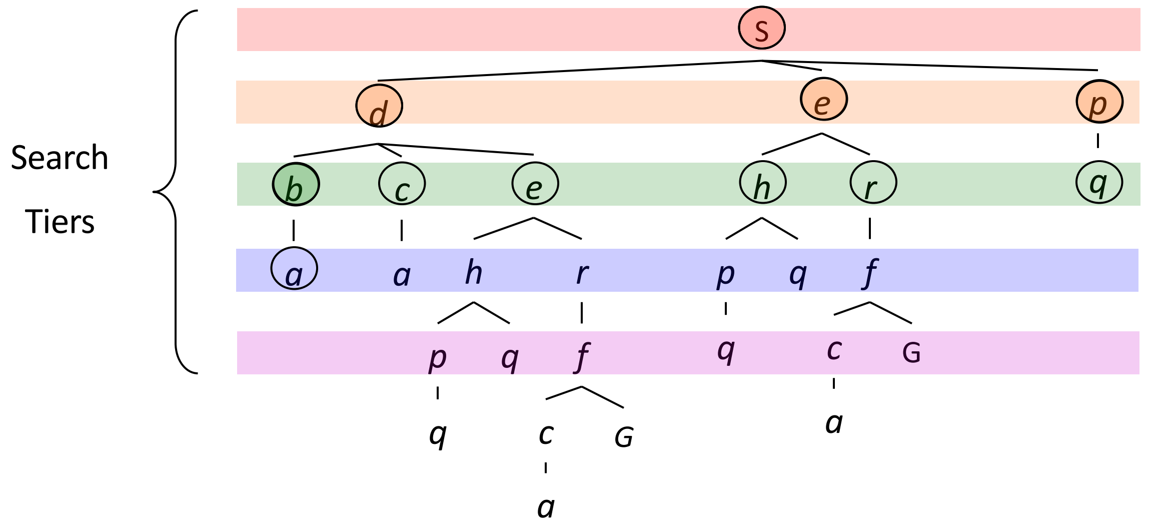


Breadth-First Search

Strategy: expand a shallowest node first

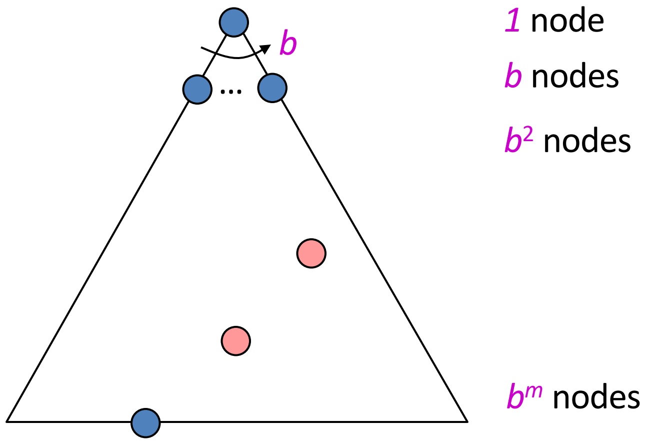
Implementation: Frontier is a FIFO queue





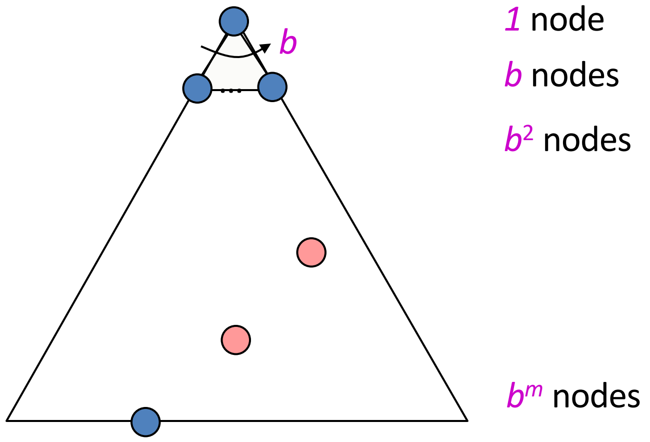
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?



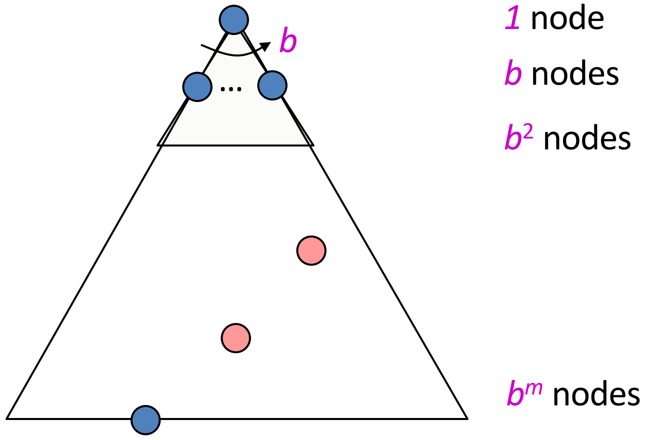
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?



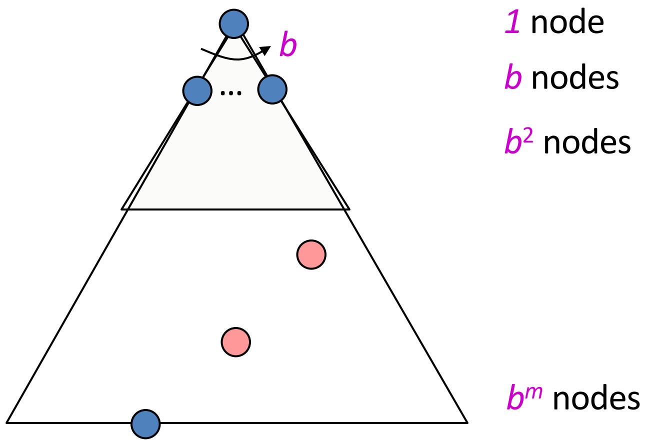
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?



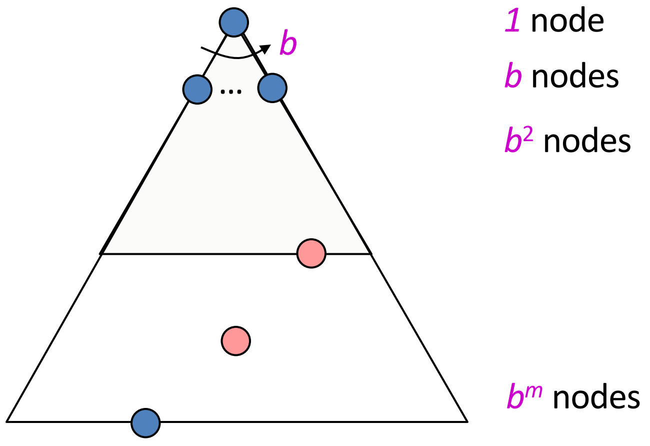
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?



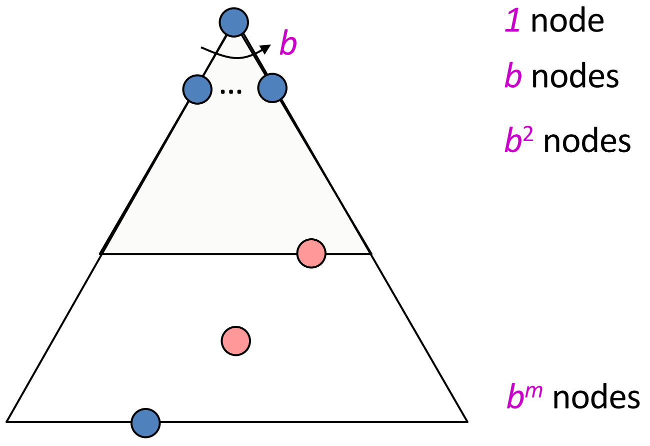
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?



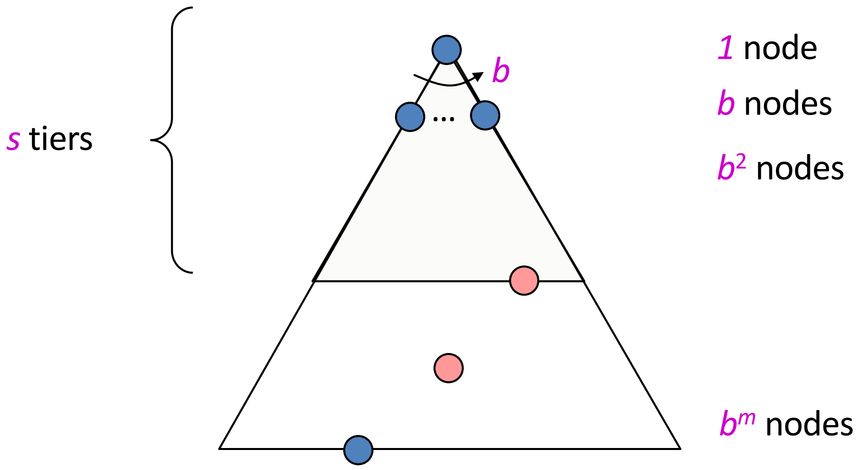
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution



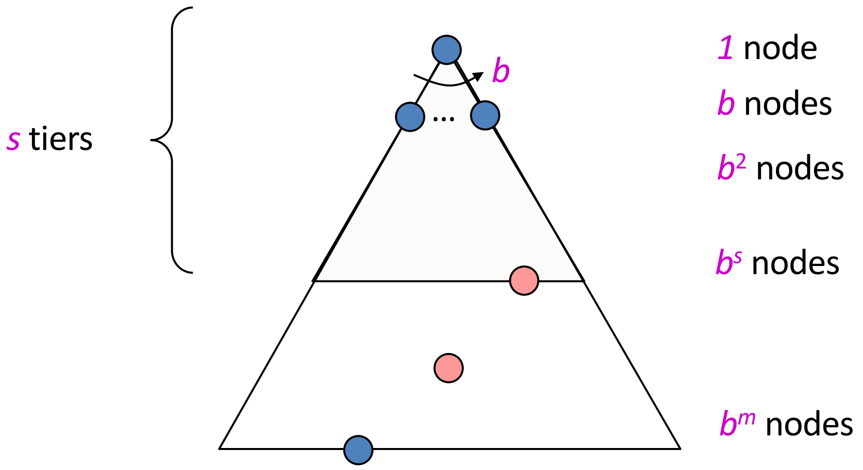
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be *s*



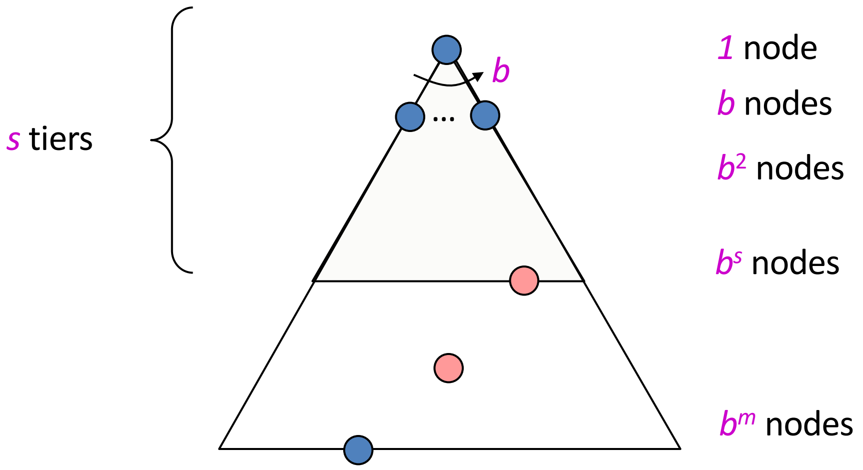
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s



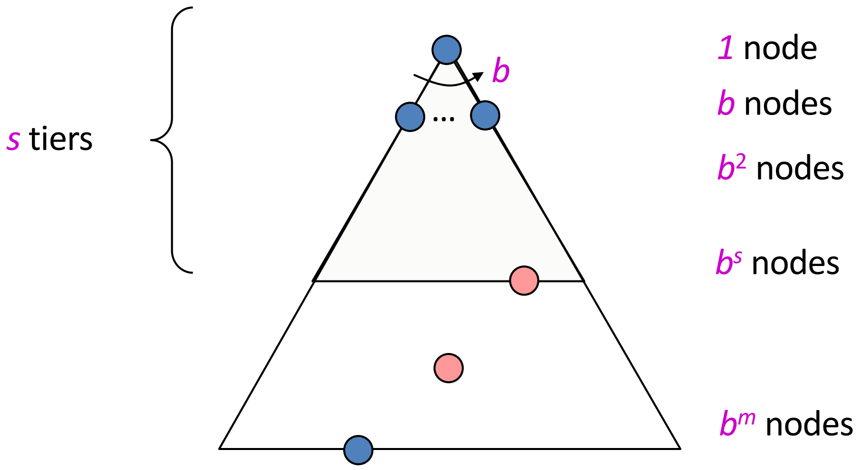
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$



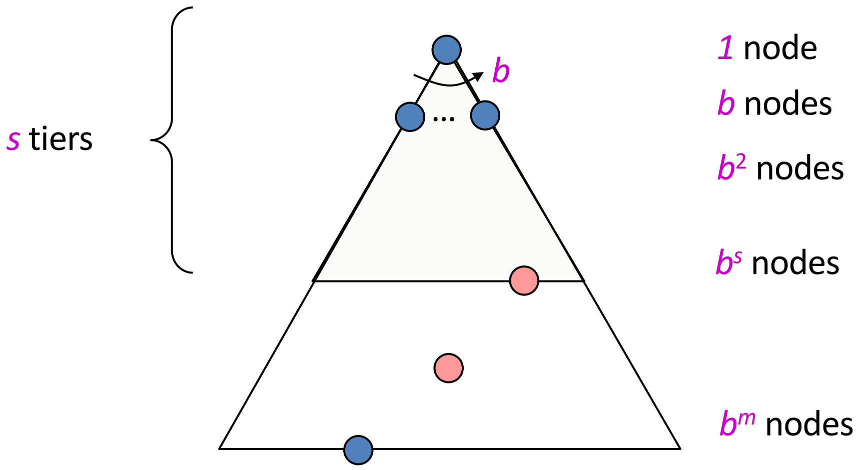
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?



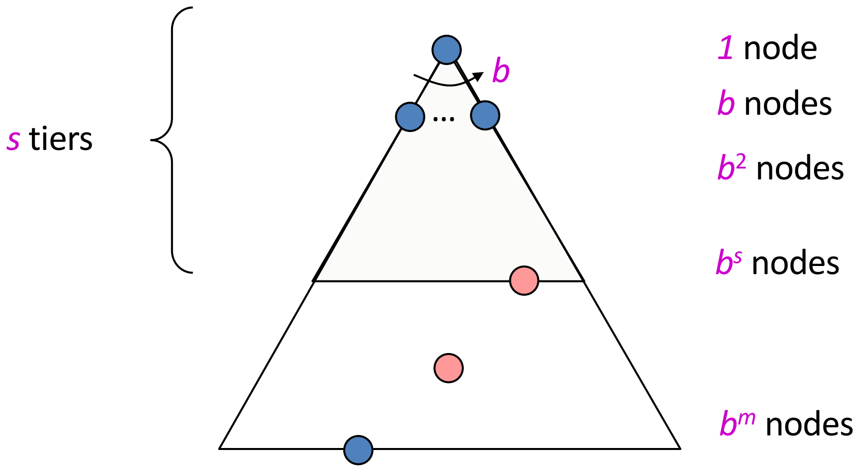
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$



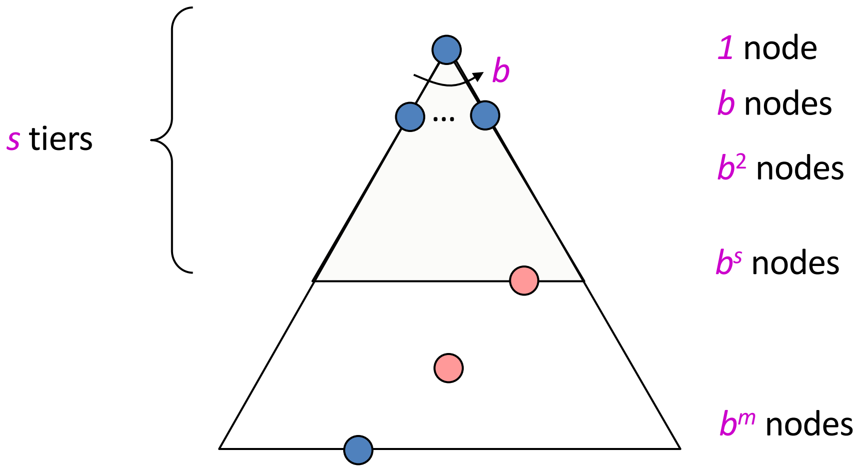
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?



