Artificial Intelligence

UNINFORMED SEARCH STRATEGIES

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

Outline

- Uninformed search strategies
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limit search
- Iterative deepening search
- Bidirectional search

Uninformed search strategies

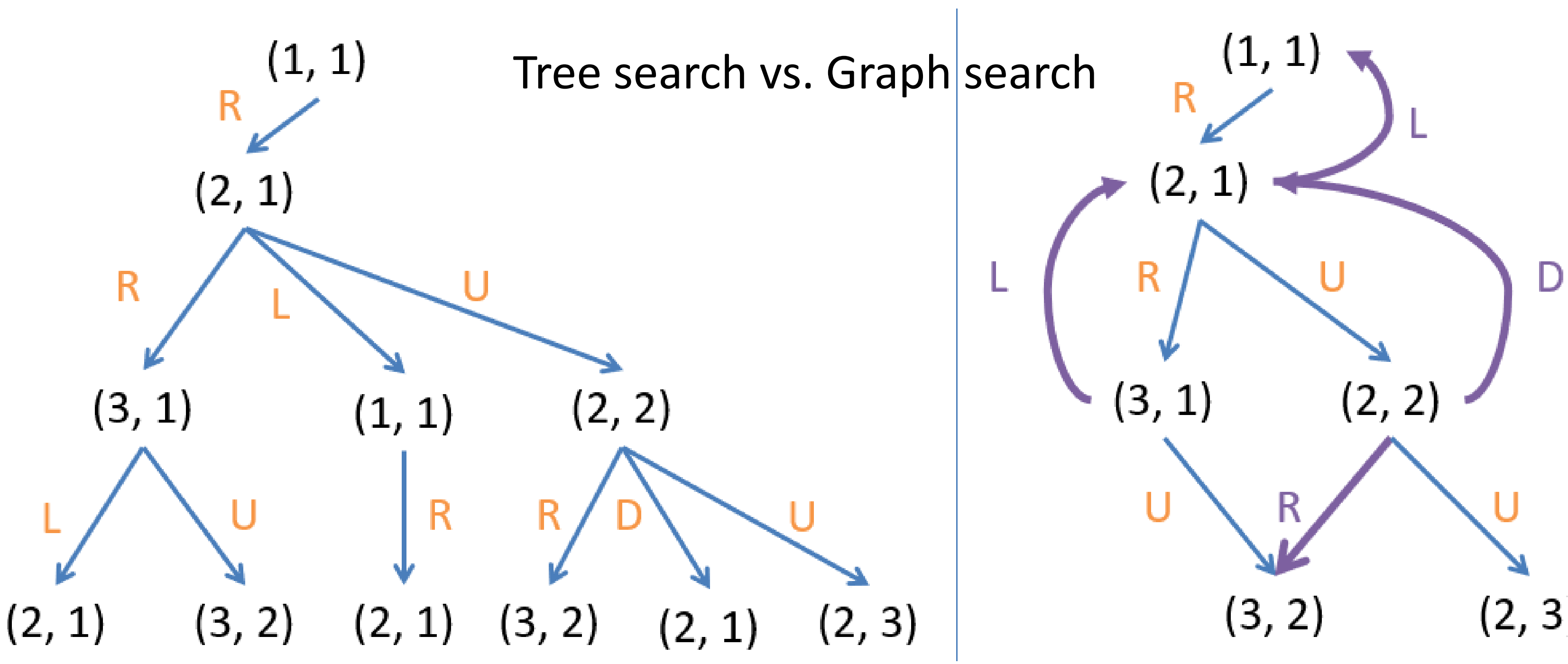
- No additional information about states beyond that provided in the problem definition
  - All they can do is to generate successors and distinguish a goal state from a non-goal state.
- Also called **Blind Search**

Uninformed search strategies

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening search
- Bidirectional search
- *Iterative lengthening search, branch and bound, etc.*

Review: Tree search vs. Graph search

- Tree search can end up repeatedly visiting the same nodes.
  - E.g., Arad-Sibiu-Arad-Sibiu-Arad-…
- *A good search algorithm avoids such paths.*
- *Graph search: frontier, explored set, etc.*



Review: Search strategies

- **Search strategies** are distinguished by the order in which nodes are expanded
- How to evaluate a search strategy?
  - Completeness
  - Time complexity
  - Space complexity
  - Optimality

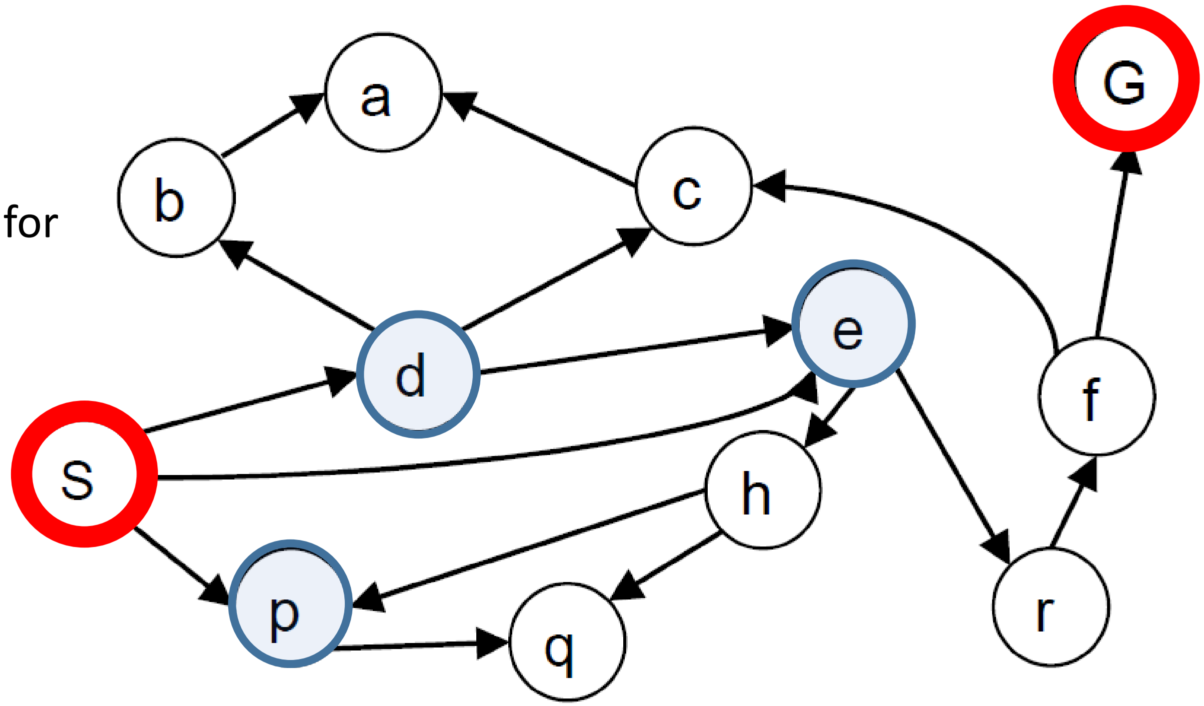
Measured by factors $b, d,$ *and* m

- b: maximum branching factor of the search tree
- d: depth of the least-cost solution
- m: maximum depth of the state space (may be ∞)

Breadth-first search

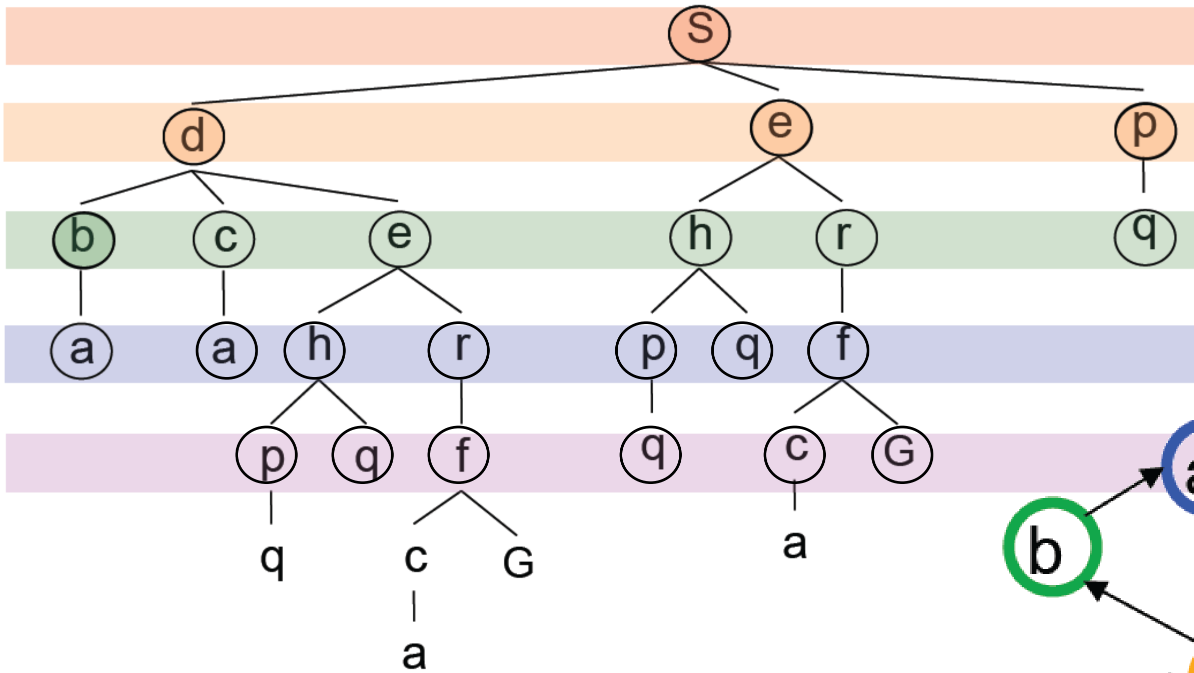
Breadth-first search (BFS)

Example state space graph for a tiny search problem



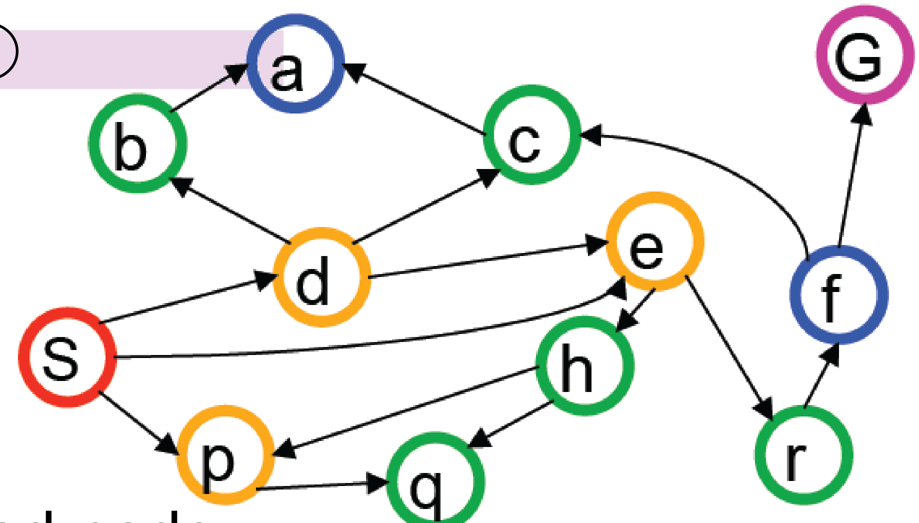
- The **root node is expanded first**, then all the successors of the root, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search (BFS)



Expansion order:

(*S, d, e, p, b, c, h, r, q, a, f, G*)



- Expand shallowest unexpanded node
- Implementation: **frontier** is a FIFO queue

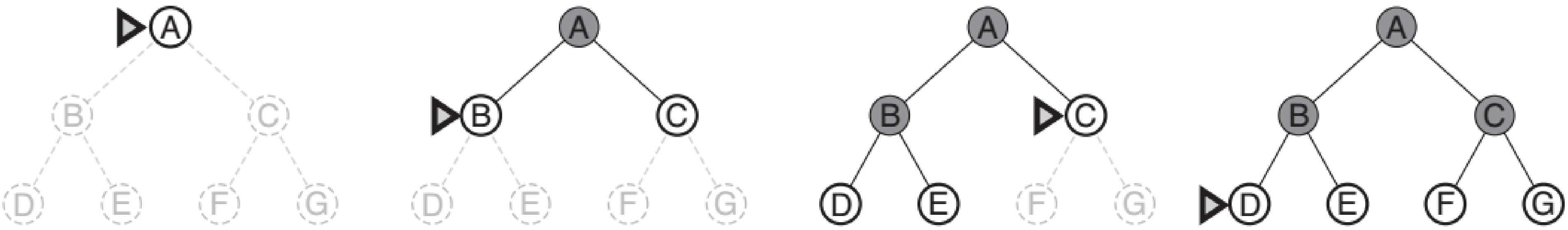
Breadth-first search on a graph

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?( frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored and not in frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Breadth-first search (BFS)

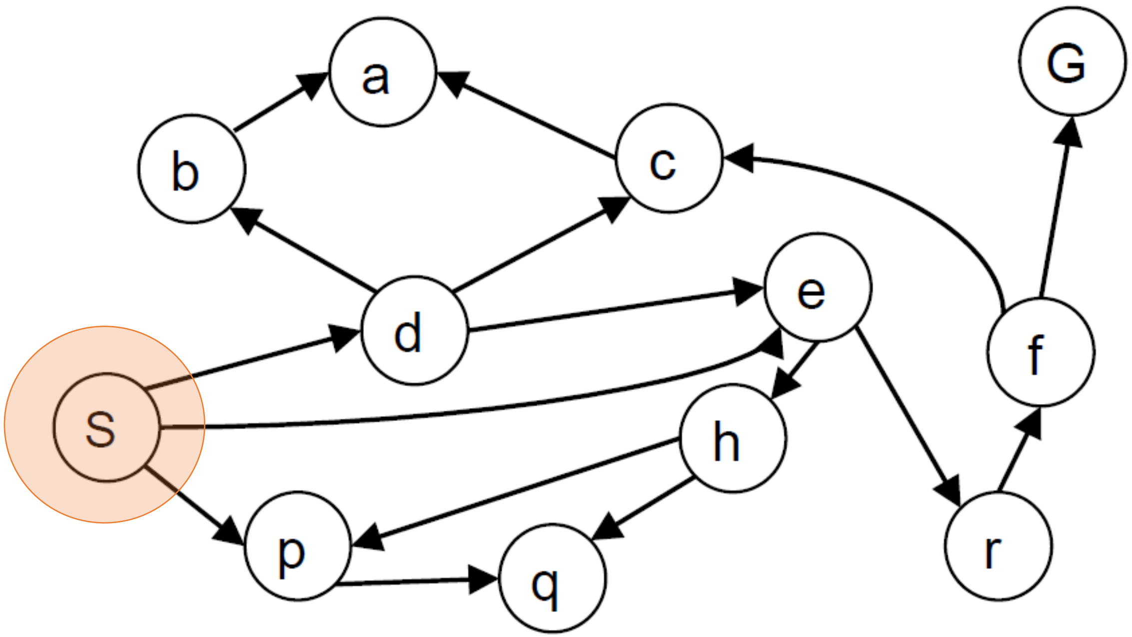
- An instance of the general graph search algorithm
- The shallowest **unexpanded** node is chosen for expansion
- The goal test is applied to each node when it is **generated** rather than when it is selected for expansion
- Discard any new path to a state already in the frontier or in the explored set

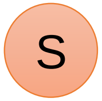
Breadth-first search on a graph



Breadth-first search on a simple binary tree.
At each stage, the node to be expanded next is indicated by a marker

