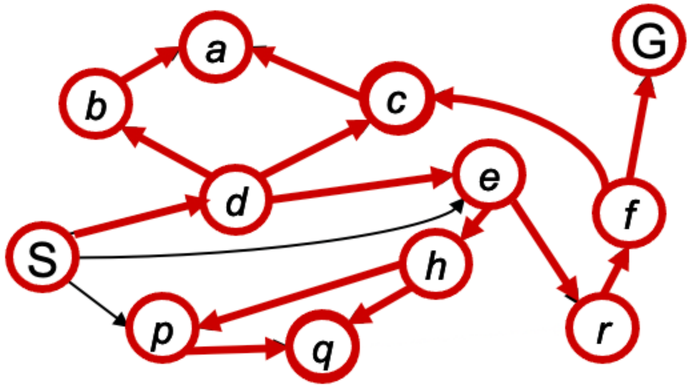
Depth-First-Search

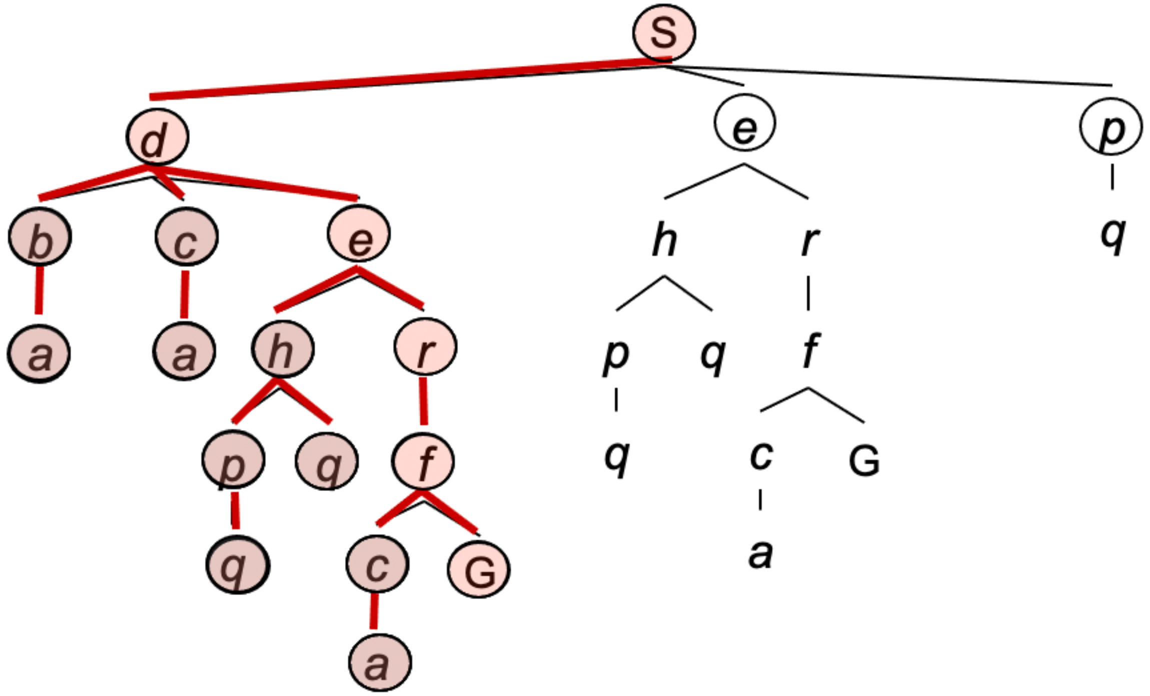
1. Put the start node *s* on OPEN
2. If OPEN is empty exit with failure.
3. Remove the first node *n* from OPEN and place it on CLOSED.
4. If *n* is a goal node, exit successfully with the solution obtained by tracing back pointers from *n* to *s*.
5. Otherwise, <u>expand *n*</u>, generating all its successors attach to them pointers back to *n*, and put them at the top of OPEN *in some order*.
6. Go to step 2.

Depth-First Search

Strategy: exp
a deepest no
first

Implementati
Fringe is a LIF
stack





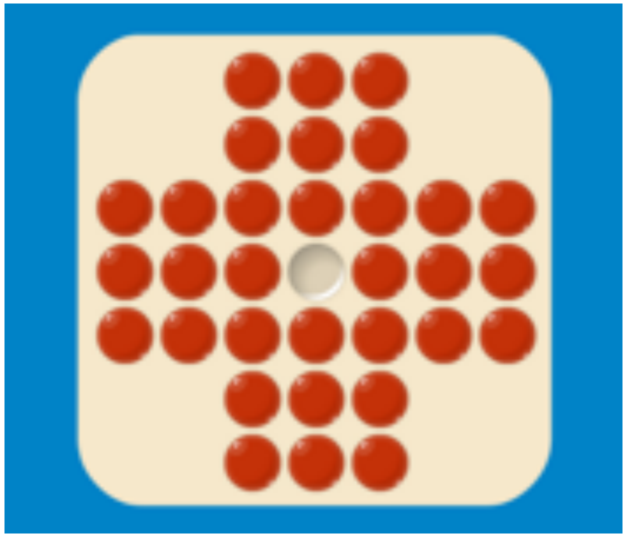
Depth-First-Search (tree search version)

Recursive version of DFS:

```
State DepthFirstSearch(node) {
  if (goalTest(node)) return node;
for each n in successors(node, operators)
   {
     result = DepthFirstSearch(n);
     if (result != FAIL) return result;
   }
return FAIL;
}
```

