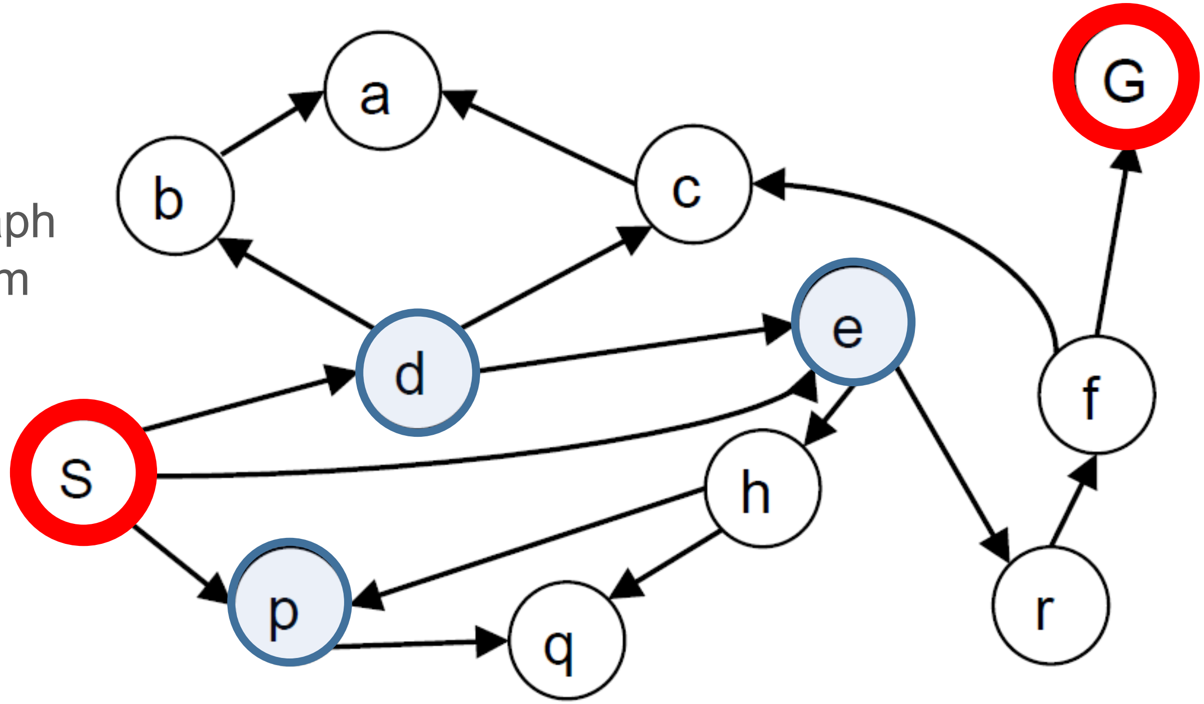
Uninformed search strategies

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

Breadth-First Search

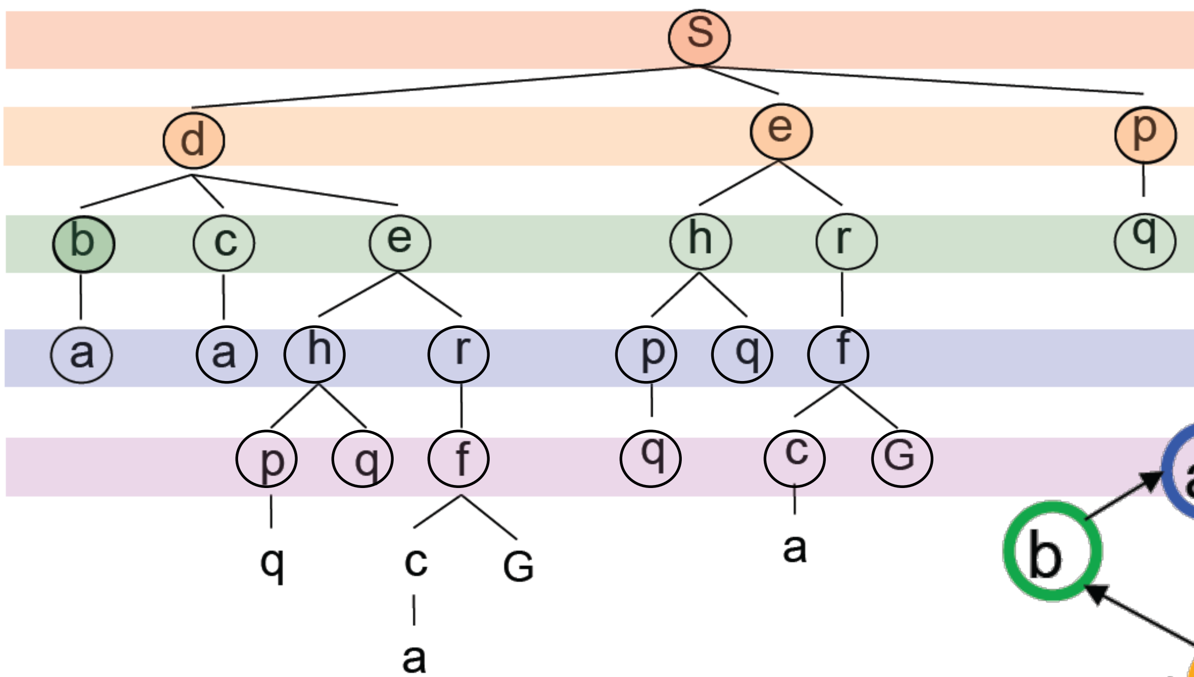
Breadth-first search (BFS)

Example state space graph
for a tiny search problem



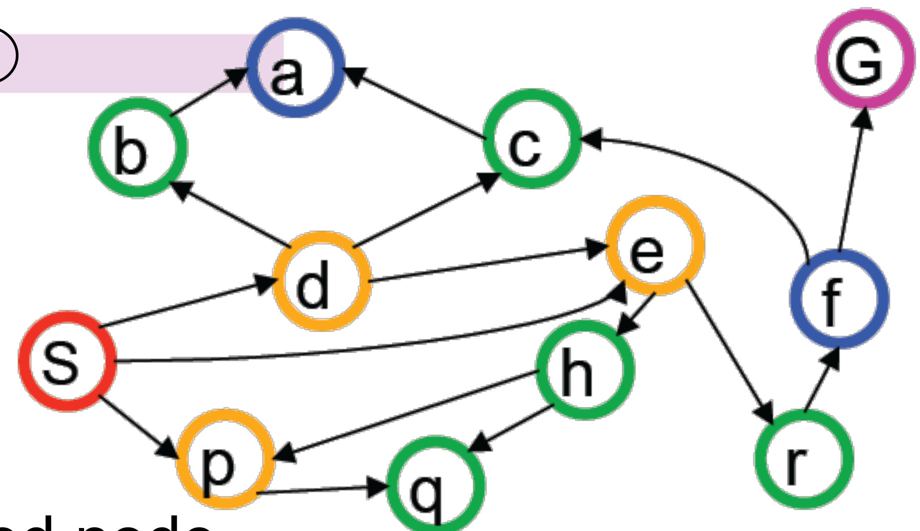
- The **root node is expanded first**, then all the successors of the root node are expanded next, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

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

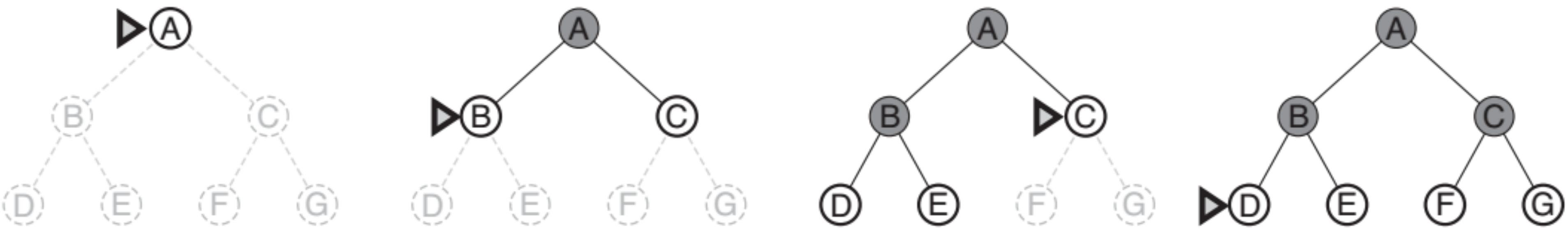
Breadth-first search (BFS)

- 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

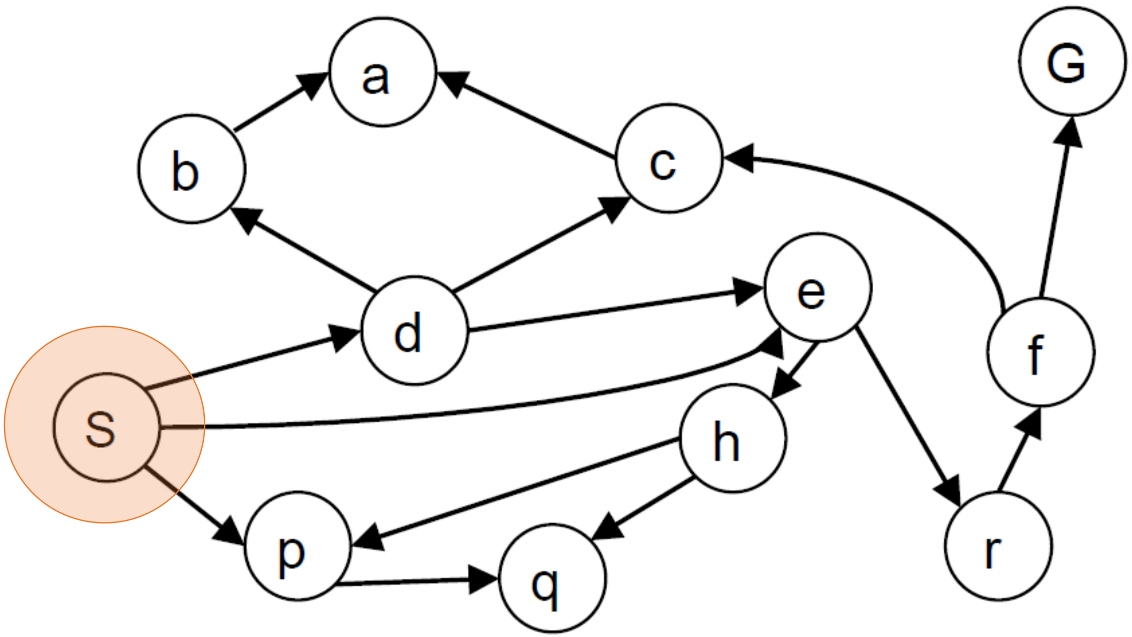
```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if 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 or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Breadth-first search on a graph

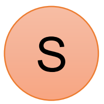


Breadth-first search on a simple binary tree.
