State-Space Formulation

- **Intelligent agents: problem solving as search**

- Search consists of
  - state space
  - operators
  - start state
  - goal states
- The search graph

- **A** Search Tree **is an effective way to represent the search process**

- **There are a variety of search algorithms, including**
  - Depth-First Search
  - Breadth-First Search
  - Others which use heuristic knowledge (in future lectures)

Uninformed search strategies

uninformed (or blind) search:

While searching you have no clue whether one non-goal state is better than any other. Your search is blind. You don’t know if your current exploration is likely to be fruitful.

common blind strategies:

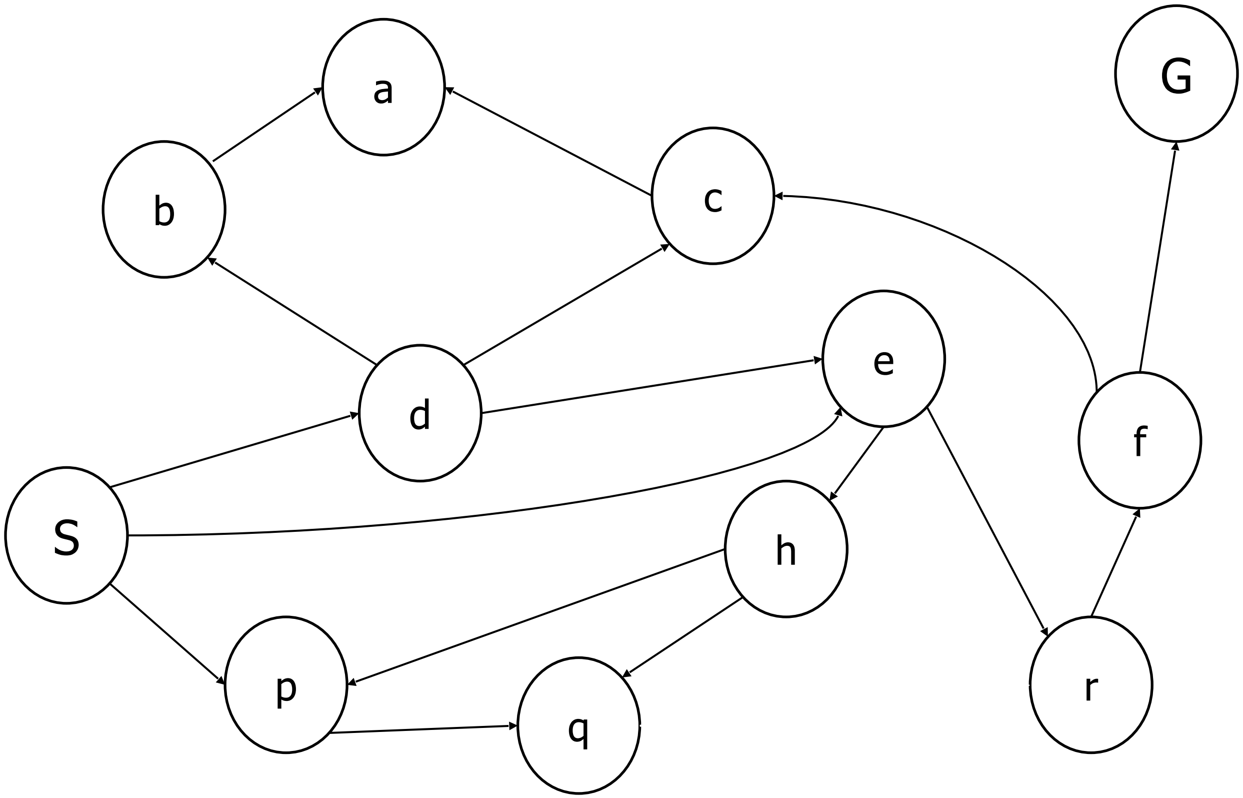
- Breadth-first search
- Uniform-cost search
- Depth-first search
- Iterative deepening search