Example 1

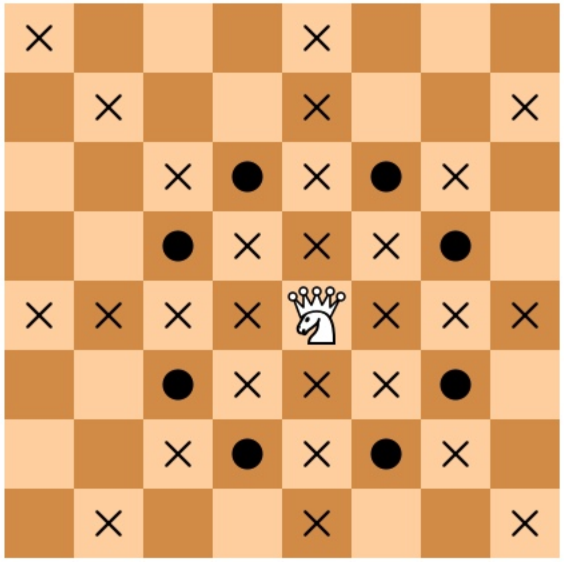
- peg solitaire



Online game: https://webgamesonline.com/peg-solitaire/

Example 2

- placement of mutually non-attacking queens (amazons) in a checker-board. Amazon in row 5 and column 5 attacks all squares marked with x or O



- Maximum of N ar on an N x N board. For what N can we actuall

Example 3

Our final toy problem was devised by Donald Knuth (1964) and illustrates how infinite state spaces can arise. Knuth conjectured that, starting with the number 4, a sequence of factorial, square root, and floor operations will reach any desired positive integer. For example, we can reach 5 from 4 as follows:

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5\,.$$

The problem definition is very simple:

- **States**: Positive numbers.
- **Initial state**: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- **Transition model**: As given by the mathematical definitions of the operations.
- **Goal test**: State is the desired positive integer.

To our knowledge there is no bound on how large a number might be constructed in the process of reaching a given target—for example, the number 620,448,401,733,239,439,360,000 is generated in the expression for 5—so the state space for this problem is infinite. Such state spaces arise frequently in tasks involving the generation of mathematical expressions, circuits, proofs, programs, and other recursively defined objects.

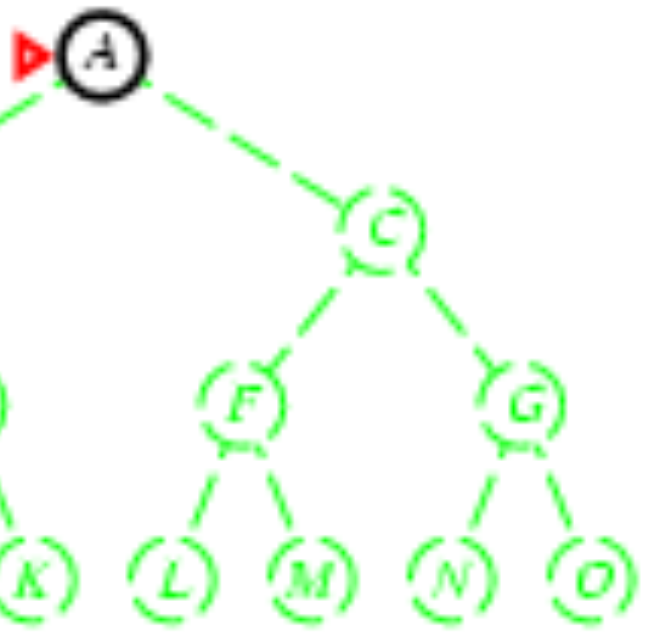
Graph structure of state space graph

- The graph could be finite or infinite
- The graph could be directed or undirected
- The graph could be cyclic or acyclic
- The graph could be a tree (or not)

- Irrespective of the structure, the search algorithms fall into tree search or graph search version.

Depth-first search

Is A a goal state?

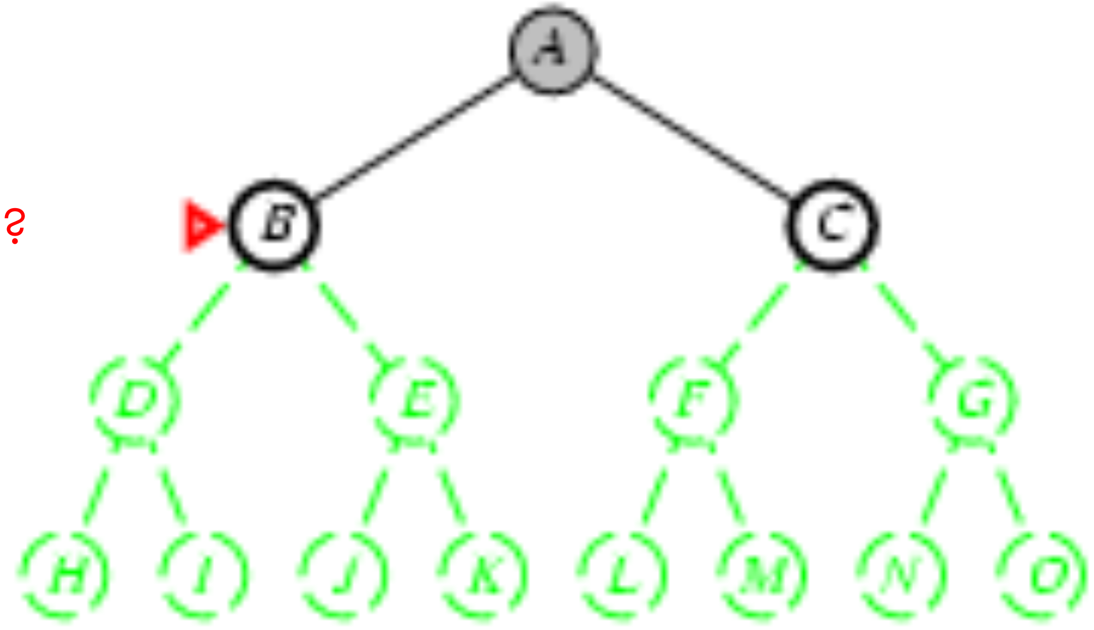


Depth-first search

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

queue=[B,C]

Is B a goal state?