At each stage, the node to be expanded next is indicated by a marker

Breadth-first search: An example

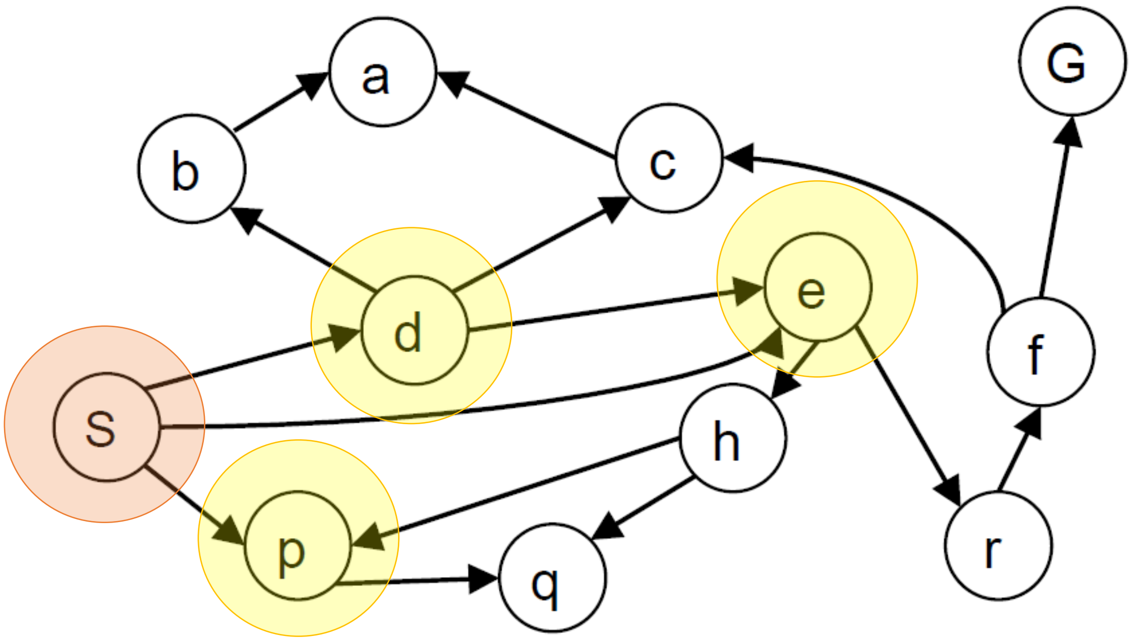


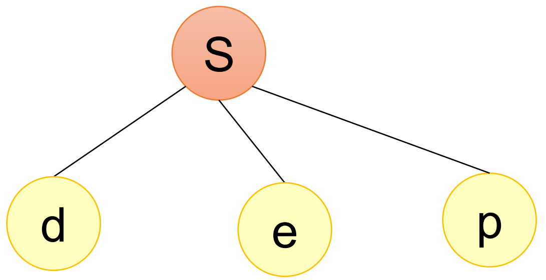
d = 0



Search Tree

Breadth-first search: An example

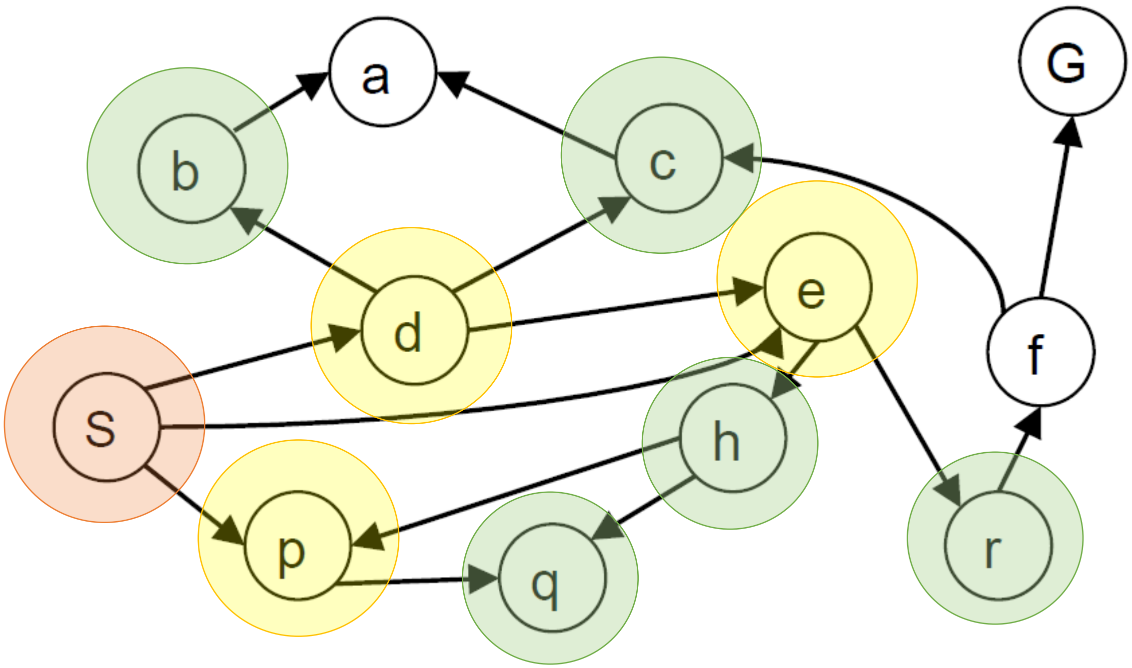




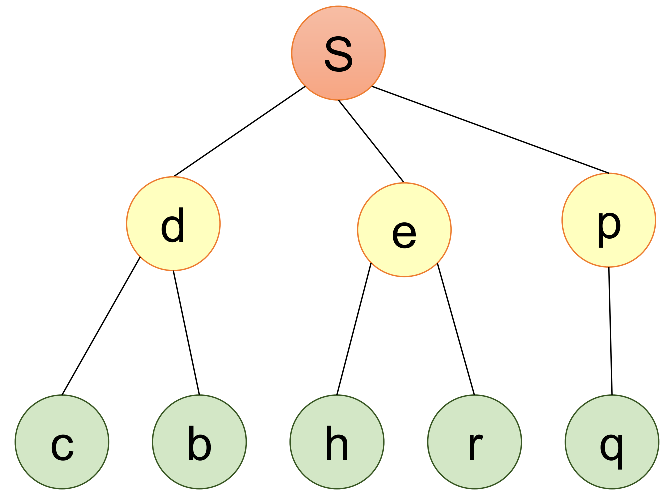
$d = 1$

Search Tree

Breadth-first search: An example

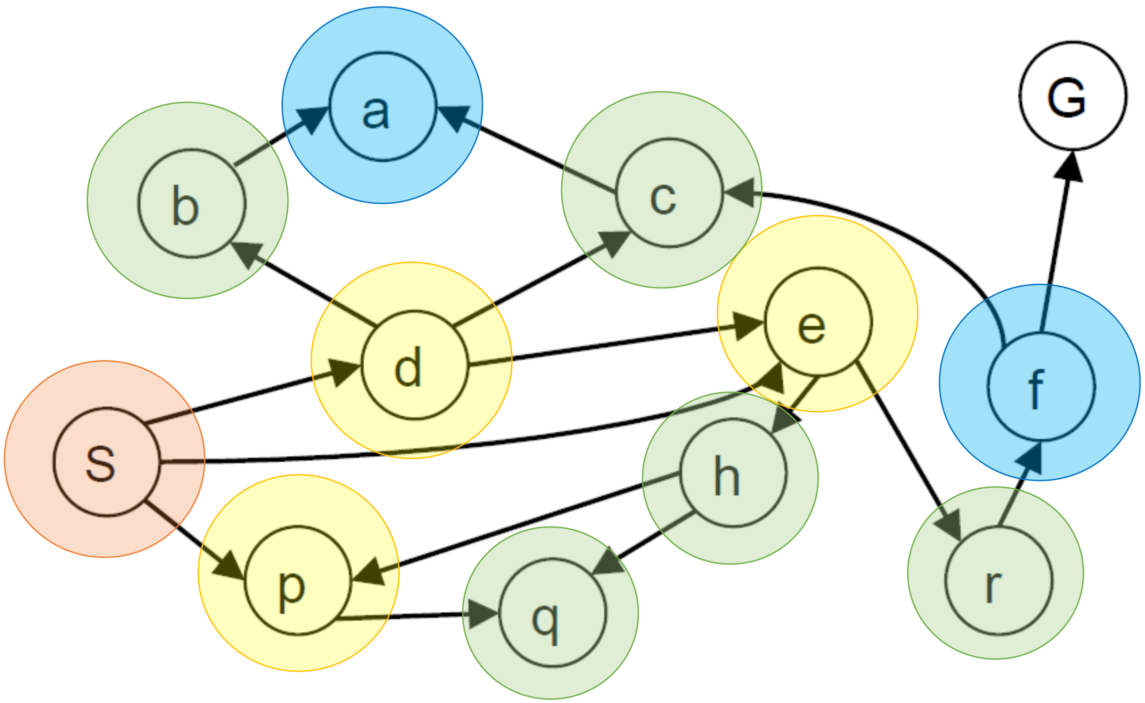


d = 2

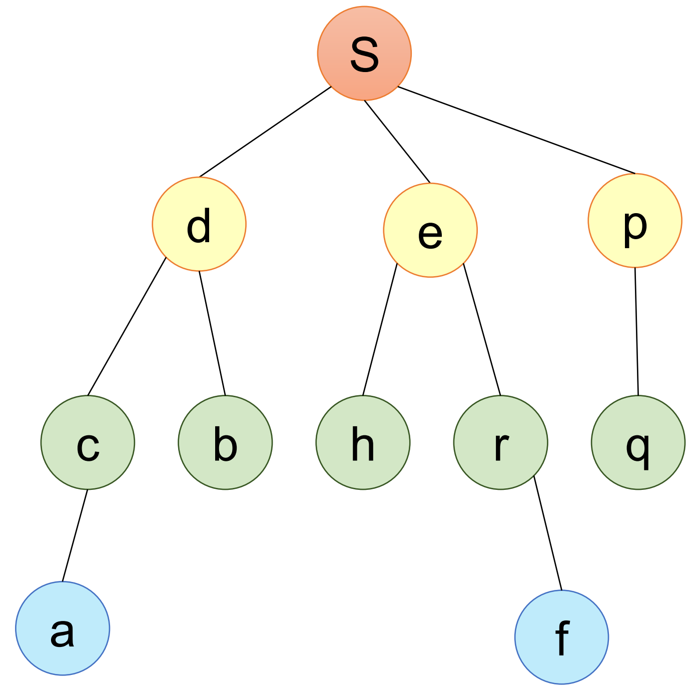


Search Tree

Breadth-first search: An example

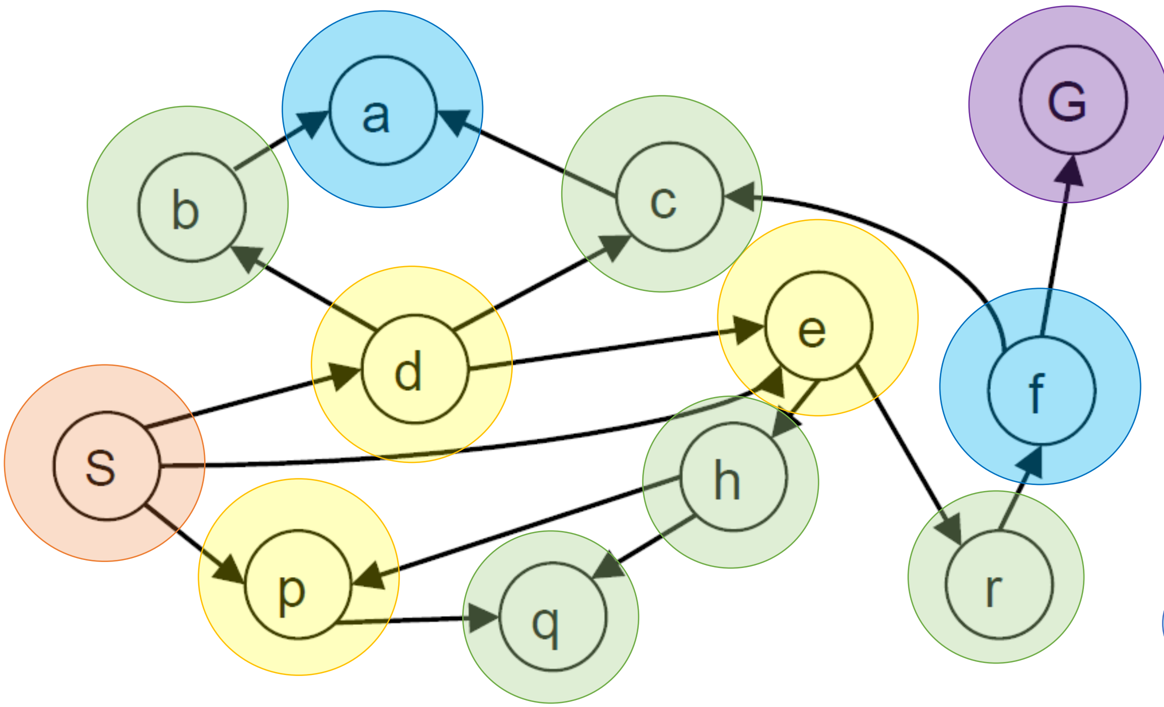


$d = 3$

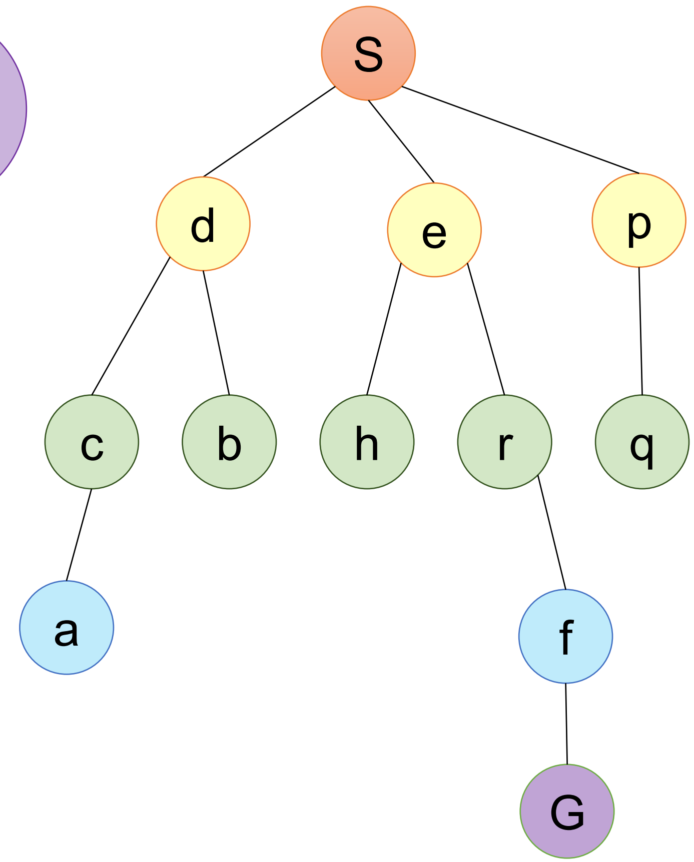


Search Tree

Breadth-first search: An example

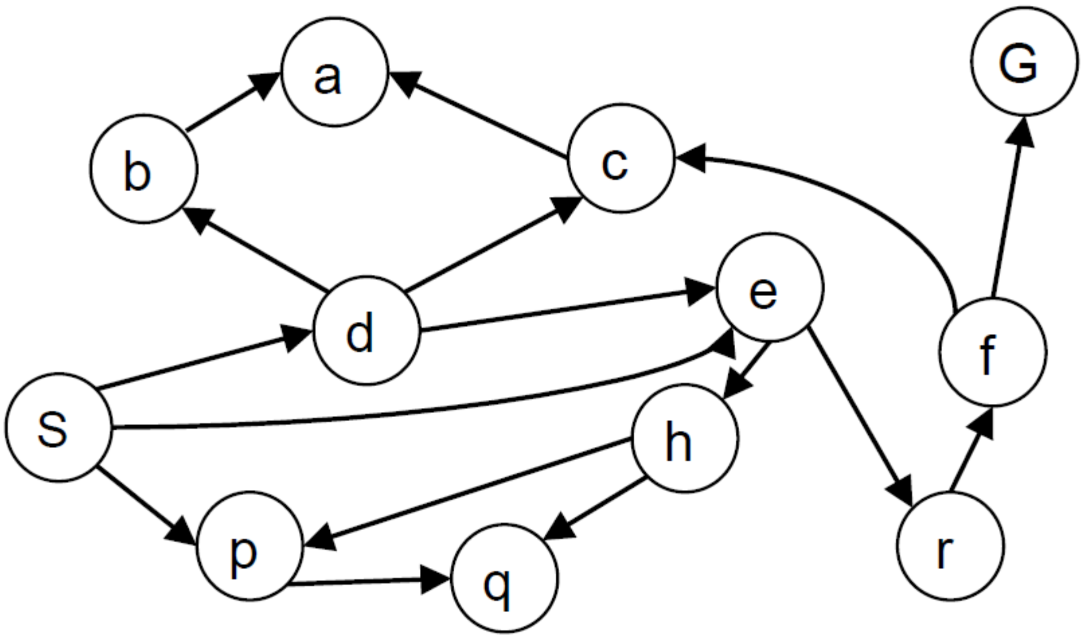