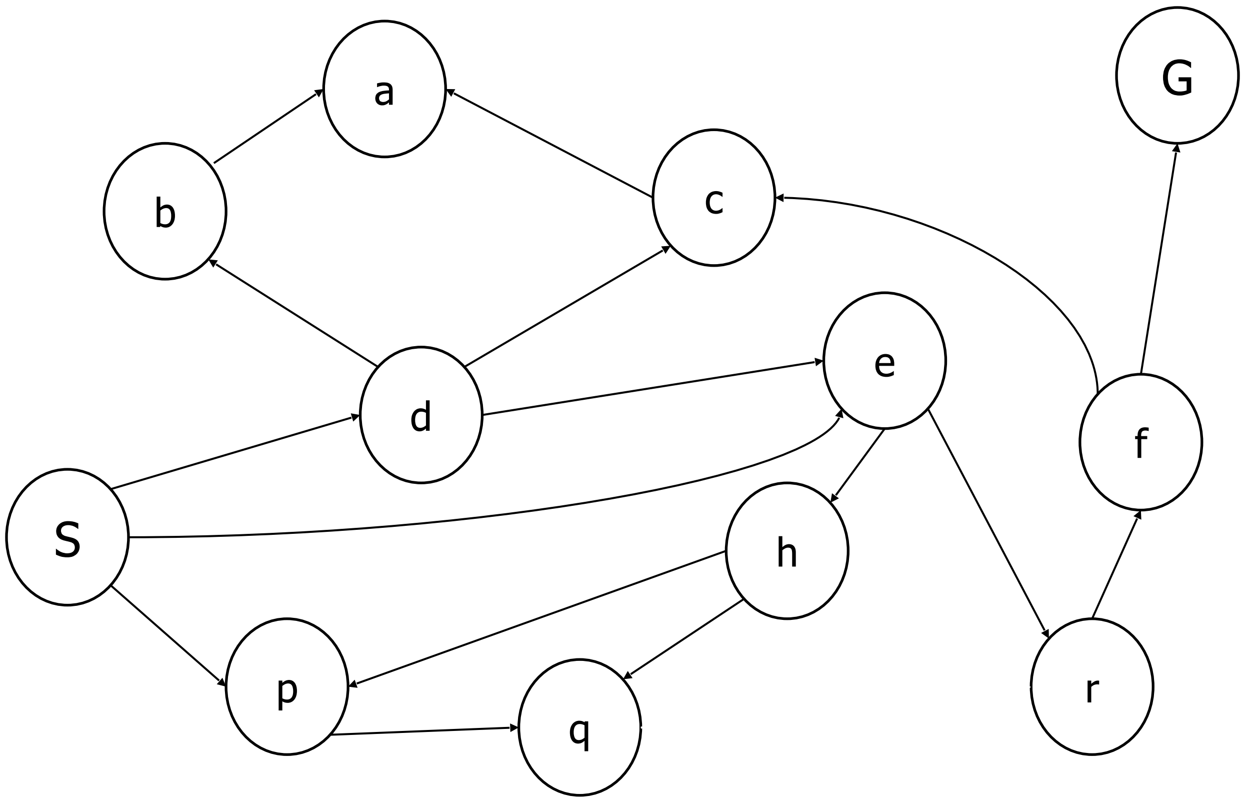
General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Important details:
    - Fringe (frontier)
    - Expansion
    - Exploration strategy
- Main question: which fringe nodes to explore?