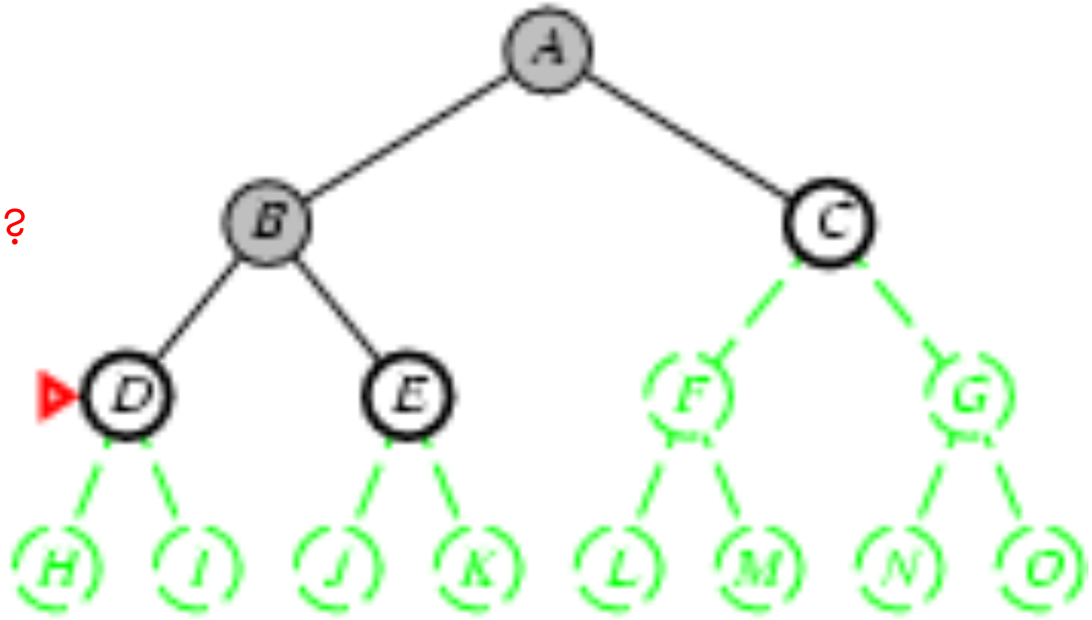


Depth-first search

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

queue=[D,E,C]

Is D = goal state?

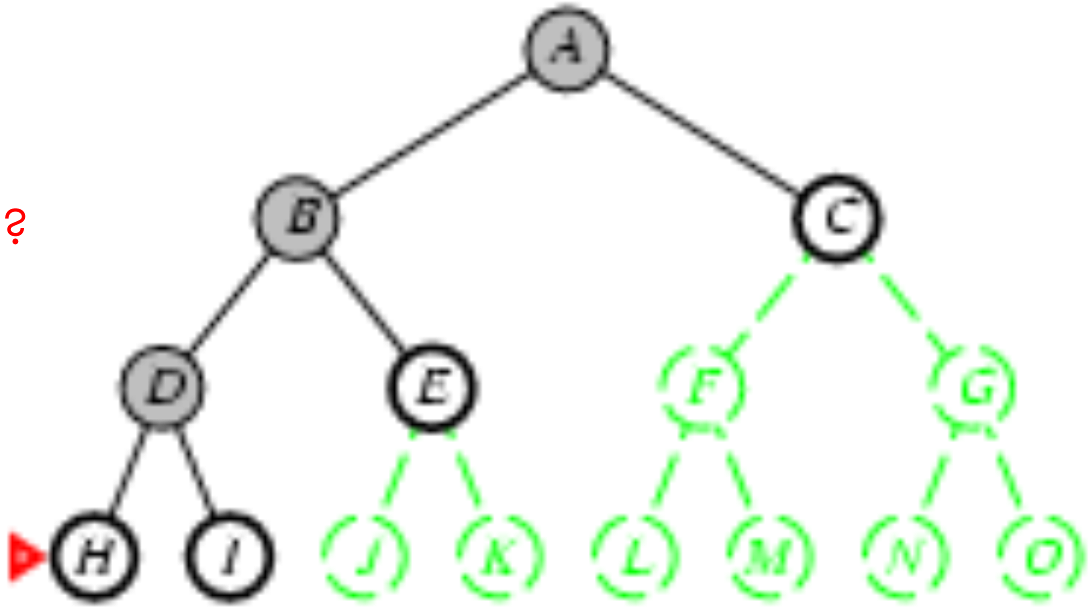


Depth-first search

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

queue=[H,I,E,C]

Is H = goal state?