$d = 4$

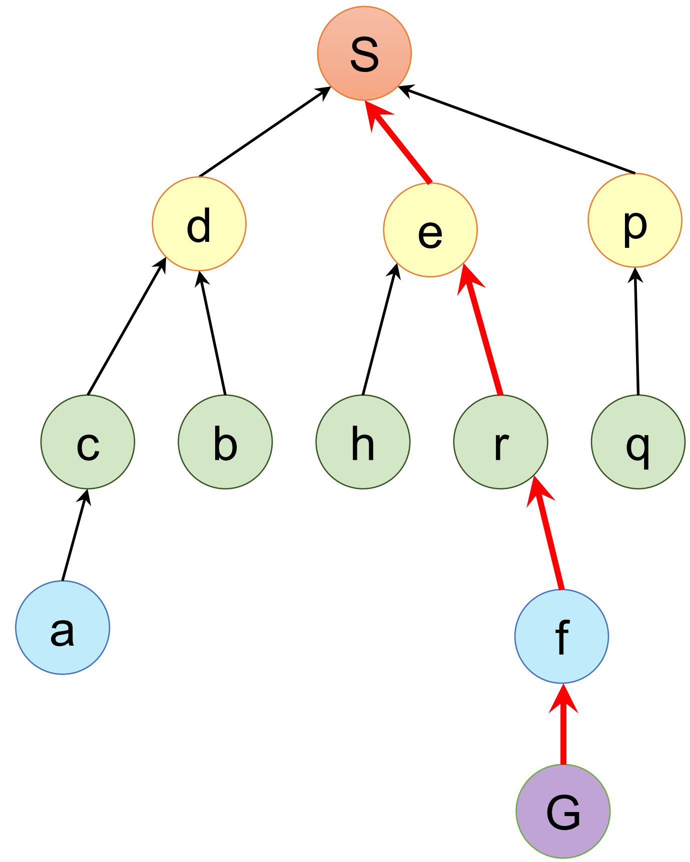


Search Tree

Breadth-first search: An example



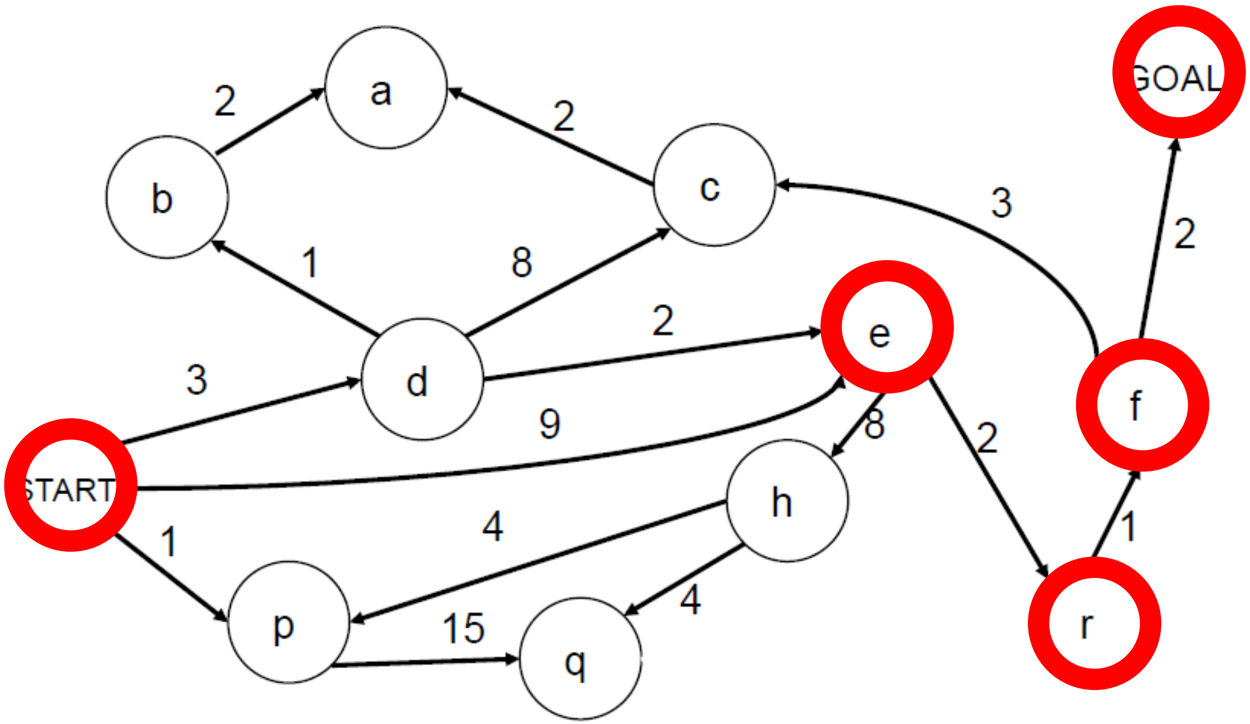
Search path: $S \rightarrow e \rightarrow r \rightarrow f \rightarrow G$



Search Tree

Uniform-Cost Search

Search with varying step costs



- BFS finds the path with the fewest steps but does not always find the cheapest path.
- *Find 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

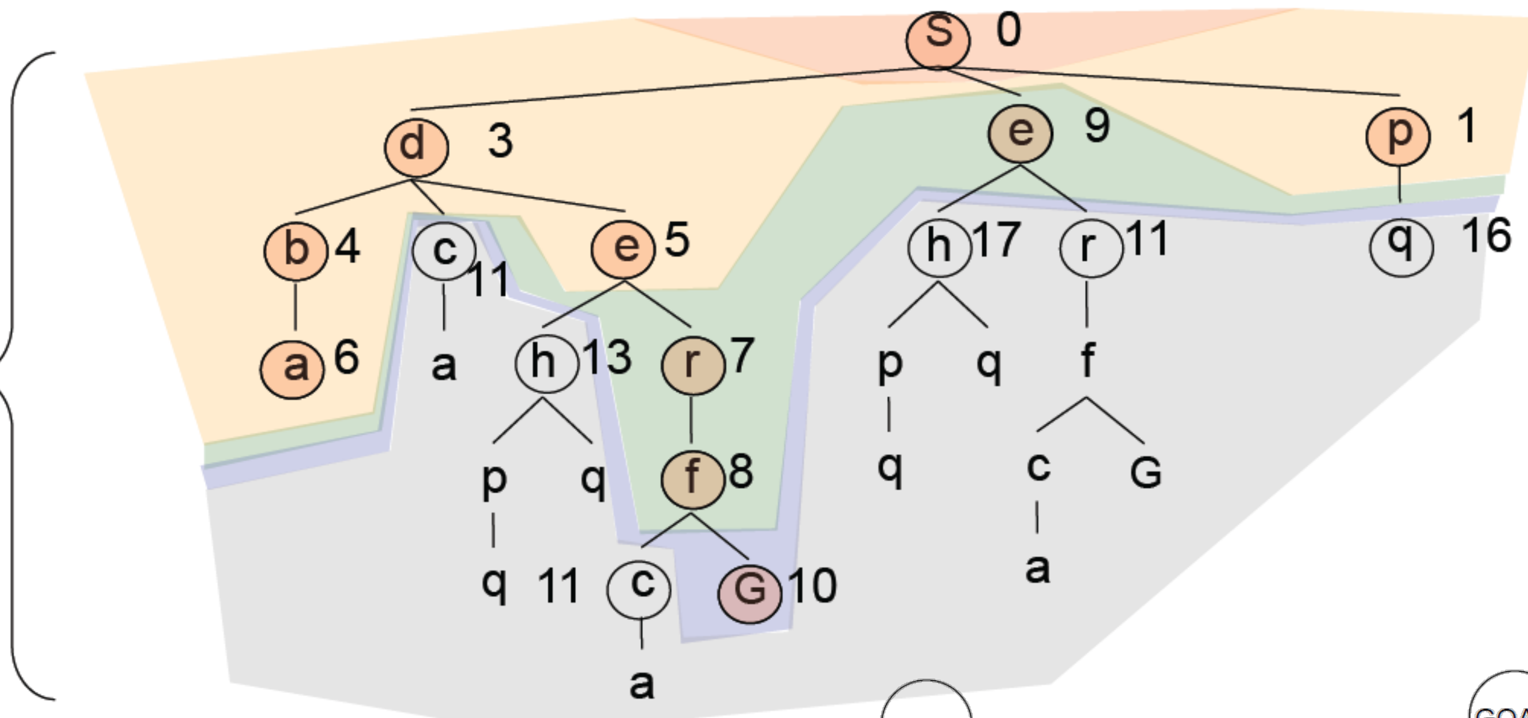
  $\rightarrow$ Equivalent to Dijkstra's algorithm in general
- The goal test is applied to a node when it is selected for expansion
- A test is added in case a better path is found to a node currently on the frontier.