Graph representation of the landscape explored

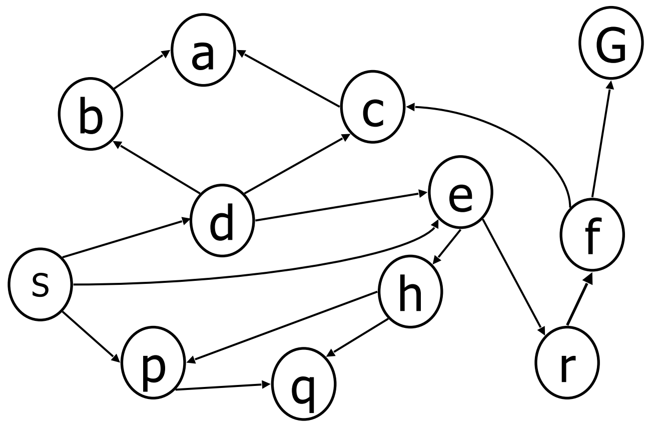


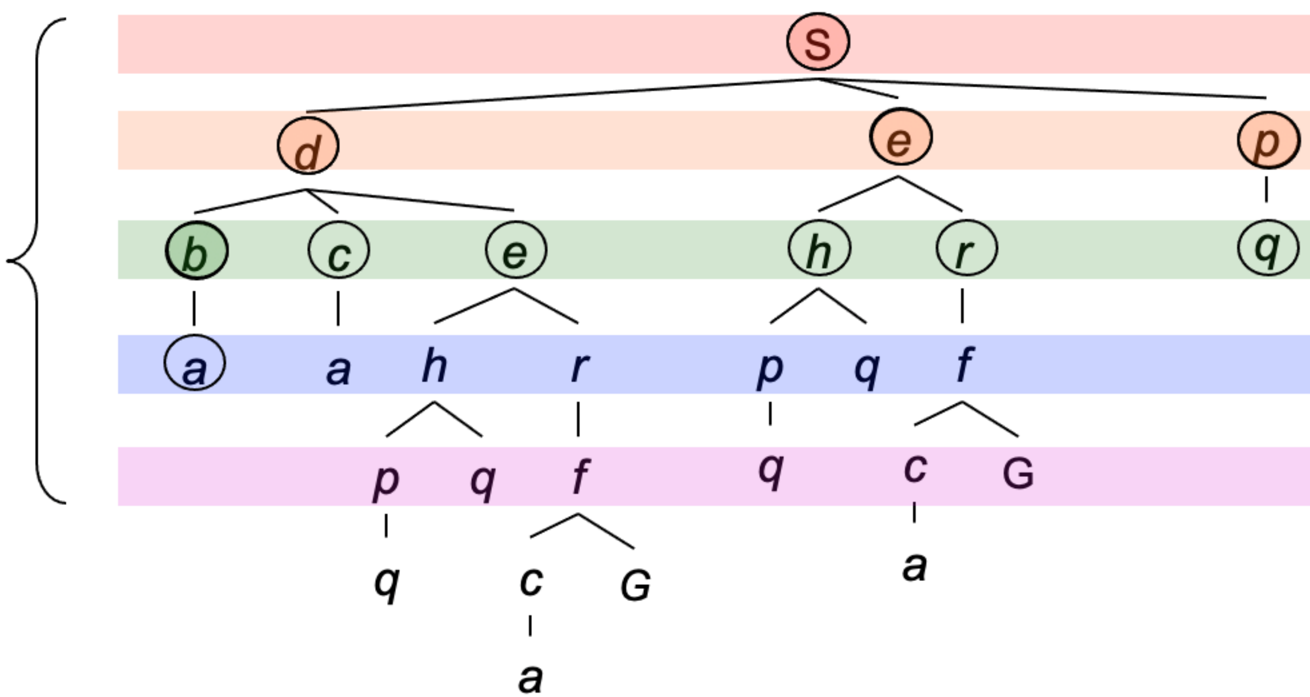

a
G
b
c
e
d
f
S
h
p
q
r

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

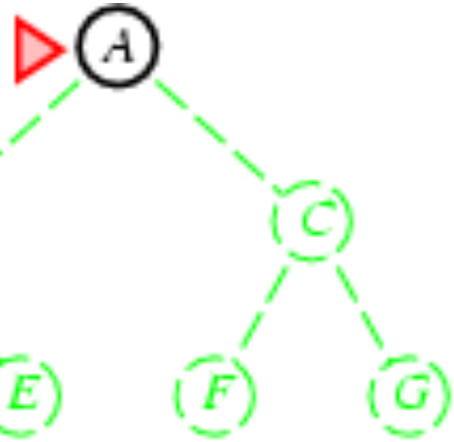
```
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest 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)
```

Figure 3.11 Breadth-first search on a graph.

Breadth-first search

- Expand shallowest unexpanded node
- Fringe: nodes waiting in a queue to be explored, also called OPEN
- Implementation:
  - *fringe* is a first-in-first-out (FIFO) queue, i.e., new successors go at end of the queue.

Is A a goal state?