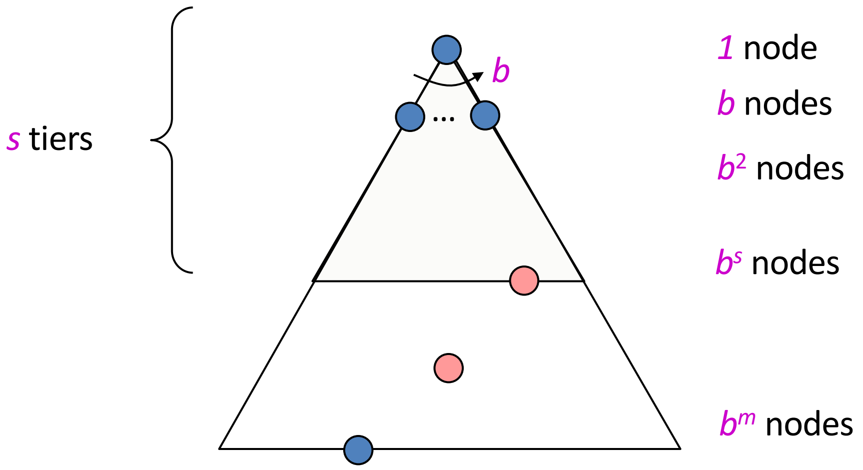
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!



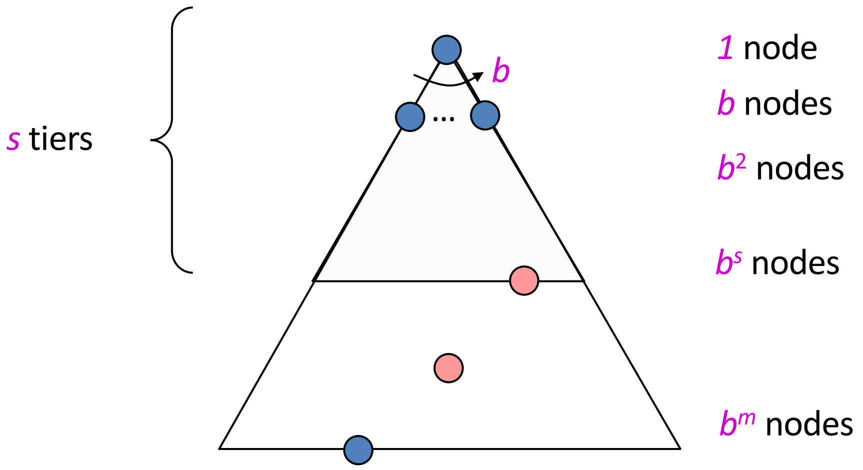
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?



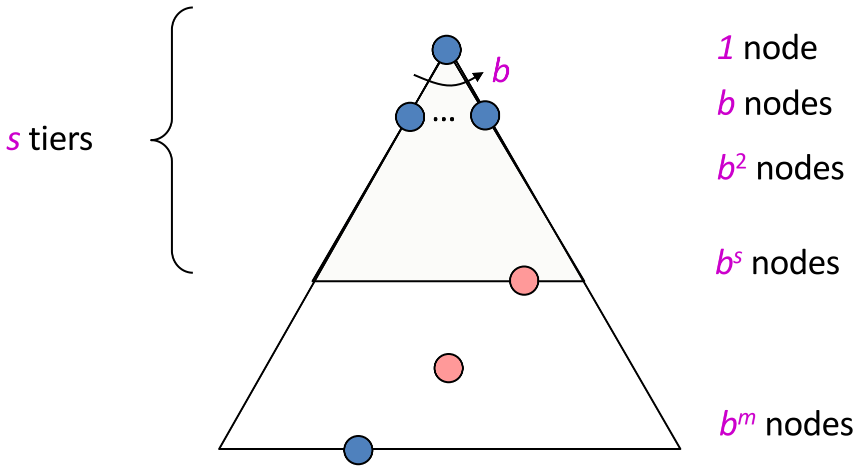
Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be *s*
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - *s* must be finite if a solution exists, so yes!
- Is it optimal?
  - If costs are equal for each operator (e.g., 1)



Breadth-First Search (BFS) Properties

- What nodes does BFS expand?
  - Processes all nodes above shallowest solution
  - Let depth of shallowest solution be s
  - Search takes time $O(b^s)$
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^s)$
- Is it complete?
  - s must be finite if a solution exists, so yes!
- Is it optimal?
  - If costs are equal for each operator (e.g., 1)



Potential issues??

Breadth-First Search

- Takes a **long time to find solutions** with large number of steps because must explore all shorter length possibilities first

Breadth-First Search

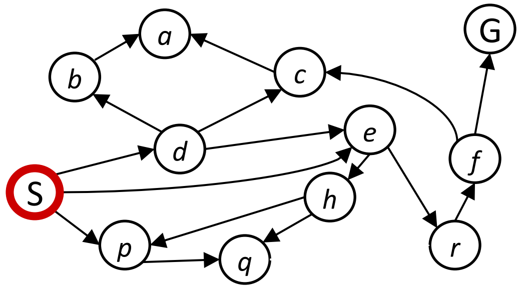
Long time to find solutions with many steps: we must look at all shorter length possibilities first

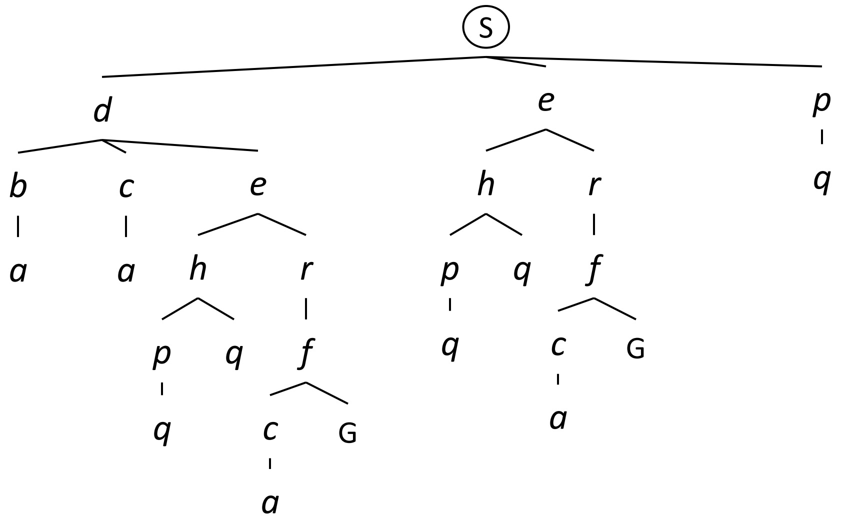
- Complete search tree of depth d where nodes have b children has $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes = $0(b^d)$
- Tree of depth 12 with branching 10 has more than a trillion nodes
- If BFS expands 1000 nodes/sec and nodes uses 100 bytes, then it may take 35 years to run and uses 111 terabytes of memory!

Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

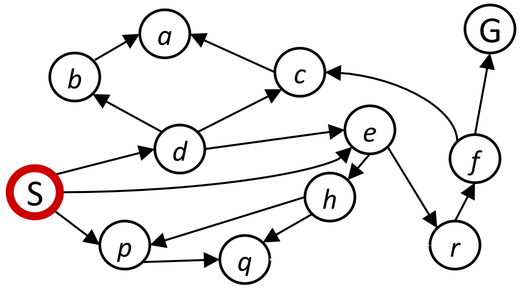


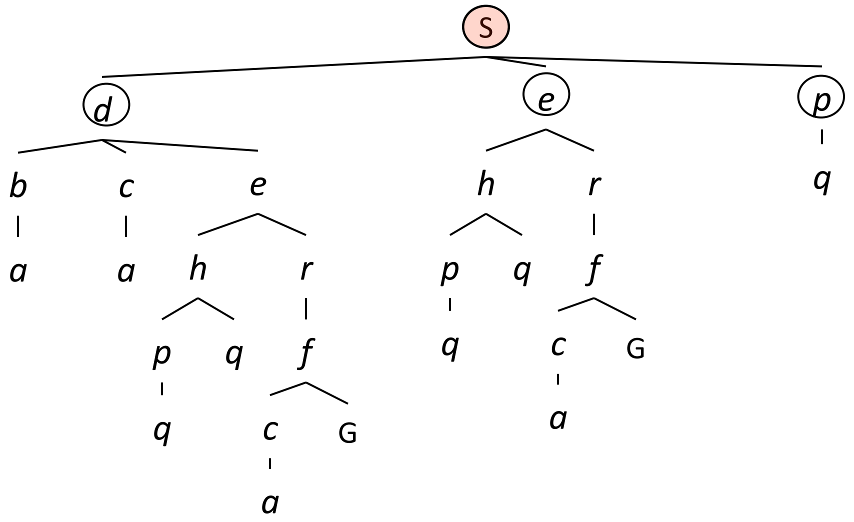


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

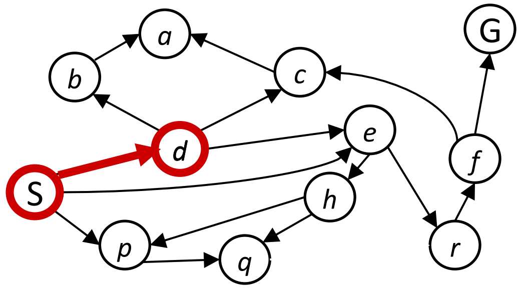


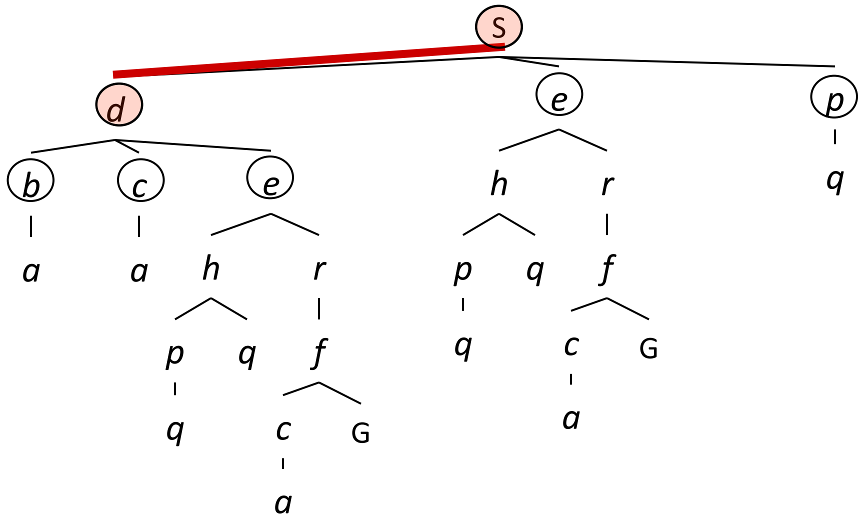


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

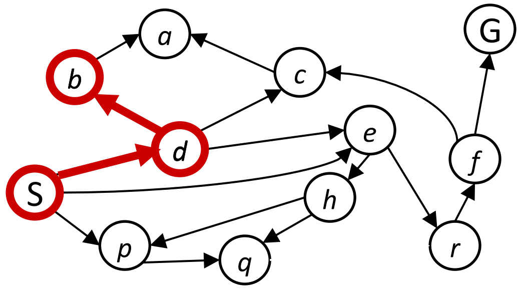


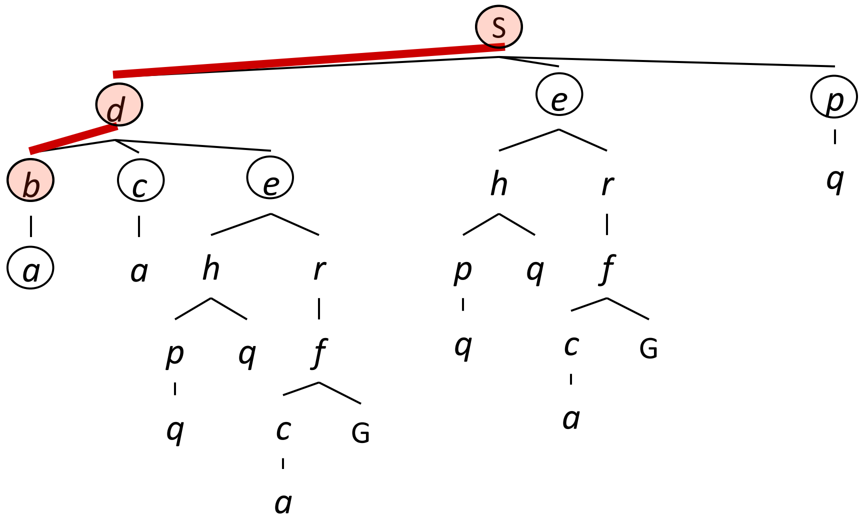


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

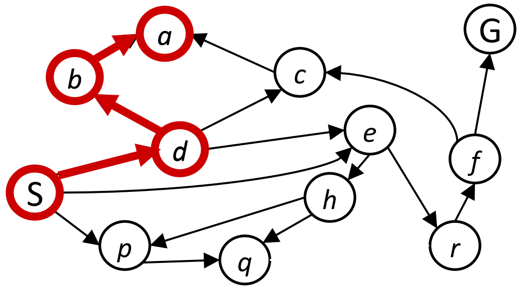


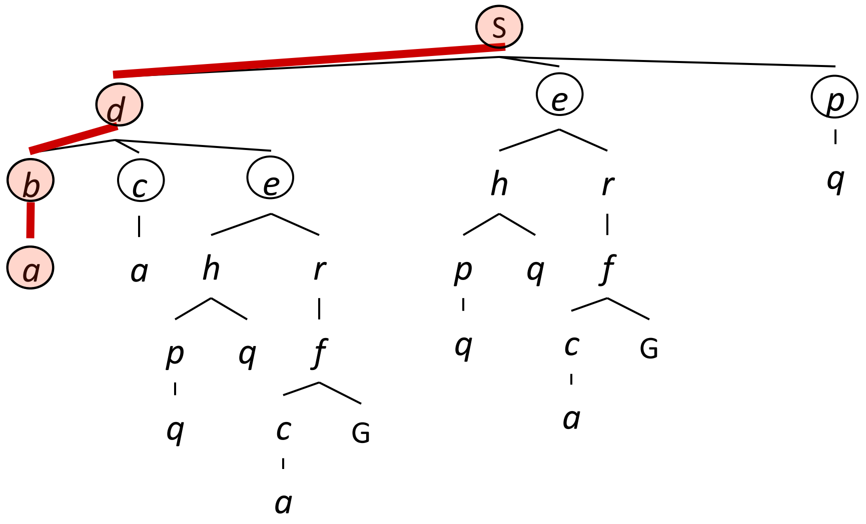


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

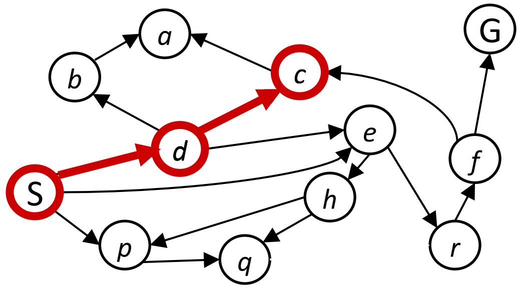


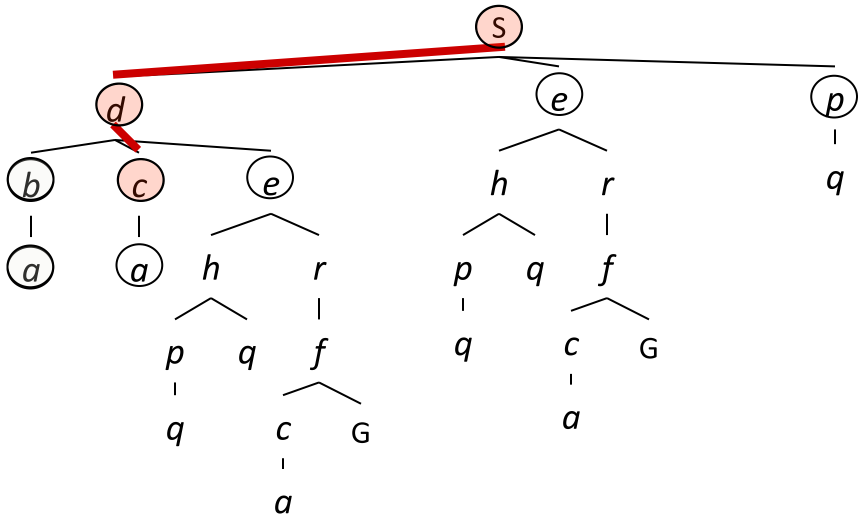


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

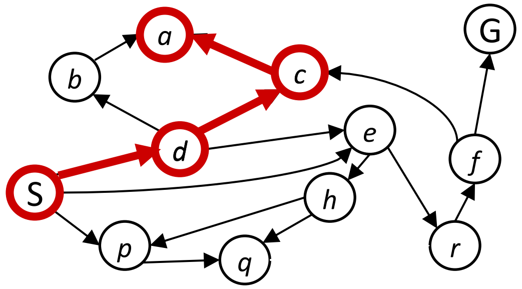


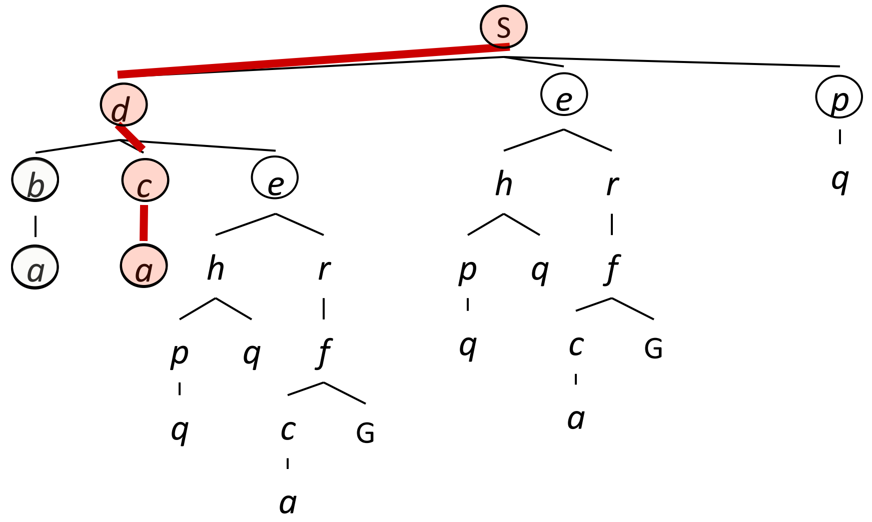


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

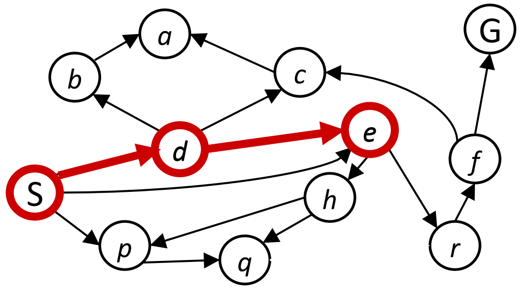


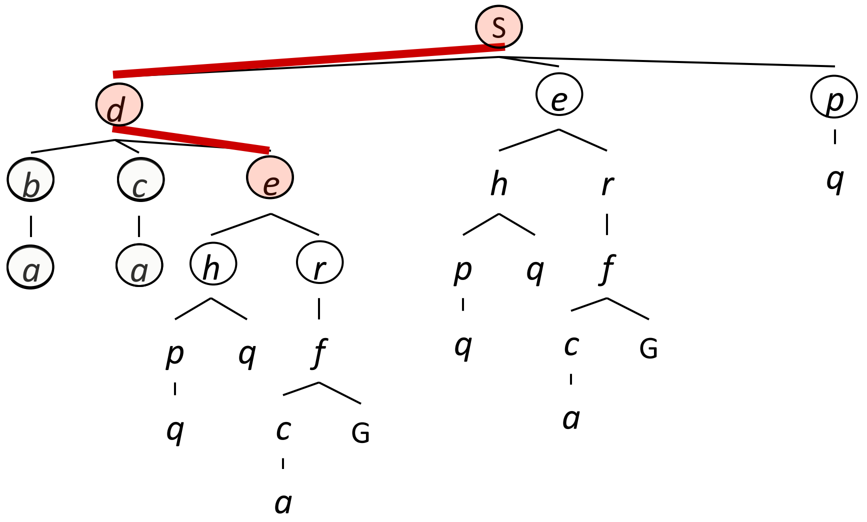


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

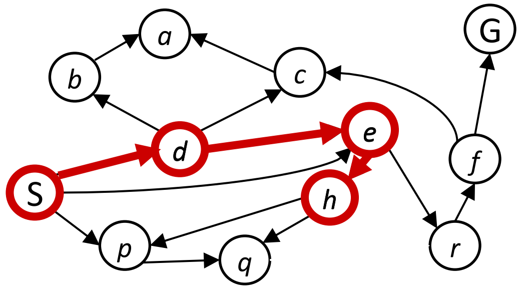


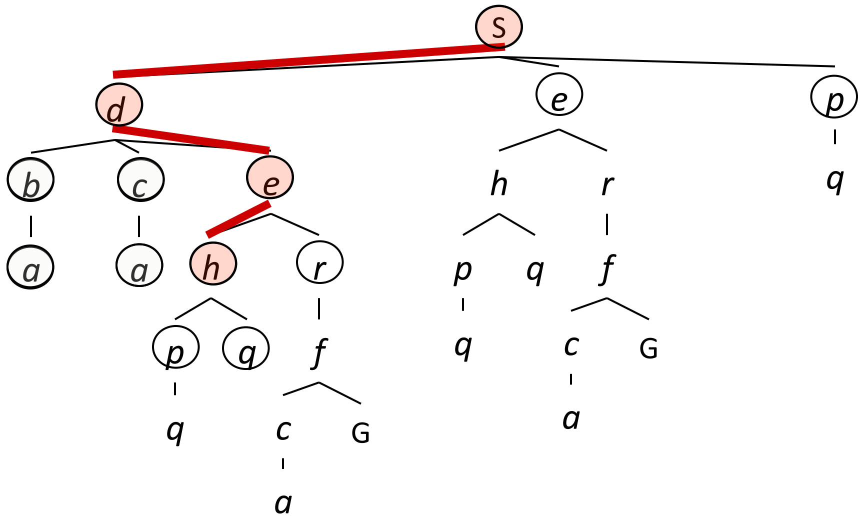


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

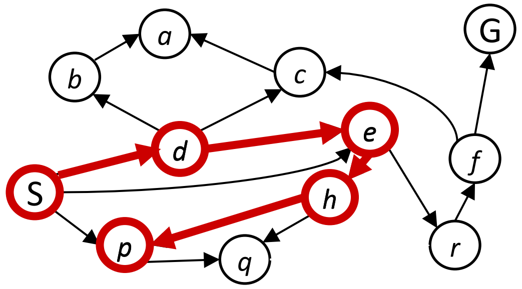


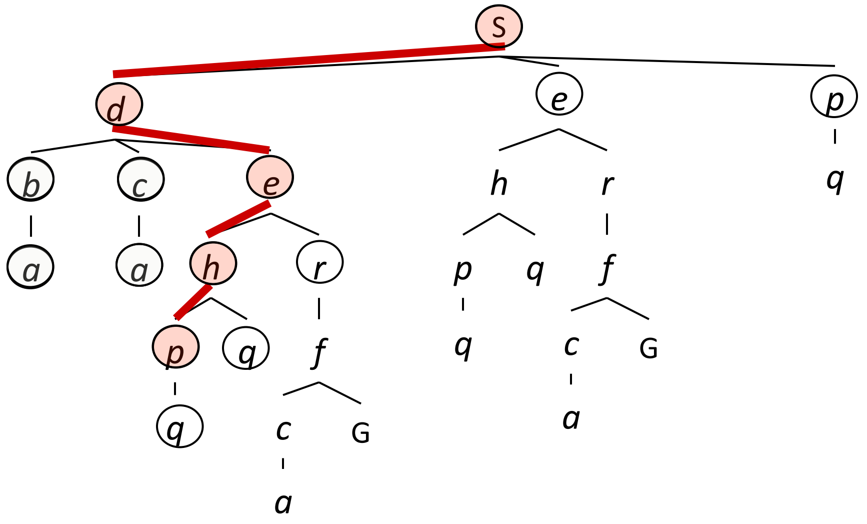


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

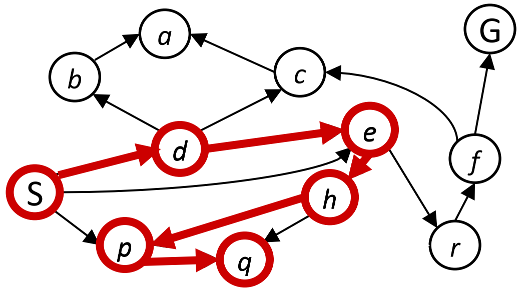


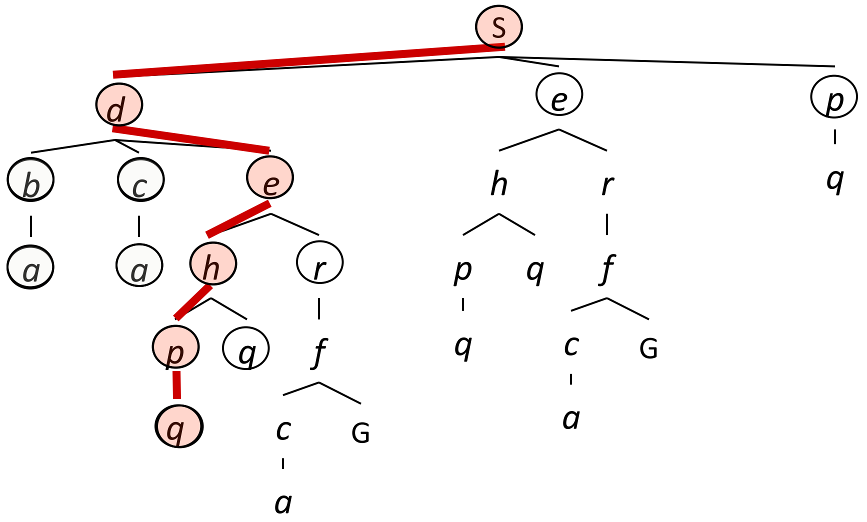


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

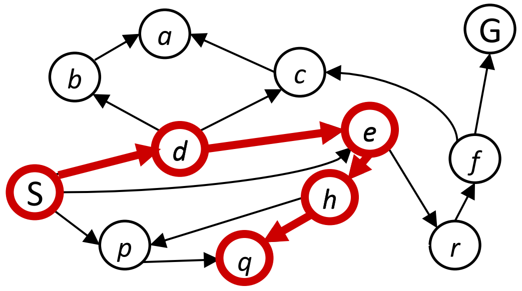


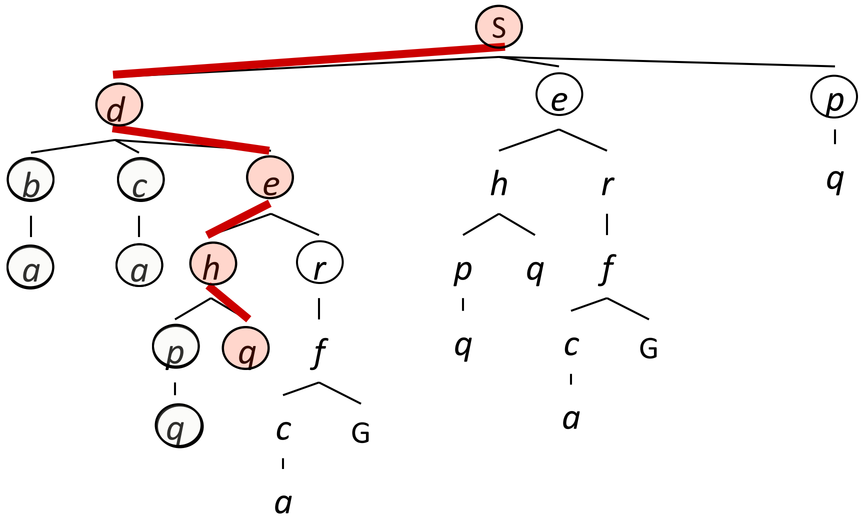