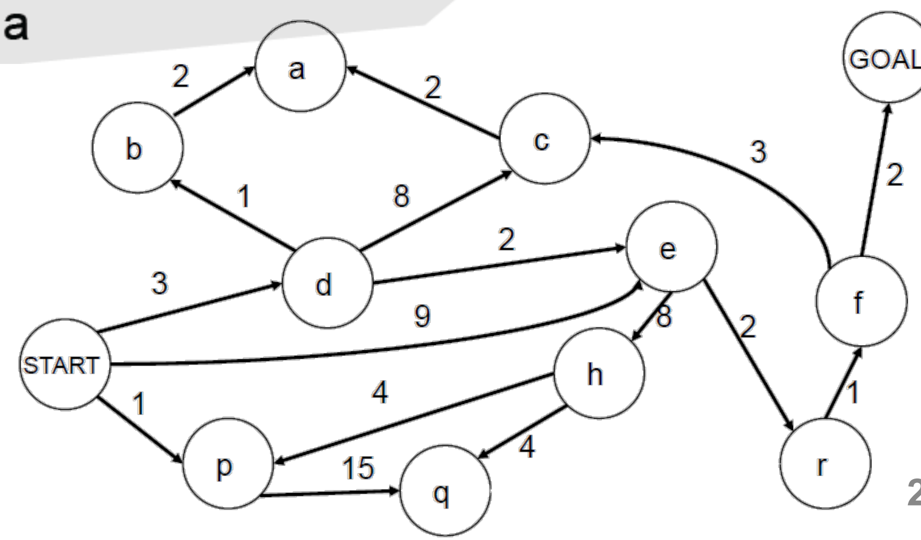
Uniform-cost search (UCS)

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* or *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

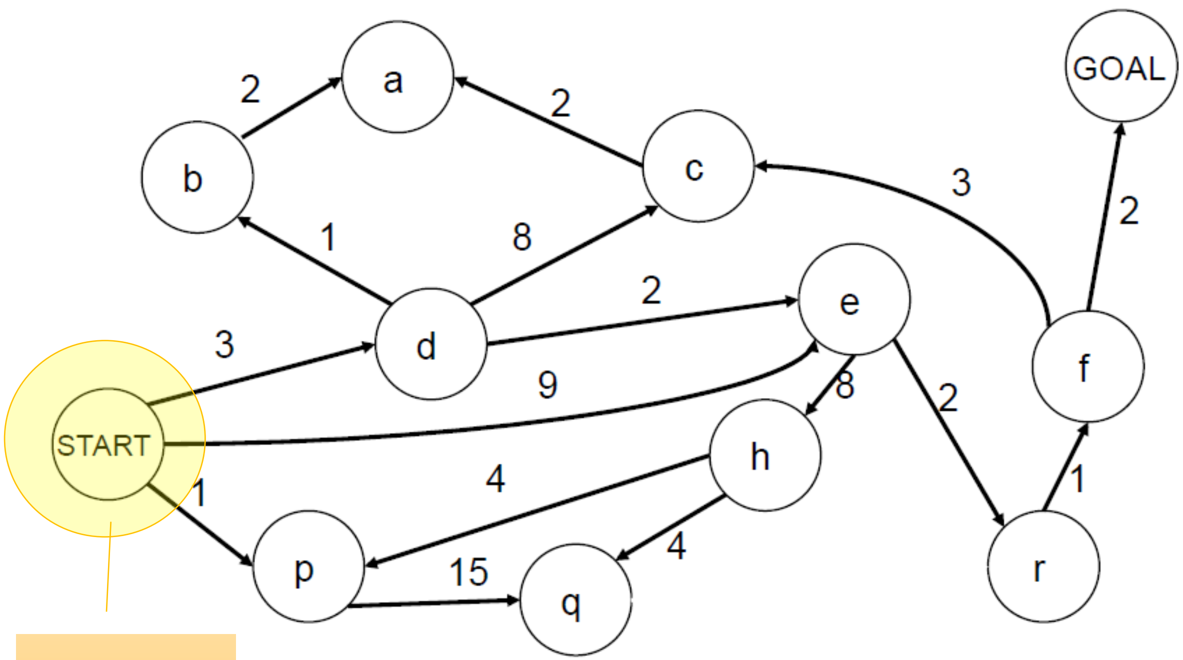




Expansion order

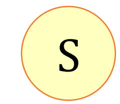
S,p,d,b,e,a,r,f,e,G

Uniform-cost search: An example



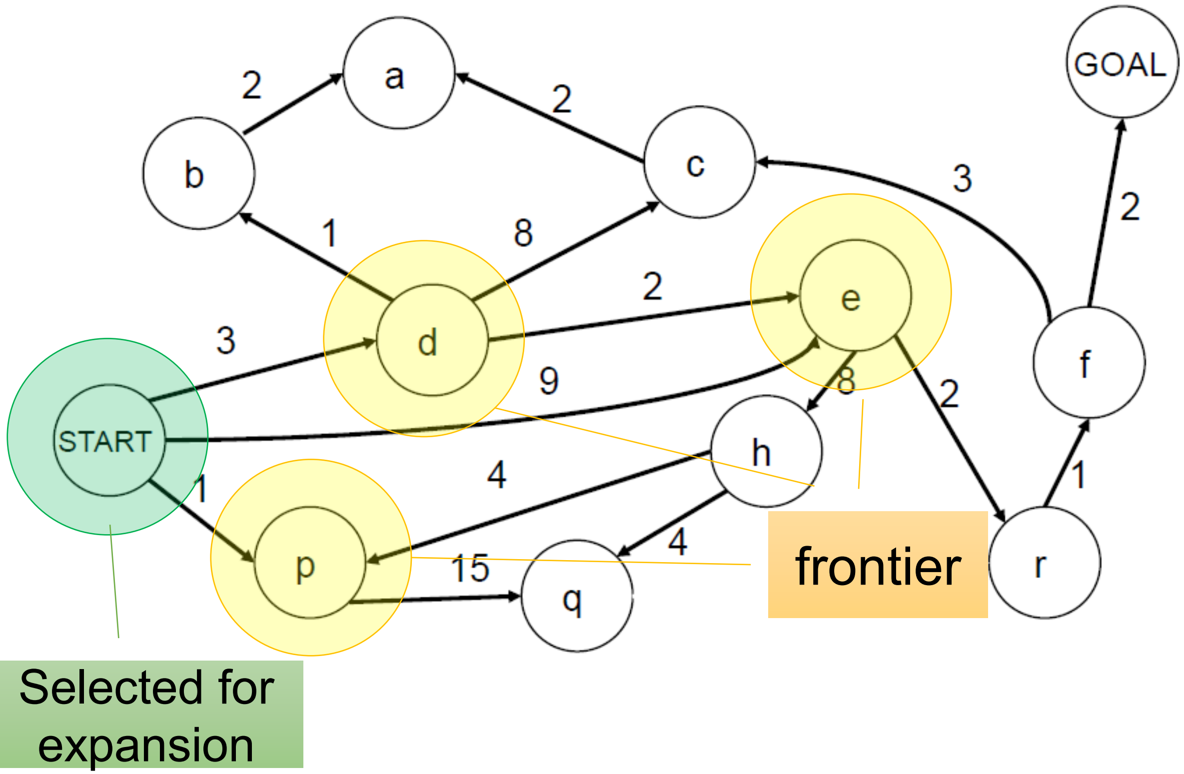
frontier

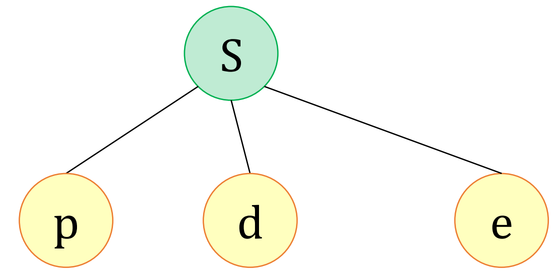
PQ = { (S:0) }



Search Tree

Uniform-cost search: An example

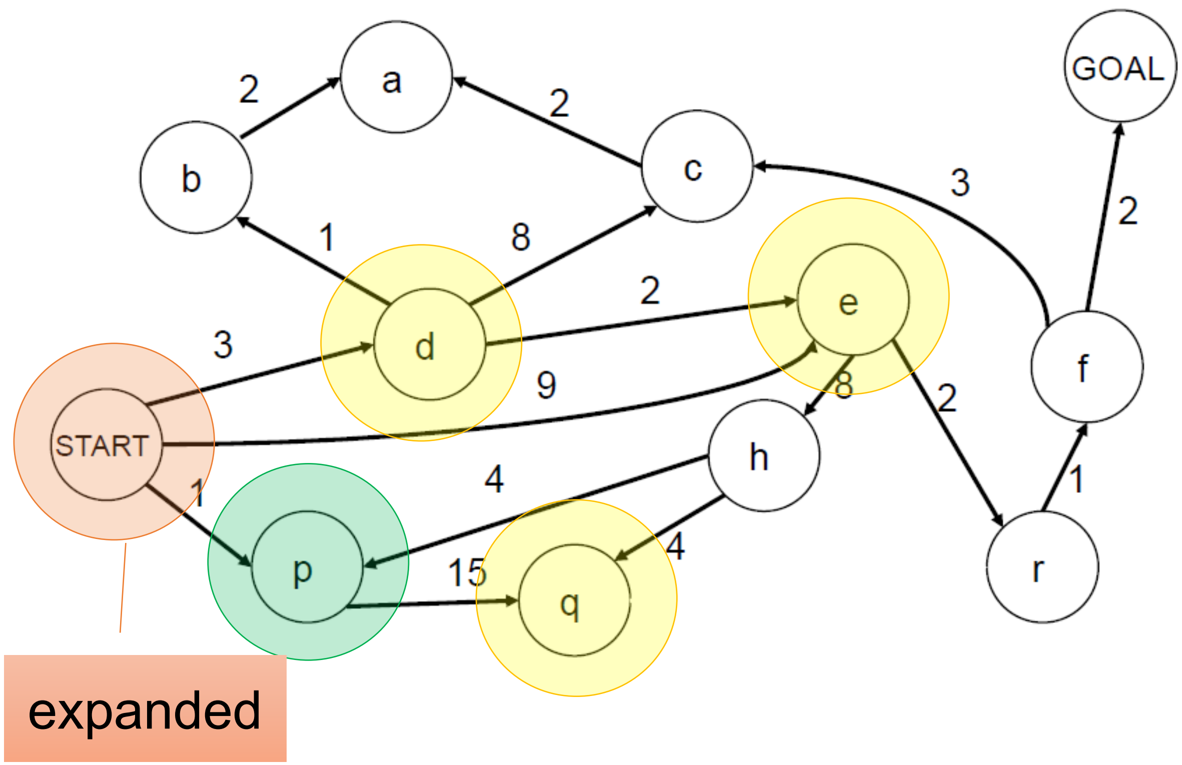




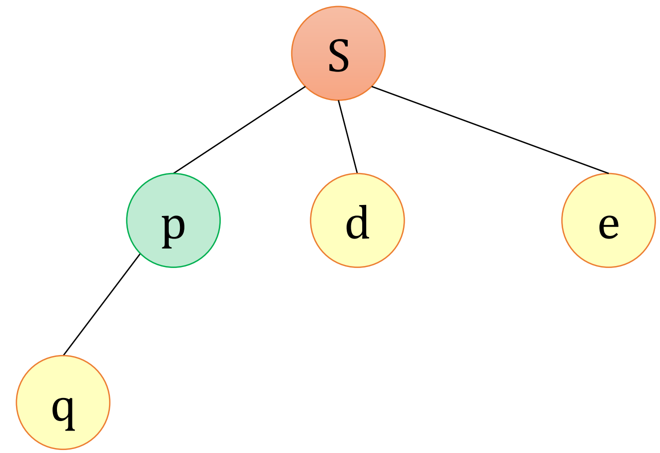
PQ = { (p:1), (d:3), (e:9) }

Search Tree