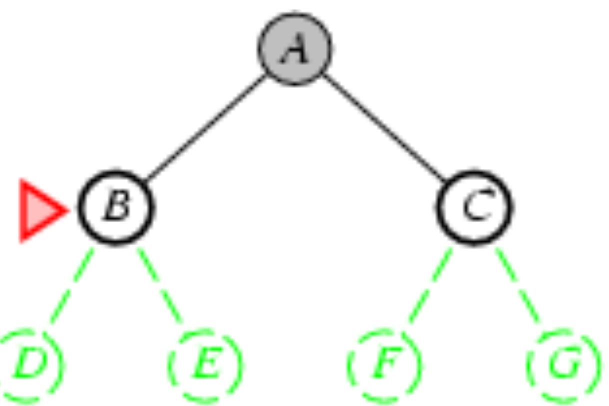


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe = [B,C]

Is B a goal state?



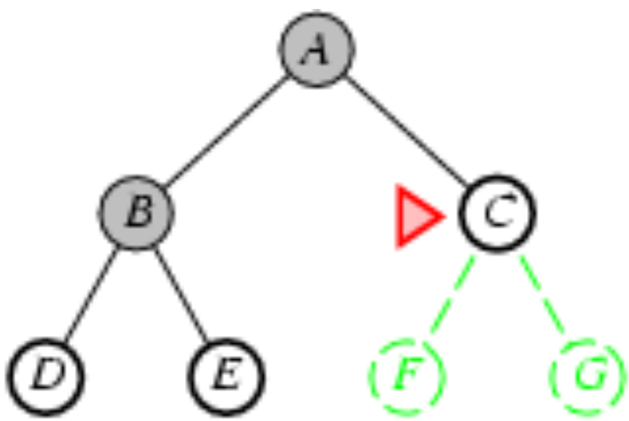
Breadth-first search

- Expand shallowest unexpanded node

- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

Expand:
fringe=[C,D,E]

Is C a goal state?

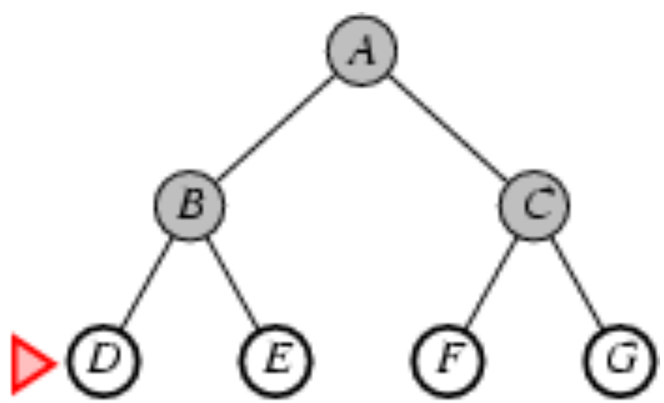


Breadth-first search

- Expand shallowest unexpanded node
- Implementation:
  - *fringe* is a FIFO queue, i.e., new successors go at end