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

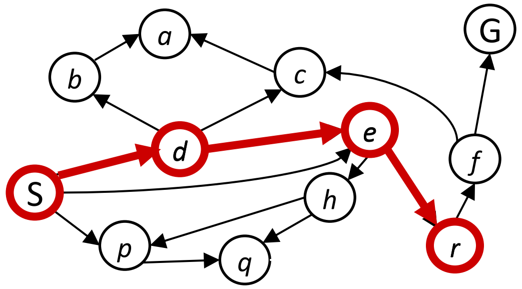


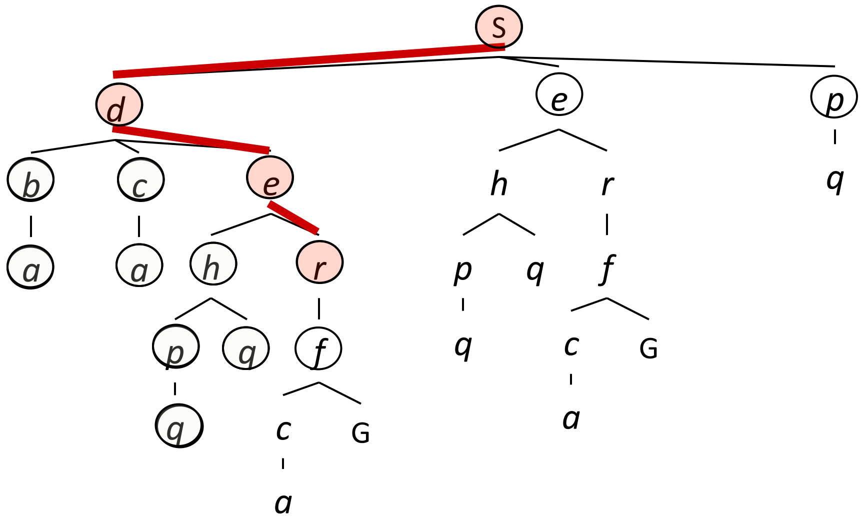


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

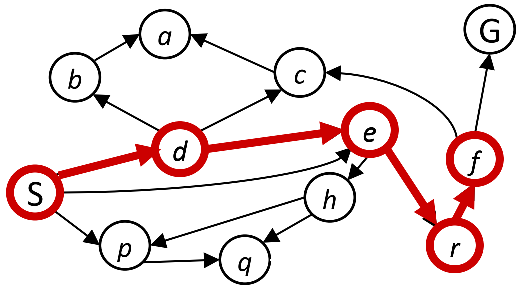


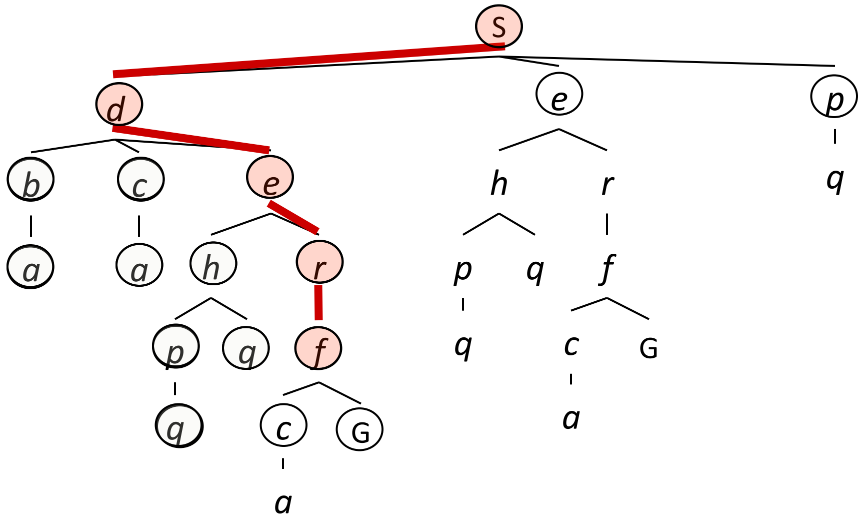


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

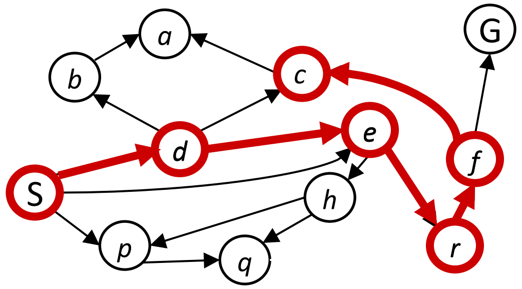


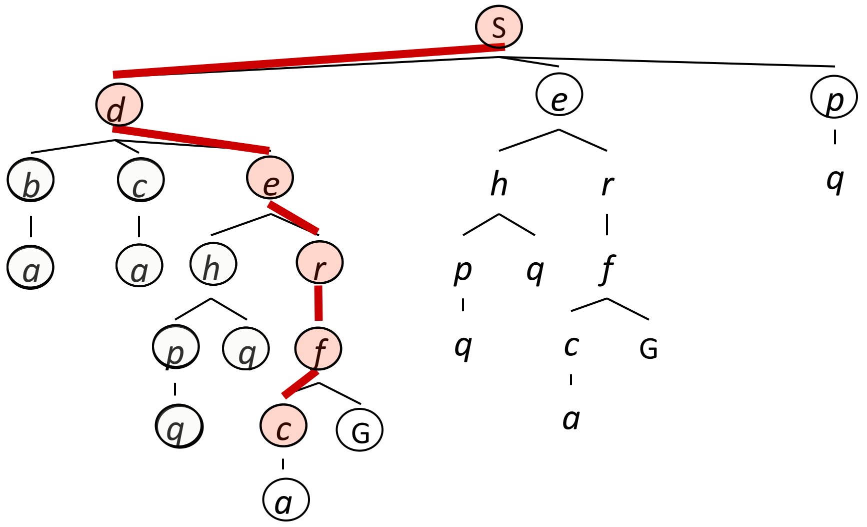


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

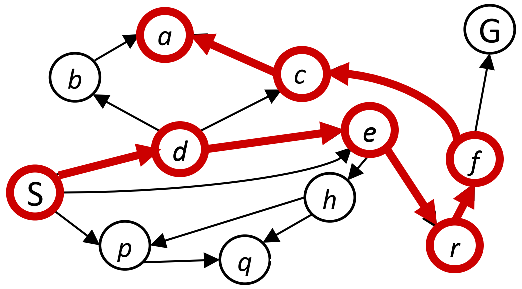


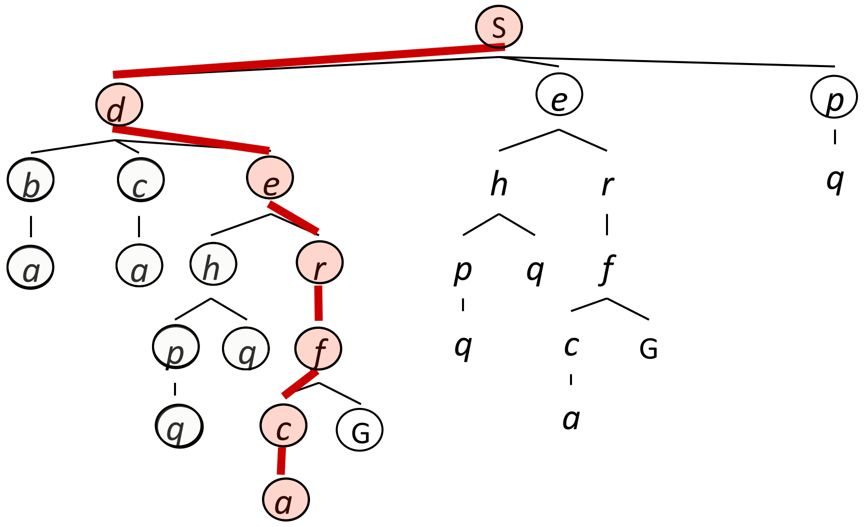


Depth-First Search

Strategy: expand a deepest node first

Implementation: Frontier is a LIFO stack

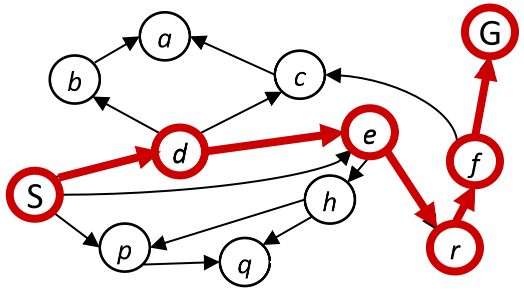


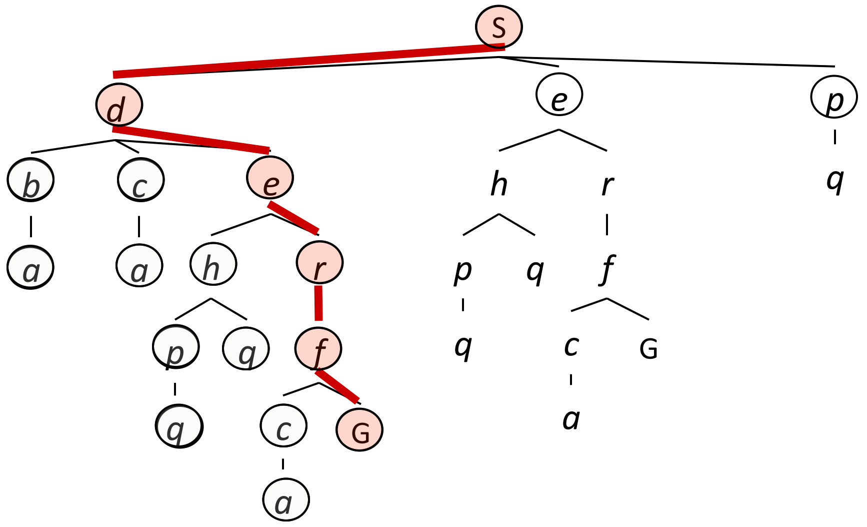


Depth-First Search

Strategy: expand a deepest node first

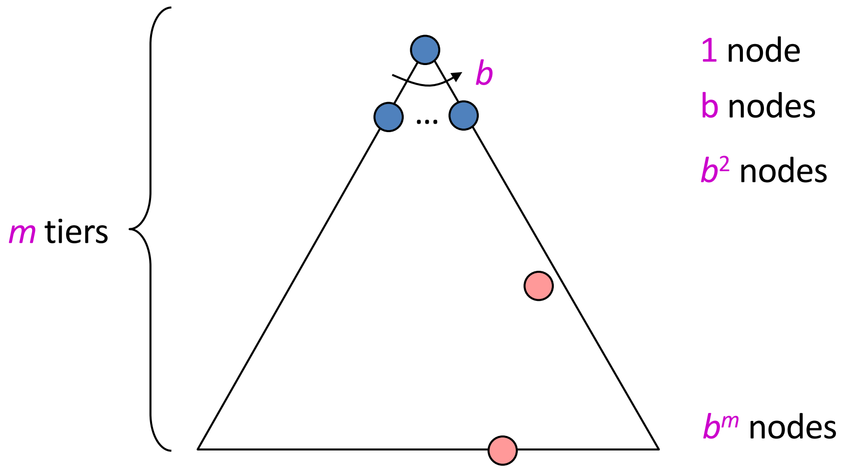
Implementation: Frontier is a LIFO stack





Depth-First Search (DFS) Properties

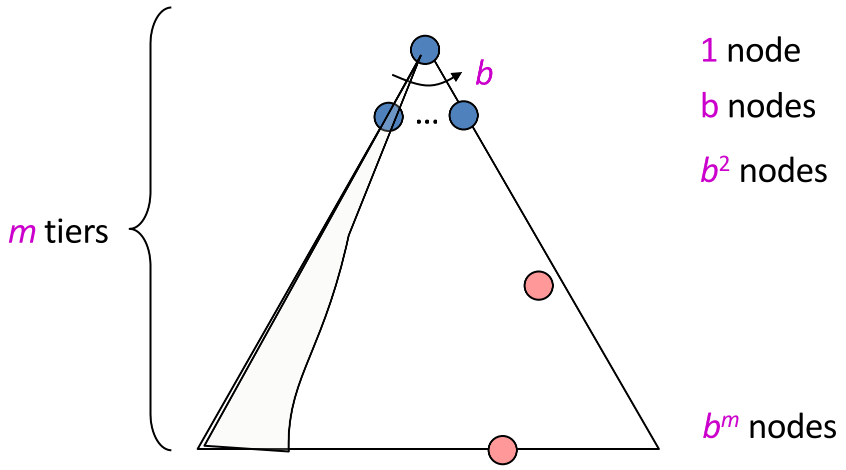
- What nodes does DFS expand?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

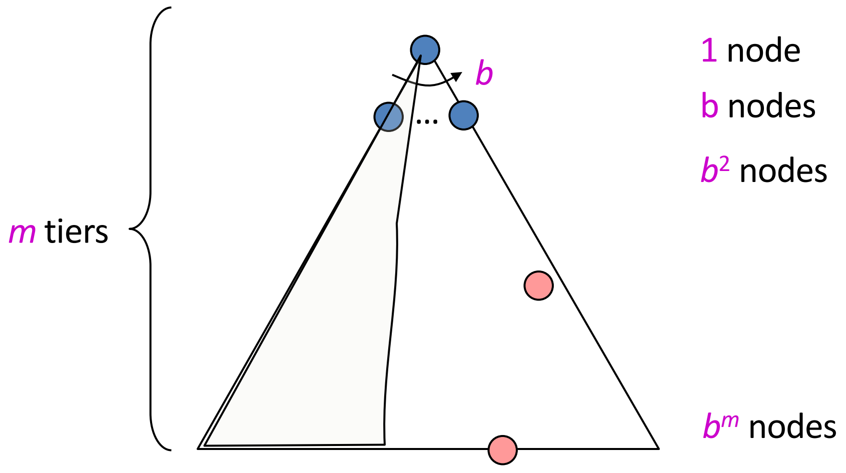
- What nodes does DFS expand?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

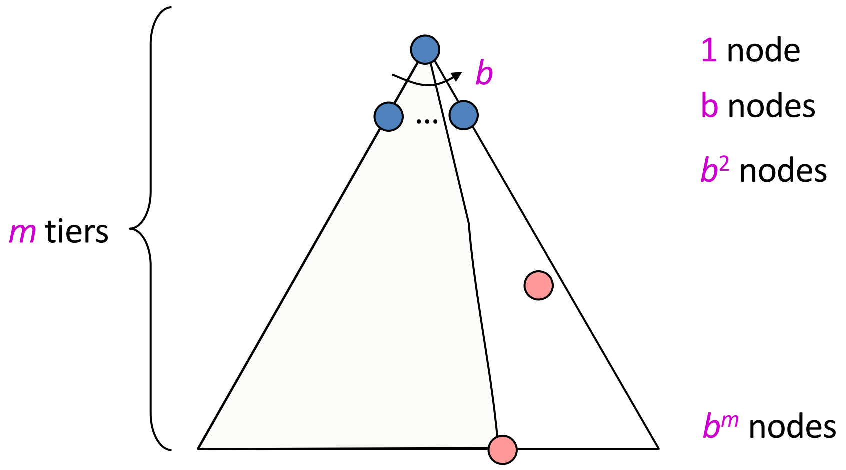
- What nodes does DFS expand?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

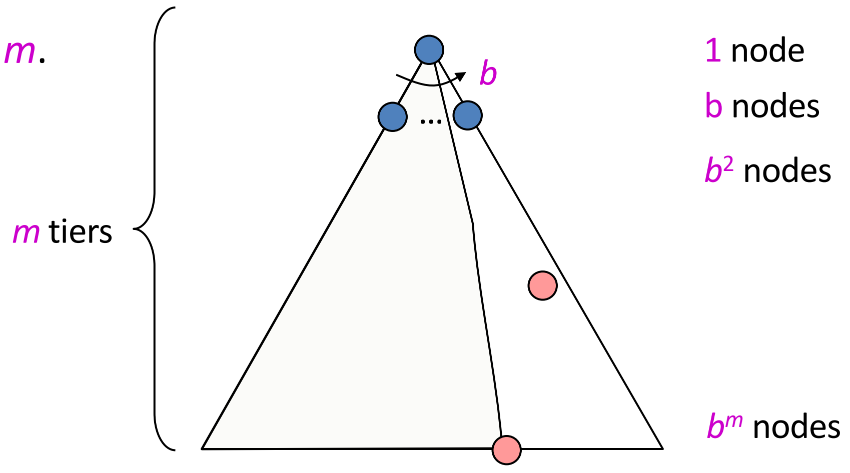
- What nodes does DFS expand?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

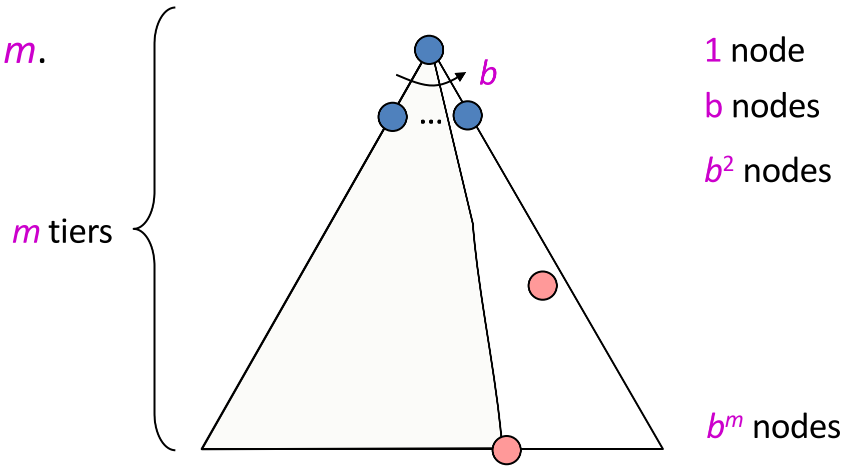
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