Uniform-cost search: An example

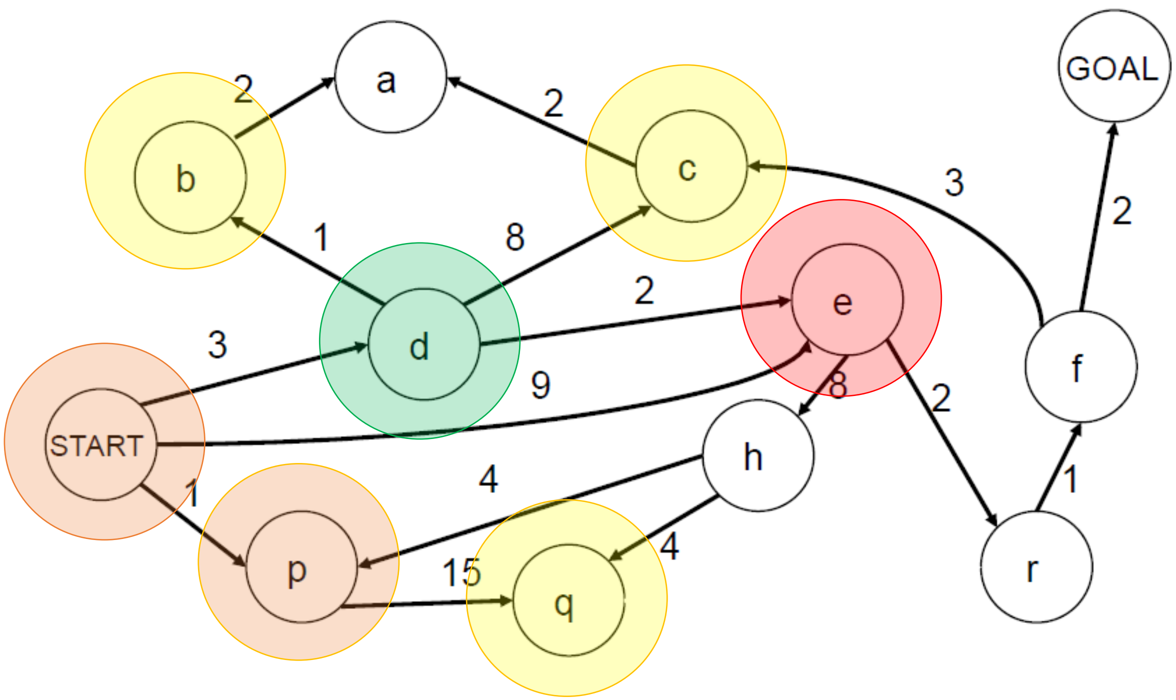


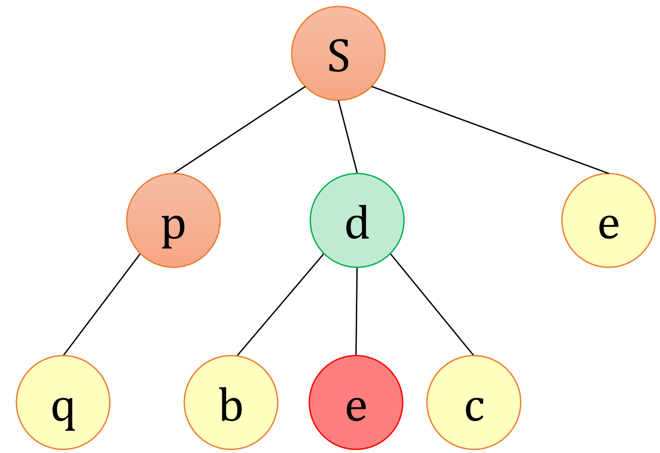
PQ = { (d:3), (e:9), (q:16) }



Search Tree

Uniform-cost search: An example



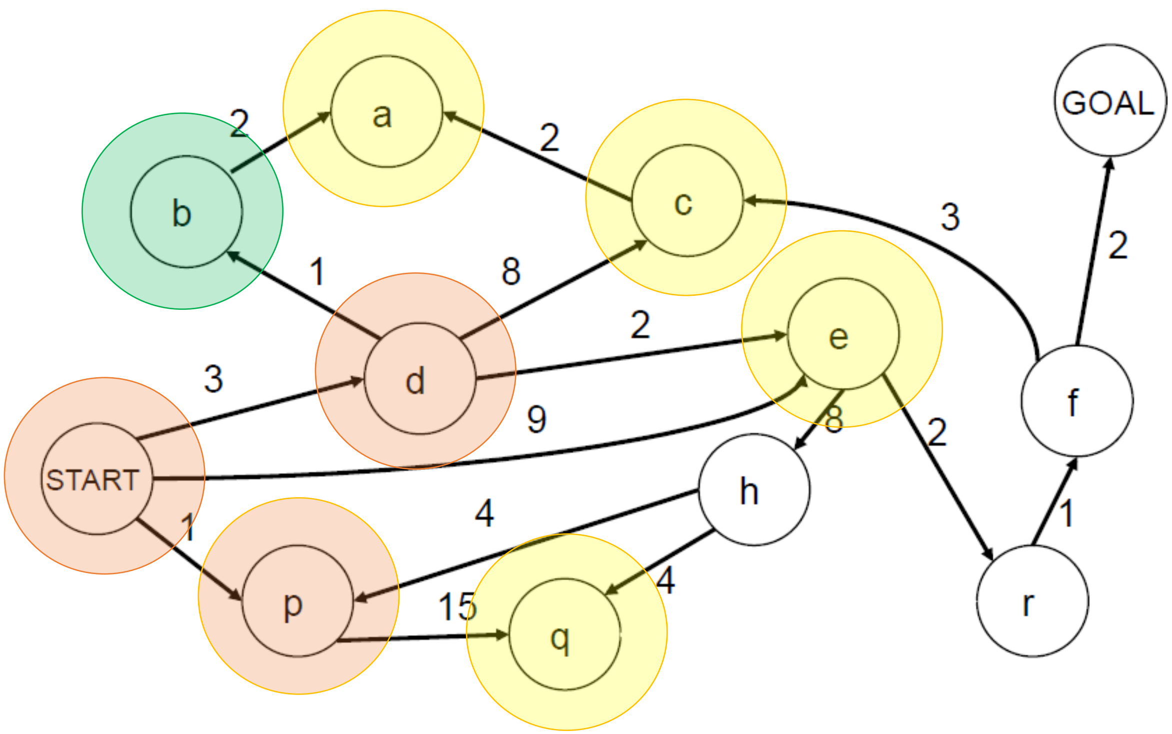


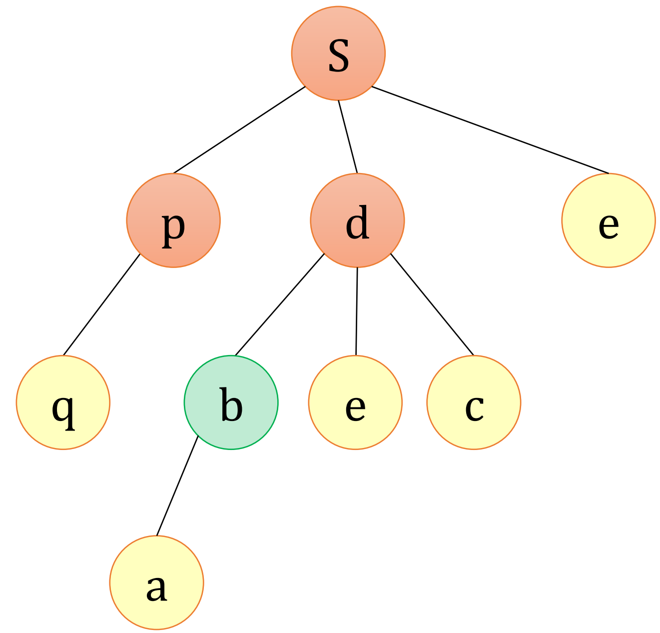
PQ = { (b:4), (e:5), (c:11), (q:16) }

Update path cost of *e*

Search Tree

Uniform-cost search: An example

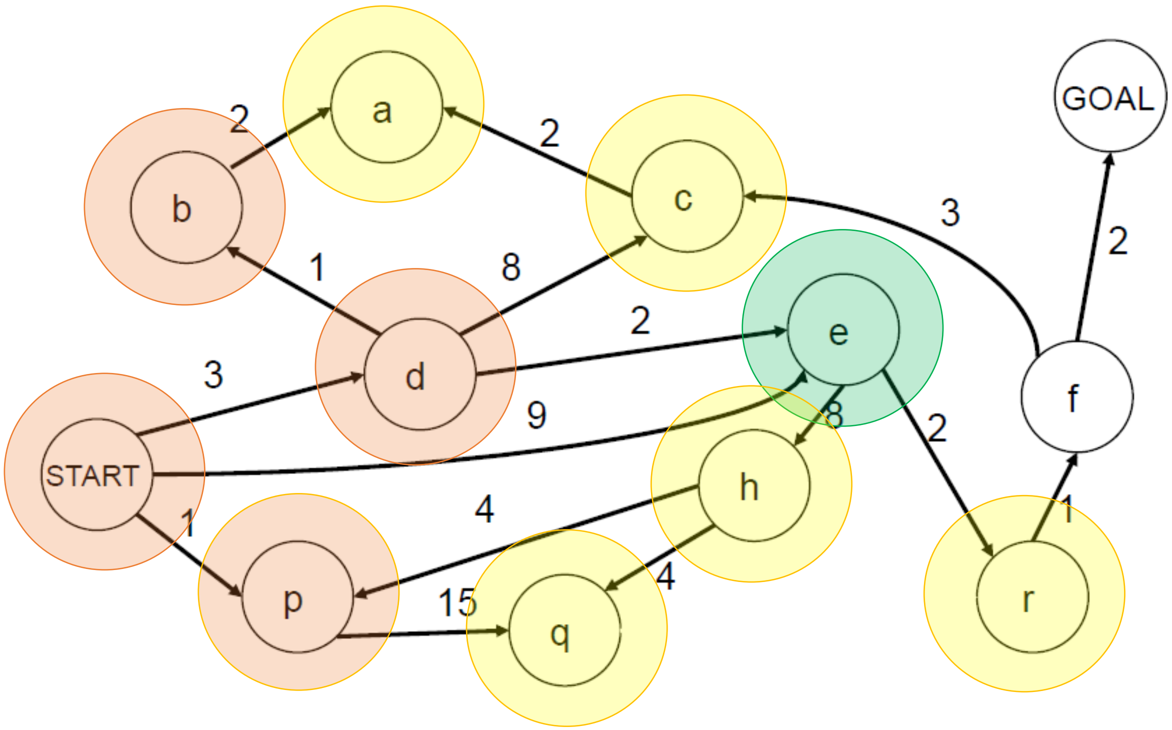


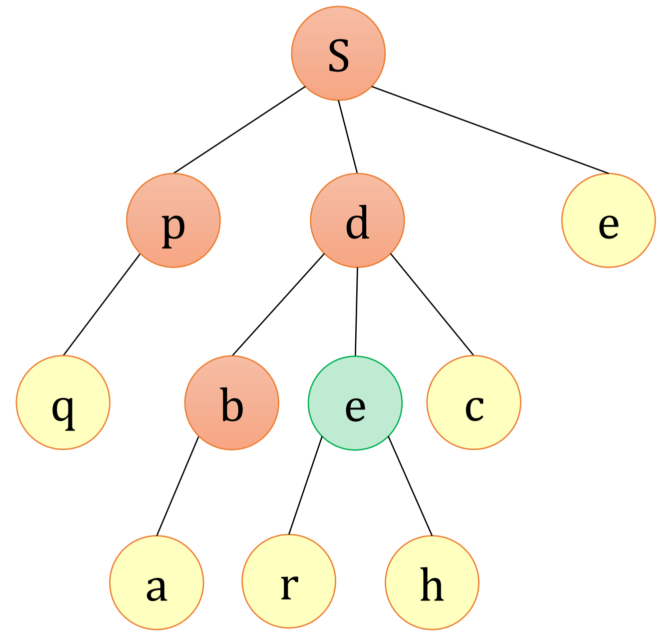


PQ = { (e:5), (a:6), (c:11), (q:16) }

Search Tree

Uniform-cost search: An example

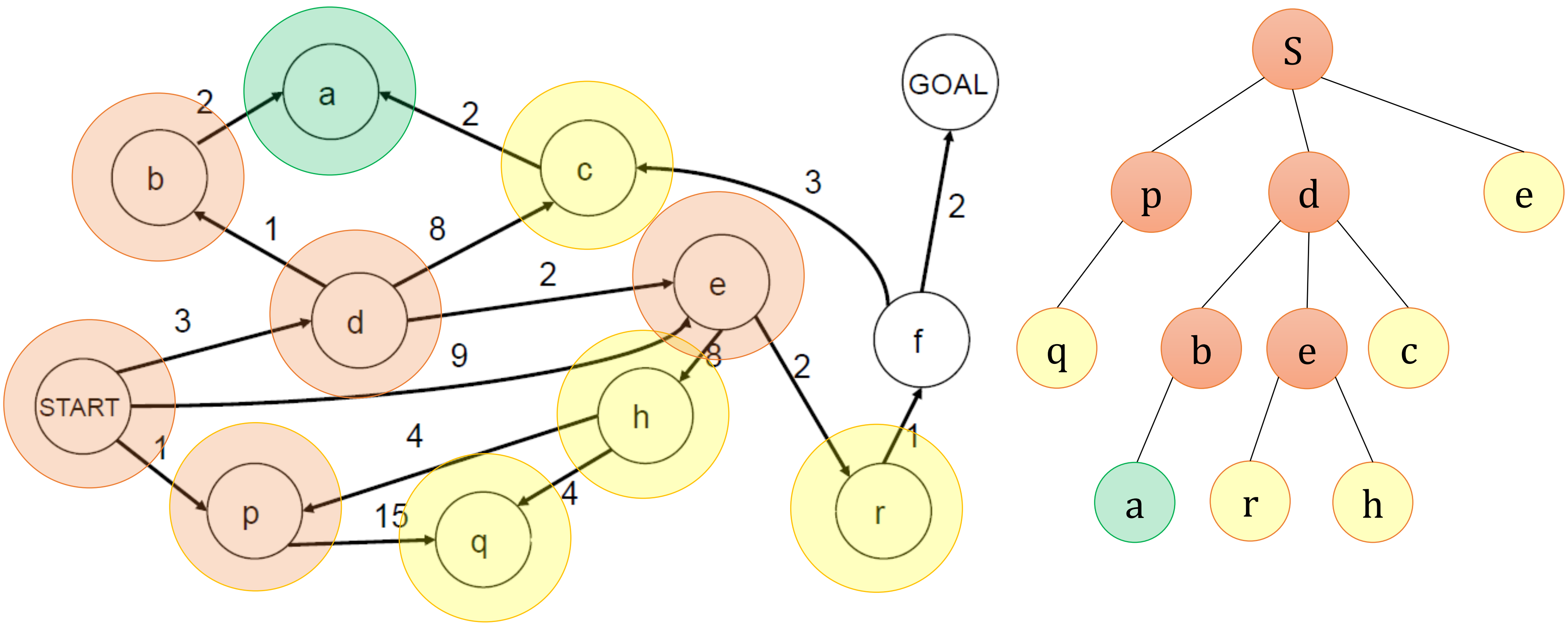




PQ = { (a:6), (r:7), (c:11), (h:13), (q:16) }