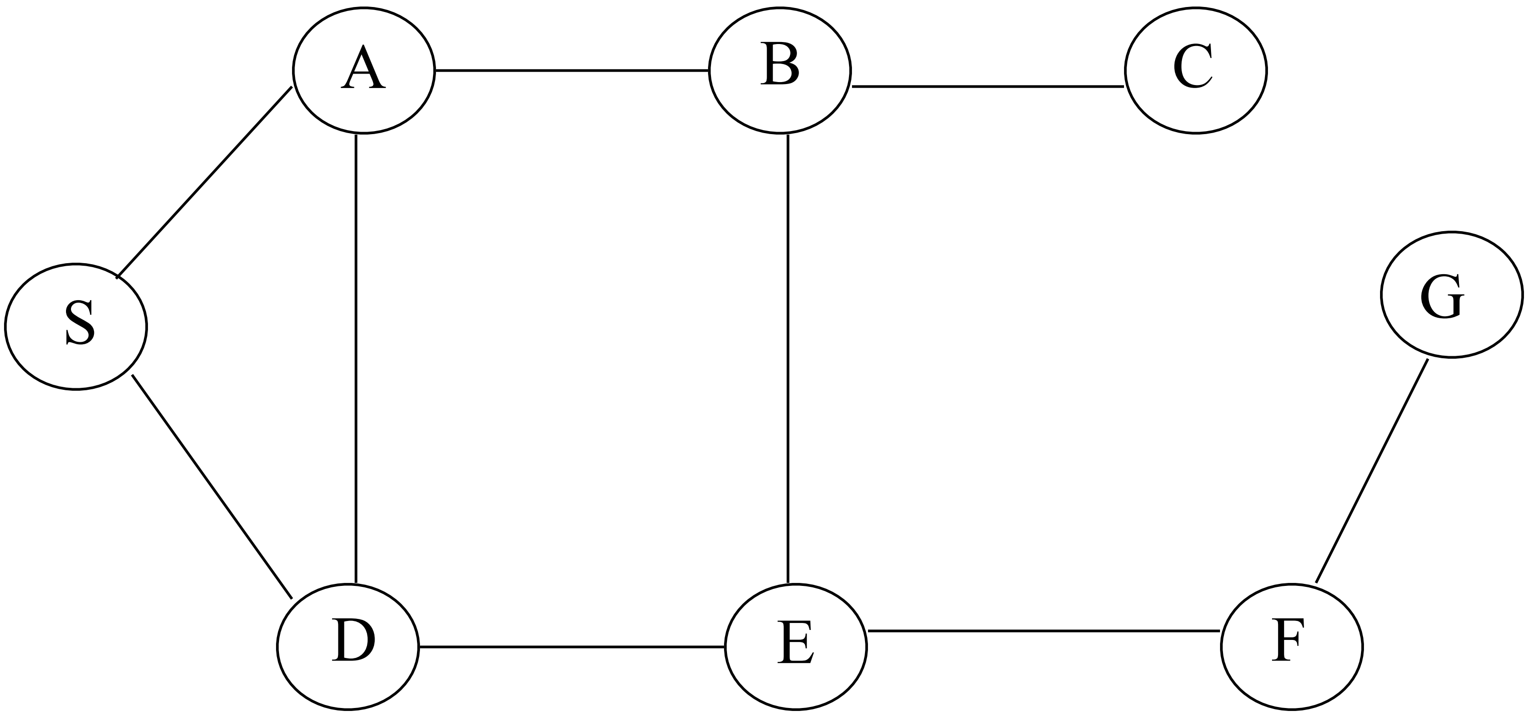
Expand:
fringe=[D,E,F,G]

Is D a goal state?