Breadth-first search: An example

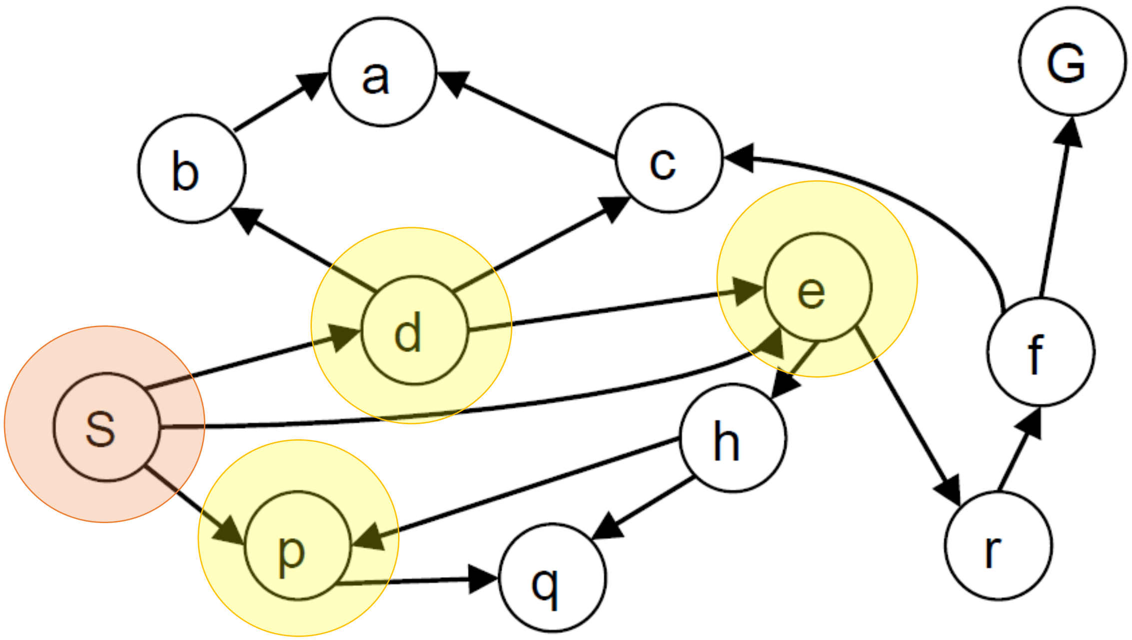




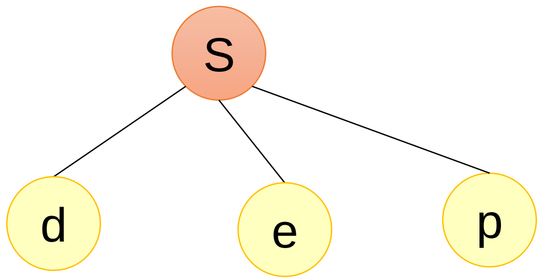
$d = 0$

Search Tree

Breadth-first search: An example

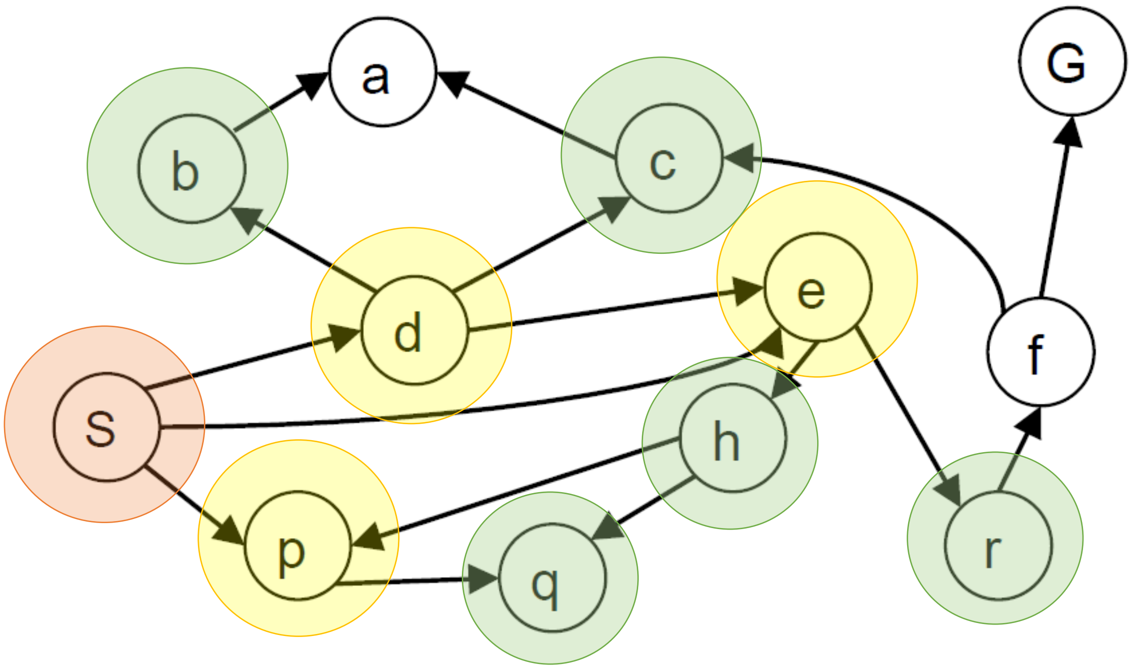


$d = 1$

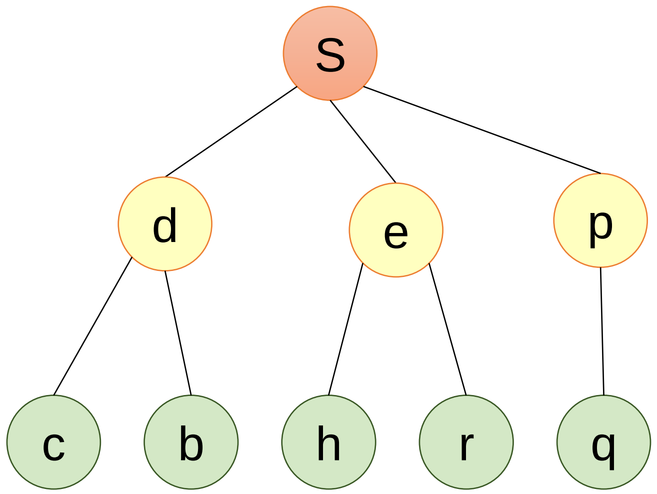


Search Tree

Breadth-first search: An example

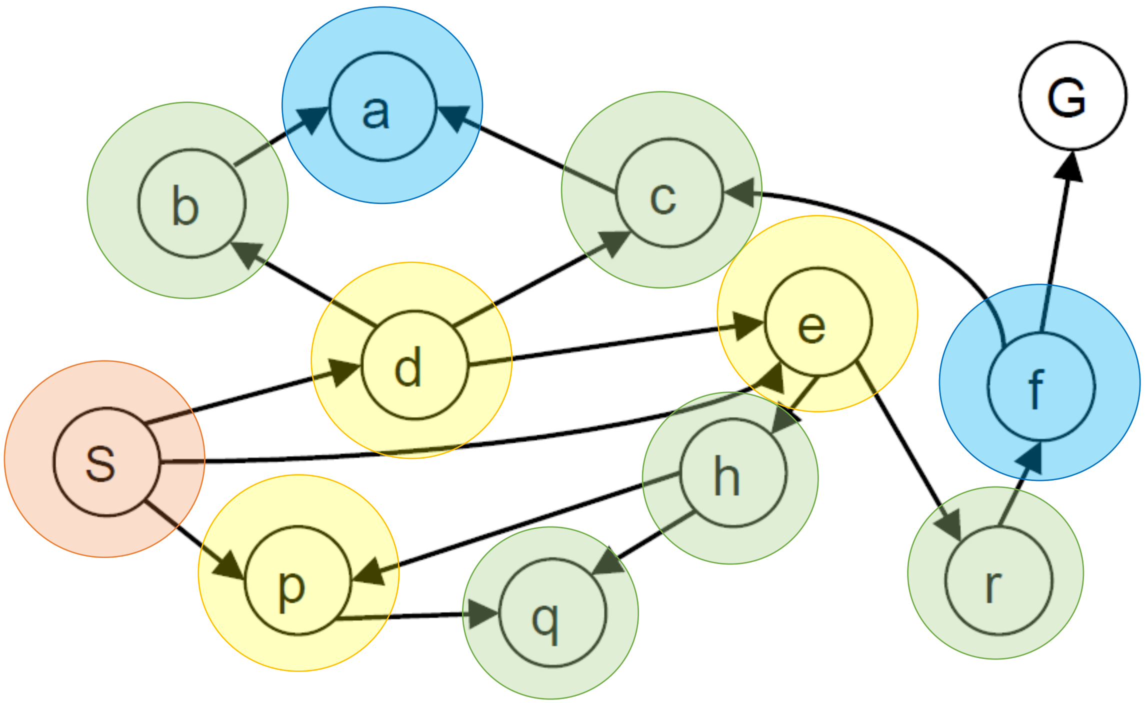


$d = 2$

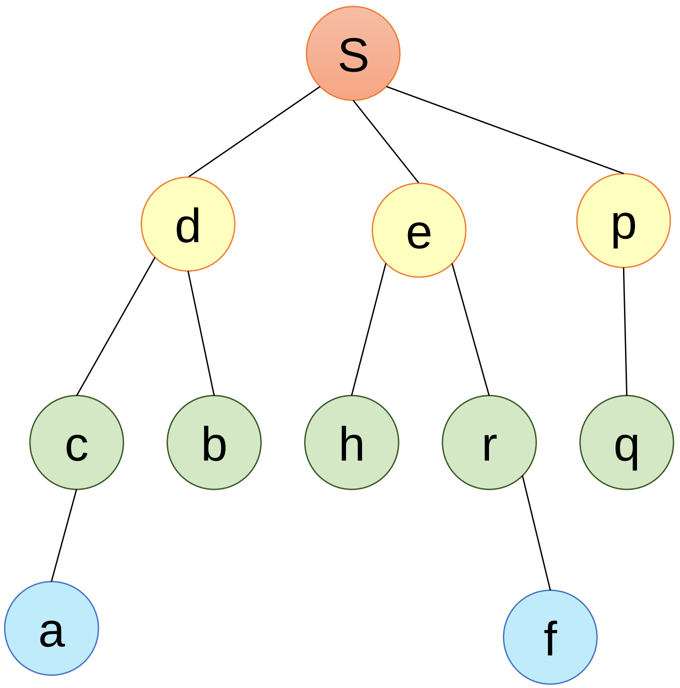


Search Tree

Breadth-first search: An example

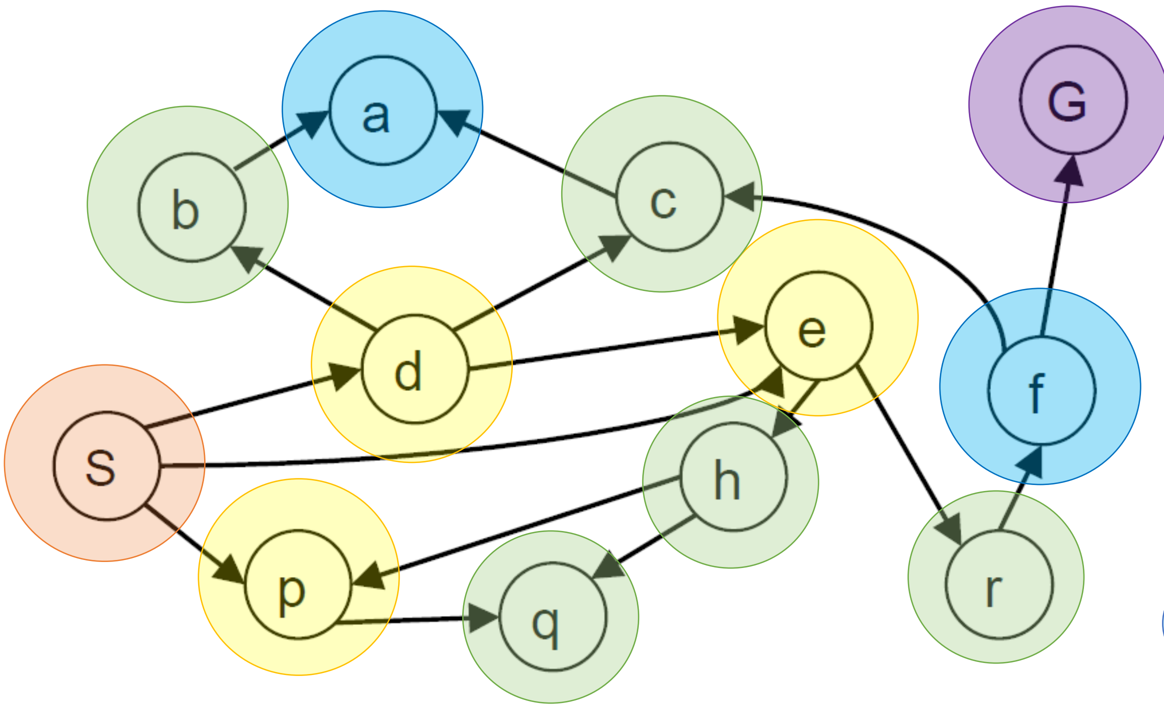


$d = 3$

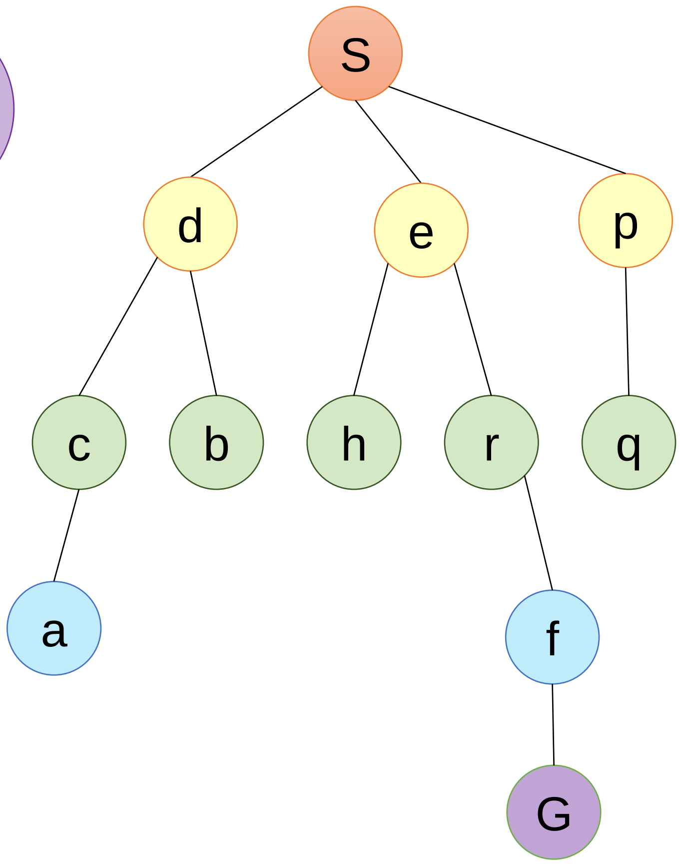


Search Tree

Breadth-first search: An example

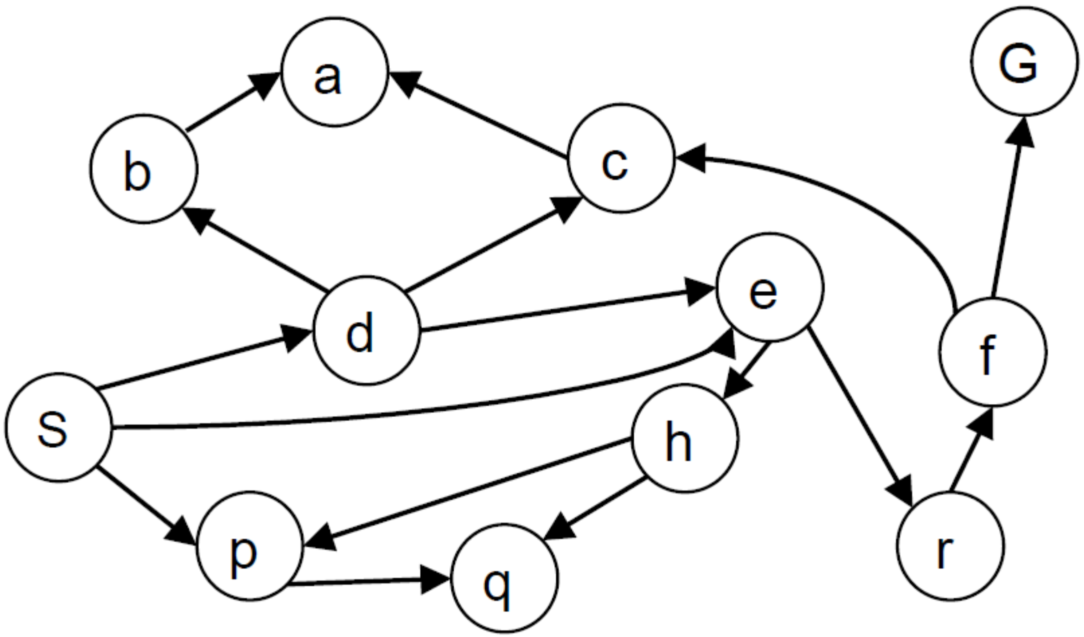


$d = 4$

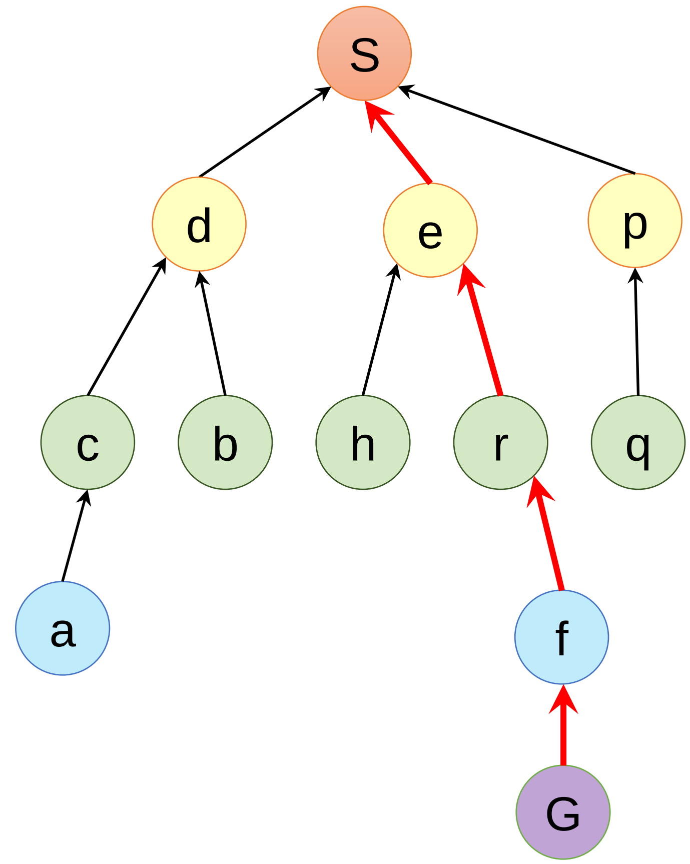


Search Tree

Breadth-first search: An example



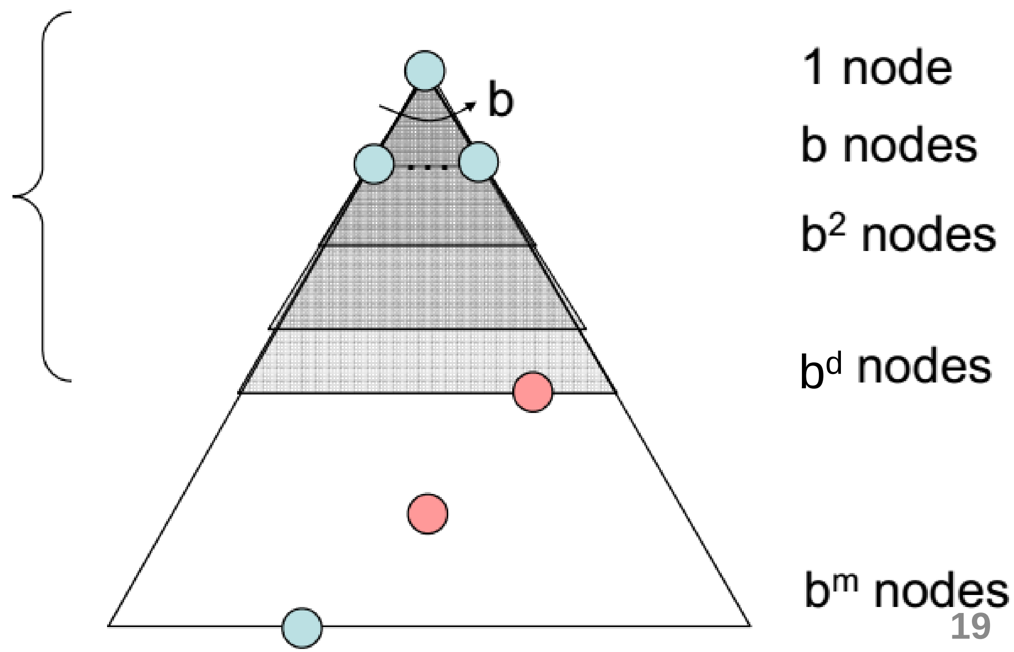
Search path: $S \rightarrow e \rightarrow r \rightarrow f \rightarrow G$



Search Tree

An evaluation of BFS

- What nodes does BFS expand?
  - Process all nodes above the shallowest solution
  - Let the shallowest solution's depth be d. Search takes time $O(b^d)$.
- How much space does the frontier take?
  - Roughly the last tier, so $O(b^d)$.
- Is it complete?
  - YES
- Is it optimal?
  - Only if costs are all uniform



The complexity of BFS

Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

