How to effectively formulate a search problem

state:

- all the information need to find a solution
- nothing but this information

useful tips:

- search state is not world state!
- search state does not need to be of the same length!

efficiency check:

- try to define states and actions in a way that decrease the size of the state space as much as possible.

SEARCH ALGORITHMS

search algorithm: takes a search problem as input and returns a solution.

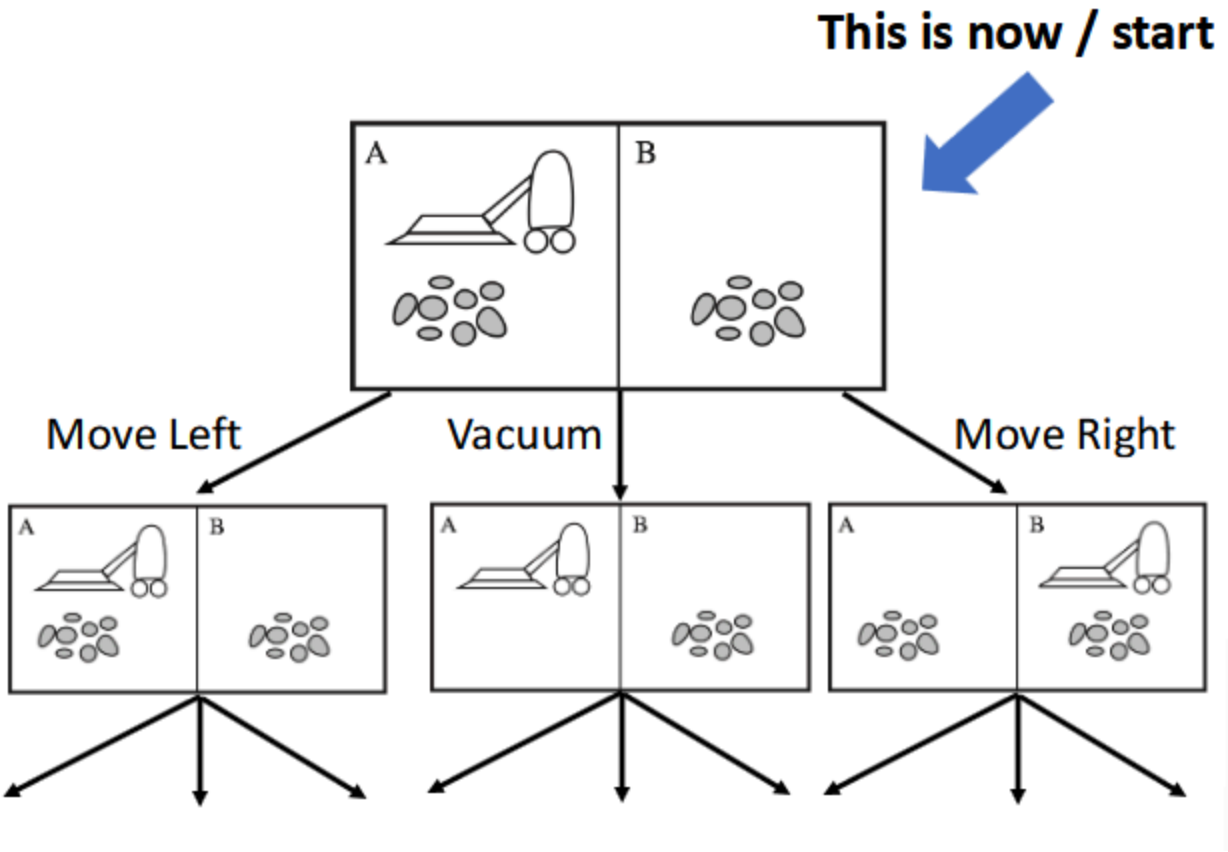
state space graph: a mathematical representation of a search problem.

- Nodes represent world configurations
- Edges represent successors (results of actions)
- Goal is to reach the goal node/s
- In a state space graph, each state occurs only once.

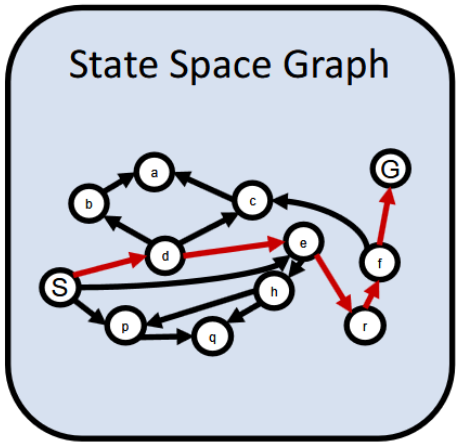
Search tree

- A "what if" tree of action and their outcomes
- root node = initial state
- Children correspond to successors
- Nodes show states
- Branches correspond to actions that lead to those states

Example:

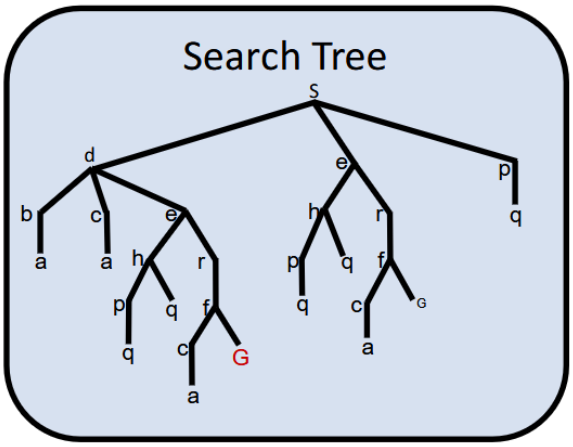


STATE SPACE GRAPHS VS SEARCH TREES



Each NODE in the search tree is reached by a PATH in the state space graph.

Algorithms construct both on demand – and they construct as little as possible.



SEARCHING WITH A SEARCH TREE

- Expand out potential plans (tree nodes)
- Maintain a **frontier (fringe -> set of nodes that have been reached but not expanded)** of partial plans under consideration
- Try to expand as few tree nodes as possible.

PSEUDOCODE FOR TREE SEARCH ALGORITHMS

```
Function TreeSearch(problem) returns a solution or failure
    frontier <- {initial_state}
    repeat
        if frontier empty then return failure
        node <- POP(frontier)
        if isGoal(node) then return corresponding solution
        for a in Actions(node):
            s' = Result(node, a)
            INSERT(frontier,s')
```

HOW TO EVALUATE A SEARCH ALGORITHM

- Completeness - Does it always find a solution? (if one exists)
- Optimality - Does it find the best solution? (Lowest path cost)
- Time complexity - How many nodes are processed to find a solution?
- Space complexity - How many nodes need to be stored in the frontier?

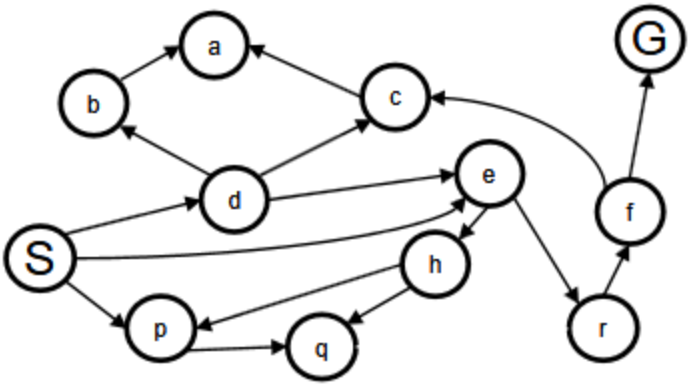
UNIFORMED SEARCH STRATEGIES

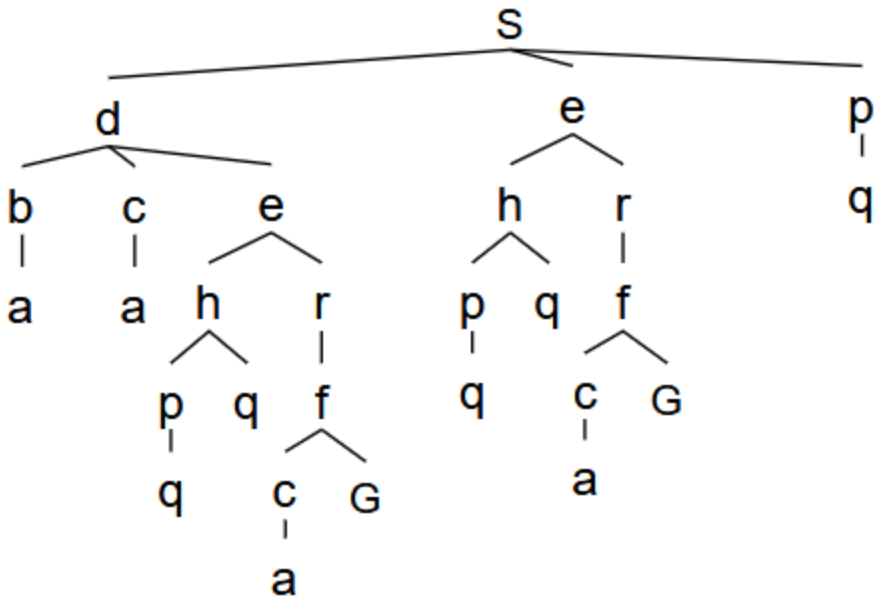
Uniformed search: The search algorithm has no additional information about states beyond what has been provided in the problem formulation.