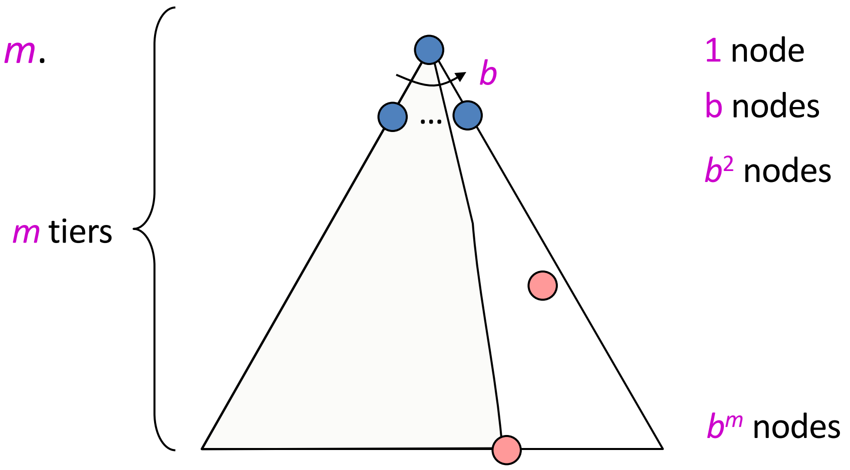
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

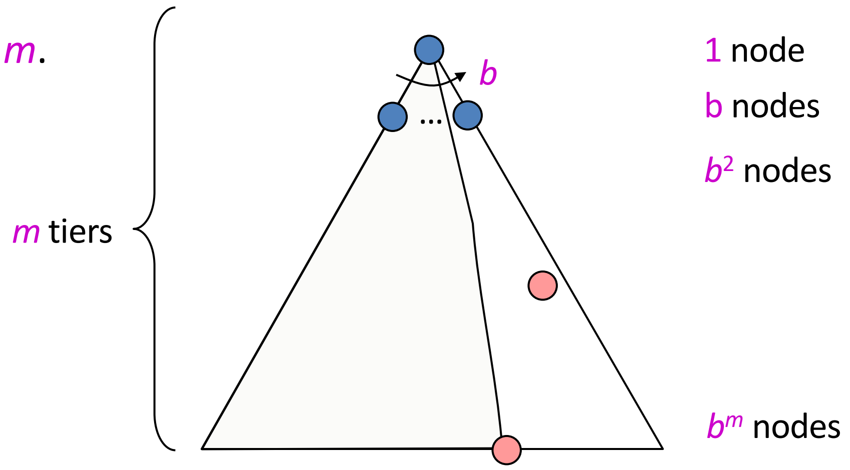
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

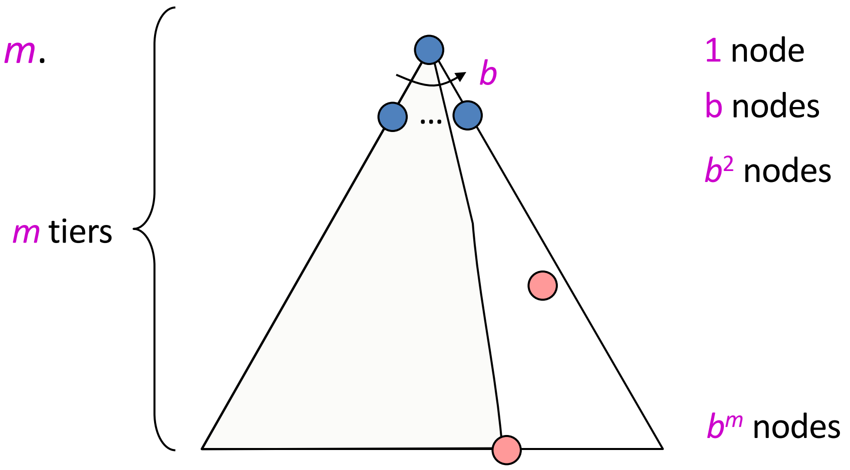
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

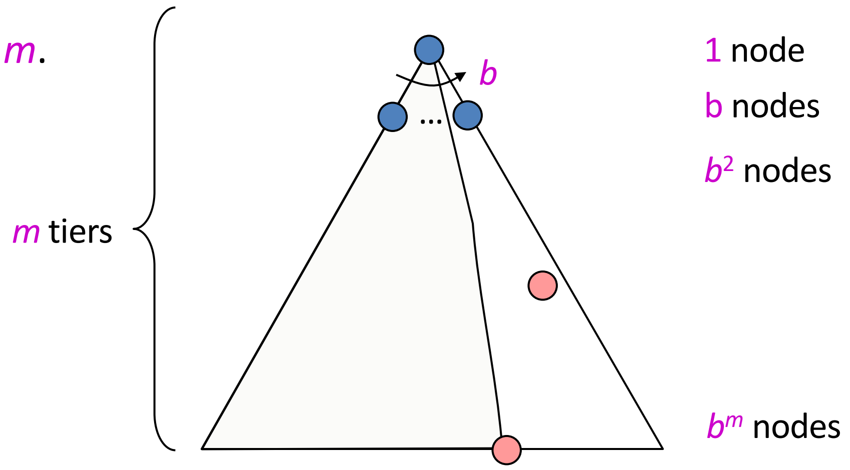
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

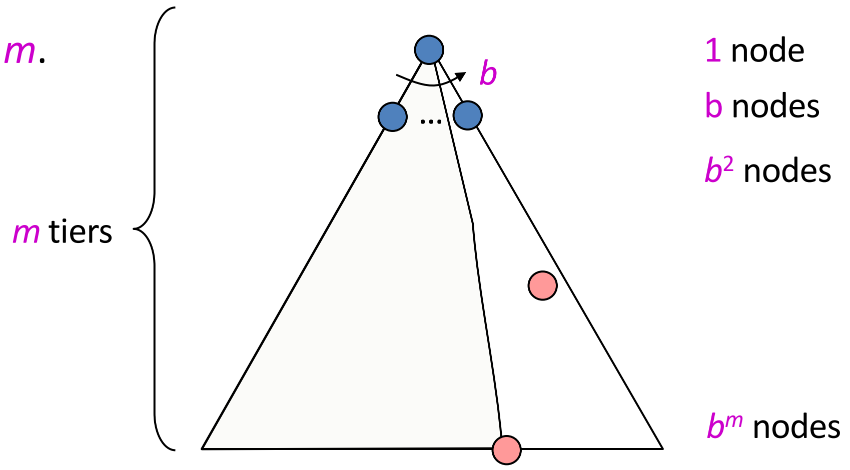
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

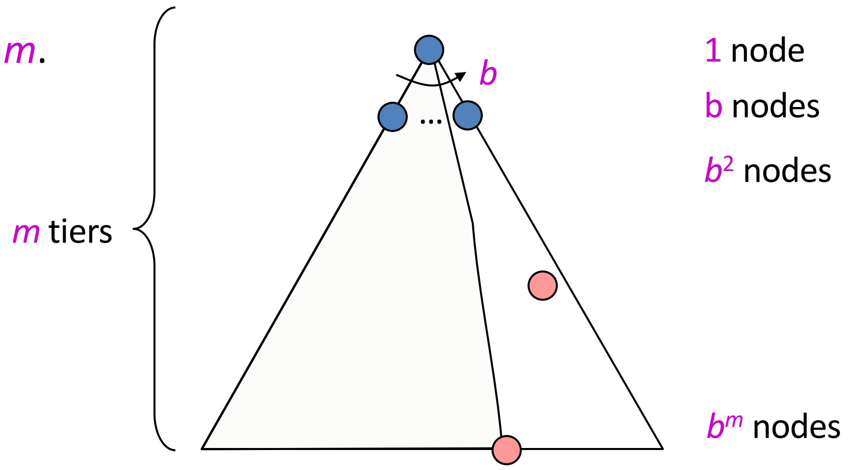
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - *m* could be infinite



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

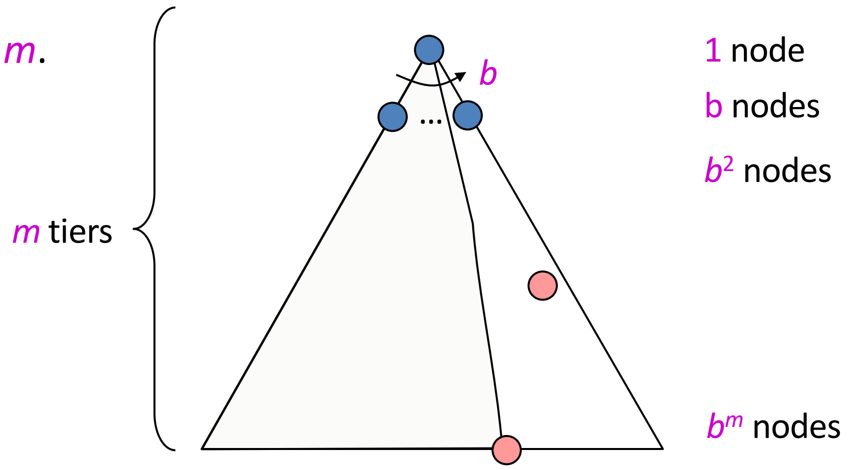
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - *m* could be infinite
  - preventing cycles may help



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

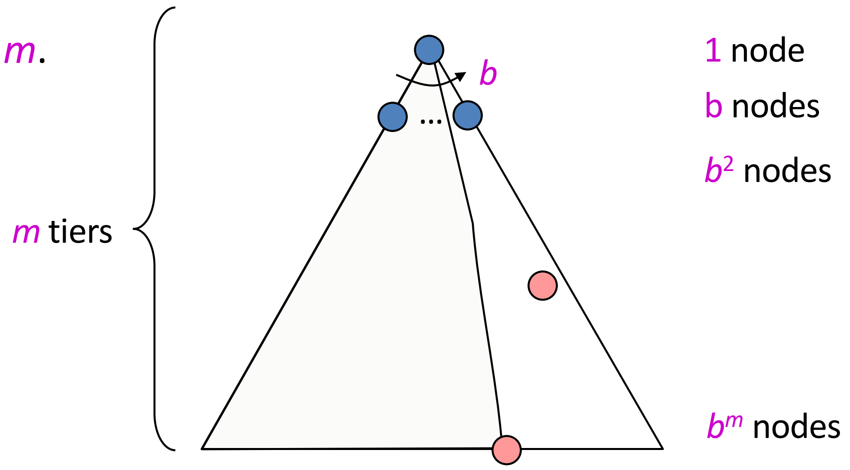
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - *m* could be infinite
  - preventing cycles may help
  - **May not terminate** w/o depth bound, i.e., ending search below fixed depth D (depth-limited search)



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

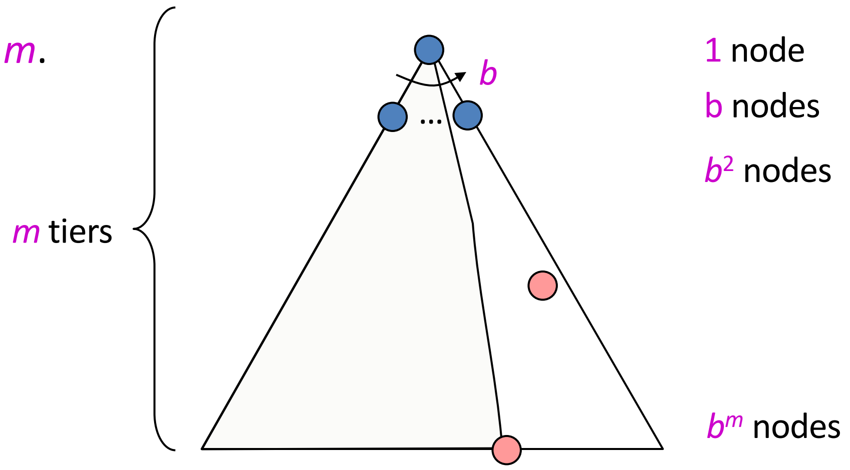
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth m.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - m could be infinite
  - preventing cycles may help
  - **May not terminate** w/o depth bound, i.e., ending search below fixed depth D (depth-limited search)
- Is it optimal?



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Depth-First Search (DFS) Properties

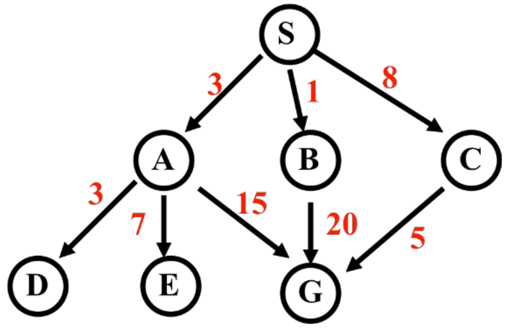
- What nodes does DFS expand?
  - Some left prefix of the tree down to depth *m*.
  - Could process the whole tree!
  - If m is finite, takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm)$
- Is it complete?
  - *m* could be infinite
  - preventing cycles may help
  - **May not terminate** w/o depth bound, i.e., ending search below fixed depth D (depth-limited search)
- Is it optimal?
  - No, it finds the "leftmost" solution, regardless of depth or cost



Can find **long solutions quickly** if lucky (and **short solutions slowly** if unlucky!)

Breadth-First Search

weighted arcs



Expanded node	Nodes list (aka Fringe)
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ B^1 C^8 D^6 E^{10} G^{18} }
B^1	{ C^8 D^6 E^{10} G^{18} G^{21} }
C^8	{ D^6 E^{10} G^{18} G^{21} G^{13} }
D^6	{ E^{10} G^{18} G^{21} G^{13} }
E^{10}	{ G^{18} G^{21} G^{13} }
G^{18}	{ G^{21} G^{13} }

Notation

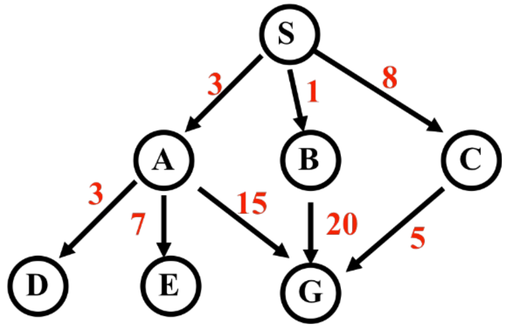
G^{18}

G is node; 18 is cost of shortest known path from

start node S

Note: we typically don’t check for goal until we expand node
Solution path found is S A G , cost 18
Number of nodes expanded (including goal node) = 7

Depth-First Search

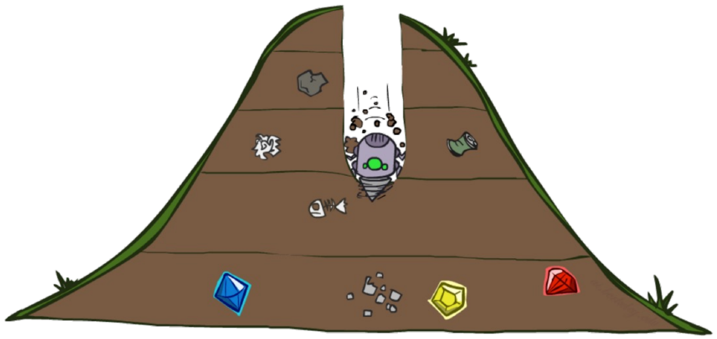


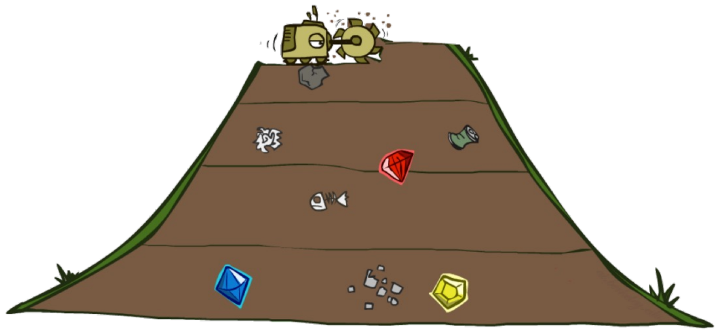
Expanded node	**Nodes list**
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Quiz: DFS vs BFS





Quiz: DFS vs BFS

- When will BFS outperform DFS?