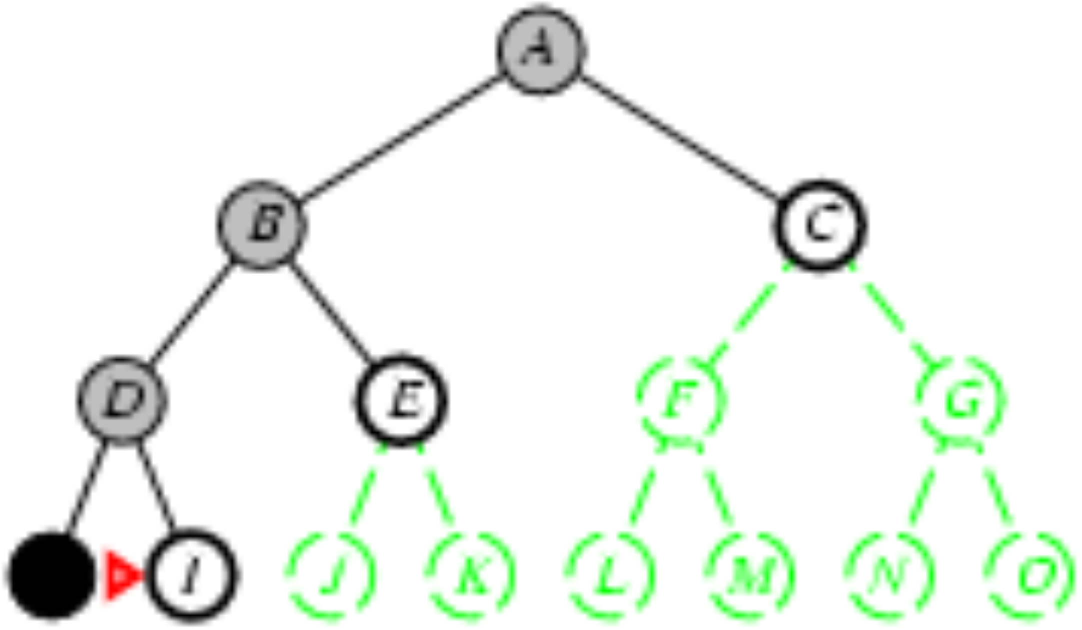


Depth-first search

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

queue=[I,E,C]

Is I = goal state?

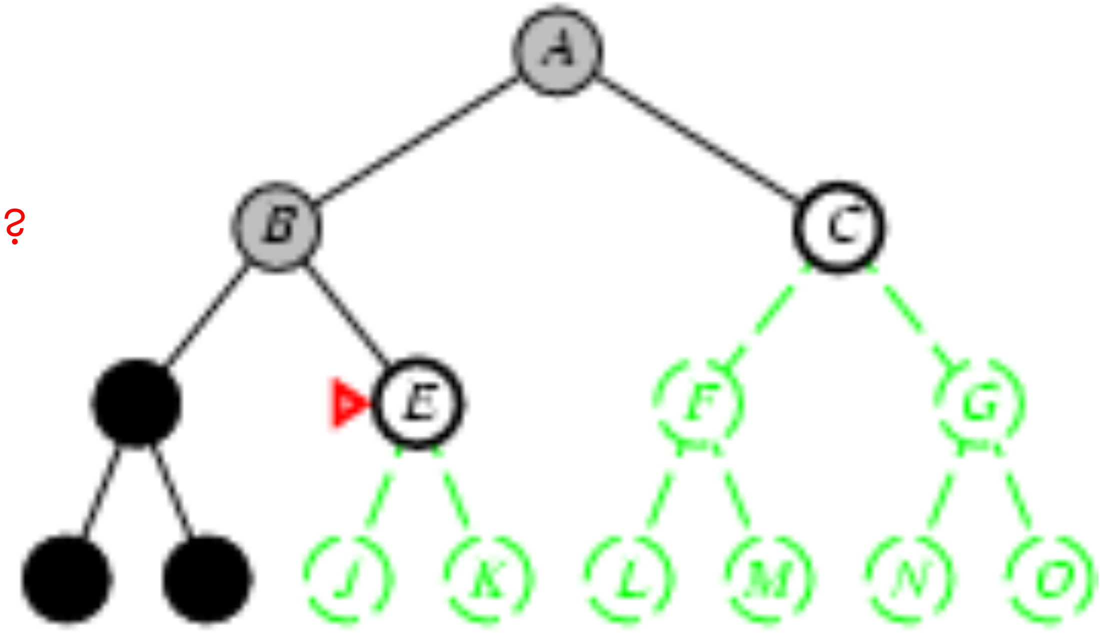


Depth-first search

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

queue=[E,C]

Is E = goal state?

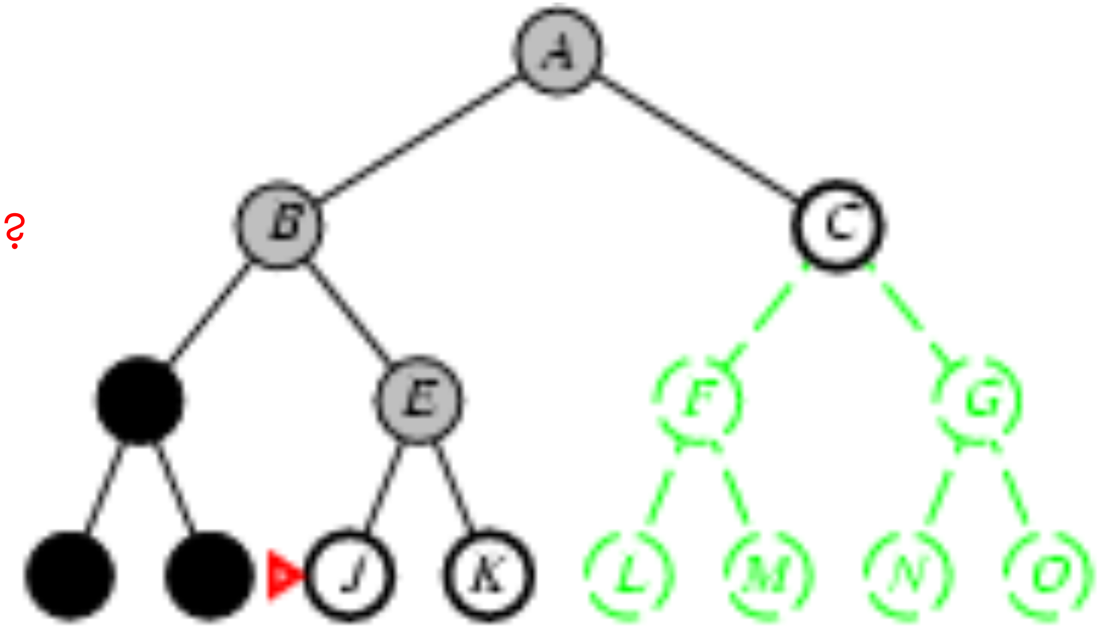


Depth-first search

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

queue=[J,K,C]

Is J = goal state?