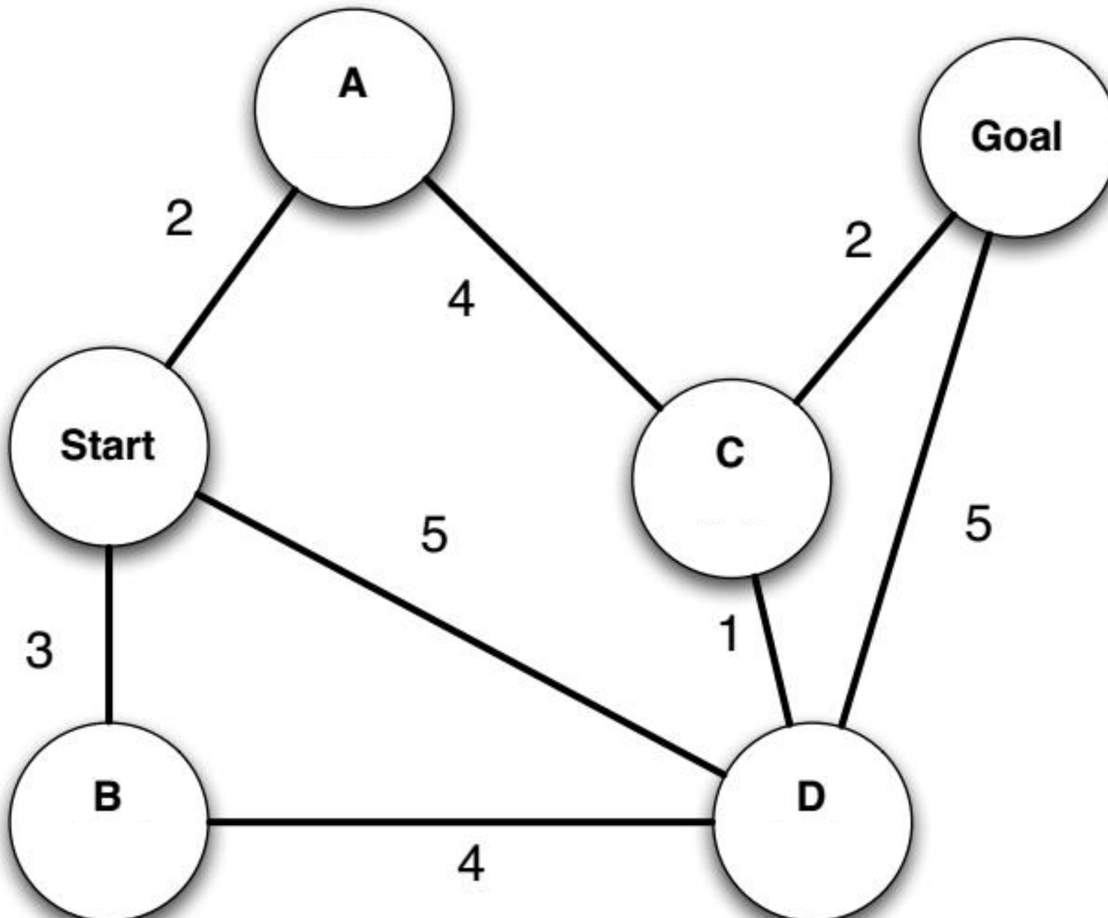
Time and memory requirements for BFS. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

The memory requirements are a bigger problem for BFS than the execution time.

In general, exponential-complexity search problems cannot be solved by uninformed methods for any but the smallest instance

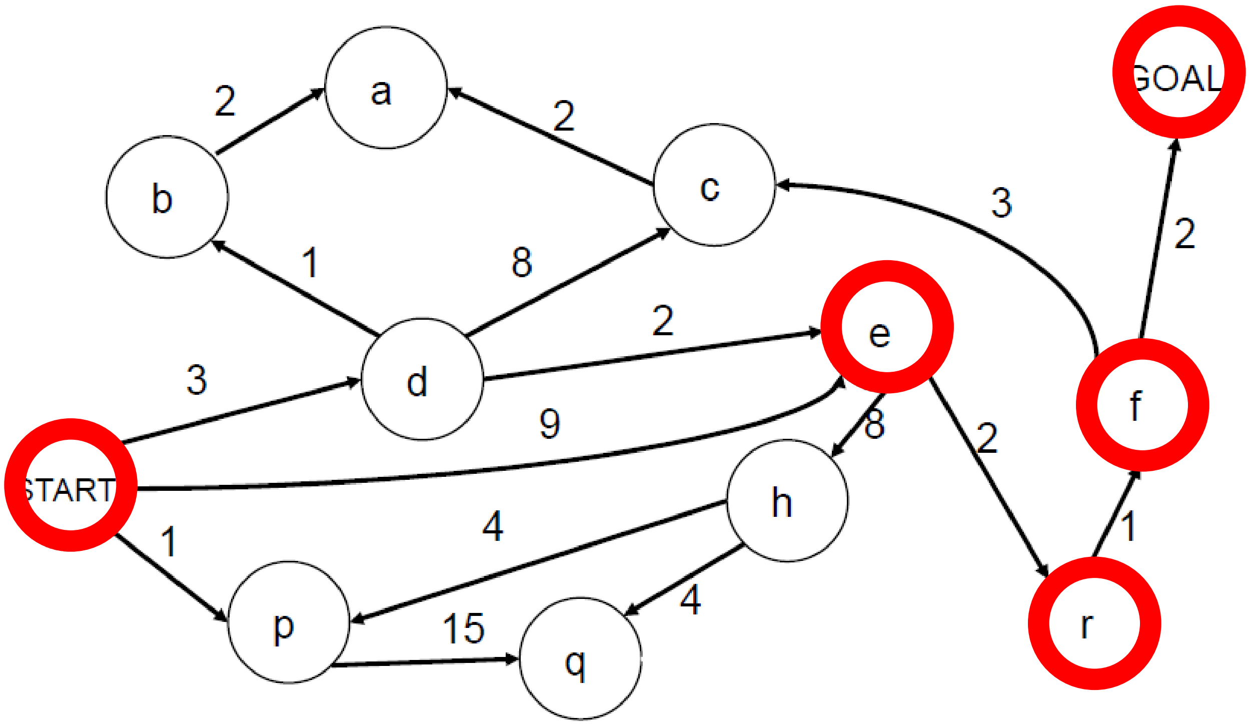
Quiz 01: Breadth-first search

- Work out the order in which states are expanded, as well as the path returned by graph search. Assume ties resolve in such a way that states with earlier alphabetical order are expanded first.



Uniform-cost search

Search with varying step costs



- BFS finds the path with the fewest steps but does not always find the cheapest path.
- *An algorithm that is optimal with any step-cost function?*

Uniform-cost search (UCS)

- UCS expands the node n with the **lowest** path cost $g(n)$
- Implementation: frontier is a priority queue ordered by g

  $\rightarrow$ Equivalent to breadth-first search if step costs all equal

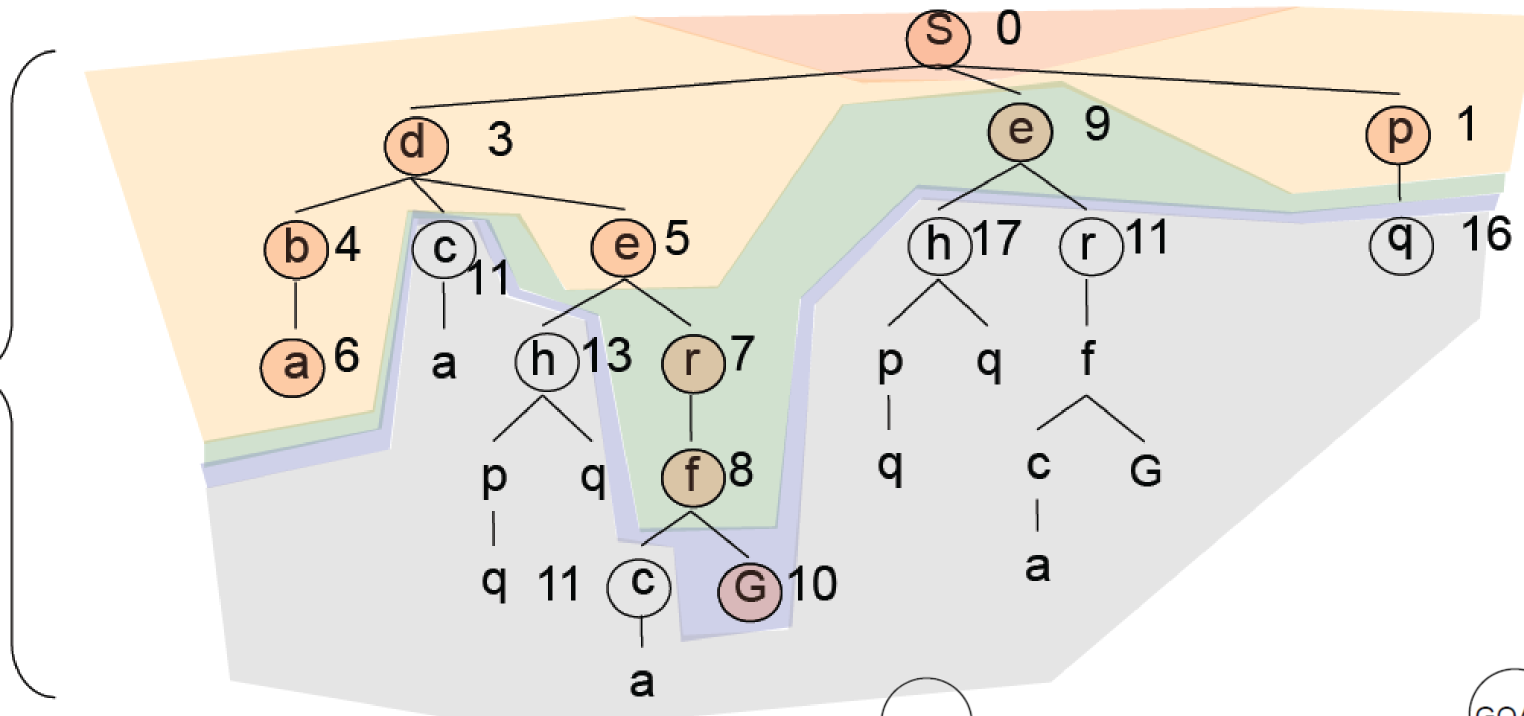
  $\rightarrow$ Equivalent to Dijkstra's algorithm in general
- The goal test is applied to a node when it is selected for expansion
- A test is added in case a better path is found to a node currently on the frontier.