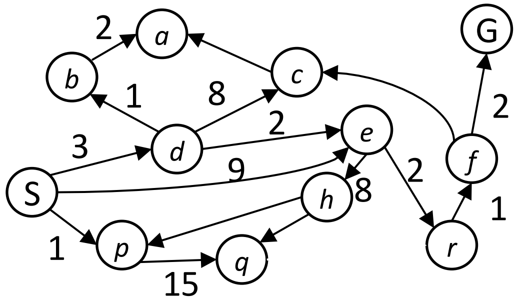
- When will DFS outperform BFS?

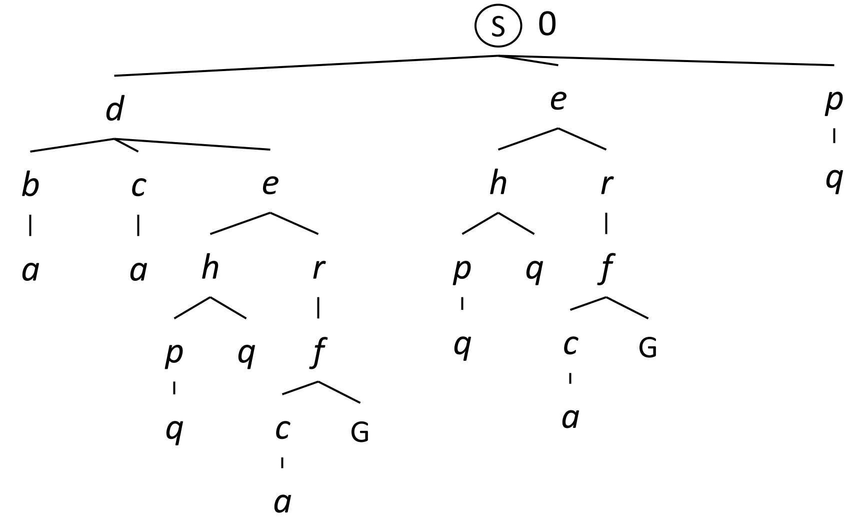
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



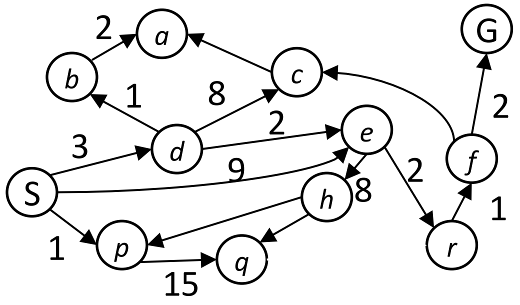


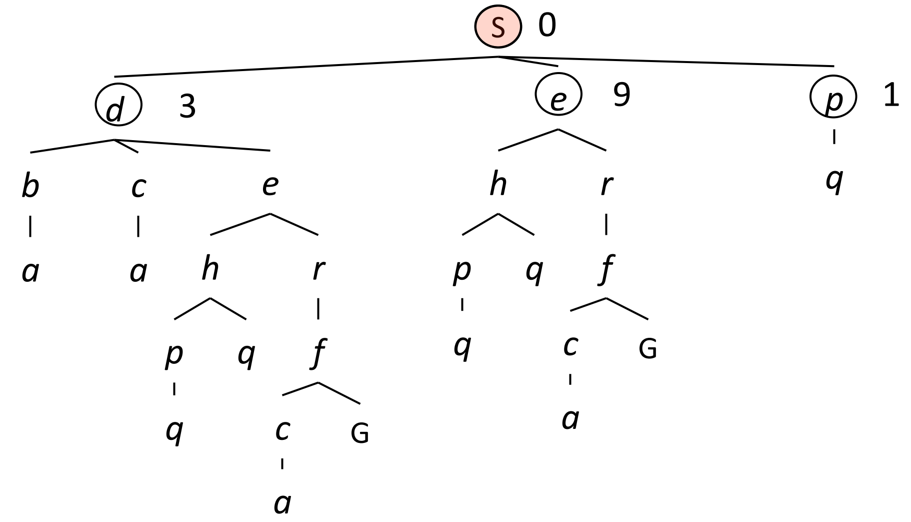
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



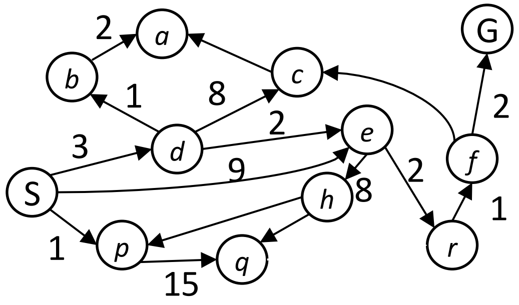


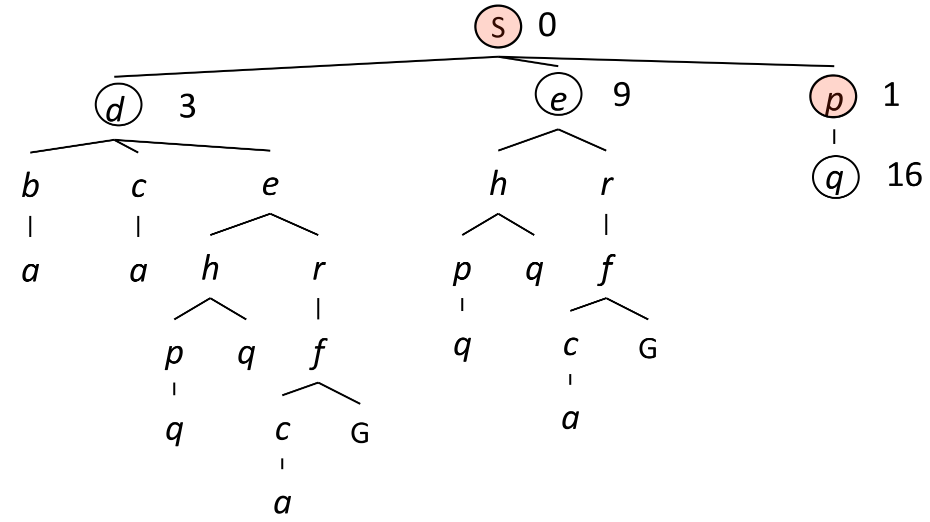
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



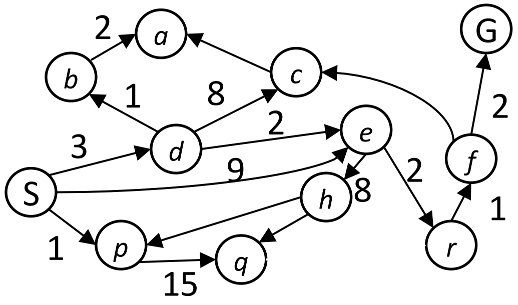


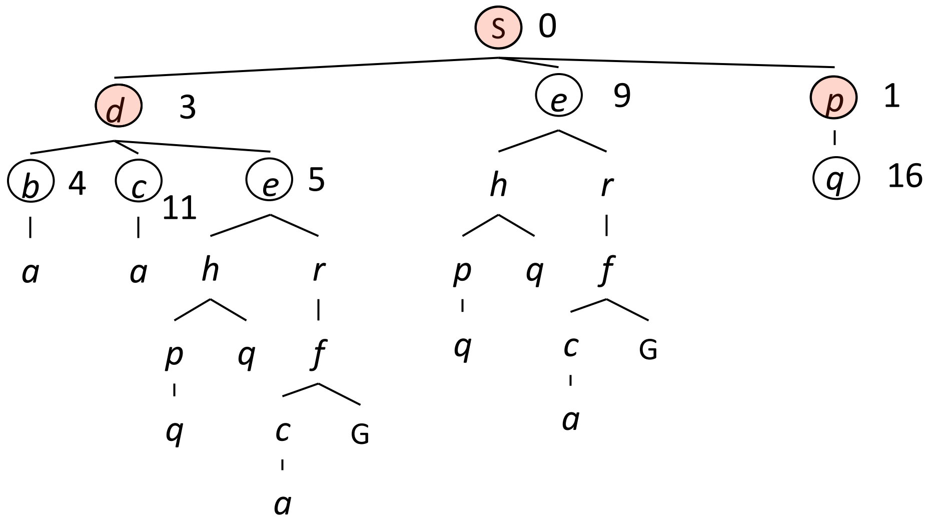
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



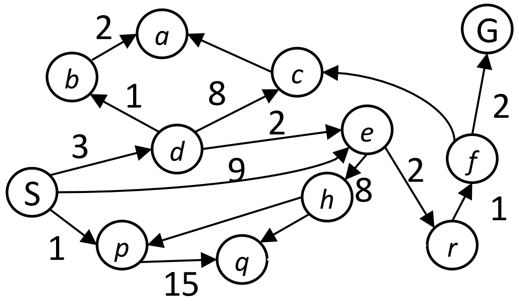


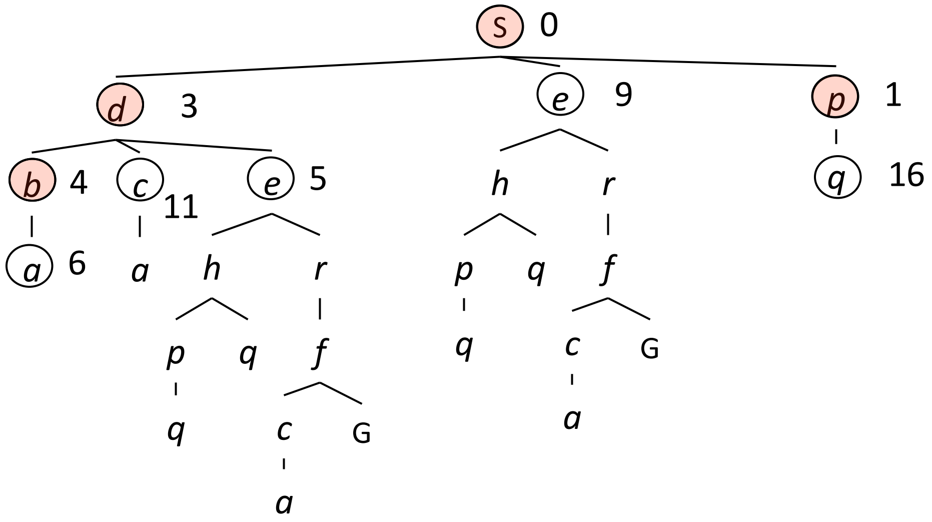
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



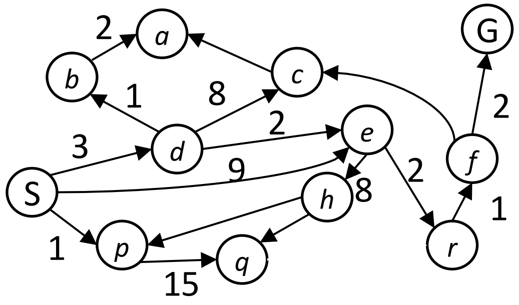


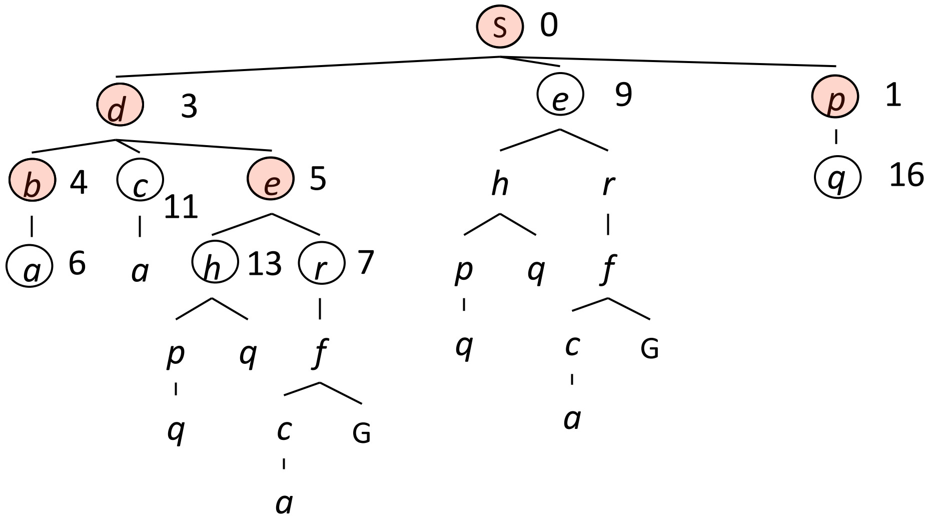
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



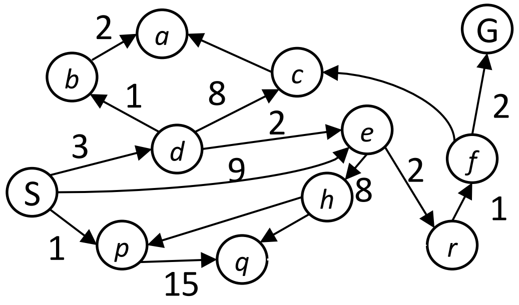


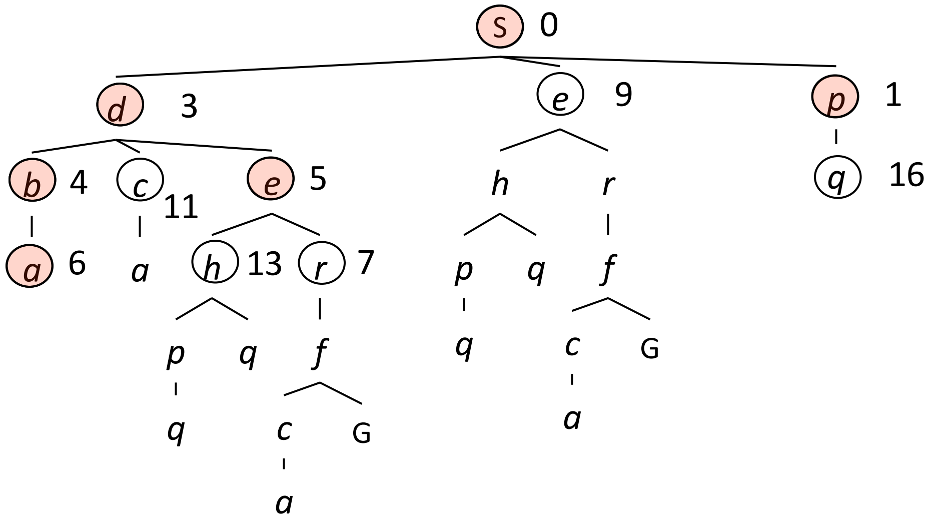
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