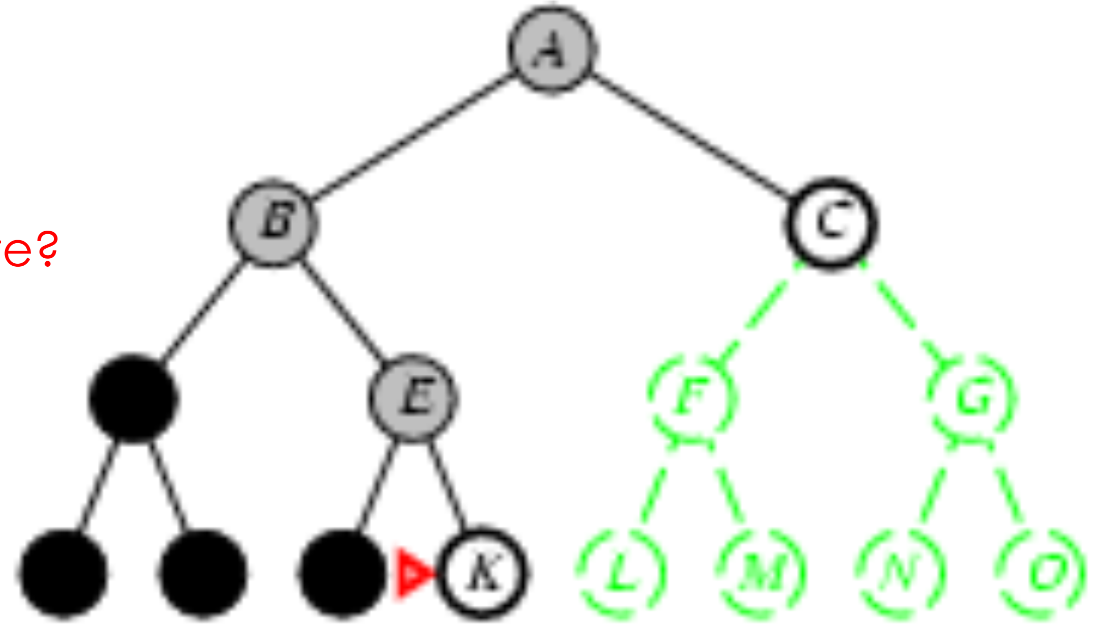


Depth-first search

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

queue=[K,C]

Is K = goal state?

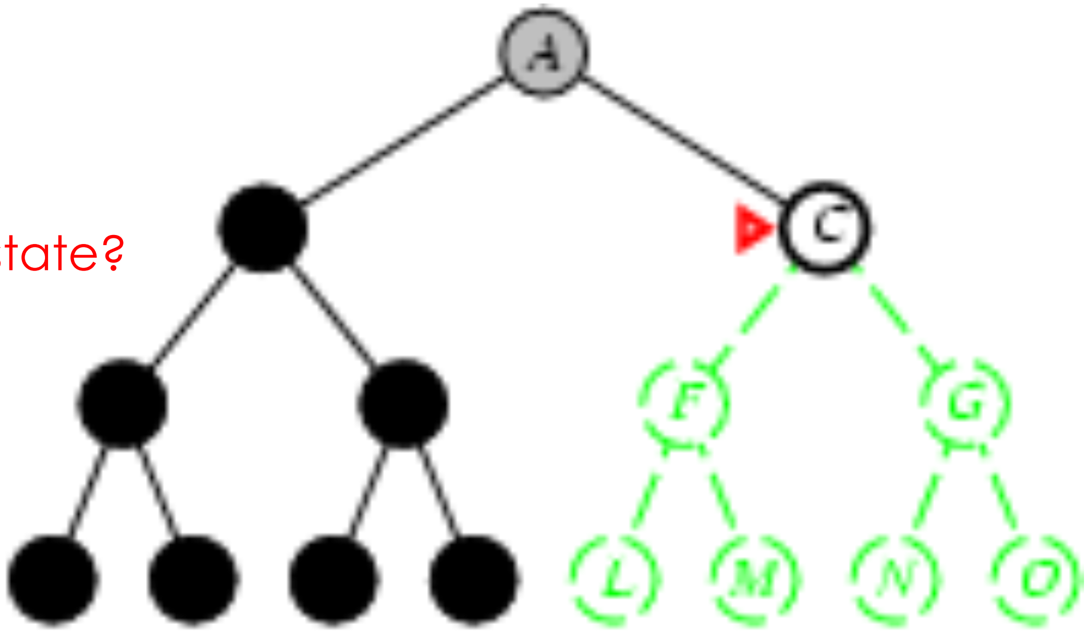


Depth-first search

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

queue=[C]

Is C = goal state?