Search Tree

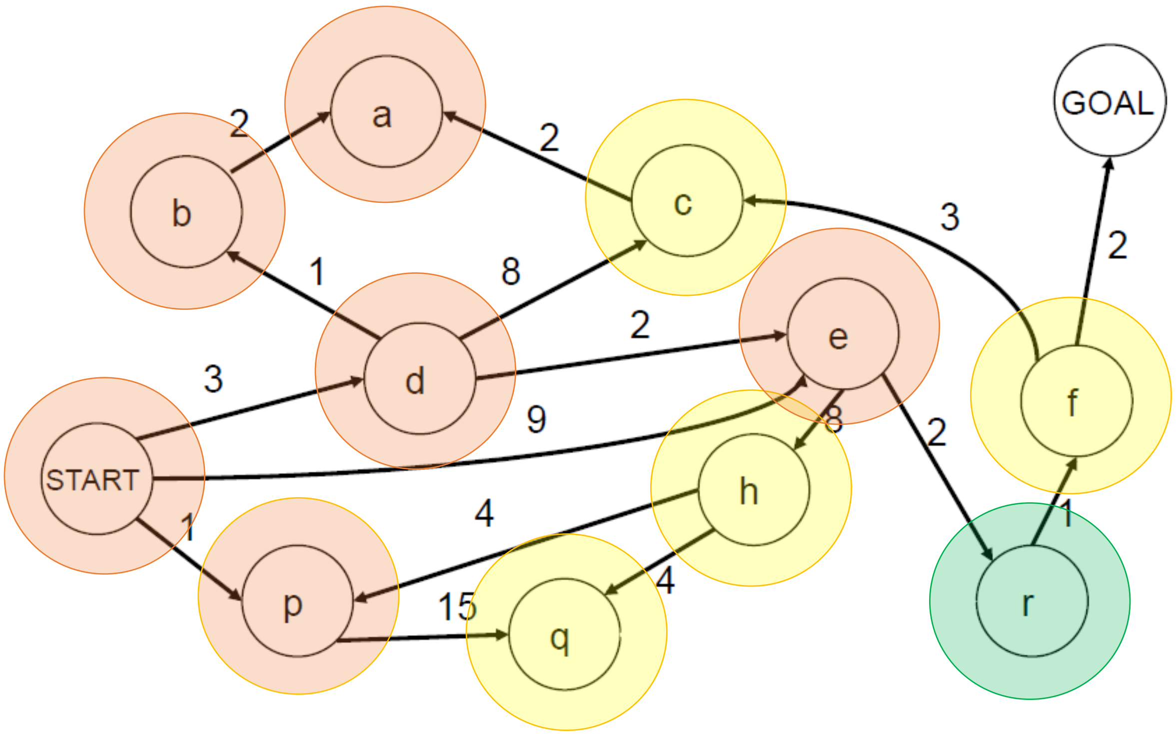
Uniform-cost search: An example

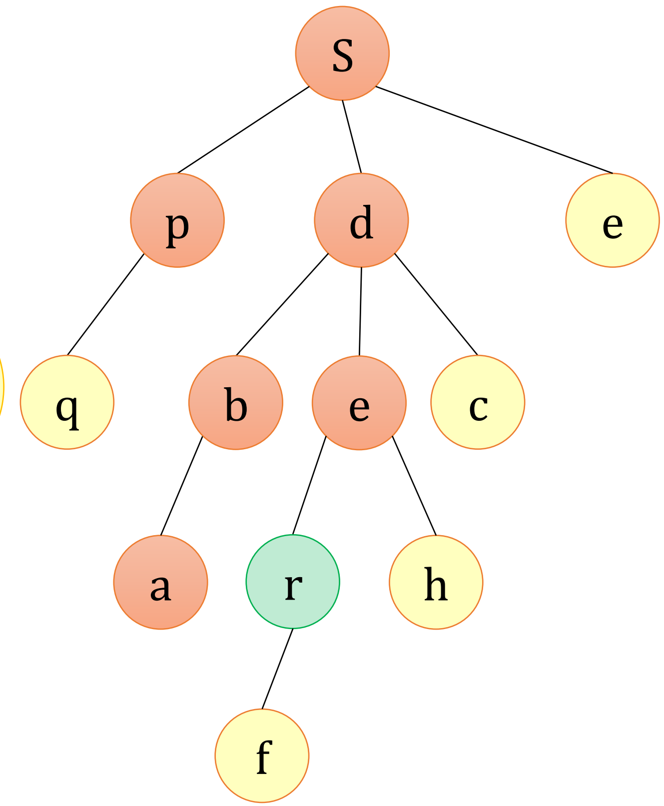


PQ = { (r:7), (c:11), (h:13), (q:16) }

Search Tree

Uniform-cost search: An example

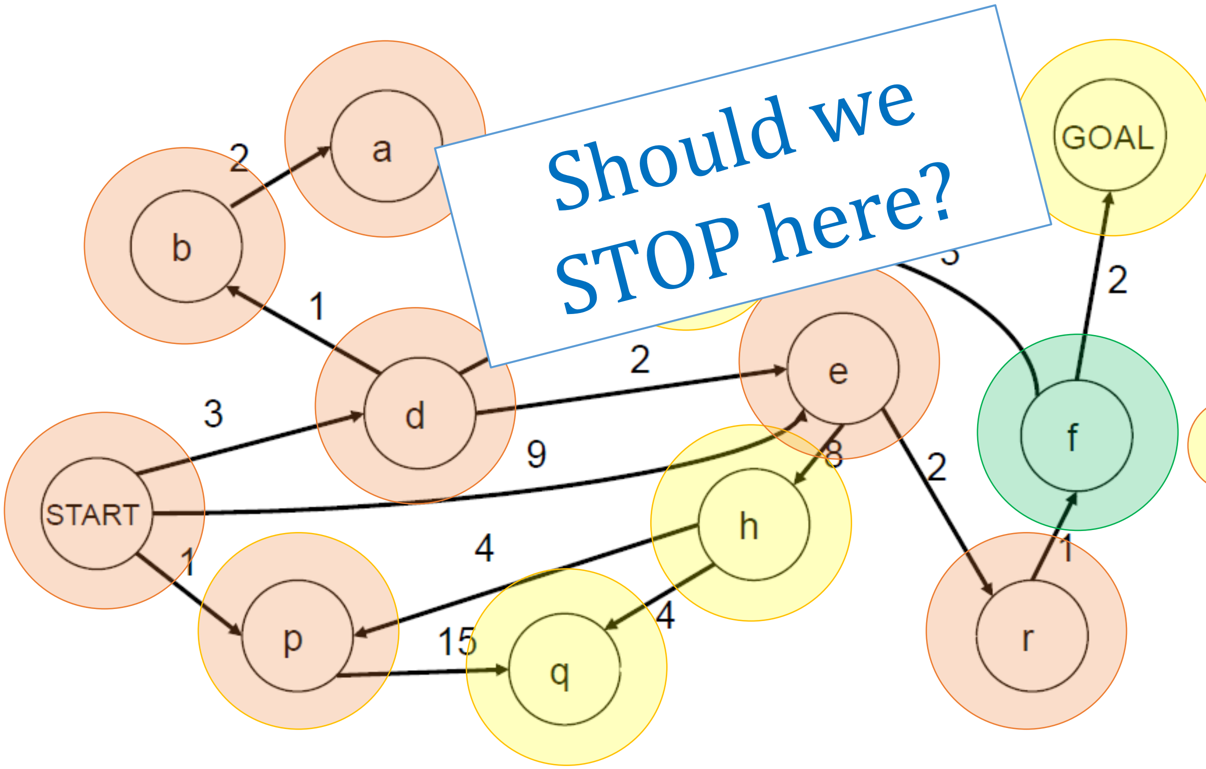




PQ = { (f:8), (c:11), (h:13), (q:16) }

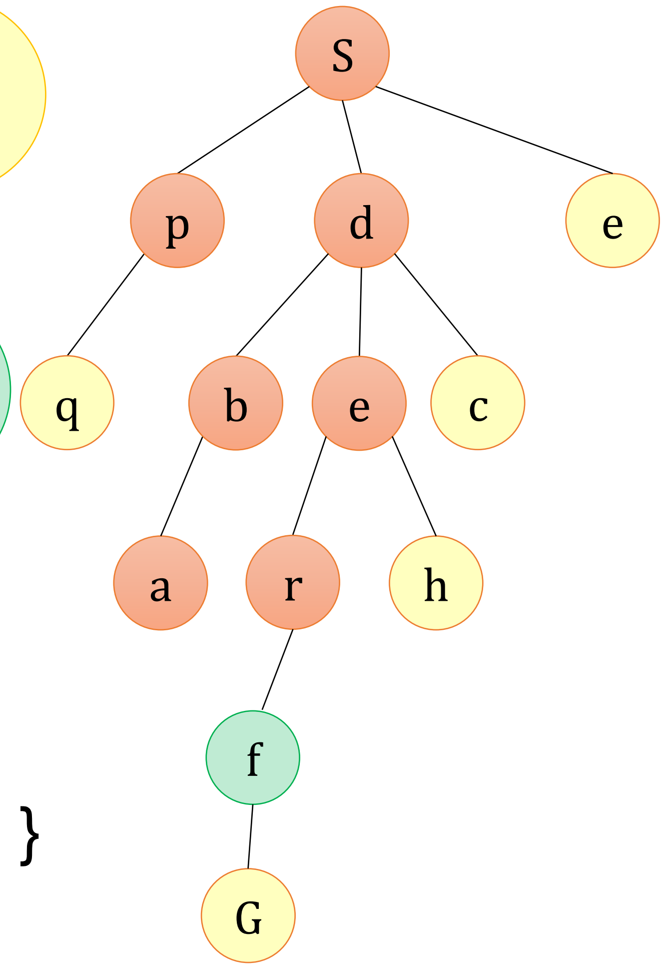
Search Tree

Uniform-cost search: An example



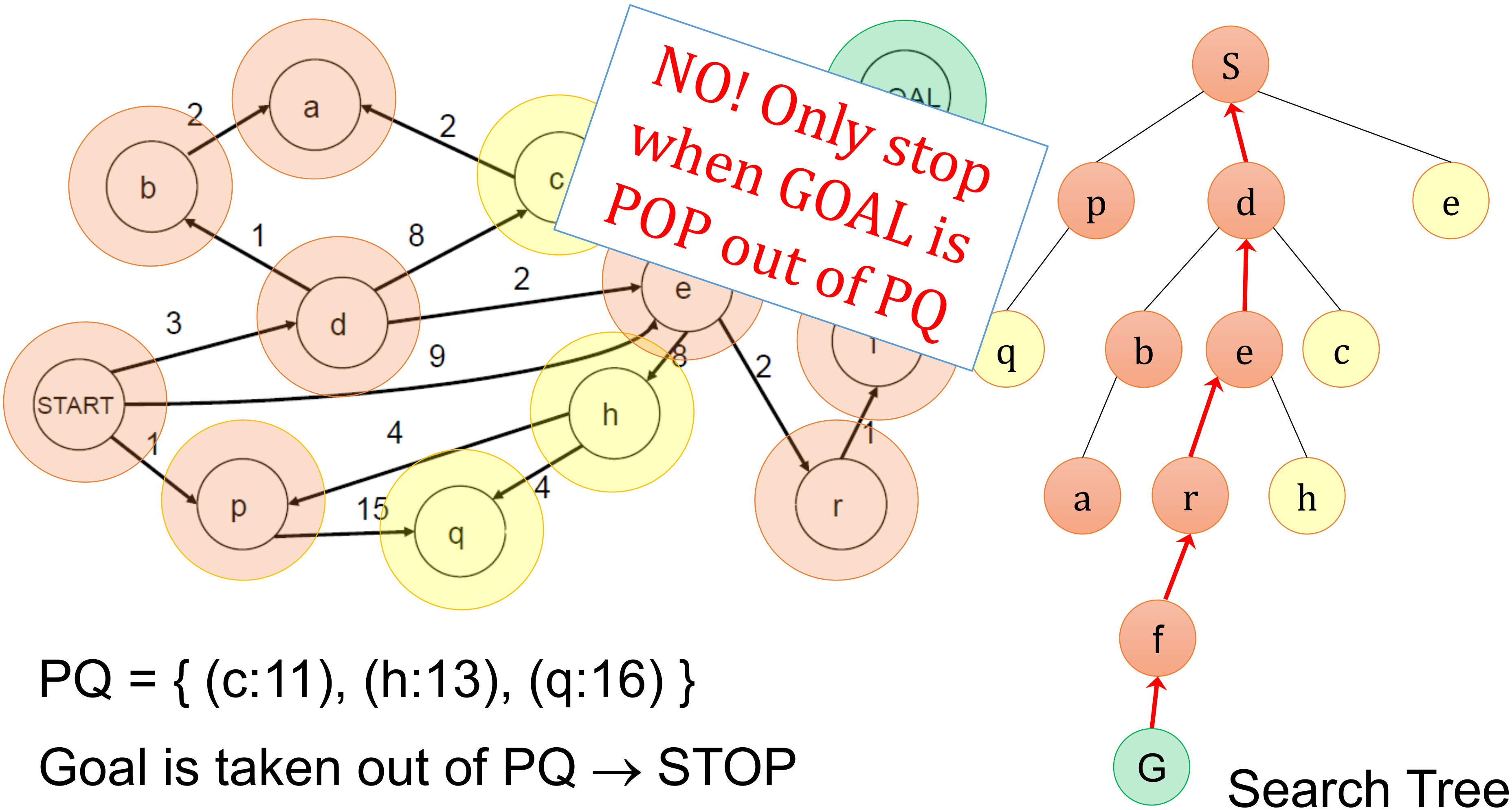
PQ = { (G:10), (c:11), (h:13), (q:16) }

Not update path cost of *c*



Search Tree

Uniform-cost search: An example

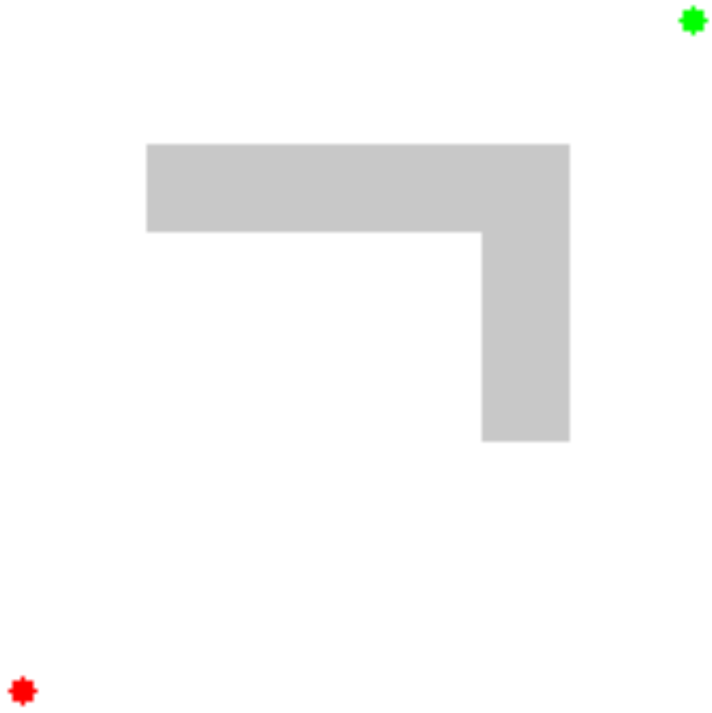