DEPTH FIRST SEARCH

strategy: expand the deepest node first
depth: number of actions in a path (node)

implementation: frontier is a LIFO stack

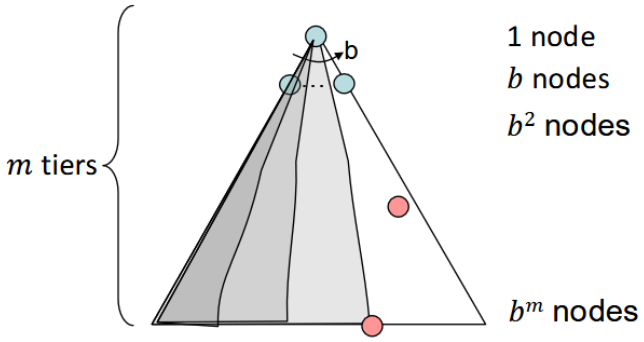




Frontier					Expand
	(S, 0)				(S, 0)
(d, 1)	(e, 1)	(p, 1)			(d, 1)
(b, 2)	(c, 2)	(e, 2)	(e, 1)	(p, 1)	(b, 2)
(a, 3)	(c, 2)	(e, 2)	(e, 1)	(p, 1)	(a, 3)
	(c, 2)	(e, 2)	(e, 1)	(p, 1)	(c, 2)
	⋮				⋮

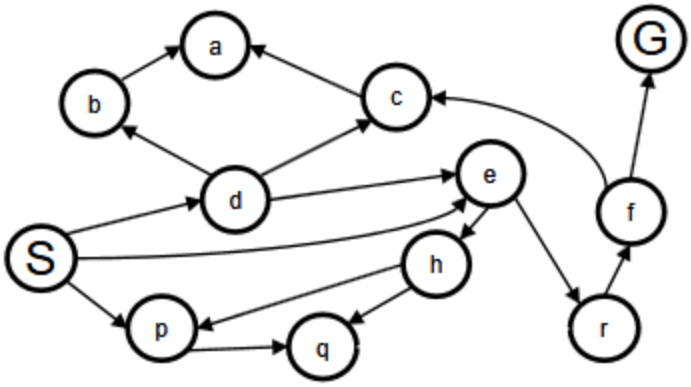
Depth-First Search (DFS) Properties

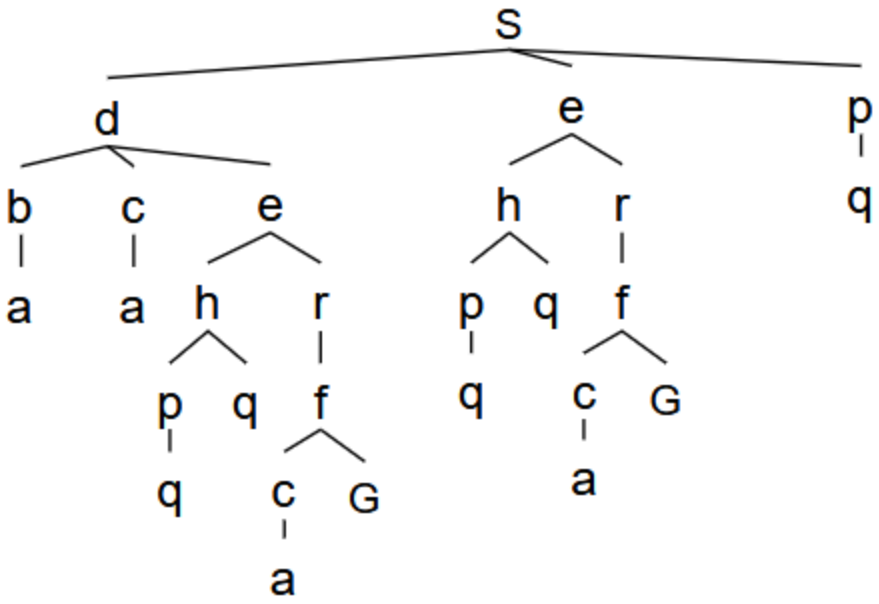
- **Time complexity**
  - DFS could process the whole tree (and $m > d$)
  - If m is finite, takes time $O(b^m)$
- **Space complexity**
  - Only need to store siblings on path to root, so $O(bm)$
- **Is it complete?**
  - m could be infinite → No
- **Is it optimal?**
  - No, it finds the "leftmost" solution, regardless of depth or path cost



Breadth-First Search

strategy: expand the shallowest node first
depth: number of actions in a path(node)
implementation: frontier is a FIFO queue


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


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

Frontier	Expand
(S, 0)	(S, 0)
(d, 1) (e, 1) (p, 1)	(d, 1)
(b, 2) (c, 2) (e, 2) (e, 1) (p, 1)	(e, 1)
(b, 2) (c, 2) (e, 2) (h, 2) (r, 2) (p, 1)	(p, 1)
(b, 2) (c, 2) (e, 2) (h, 2) (r, 2) (q, 2)	(b, 2)
⋮	⋮

Breadth-First Search (BFS) Properties

- **Time complexity**
  - BFS processes all nodes above shallowest solution
  - Let depth of shallowest solution be d
  - Search takes time $O(b^d)$
- **Space complexity**
  - Needs to store roughly the successors of the last tier, so $O(b^d)$
- **Is it complete?**
  - Yes, d must be finite if a solution exists
- **Is it optimal?**
  - Only if costs are all 1