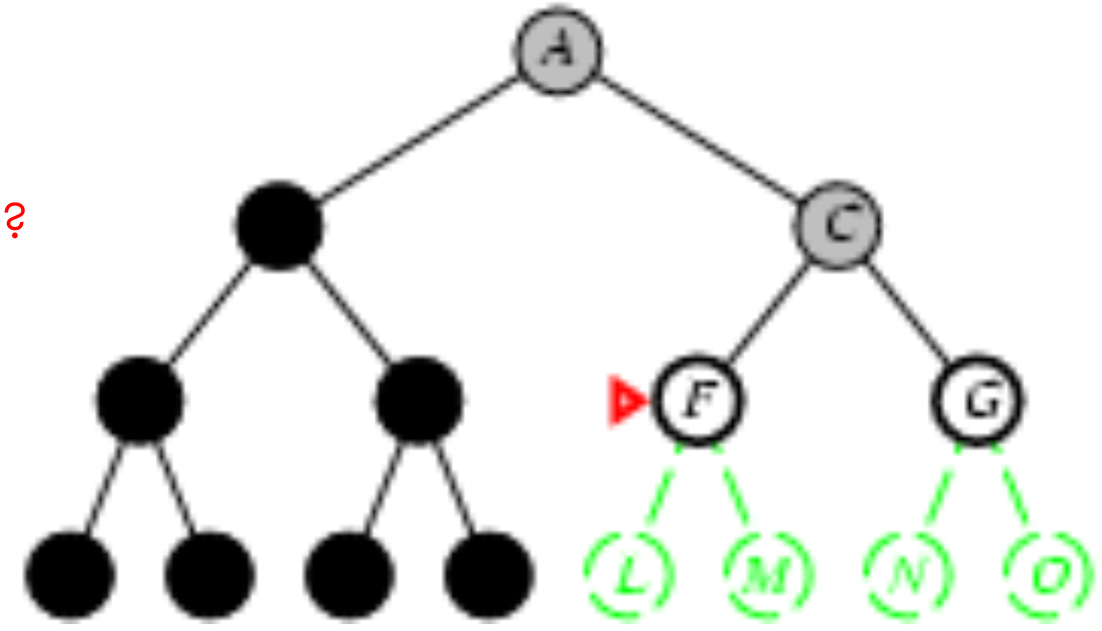


Depth-first search

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

queue=[F,G]

Is F = goal state?

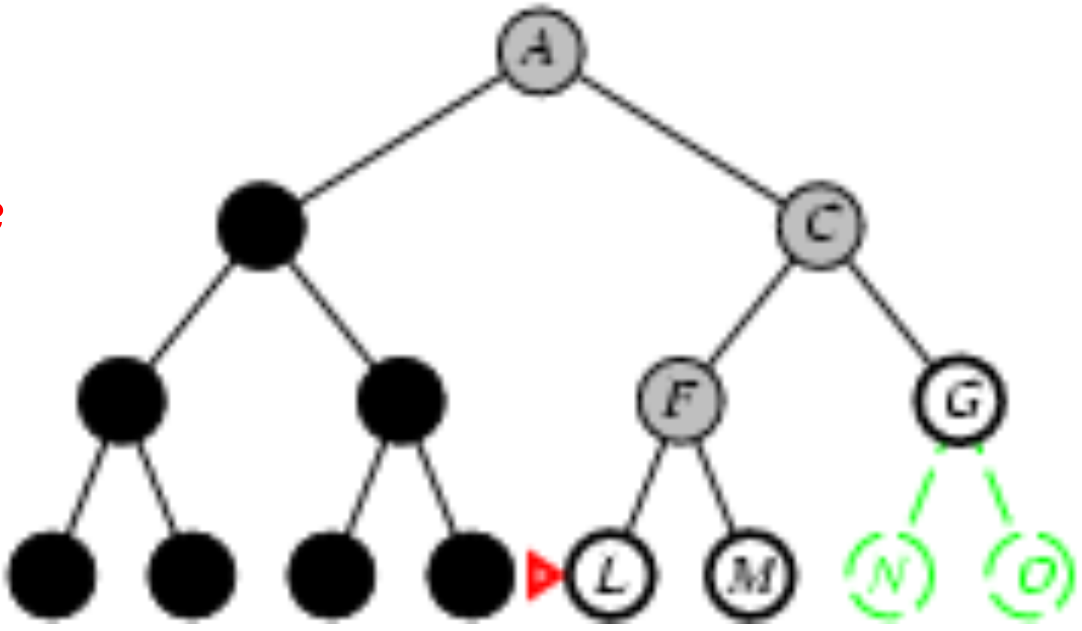


Depth-first search

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

queue=[L,M,G]

Is L = goal state?

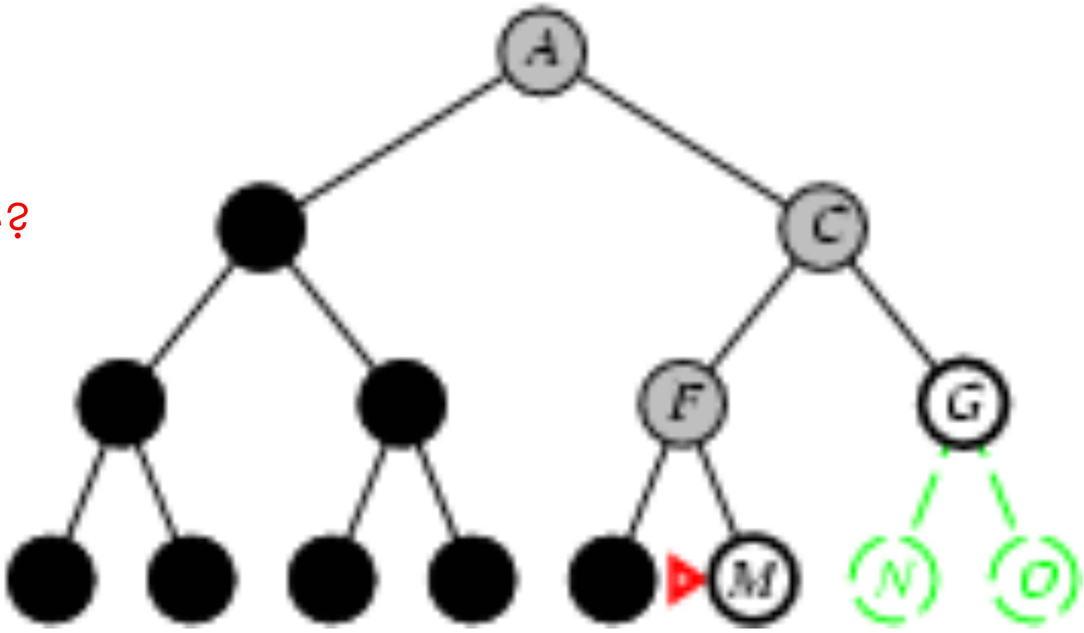


Depth-first search

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

queue=[M,G]

Is M = goal state?