PQ = { (c:11), (h:13), (q:16) }

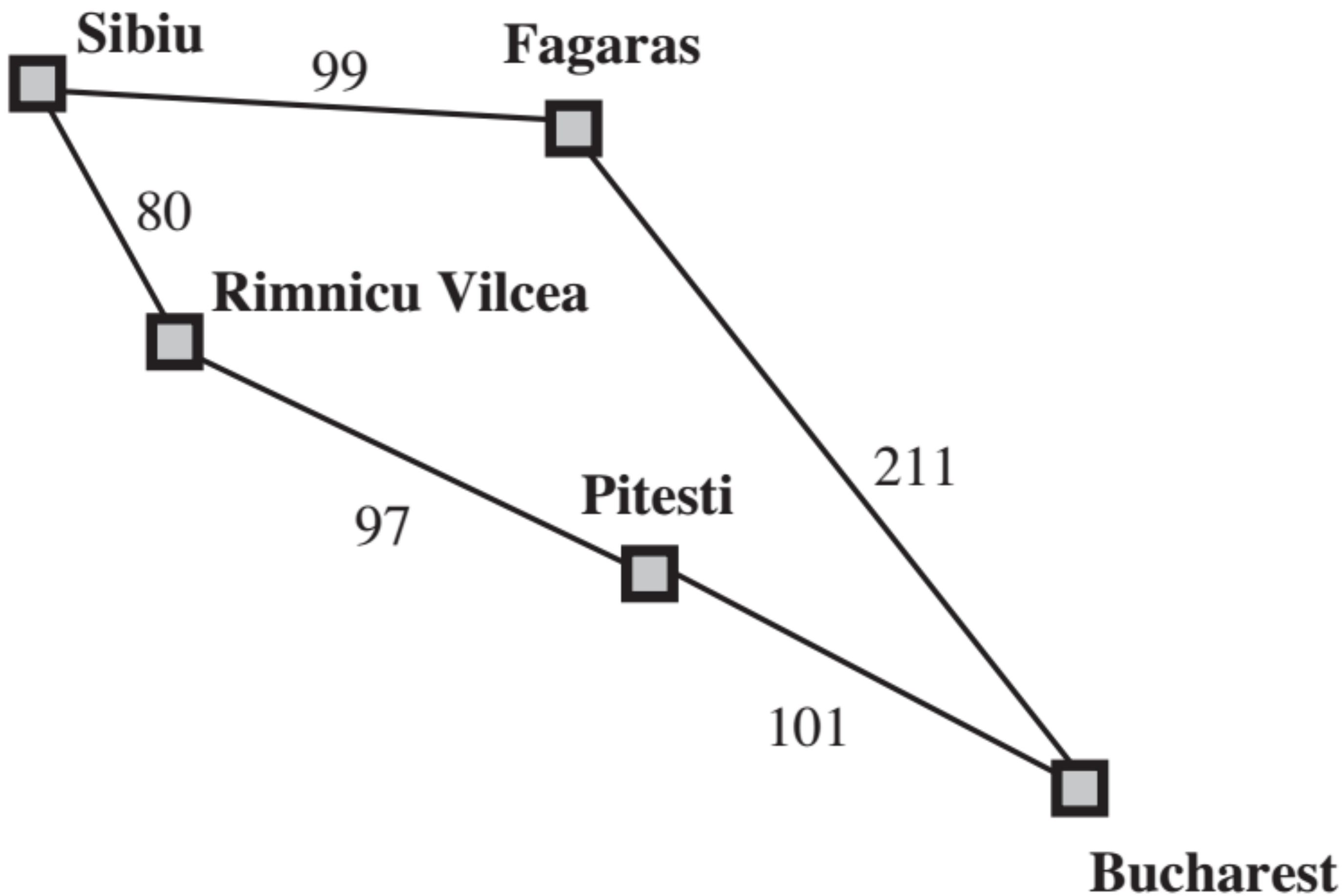
Goal is taken out of PQ → STOP

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

Uniform-cost search: An example



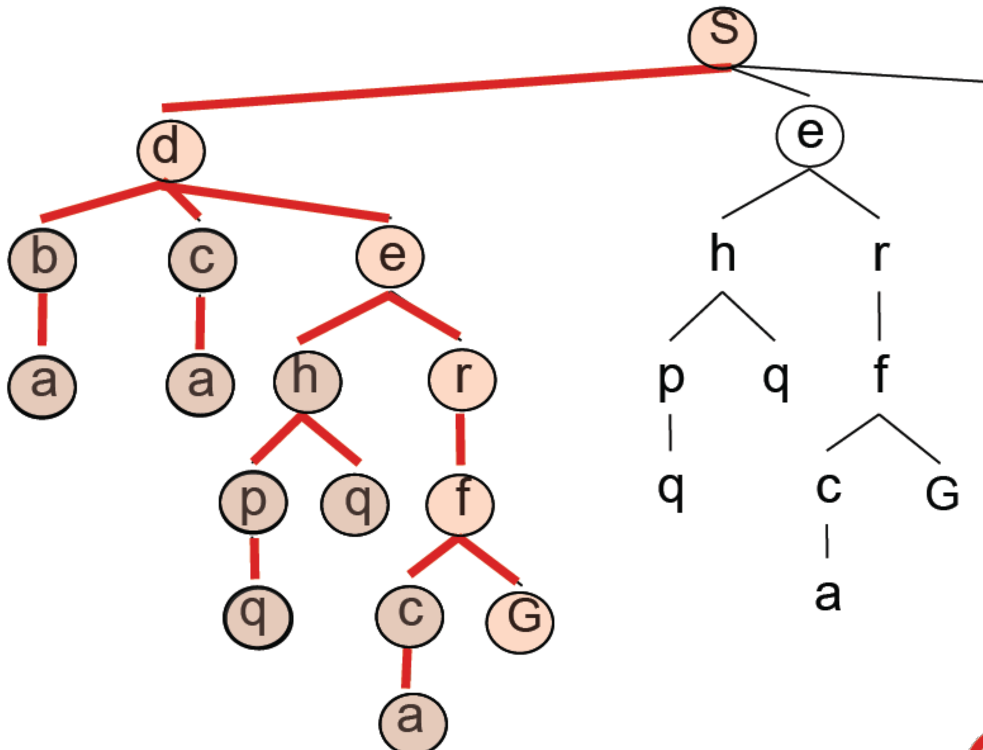
Uniform-cost search: Suboptimal path



Depth-First Search

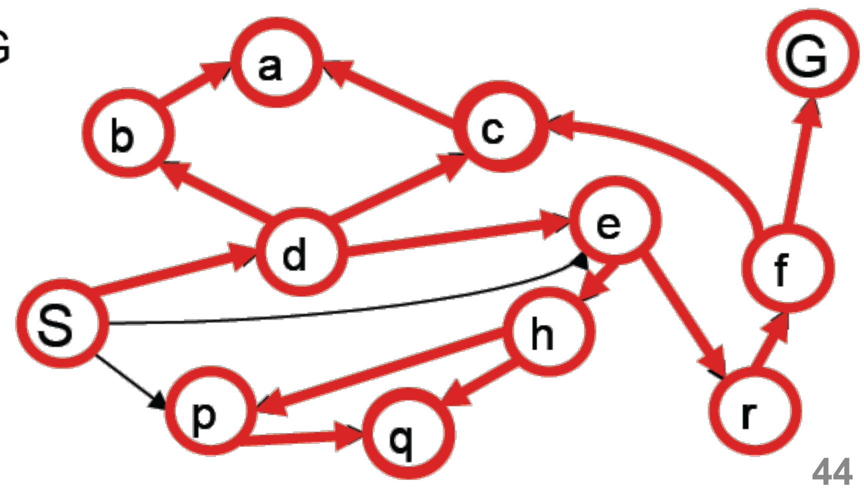
Depth-first search (DFS)

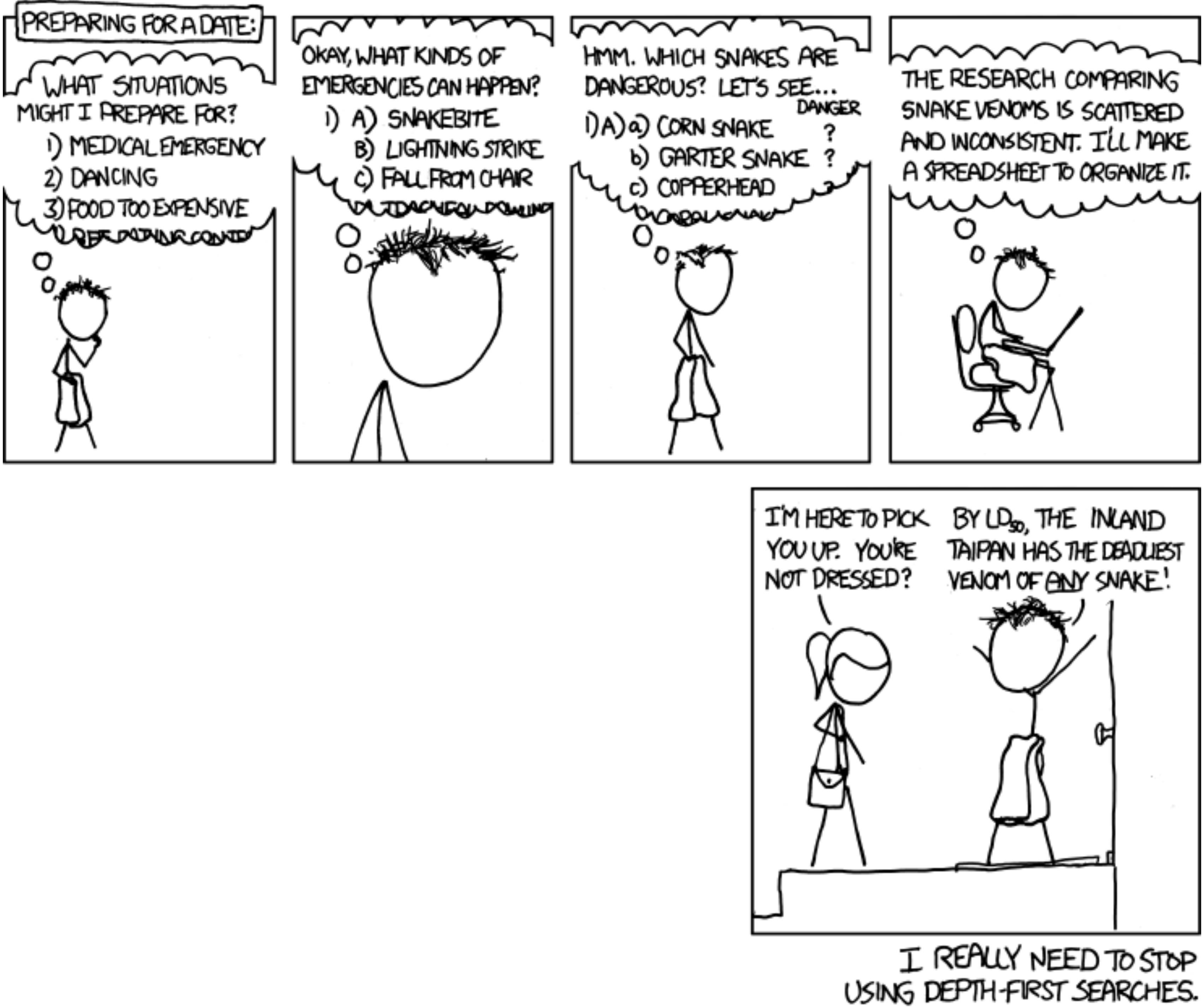
- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO Stack



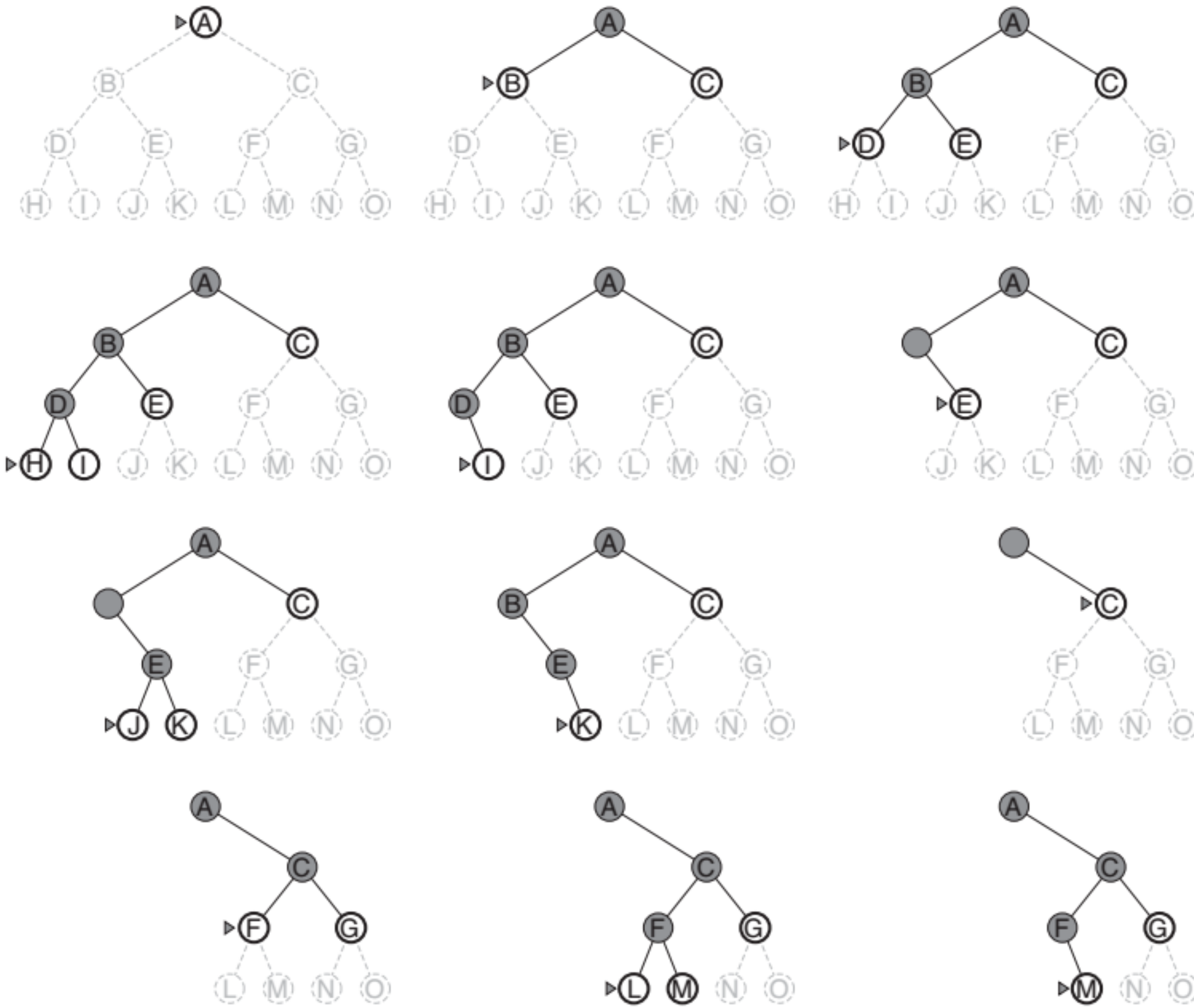
Expansion order:

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




PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR?