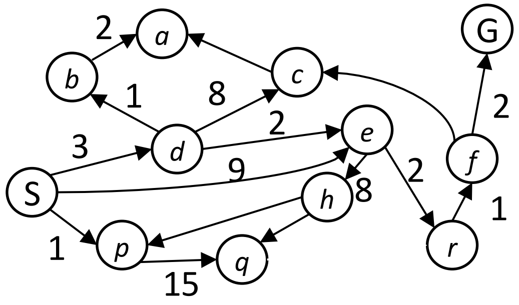


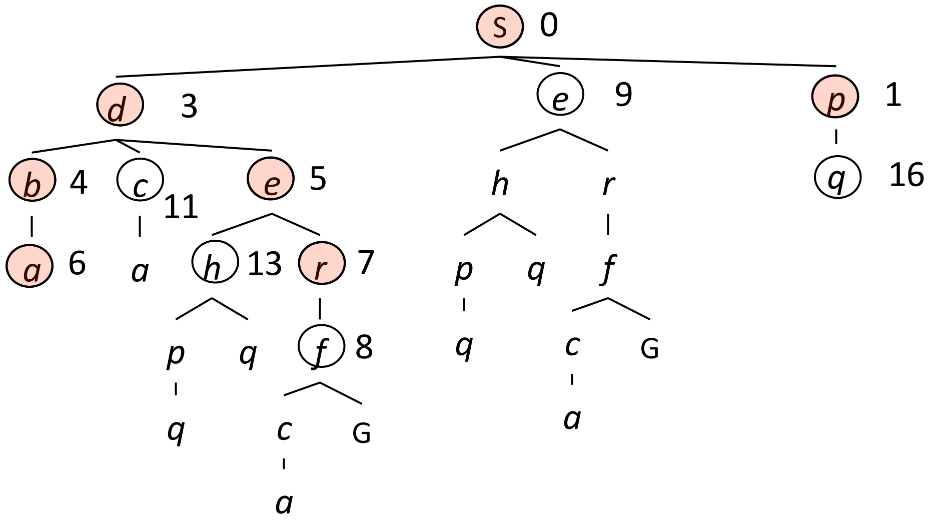
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



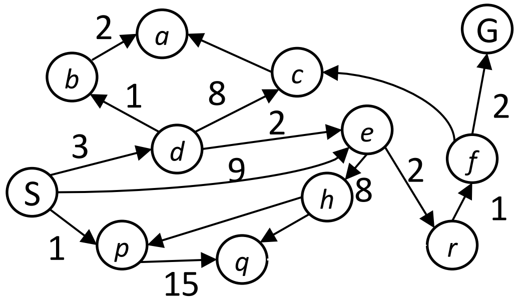


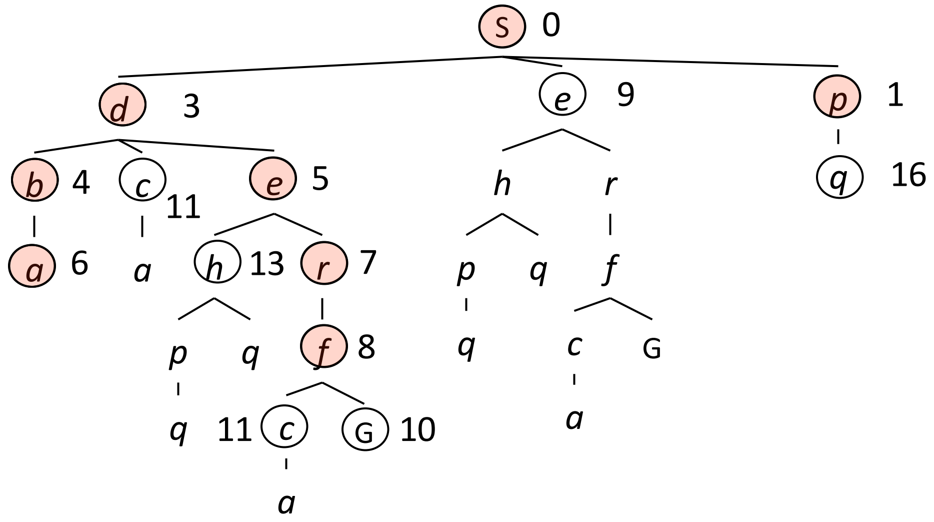
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



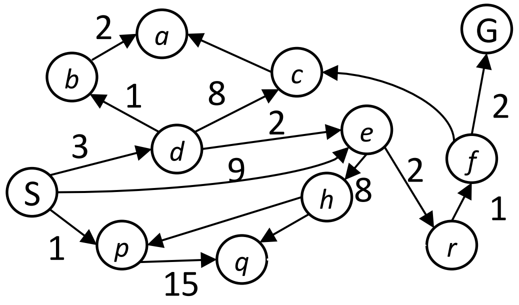


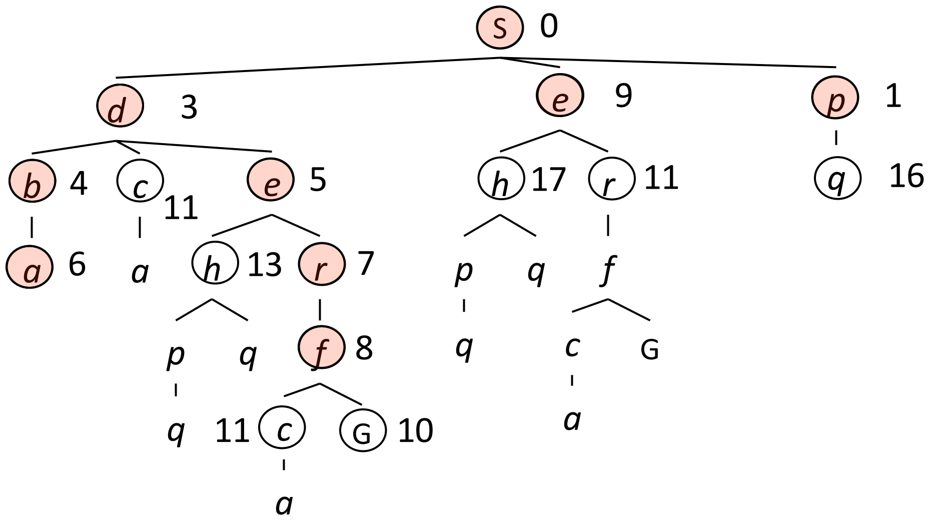
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



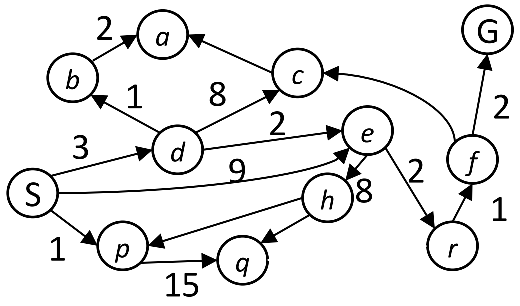


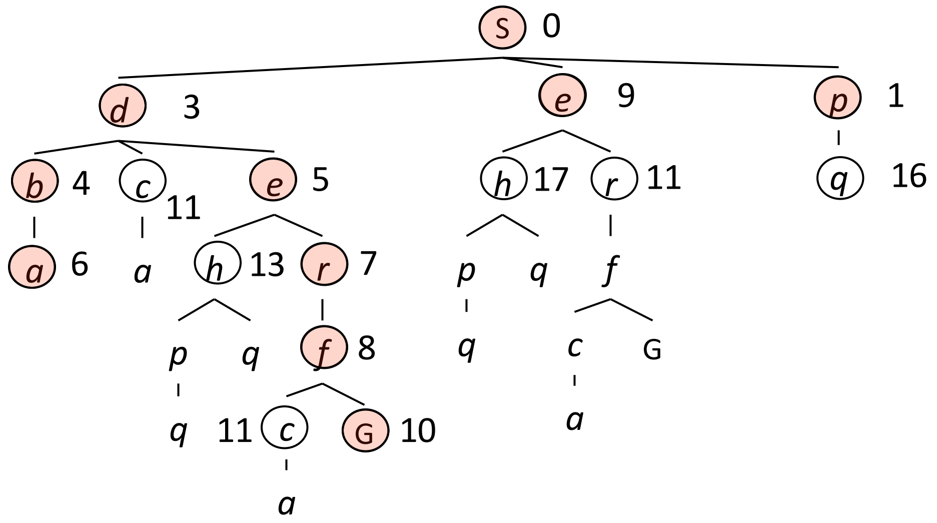
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)



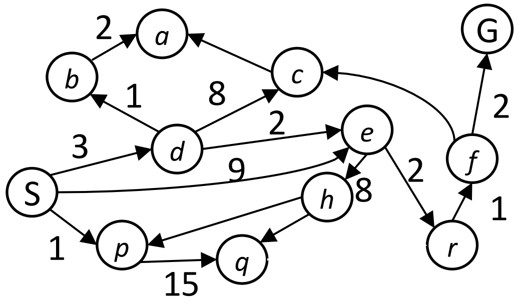


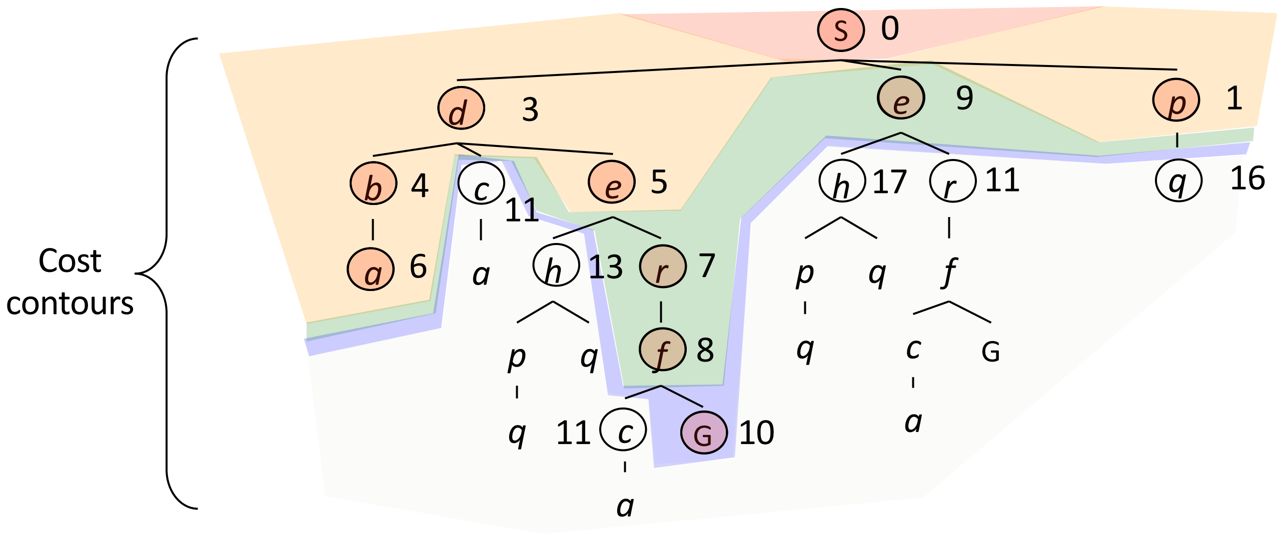
Uniform Cost Search

g(n) = cost from root to n

Strategy: expand lowest g(n)

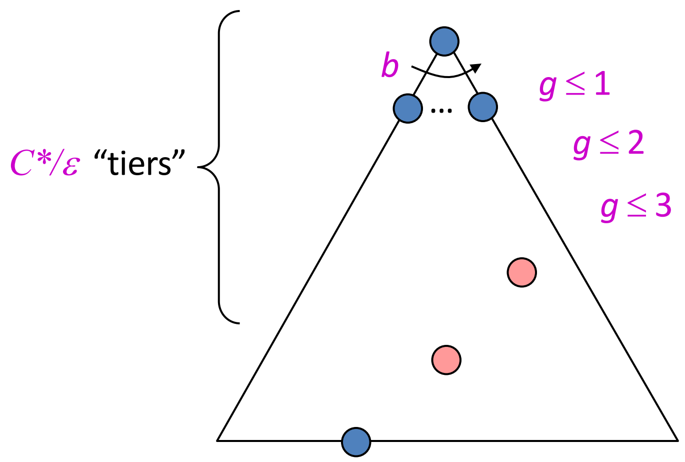
Frontier is a priority queue sorted by g(n)





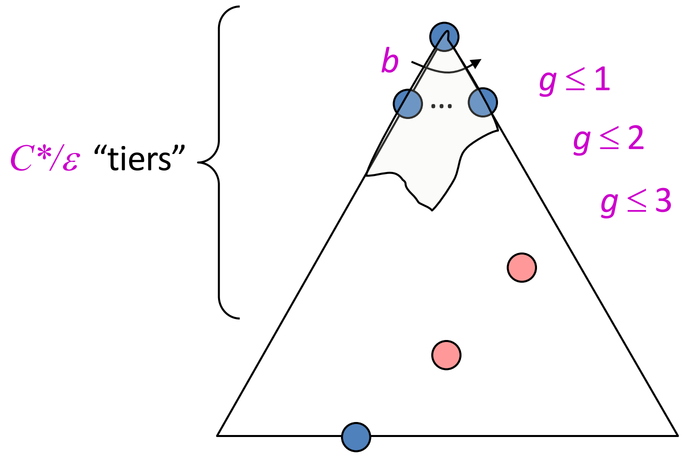
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?



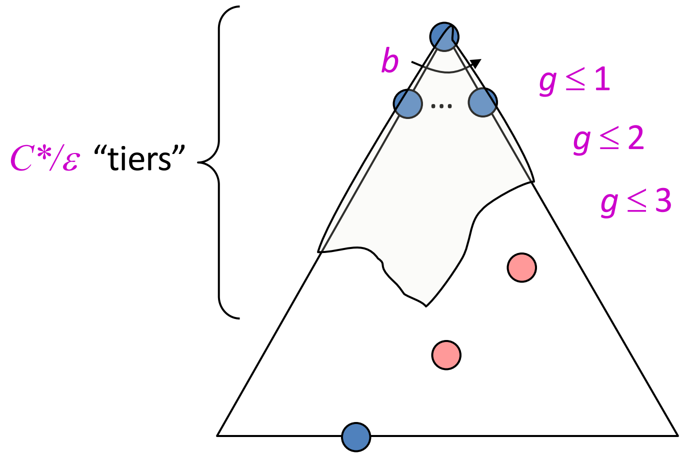
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?



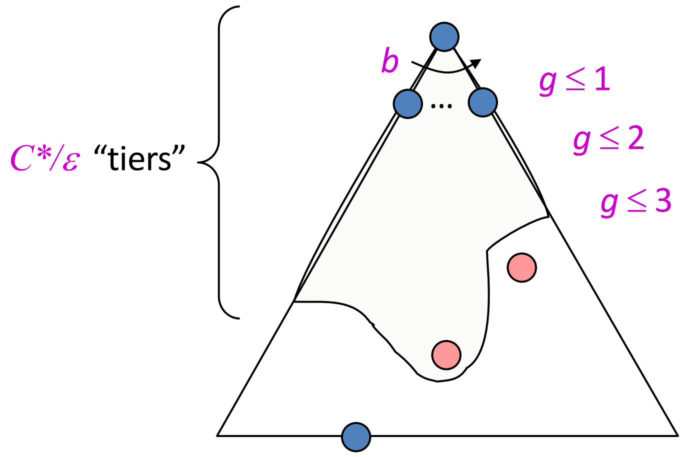
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?



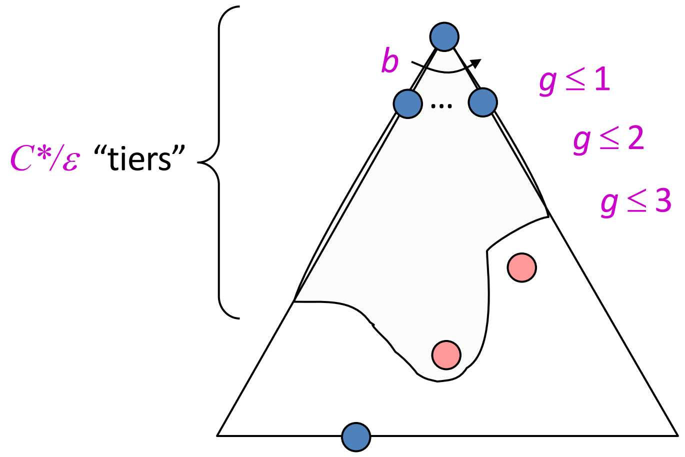
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?