Example DFS

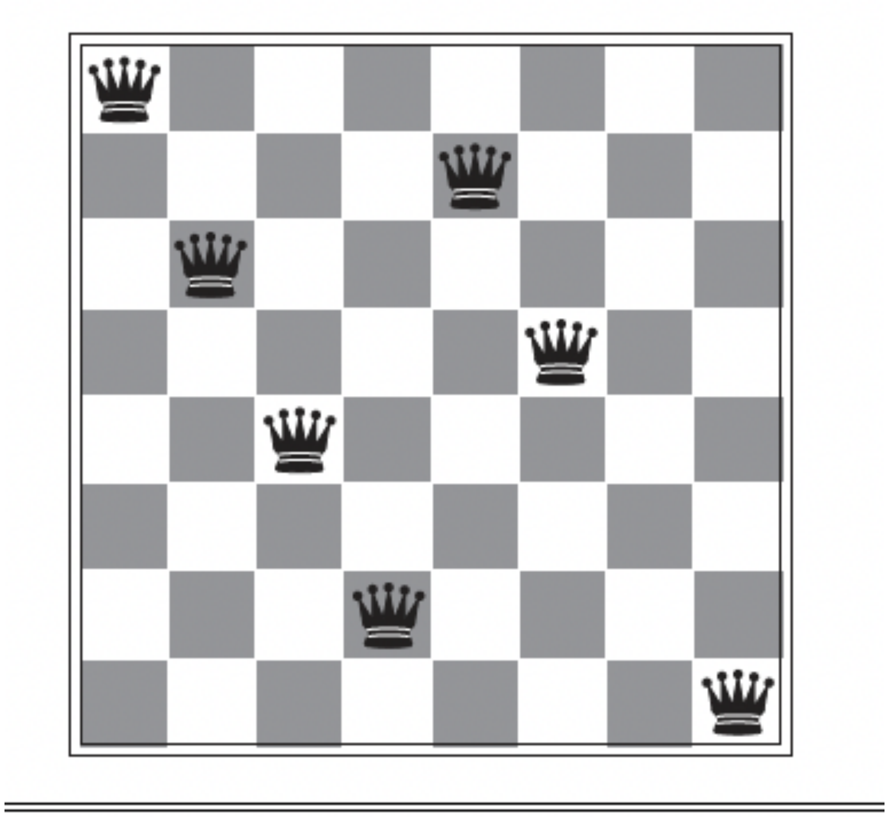
Project 1 – placing amazons

```
def amazonDFS(B, n):
    # B is a vector such that B[i] = the col number in
which the amazon is placed in row i
    # n is the dimension of the board for which a solution
is sought
    # pre-condition: B is a valid partial solution
    if len(B) == n:   # solved
        print(B)
        return True
    for j in range(n):
        # looking for a candidate for B[k] which will be an
integer from 0 to n-1
        if not(attack(B, n, j)):
            B.append(j)
            if amazonDFS(B, n):
                return True
            else:
                B.remove(j)
    print(B)
    return False
```

Project 1 – placing max no of amazons

```
function PlaceAmazons(Board, k, n):
# return Board by adding as many more amazons as possible
# Board contains amazons in some of the first k rows
# For empty Board set k = -1
Best = Board
for each square (r,s) in Rows k+1 to n:
   if (r,s) is not under attack in Board:
       add an amazon in (r,s) in Board
       out = PlaceAmazons(Board,r+1,n)
       if size(out) > size(Best):
           Best = out
return Best
```