1) MEDICAL EMERGENCY
2) DANCING
3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN?
1) A) SNAKEBITE
B) LIGHTNING STRIKE
C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE...
DANGER
1) A) a) CORN SNAKE ?
b) GARTER SNAKE ?
c) COPPERHEAD ?
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD_{50}, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

Depth-first search: An example



Depth-Limited Search

Depth-limited Search (DLS)

- Standard DFS with a predetermined depth limit l, i.e., 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

Iterative Deepening Search

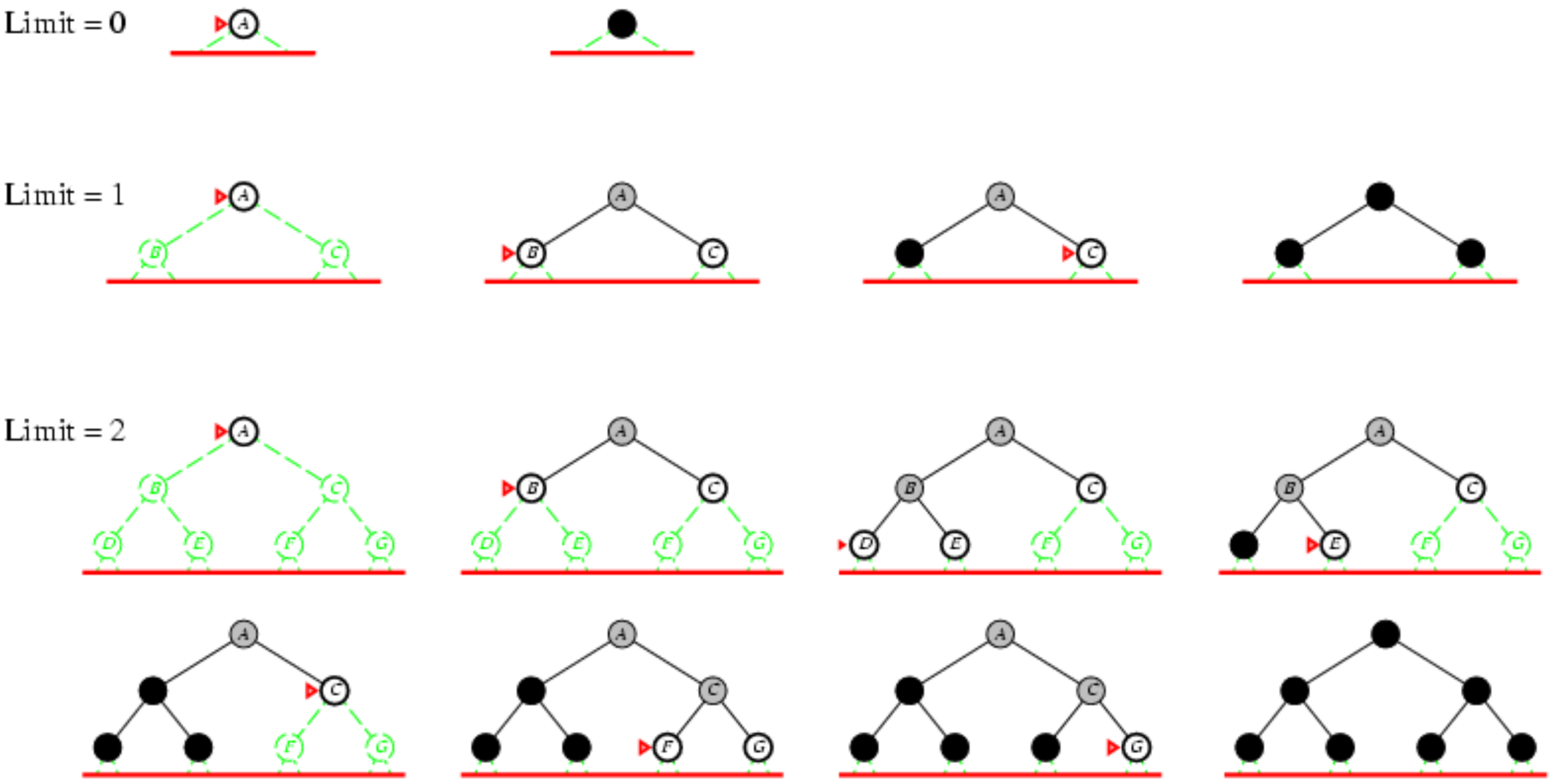
Iterative deepening search (IDS)

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

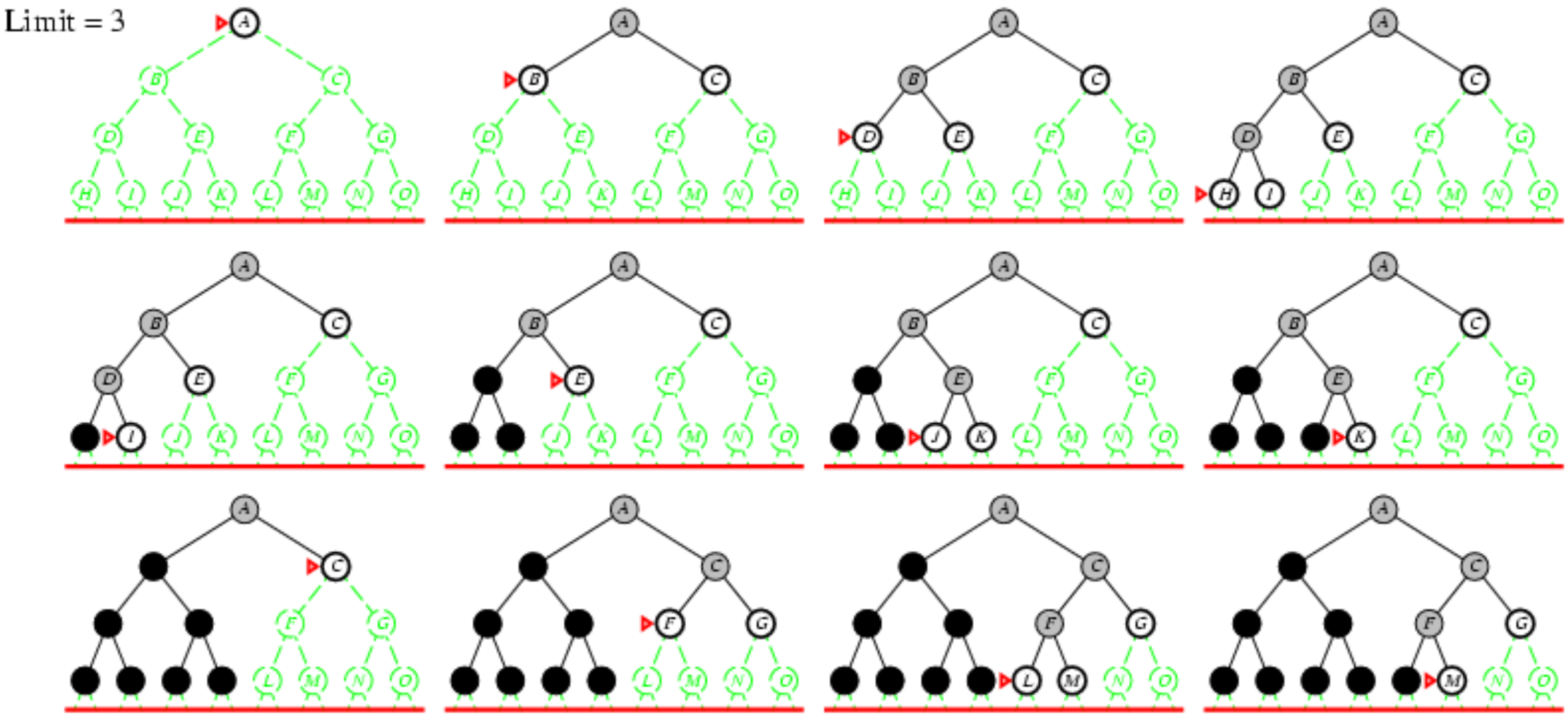
```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution, or failure
   for depth = 0 to ∞ do
      result ← DEPTH-LIMITED-SEARCH(problem, depth)
   if result ≠ cutoff then return result
```

- 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)



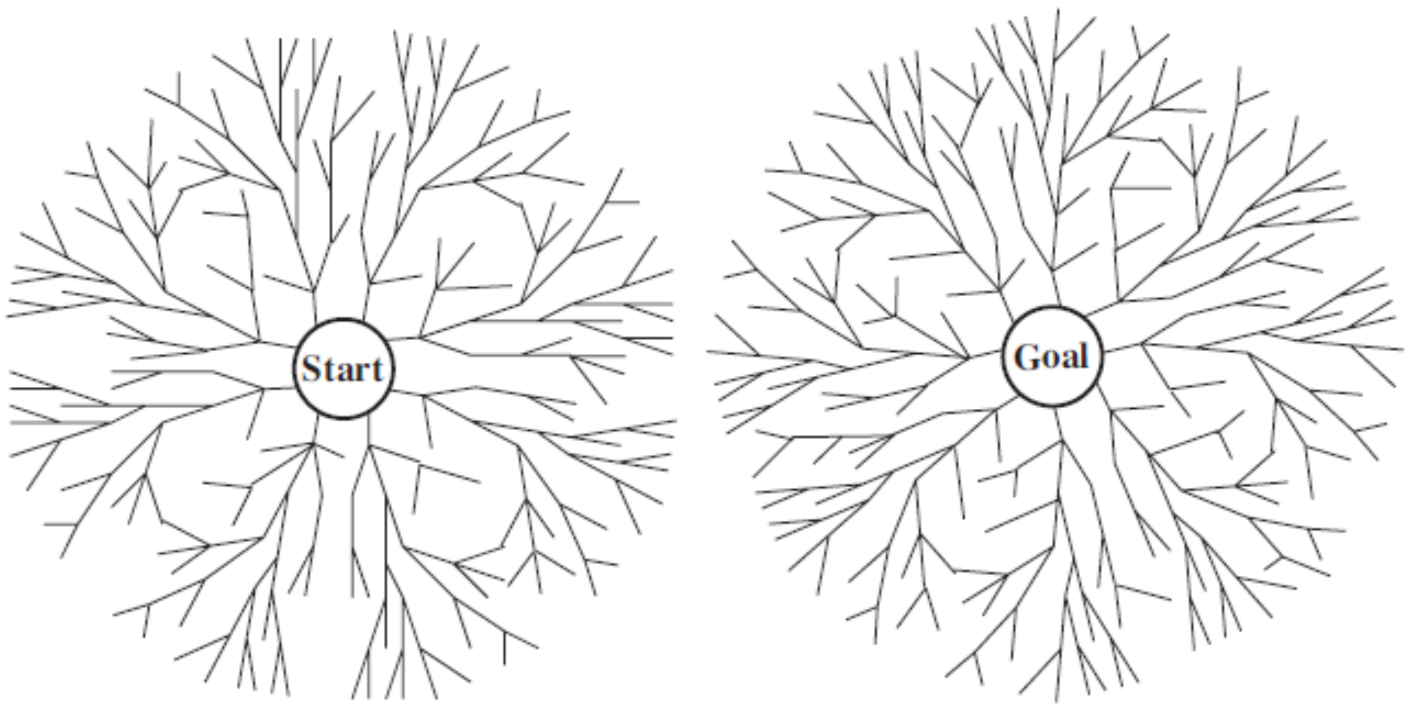
Iterative deepening search (IDS)



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



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.