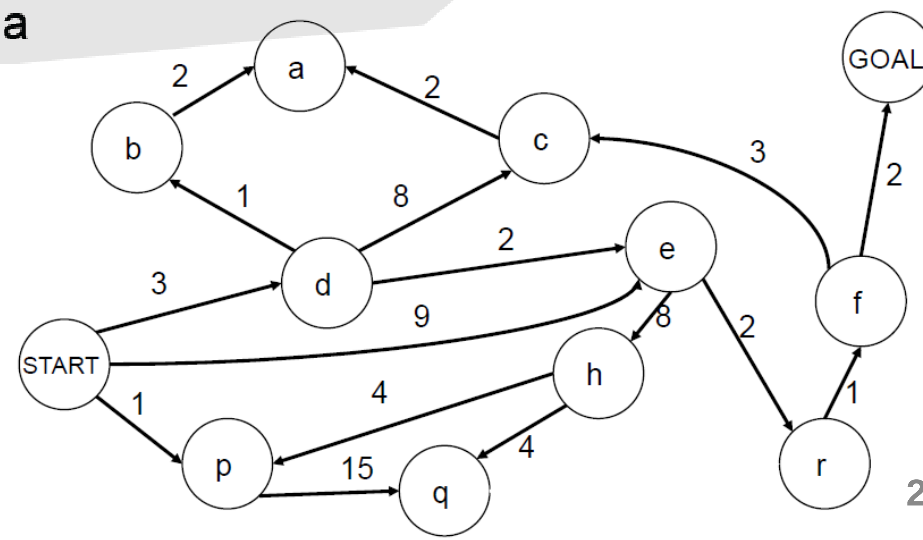
Uniform-cost search (UCS)

function UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with *node* as the element
  explored ← an empty set
  loop do
    if EMPTY?(*frontier*) **then return** failure
    node ← POP(*frontier*) /* chooses the lowest-cost node in *frontier* */
    if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
    add *node*.STATE to *explored*
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← CHILD-NODE(*problem, node, action*)
      if *child*.STATE is not in *explored* and not in *frontier* **then**
        frontier ← INSERT(*child, frontier*)
      else if *child*.STATE is in *frontier* with higher PATH-COST **then**
        replace that *frontier* node with *child*