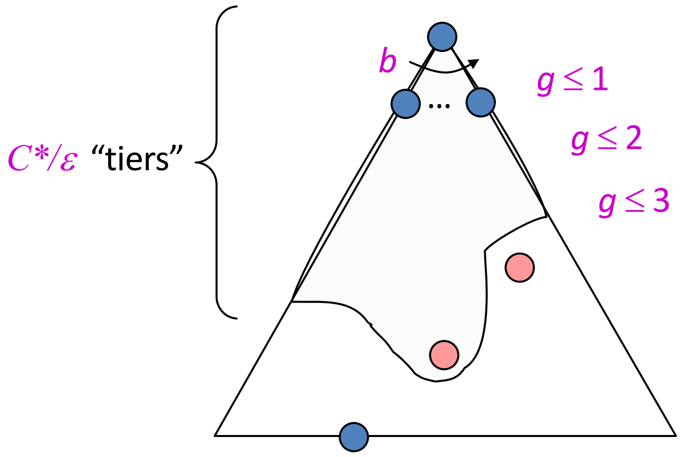
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!



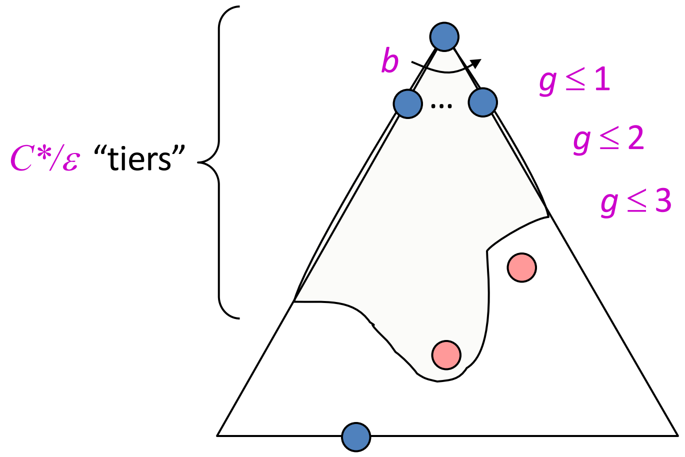
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε



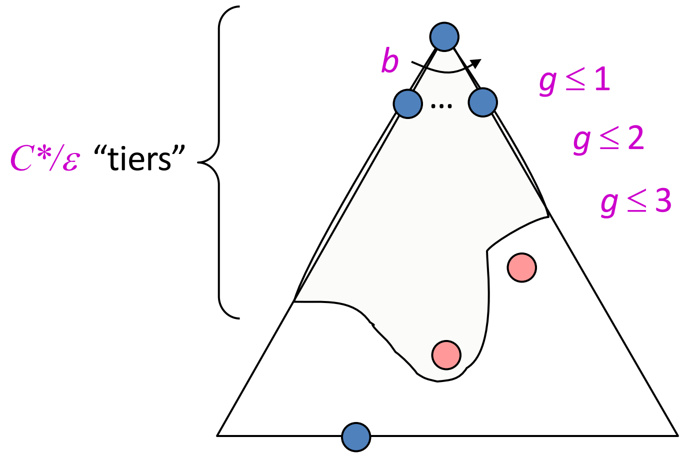
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)



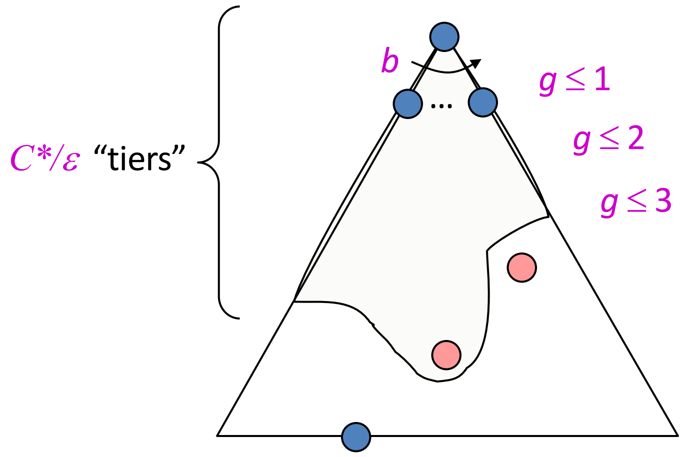
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?



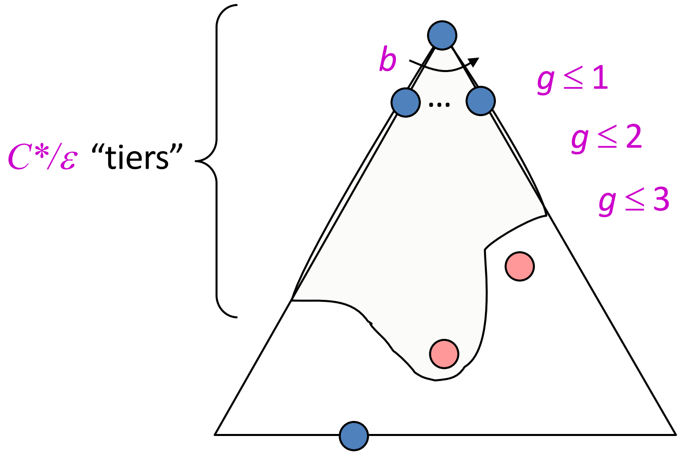
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$



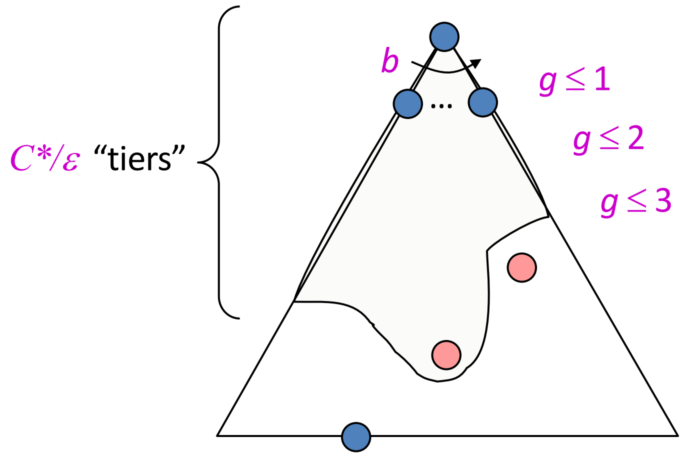
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?



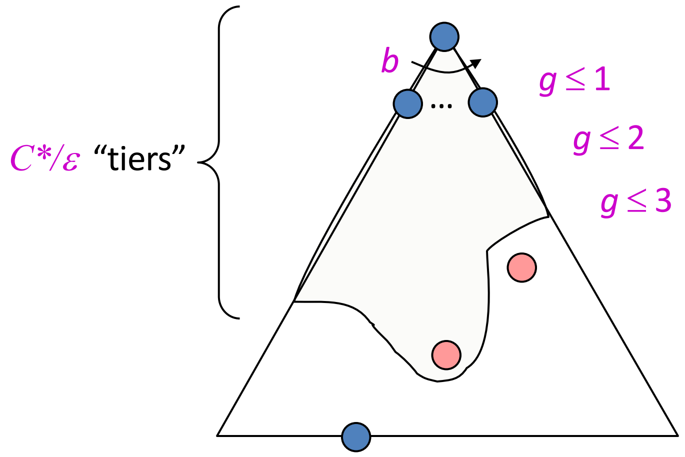
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming C^* is finite and $\varepsilon > 0$, yes!



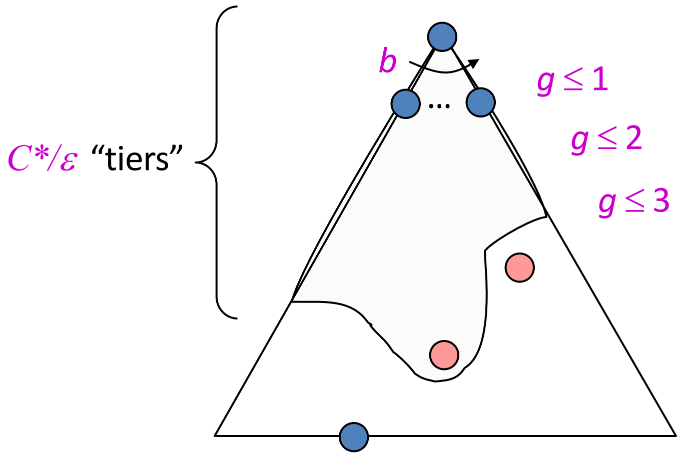
Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming C^* is finite and $\varepsilon > 0$, yes!
- Is it optimal?



Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)
- How much space does the frontier take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$
- Is it complete?
  - Assuming C^* is finite and $\varepsilon > 0$, yes!
- Is it optimal?
  - Yes! (Proof next lecture via A*)



Depth-First Iterative Deepening (DFID)

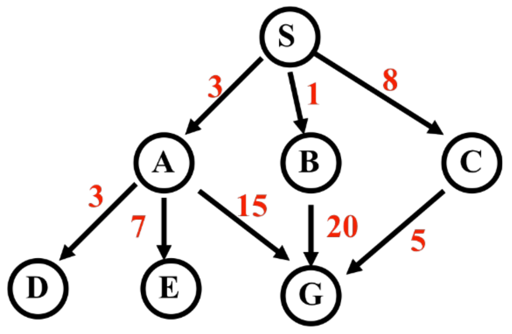
- Do DFS to depth 0, then (if no solution) DFS to depth 1, etc.
- Usually used with a tree search
- **Complete**
- **Optimal/Admissible** if all operators have unit cost, else finds shortest solution (like BFS)
- Time complexity a bit worse than BFS or DFS

  Nodes near top of search tree generated many times, but since almost all nodes are near tree bottom, worst case time complexity still exponential, $O(b^d)$

Depth-First Iterative Deepening (DFID)

- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc.
  - Hence $b^d + 2b^{(d-1)} + ... + db <= b^d / (1 - 1/b)^2 = O(b^d)$.
  - If b=4, worst case is $1.78 * 4^d$, i.e., 78% more nodes searched than exist at depth d (in worst case)
- **Linear space complexity**, O(bd), like DFS
- Has advantages of BFS (completeness) and DFS (i.e., limited space, finds longer paths quickly)
- Preferred for **large state spaces** where **solution depth is unknown**

How they perform



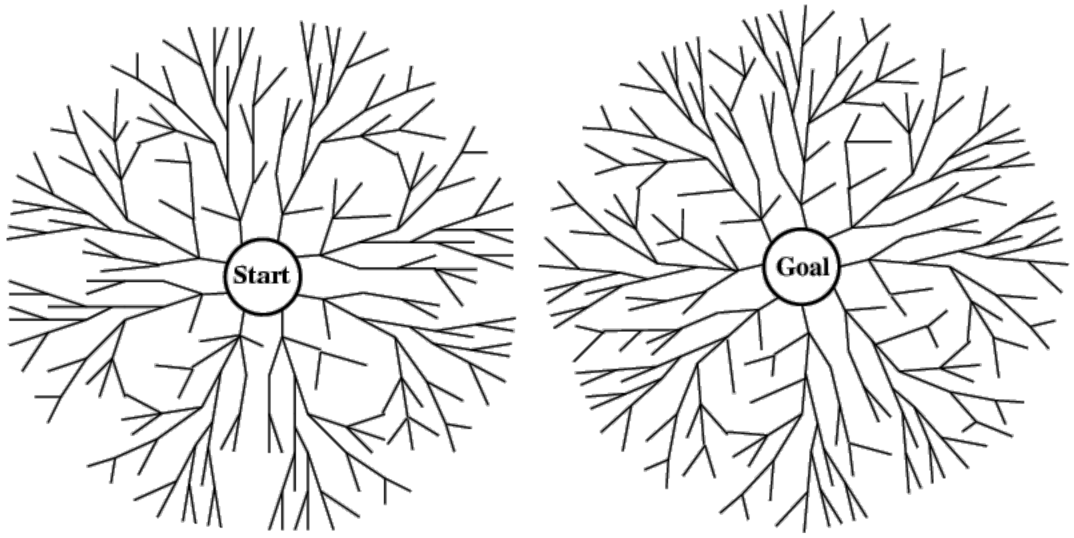
- **Depth-First Search:**
  - 4 Expanded nodes: S A D E G
  - Solution found: S A G (cost 18)
- **Breadth-First Search**:
  - 7 Expanded nodes: S A B C D E G
  - Solution found: S A G (cost 18)
- **Uniform-Cost Search**:
  - 7 Expanded nodes: S A D B C E G
  - Solution found: S C G (cost 13)

  Only uninformed search that worries about costs
- **Iterative-Deepening Search**:
  - 10 nodes expanded: S S A B C S A D E G
  - Solution found: S A G (cost 18)

Searching Backward from Goal

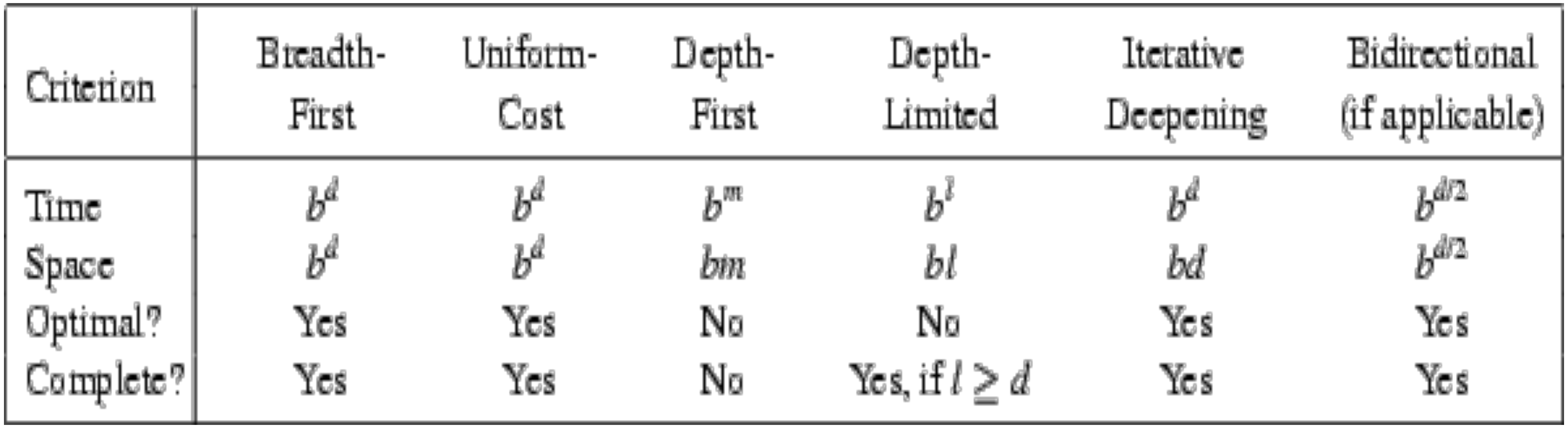
- Usually a successor function is reversible
  - i.e., can generate a node's predecessors in graph
- If we know a single goal (rather than a goal's properties), we could search backward to the initial state
- It might be more efficient
  - Depends on whether the graph fans in or out

Bi-directional search



- Alternate searching from the start state toward the goal and from the goal state toward the start
- Stop when the frontiers intersect
- Works well only when there are unique start & goal states
- Requires ability to generate “predecessor” states
- Can (sometimes) lead to finding a solution more quickly

Comparing Search Strategies



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Summary

- Search in a problem space is at the heart of many AI systems
- Formalizing the search in terms of **states**, **actions**, and **goals** is key
- The simple “uninformed” algorithms we examined can be augmented to heuristics to improve them in various ways
- But for some problems, a simple algorithm is best