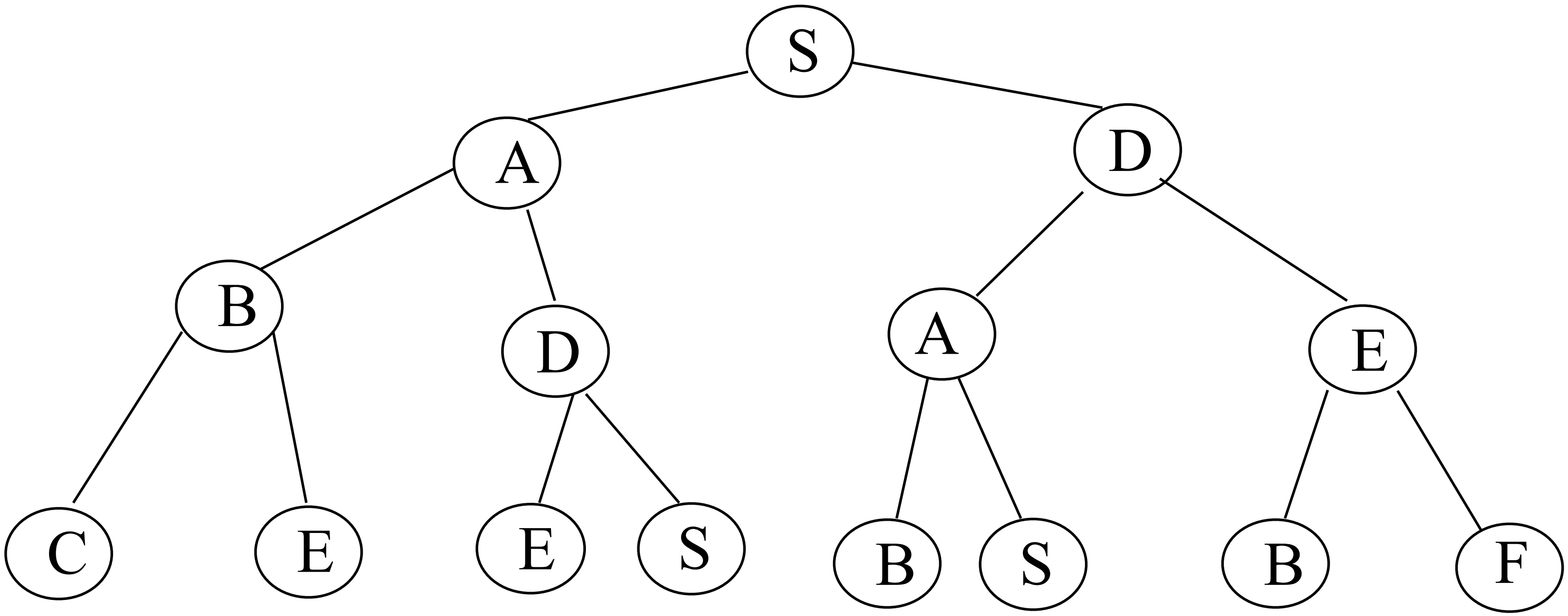
BFS for 8 puzzle

Example: Map Navigation



S = start,   G = goal, other nodes = intermediate states, links = legal transitions

Breadth First Search Tree (BFS)

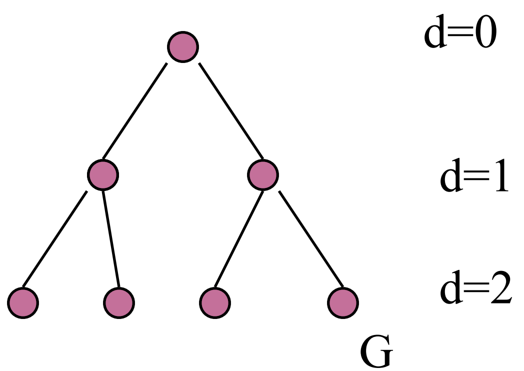


Here BFS is implemented as a tree search with only parent node not added as a child node.

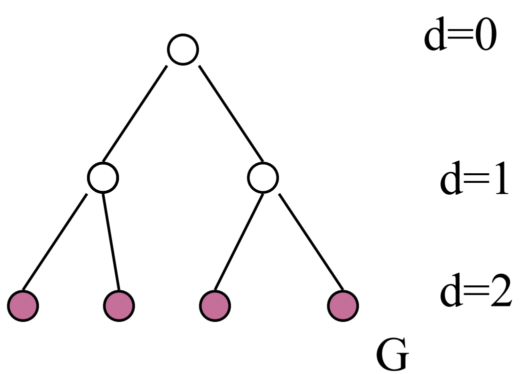
What is the Complexity of Breadth-First Search?

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

    $= 1 + b + b^2 + \;\; \dots\dots\dots + b^d$
    $= \mathbf{O\ (b^{d+1})}$



- Space Complexity
  - how many nodes can be in the queue (worst-case)?
  - at depth d there are b^d **unexpanded** nodes in the Q = $\mathbf{O\ (b^d)}$



- Time and space of number of generated nodes is $O\ (b^{d+1})$

Examples of Time and Memory Requirements for *tree search version* of Breadth-First Search

Depth of Solution	Nodes Expanded	Time	Memory
0	1	1 millisecond	100 bytes
2	111	0.1 seconds	11 kbytes
4	11,111	11 seconds	1 megabyte
8	10^8	31 hours	11 giabytes
12	10^{12}	35 years	111 terabytes

Assuming b=10, 1000 nodes/sec, 100 bytes/node

Breadth-First Search (BFS) Properties

- Complete (with find a solution in a finite number of steps if one exists)
- Solution Length: optimal (assuming unit cost per move)
- (Can) expand each node once (if checks for duplicates)
- Search Time: $O(b^d)$ which is the size of the state space
- Memory Required: $O(b^d)$
- Drawback: requires space proportional to the state-space (Search time is unavoidable)