Uniform-cost search

Cost contours

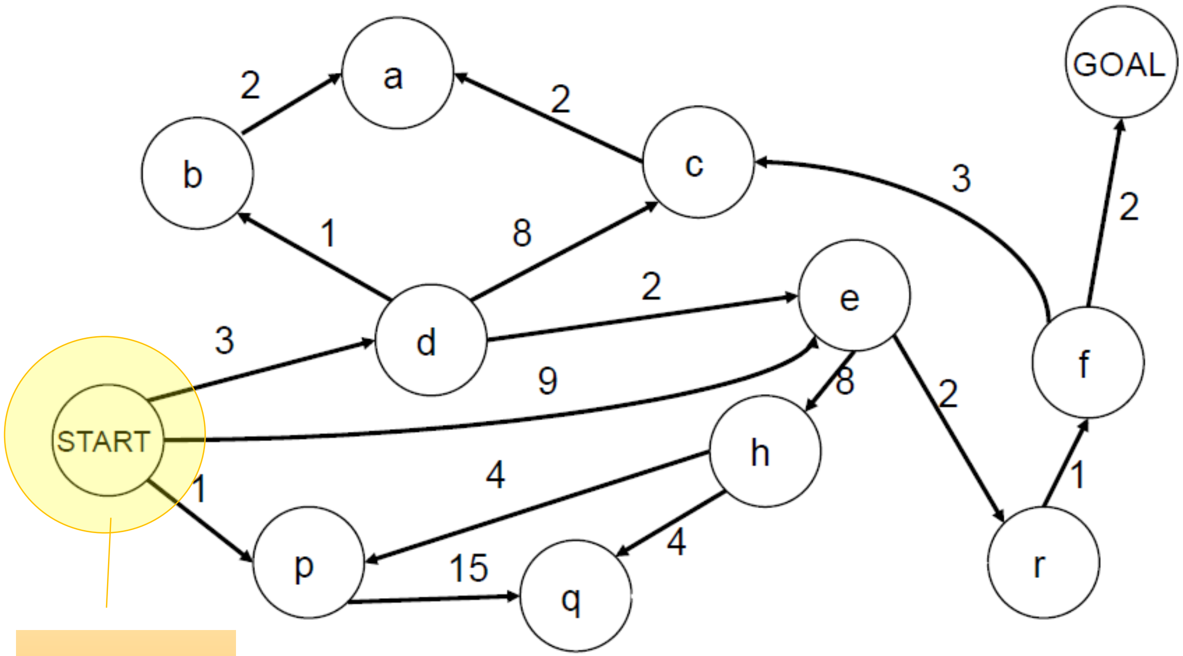




Expansion order

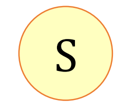
S,p,d,b,e,a,r,f,e,G

Uniform-cost search: An example



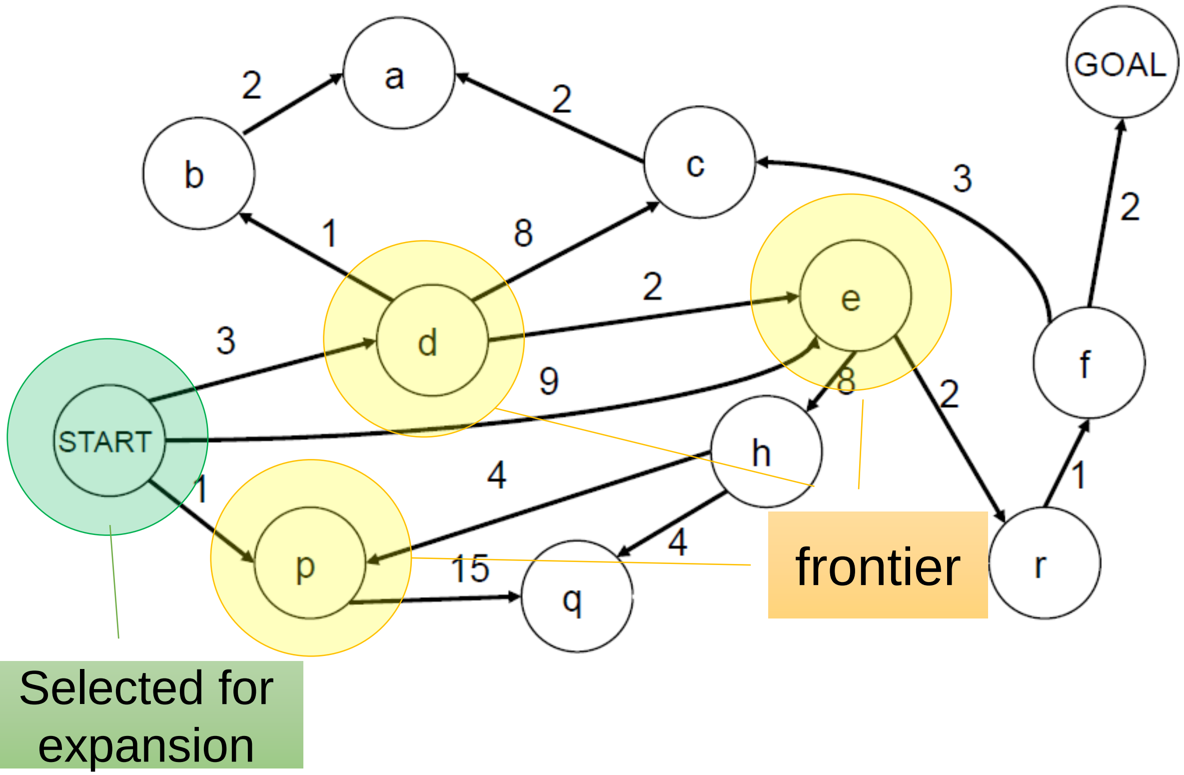
frontier

PQ = { (S:0) }

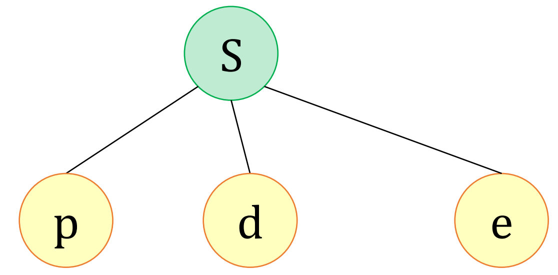


Search Tree

Uniform-cost search: An example

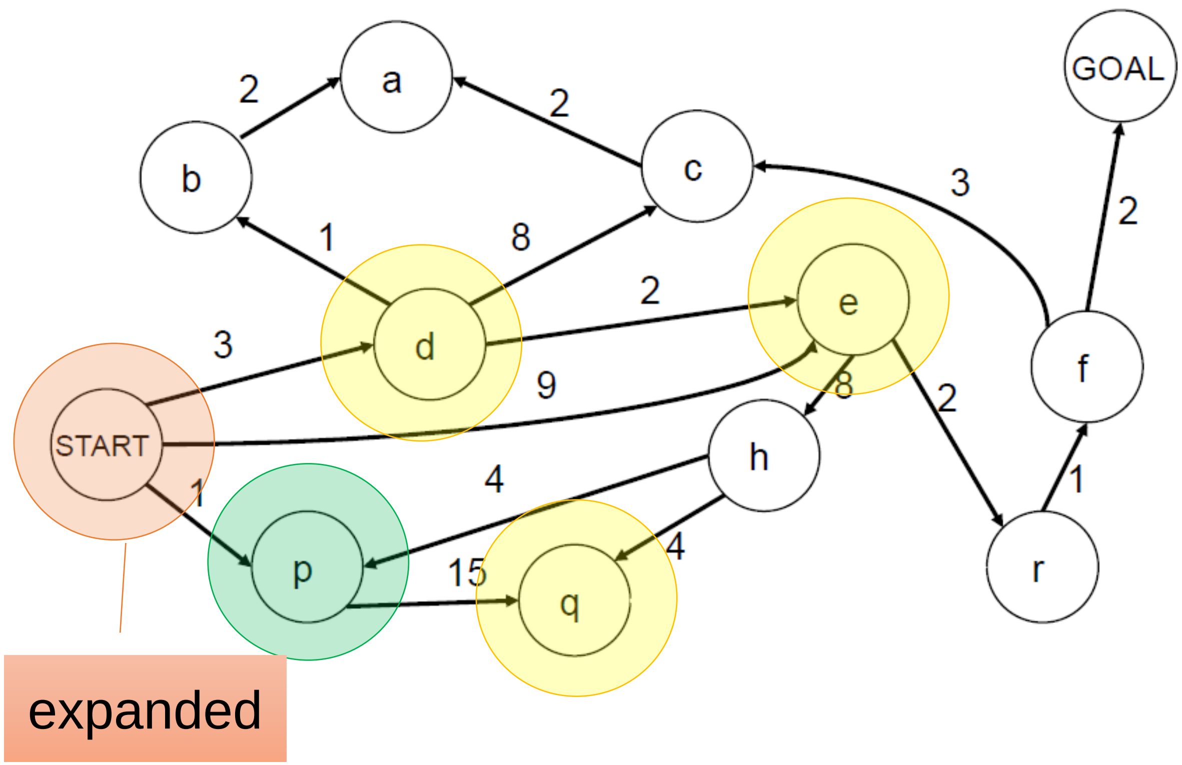


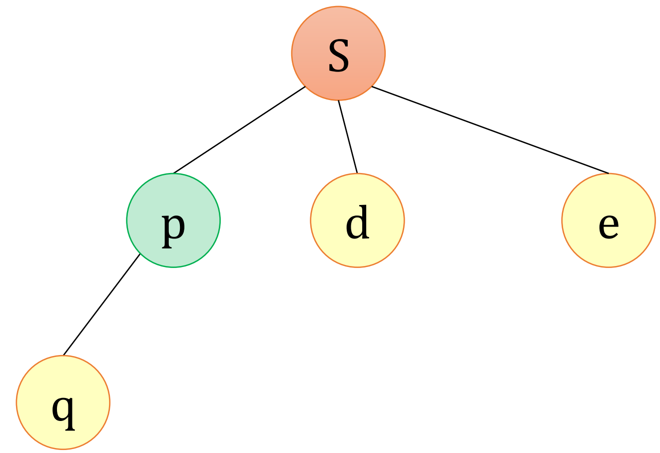
PQ = { (p:1), (d:3), (e:9) }



Search Tree

Uniform-cost search: An example

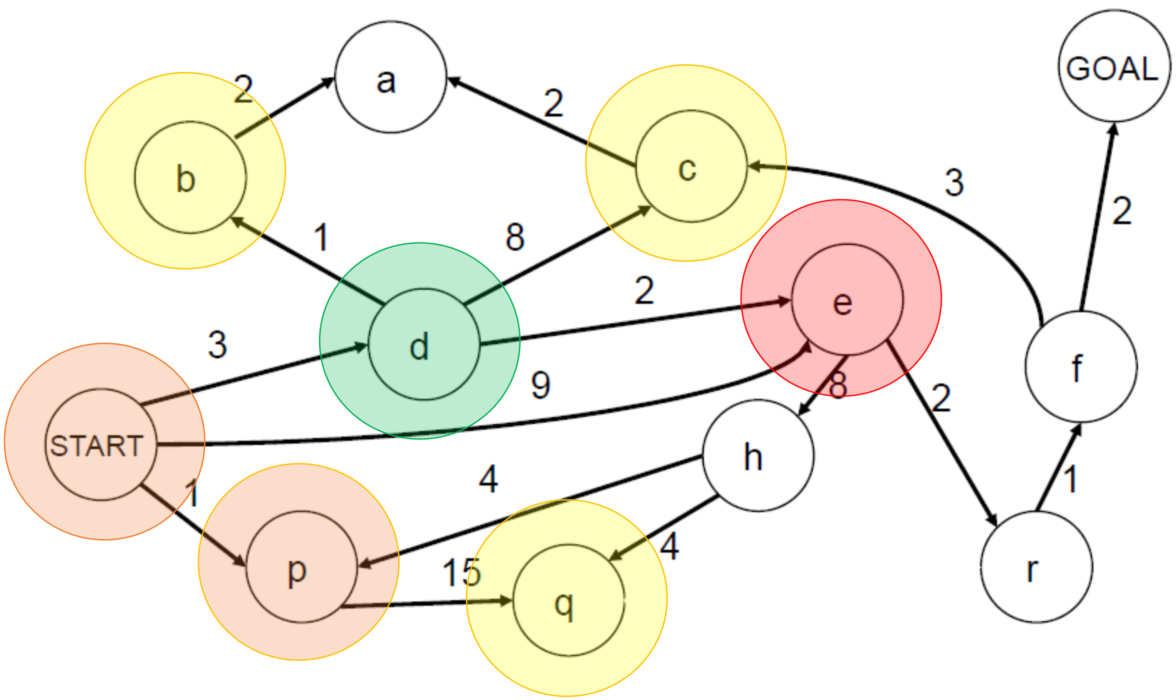


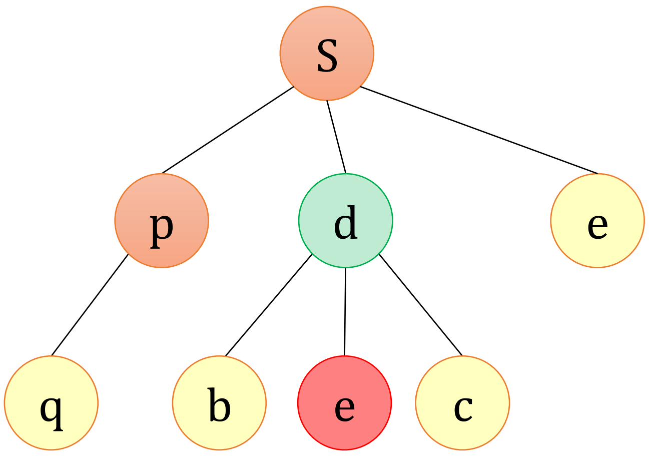


PQ = { (d:3), (e:9), (q:16) }

Search Tree

Uniform-cost search: An example



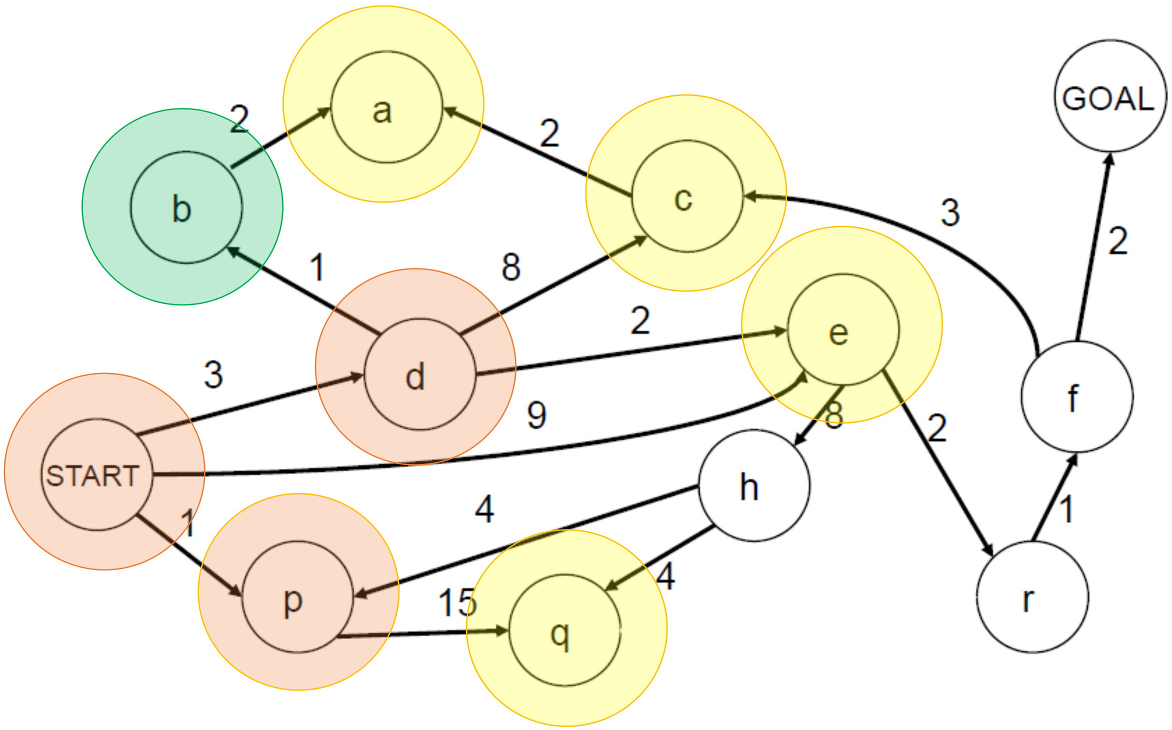


PQ = { (b:4), (e:5), (c:11), (q:16) }

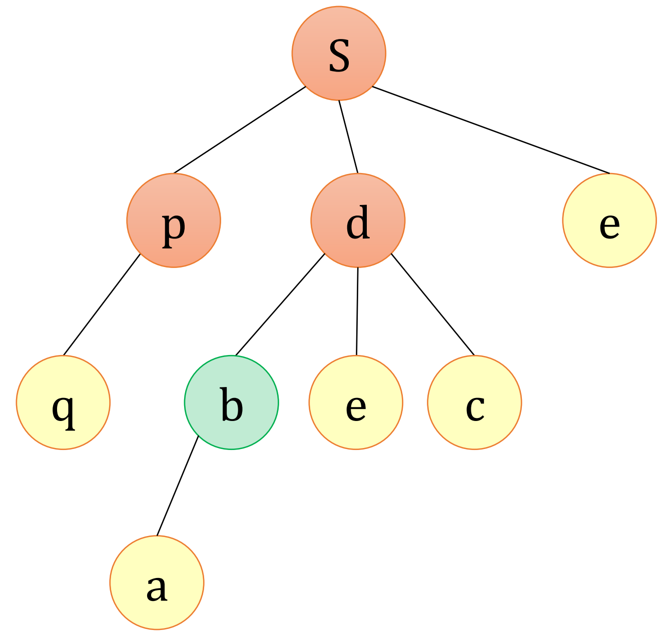
Update path cost of *e*

Search Tree

Uniform-cost search: An example

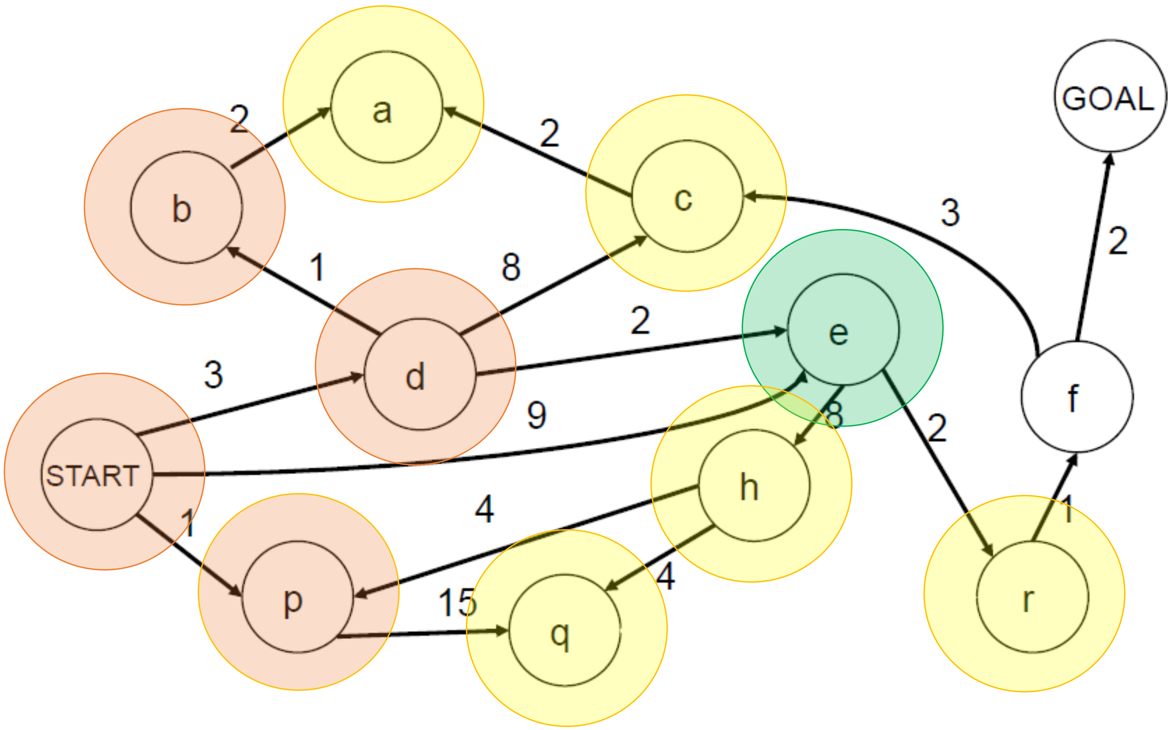


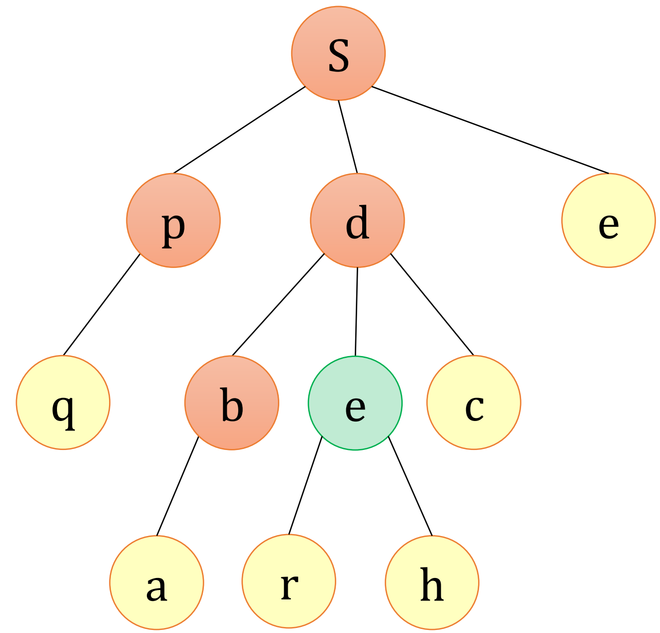
PQ = { (e:5), (a:6), (c:11), (q:16) }



Search Tree

Uniform-cost search: An example

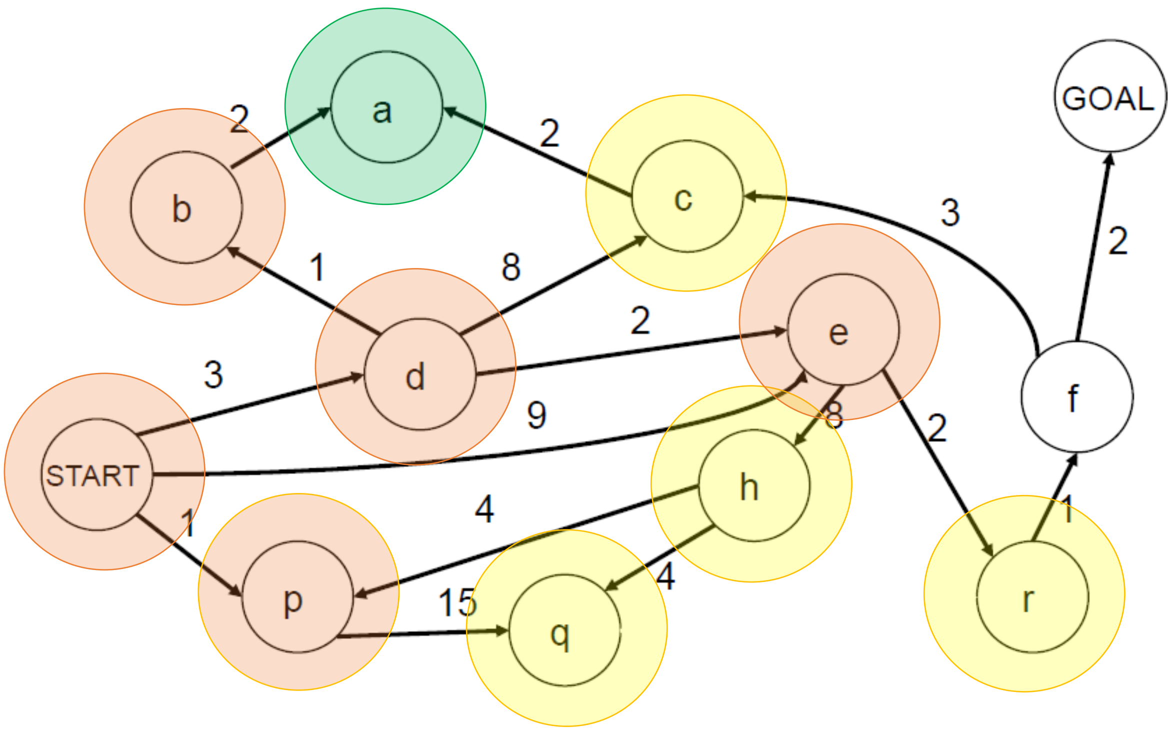


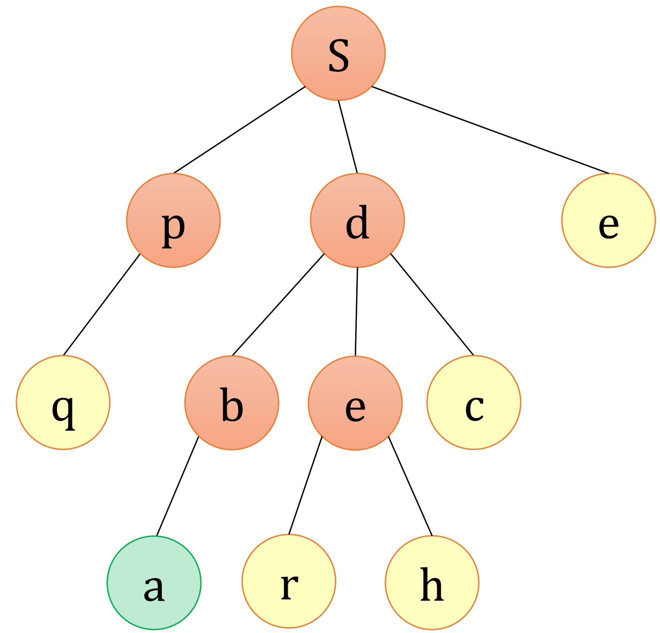


PQ = { (a:6), (r:7), (c:11), (h:13), (q:16) }

Search Tree

Uniform-cost search: An example

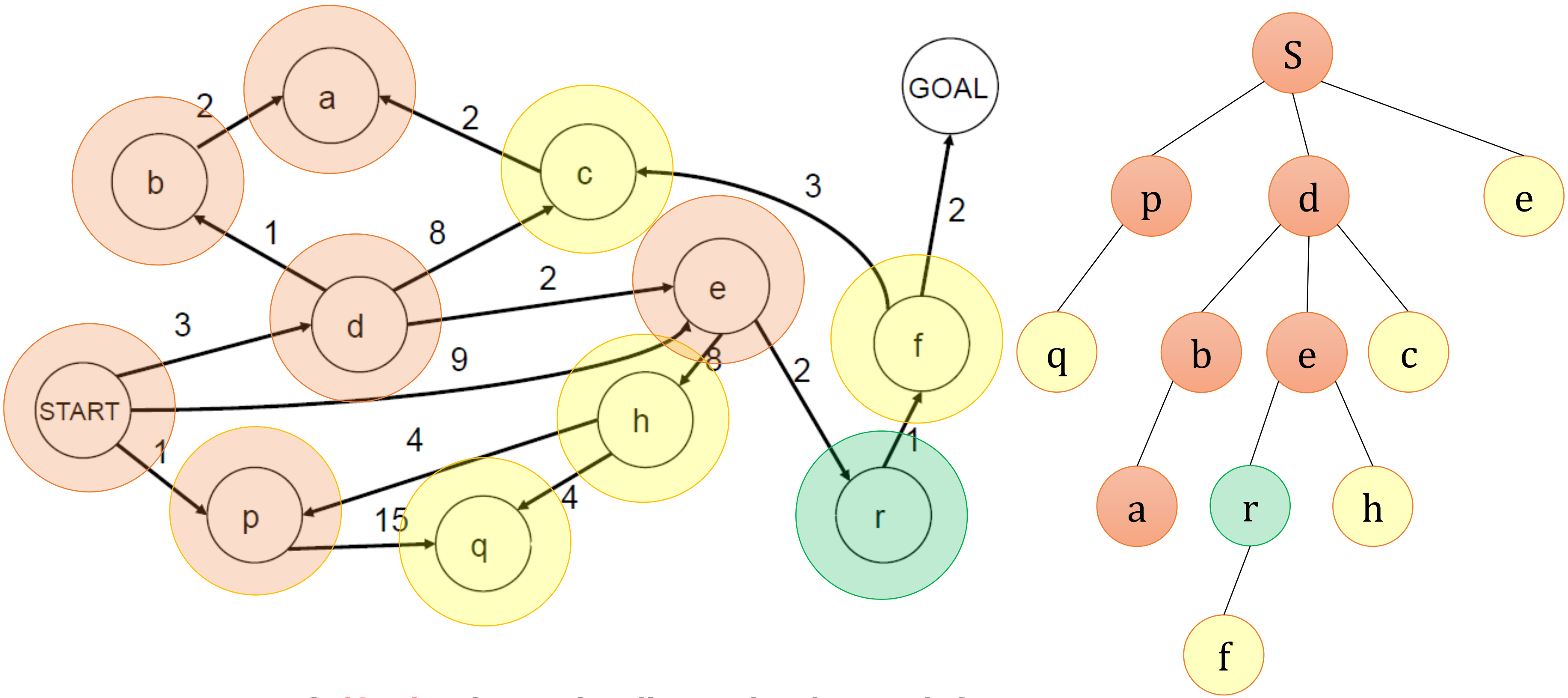




PQ = { (r:7), (c:11), (h:13), (q:16) }