A solution to 8 queens problem

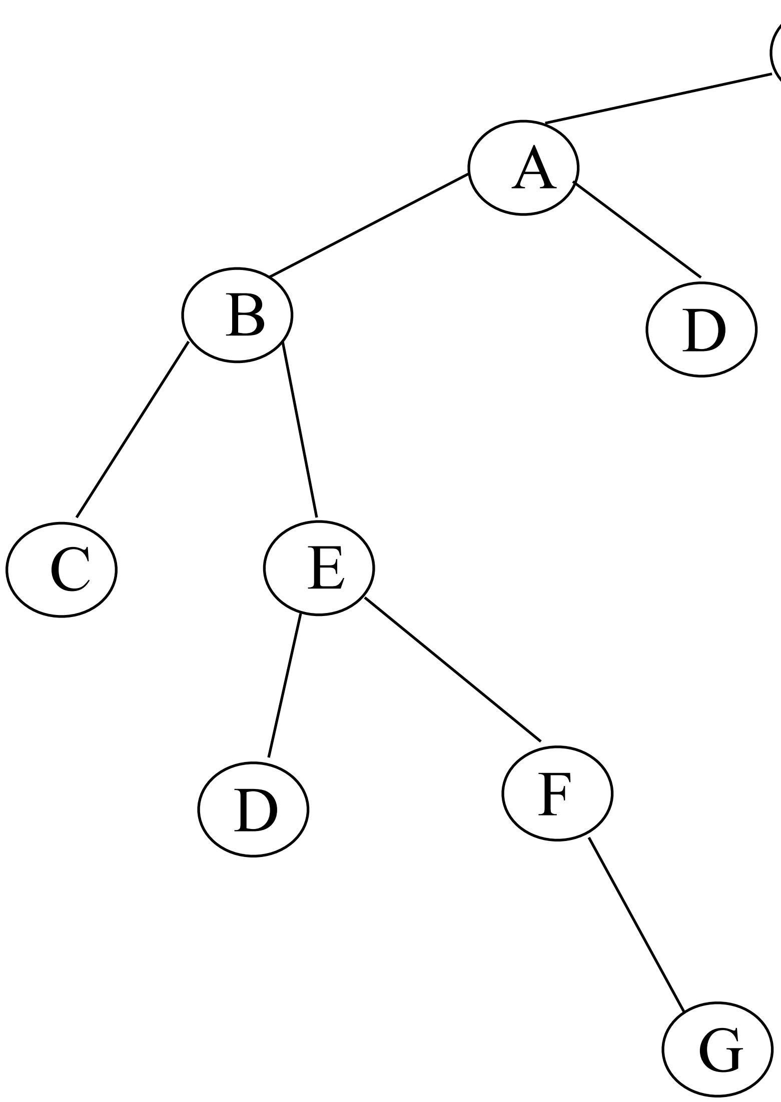


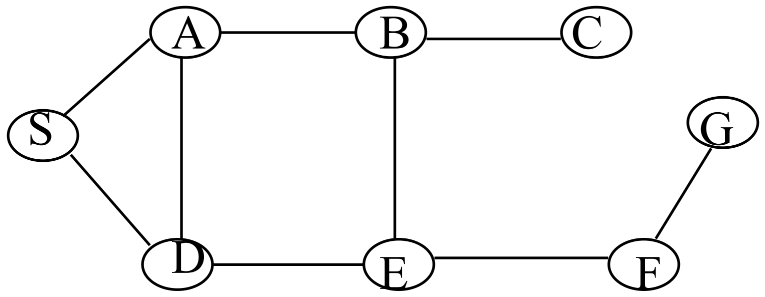
Tree Search vs. Graph Search (of BFS, DFS)

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      expand the chosen node, adding the resulting nodes to the frontier
```

```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      add the node to the explored set
      expand the chosen node, adding the resulting nodes to the frontier
          only if not in the frontier or explored set
```

Depth First Search tree – Graph search version





Here, to avoid repeated states assume we don't expand any child node which appears already in the path from the root S to the parent.

3.3.2 Measuring problem-solving performance

Before we get into the design of specific search algorithms, we need to consider the criteria that might be used to choose among them. We can evaluate an algorithm's performance in four ways:

- **Completeness**: Is the algorithm guaranteed to find a solution when there is one?
- **Optimality**: Does the strategy find the optimal solution, as defined on page 68?
- **Time complexity**: How long does it take to find a solution?
- **Space complexity**: How much memory is needed to perform the search?

What is the Complexity of Depth-First Search?

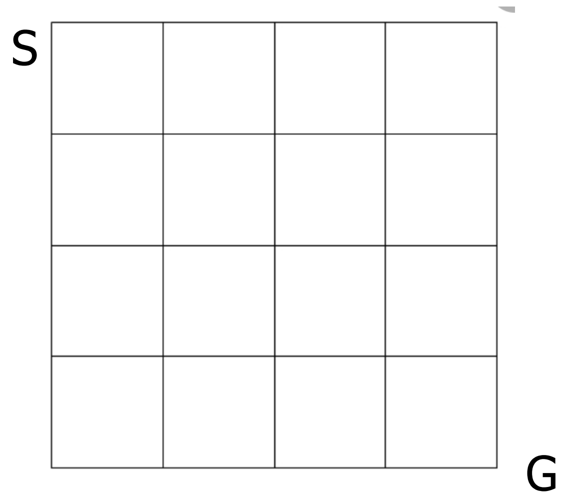
- Time Complexity
  - assume (worst case) that there is 1 goal leaf at the RHS
  - DFS will expand all nodes

    $= 1 + b + b^2 + \dots + b^d$

    $= O(\mathbf{b^d})$
- Space Complexity (iterative version)
  - how many nodes can be in the stack (worst-case)?
  - at depth $l < d$ we have $b - 1$ nodes
  - at depth d we have b nodes
  - total $= (d-1)*(b-1) + b =$ **$O(bd)$**
- In recursive version, the actual nodes being saved in recursive call stack so implicitly $O(bd)$ nodes are kept.

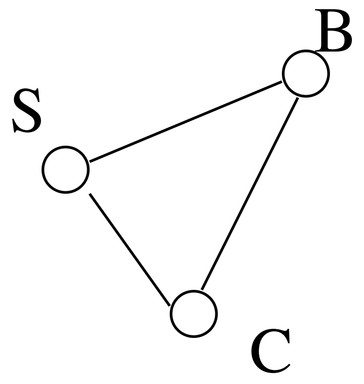
Repeated states

- Failure to detect repeated states can make the search tree much larger than N = size of the size space. (Also true in the case of BFS).

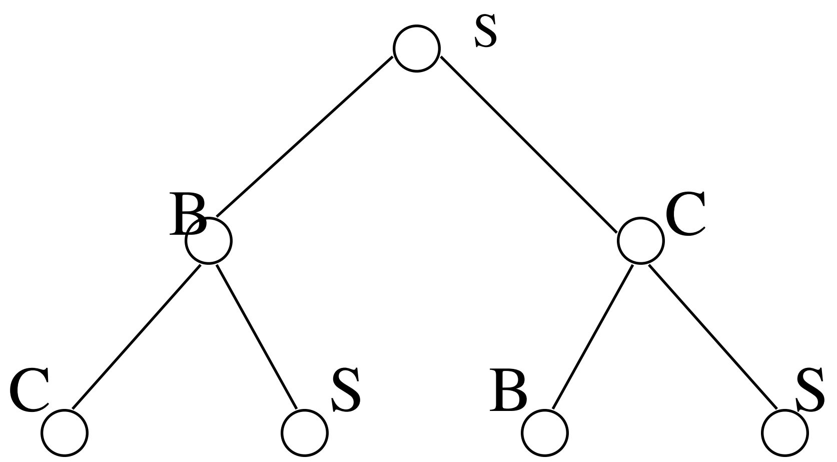


Question: How many times will a node appear in the DFS tree?

Solutions to repeated states



State Space



Example of a Search Tree

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

- Method 1 is most practical, works well on most problems

Properties of depth-first search

- Complete? No.
  - If state space is not finite, the search may never terminate even if solution 2 is used. With finite state-space, both solutions will terminate.
- Time? $O(b^m)$ with m = maximum depth
- terrible if *m* is much larger than *d*
  - but if there are many solutions, DFS can be much faster than BFS.
- Space? $O(bm)$, i.e., linear space! (we only need to remember a single path + expanded unexplored nodes)
- Optimal? No (It may find a non-optimal goal first)