Search Tree

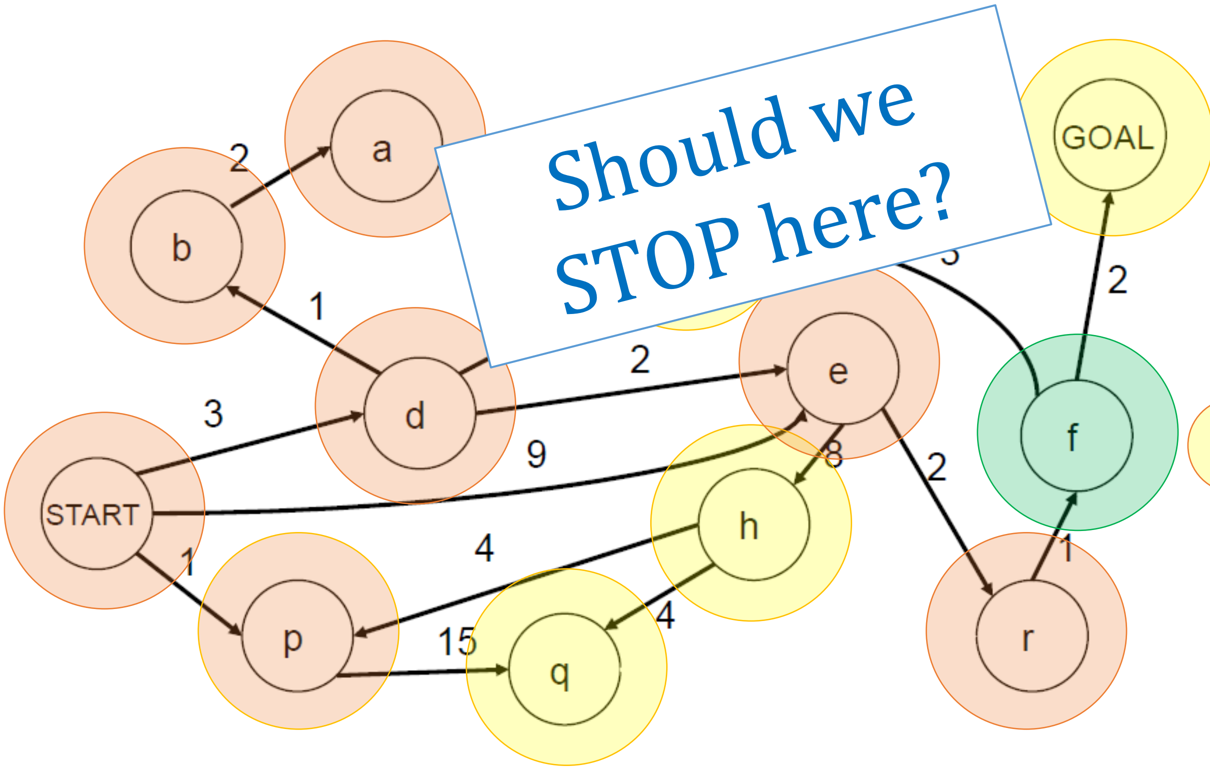
Uniform-cost search: An example



PQ = { (f:8), (c:11), (h:13), (q:16) }

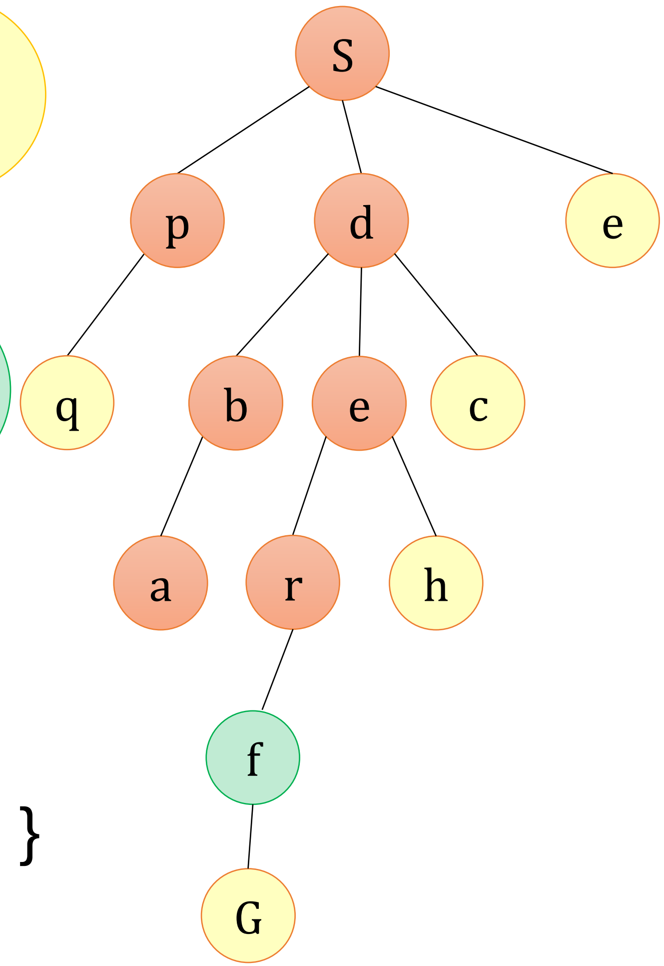
Search Tree

Uniform-cost search: An example



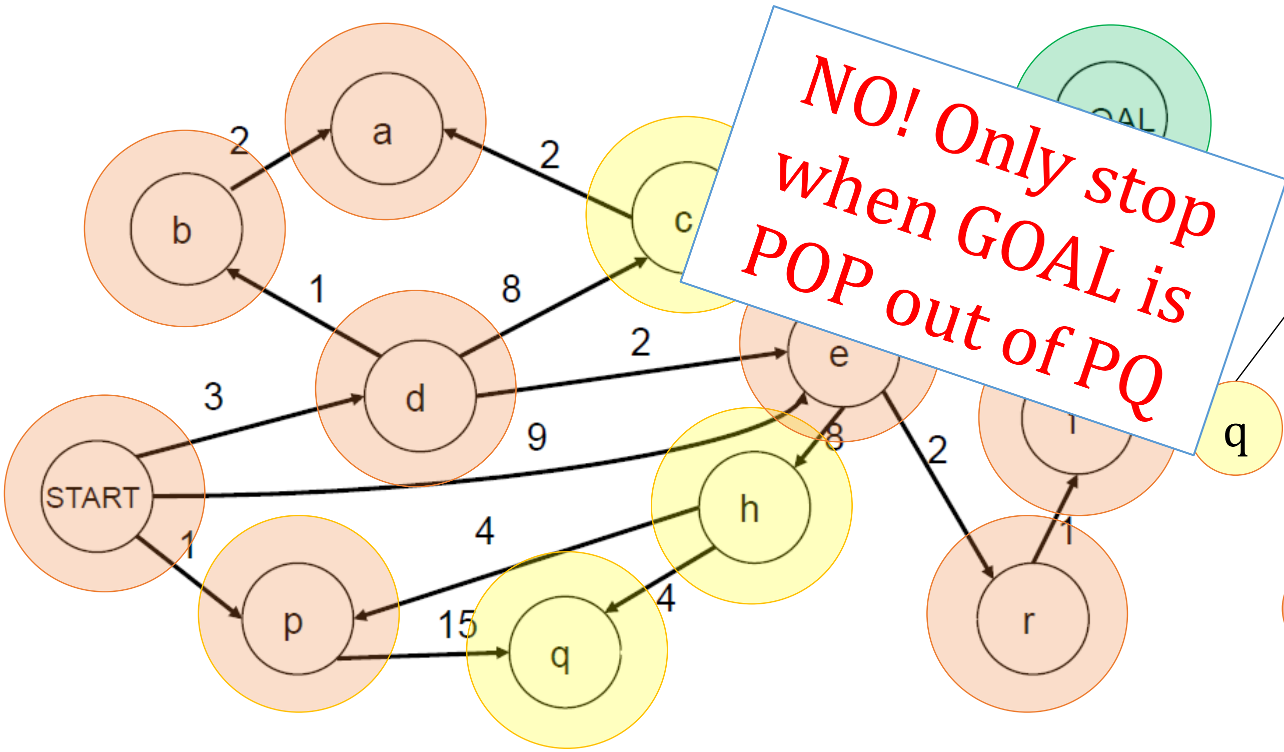
PQ = { (G:10), (c:11), (h:13), (q:16) }

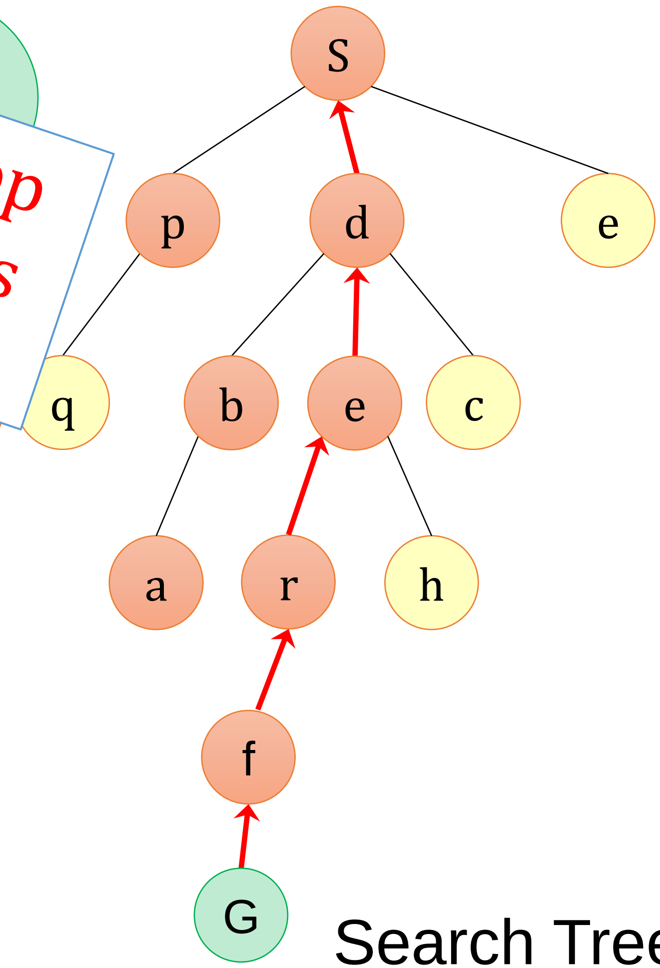
Not update path cost of *c*



Search Tree

Uniform-cost search: An example





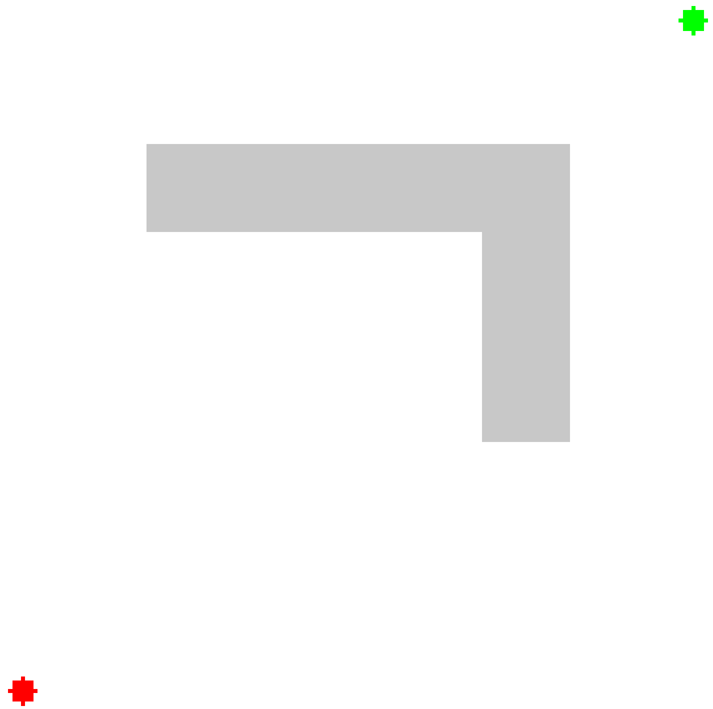
PQ = { (c:11), (h:13), (q:16) }

Goal is taken out of PQ → STOP

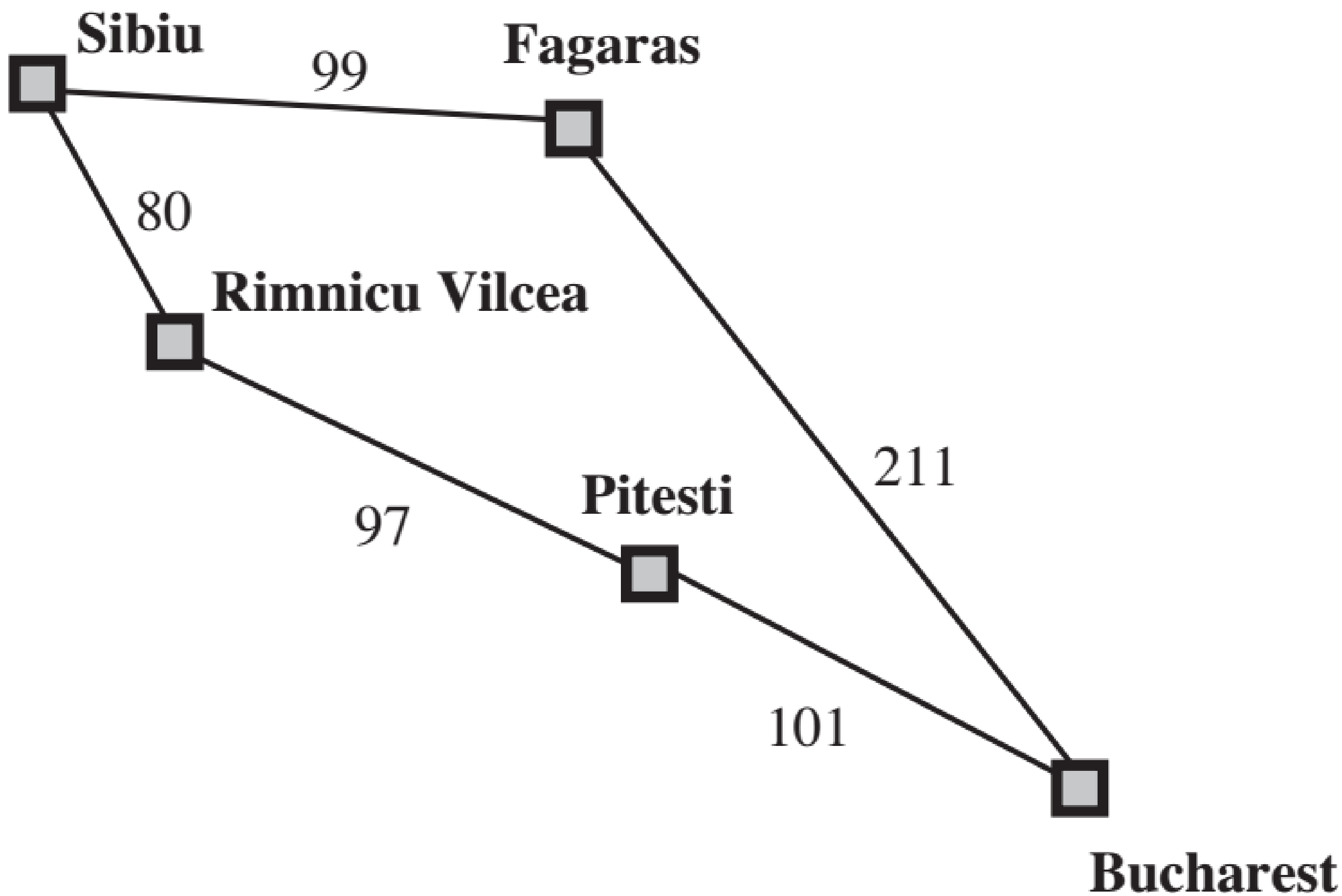
Search path: S → d → e → r → f → G, cost = 10

Search Tree

Uniform-cost search: An example

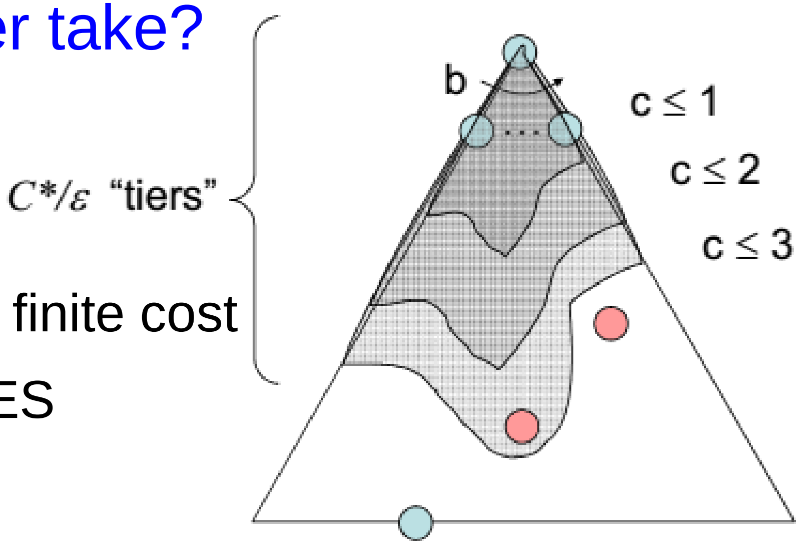


Uniform-cost search: Suboptimal path



An evaluation of UCS

- What nodes does UCS expand?
  - Process all nodes with cost less than cheapest solution!
  - Let C^* be the cost of the optimal solution and assume that every action costs at least ϵ.
  - Take time $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ (exponential in effective depth)
- How much space does the frontier take?
  - Roughly the last tier, so $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
- Is it complete?
  - Assume that the best solution has a finite cost and minimum arc cost is positive, YES
- Is it optimal?
  - YES



An evaluation of UCS

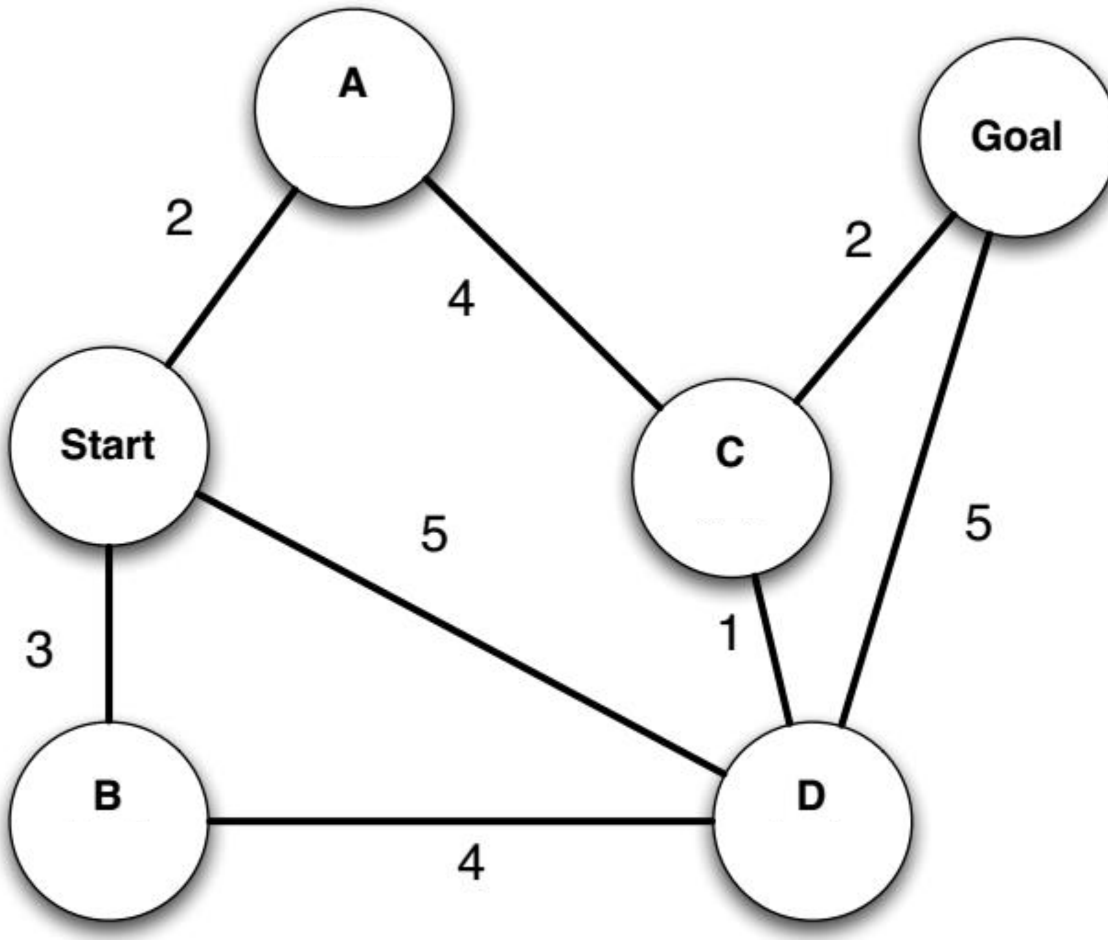
- Graph separation property: every path from the initial state to an unexplored state must pass through a state on the frontier.
  - Proved inductively
- Optimality of UCS: proof by contradiction
  - Suppose UCS terminates at goal state n with path cost $g(n) = C$ but there exists another goal state n' with $g(n') < C$
  - There must exist a node n'' on the frontier that is on the optimal path to n'.
  - Since $g(n'') < g(n') < g(n)$, n'' should have been expanded first!
- UCS expands nodes in order of their optimal path cost.

An evaluation of UCS

- The complexity of $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ can be greater than $O(b^d)$.
  - UCS can explore large trees of small steps before exploring paths involving large and perhaps useful steps.
- When all step costs are equal, $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ is just $O(b^{d+1})$.
  - UCS does strictly more work by unnecessarily expanding nodes at depth d, while BFS stops as soon as it generates a goal.

Quiz 02: Uniform-cost search

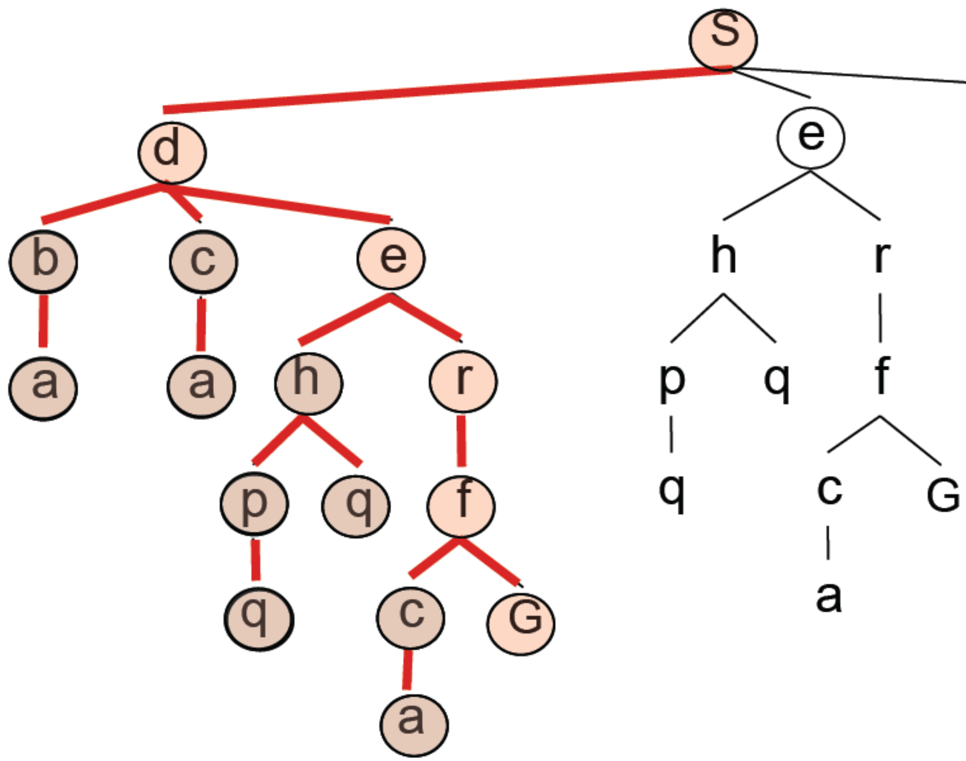
- Work out the order in which states are expanded, as well as the path returned by graph search. Assume ties resolve in such a way that states with earlier alphabetical order are expanded first.

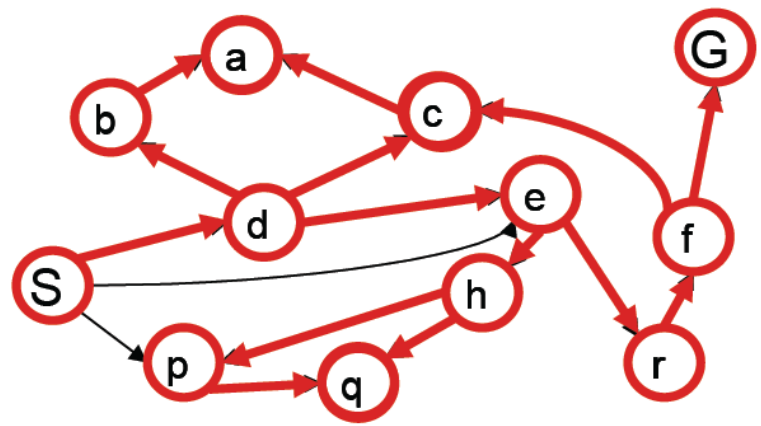


Depth-first search

Depth-first search (DFS)

- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO Stack

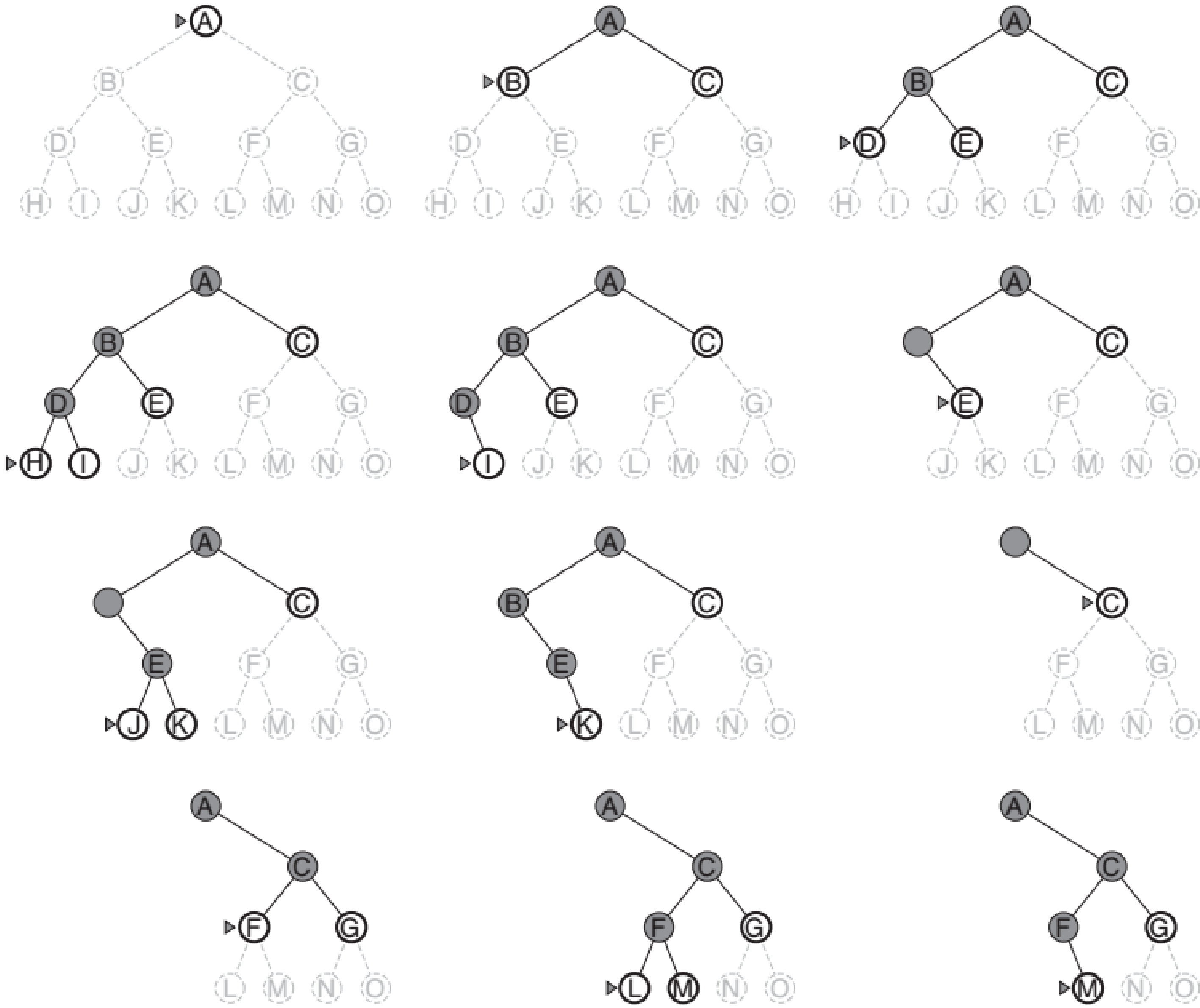




Expansion order:

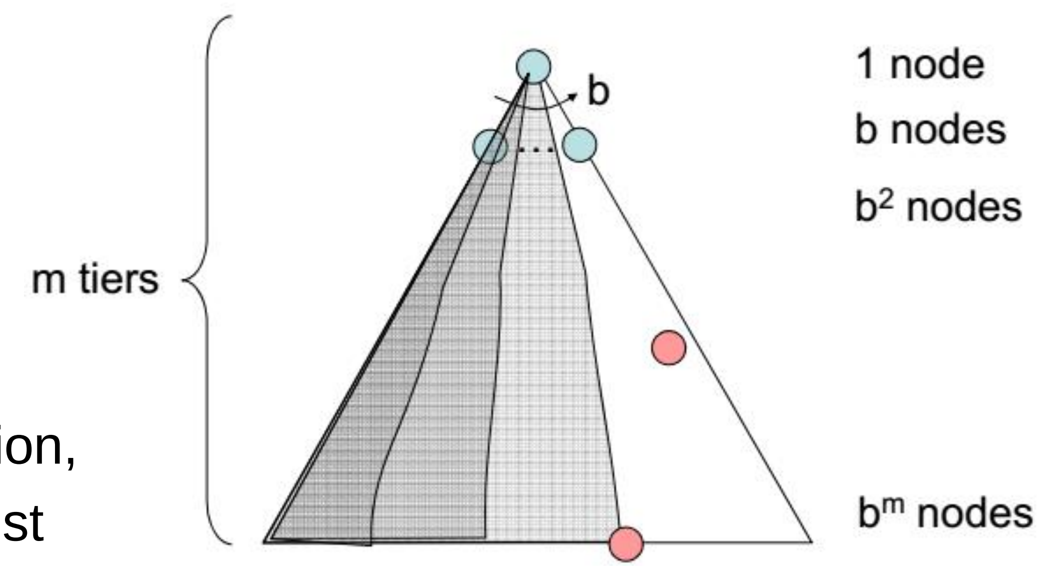
S,*d,b,a,c,a,e,h,p,q,q,r,f,c,a,G*

Depth-first search: An example



An evaluation of DFS

- What nodes DFS expand?
  - Some left prefix of the tree, and it could process the whole tree!
  - If the maximum depth m is finite, it takes time $O(b^m)$
- How much space does the frontier take?
  - Only has siblings on path to root, so $O(bm) \rightarrow$ linear space
- Is it complete?
  - m could be infinite
  - YES if loops prevented
- Is it optimal?
  - NO, the "leftmost" solution, regardless of depth or cost



Completeness of DFS

- **Graph-search: complete**, while **tree-search: not complete**
- Avoid repeated states by checking new states against those on the path from the root to the current node.
  - Infinite loops in finite state spaces are avoided, but the proliferation of redundant paths remains.
- Infinite state spaces: both versions fail if an infinite non-goal path is encountered.
  - E.g., the Knuth's 4 problem $\rightarrow$ keep applying the factorial operator

Comparison of BFS and DFS

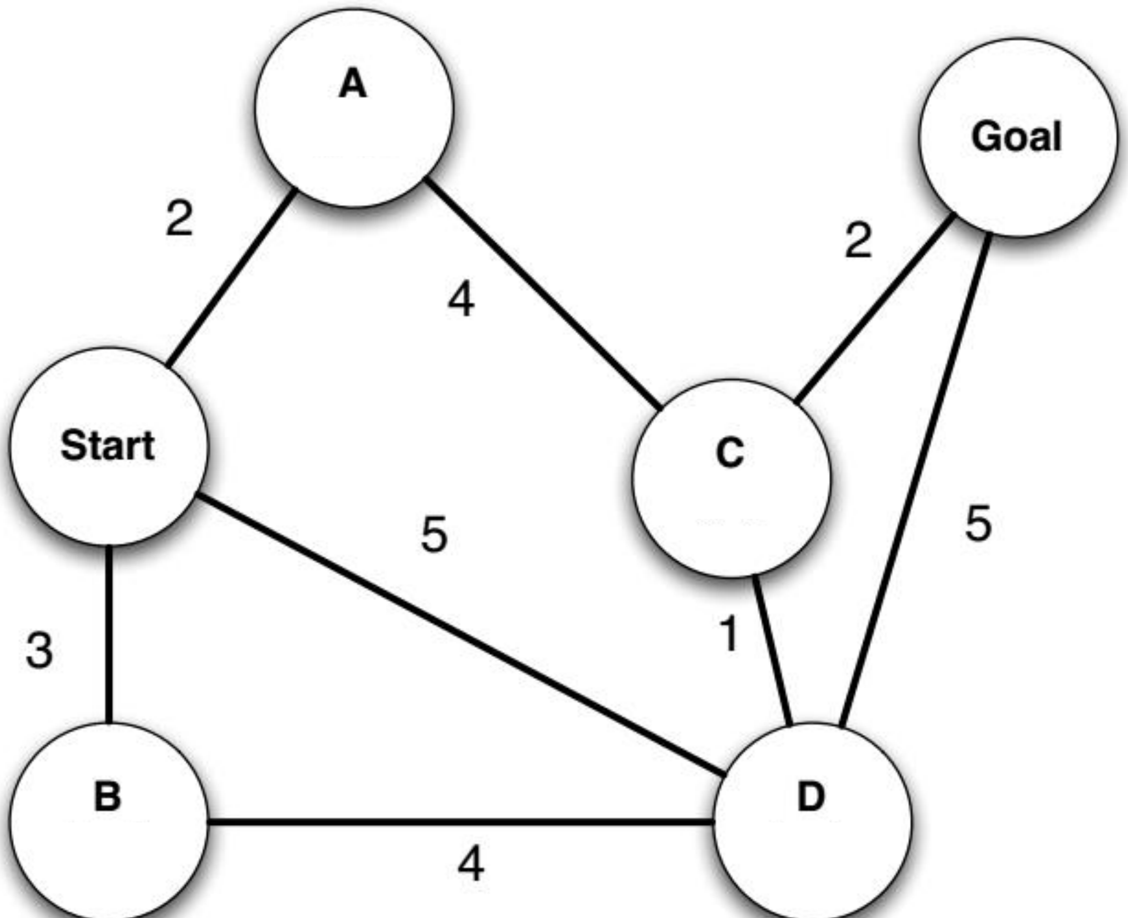
	DFS	BFS
Space complexity	Linear space	Maybe the whole search space
Time complexity	Same, better on the average (many goals, no loops, and no infinite paths)	Same, better in worst-cases
In general	better if many goals, not many loops, and much better in terms of memory.	better if goal is not deep, infinite paths, many loops, or small search space

DFS in use

- The goal test is applied to each node when it is **generated** rather than when it is selected for expansion.
- Avoid repeated states by checking new states against those on the path from the root to the current node.

Quiz 03: Depth-first search

- Work out the order in which states are expanded, as well as the path returned by the algorithm. Assume ties resolve in such a way that states with earlier alphabetical order are expanded first.

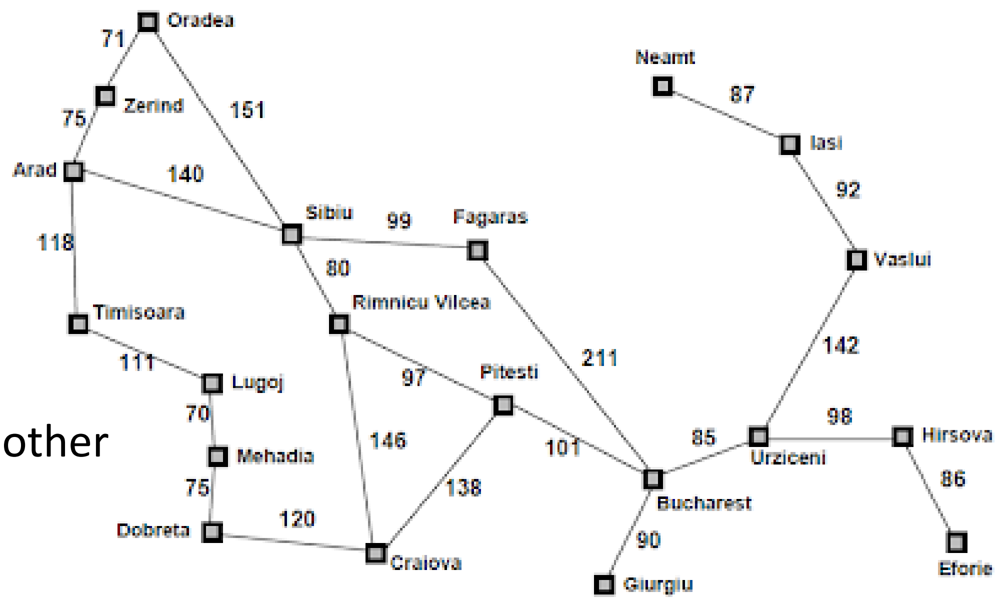


Depth-limited search

Depth-limited search (DLS)

- Standard DFS with a predetermined depth limit l
  - Nodes at depth l are treated as if they have no successors $\rightarrow$ infinite problems solved.
- Depth limits can be based on knowledge of the problem.
  - Diameter of state-space, typically unknown ahead of time in practice

E.g., 20 cities in the Romania map
$\rightarrow l = 19$
but any city is reached from any other city in at most 9 steps
$\rightarrow l = 9$ is better



Depth-limited search (DLS)

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or
                                                        failure/cutoff
return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE),
                                              problem, limit)
```

```
function RECURSIVE-DLS(node, problem, limit) returns a solution, or
                                              failure/cutoff
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  else if limit = 0 then return cutoff
  else cutoff_occurred? ← false
  for each action in problem.ACTIONS(node.STATE) do
    child ← CHILD-NODE(problem, node, action)
    result ← RECURSIVE-DLS(child, problem, limit – 1)
    if result = cutoff then cutoff occurred? ← true
    else if result ≠ failure then return result
  if cutoff occurred? then return cutoff else return failure
```

- Failure: no solution
- Cutoff: no solution within the depth limit

An evaluation of DLS

- Completeness
  - Maybe NO if $l < d$
- Optimality
  - NO if $l > d$
- Time complexity
  - $O(b^l)$
- Space complexity
  - $O(bl)$

DFS is a special case of DLS when $l = \infty$

Iterative deepening search

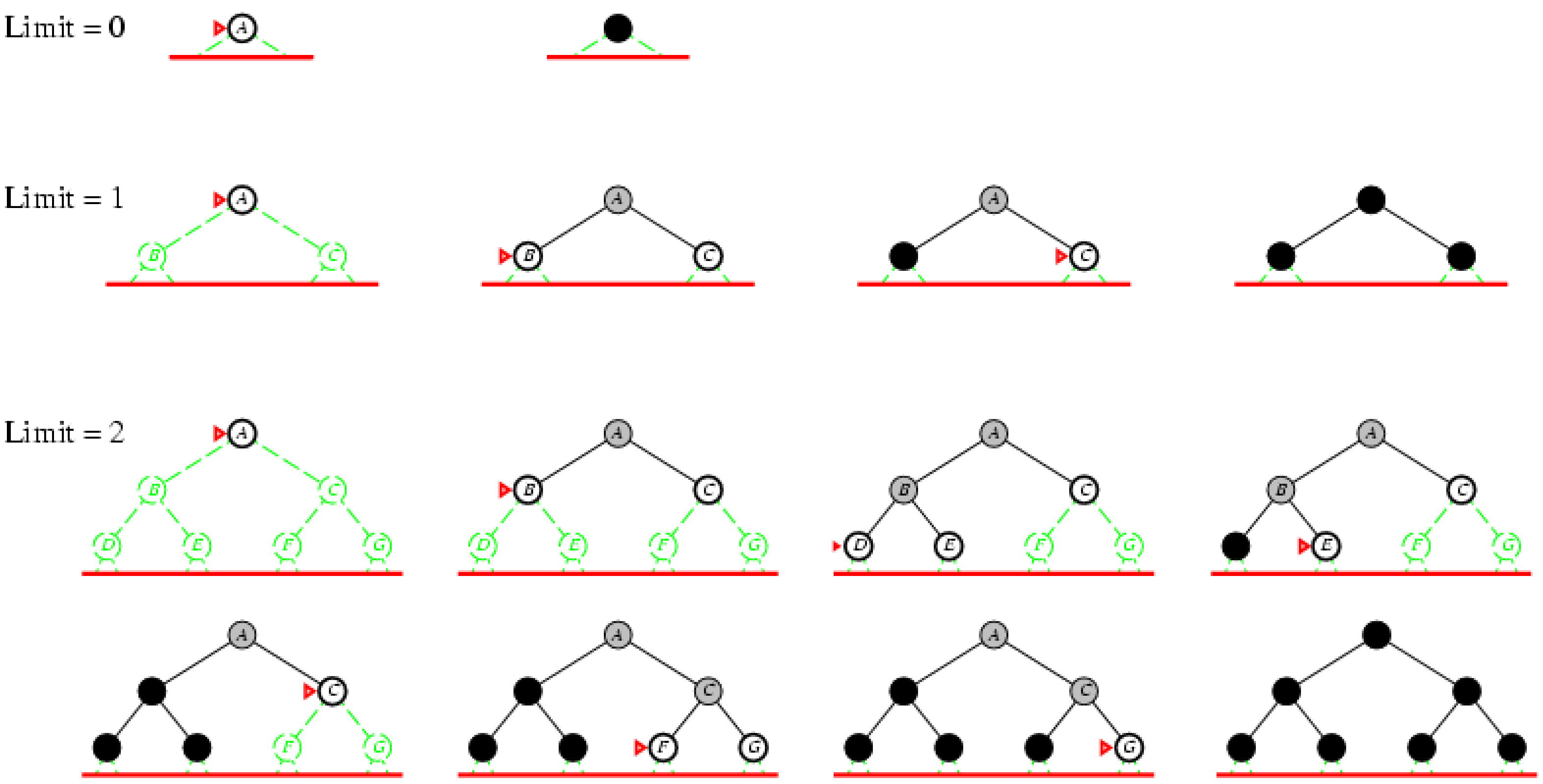
Iterative deepening search (IDS)

- General strategy, often used in combination with depth-first tree search to find the best depth limit

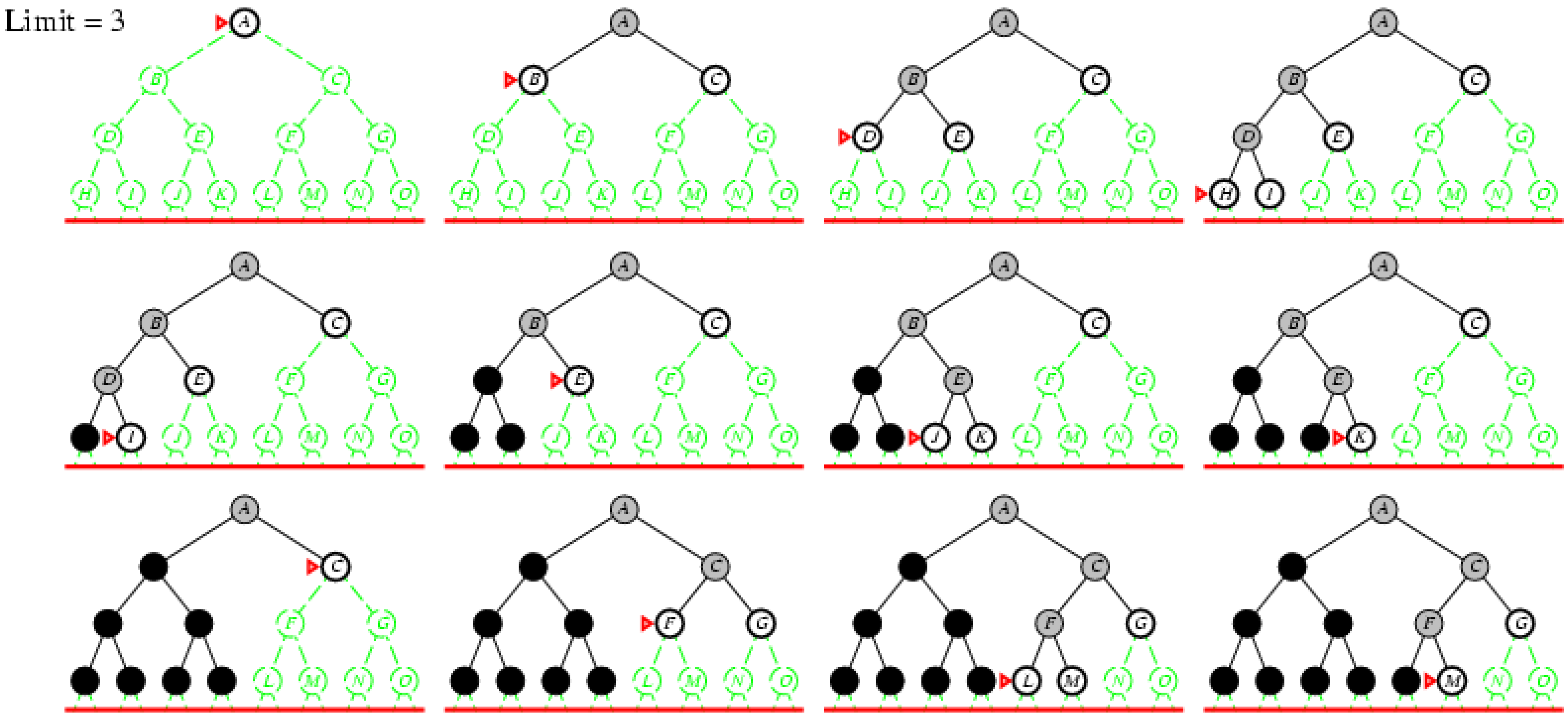
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
  if result ≠ cutoff then return result
```

- Gradually increase the limit until a goal is found.
  - The depth limit reaches the depth d of the shallowest goal node.

Iterative deepening search (IDS)



Iterative deepening search (IDS)



An evaluation of IDS

- Completeness
  - YES when the branching factor is finite
- Optimality
  - YES if step cost = 1

Similar to BFS

- Time complexity
  - $(d+1)b^0 + db^1 + (d-1)b^d = O(b^d)$
- Space complexity
  - $O(bd)$, similar to DFS

- Preferred when the search space is large and the depth of the solution is not known

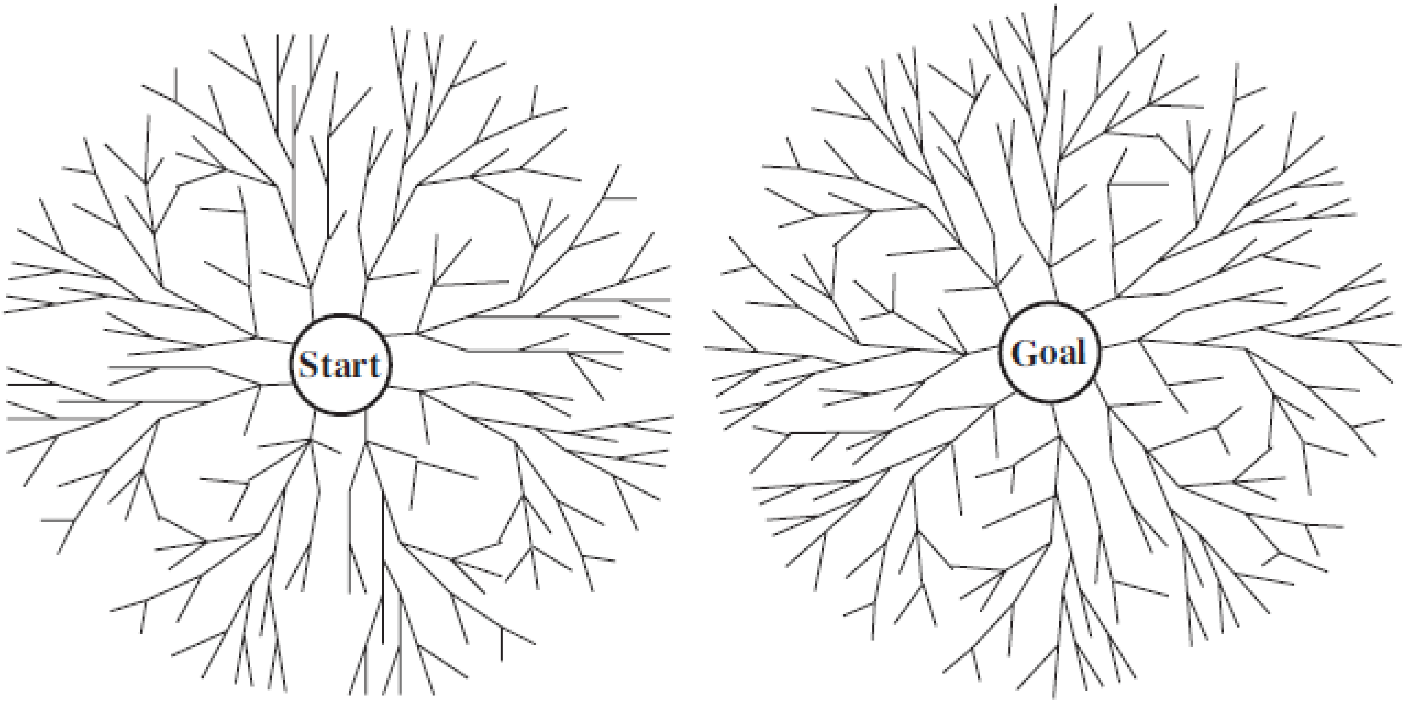
Quiz 04: Iterative deepening search

- IDS seem to be wasteful because states are generated multiple times. However, it turns out to be not too costly, compared to BFS.
- Why?

Bidirectional search

Bidirectional search

- Two simultaneous searches: one from the initial state towards, and the other from the goal state backwards
- Hoping that two searches meet in the middle



Bidirectional search

- Goal test: whether the frontiers of two searches intersect
- Optimality: maybe NO
- Time and Space complexity: $O(b^{d/2})$

- It sounds attractive, but what is the **tradeoff**?
- Space requirement for the frontiers of at least one search
- Not easy to search backwards (predecessors required)
  - In case there are more than 1 goals
  - Especially if the goal is an abstract description (no queen attacks another queen)

A summary of uninformed search

- Comparison of uninformed algorithms (tree-search versions)

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

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.

THE END