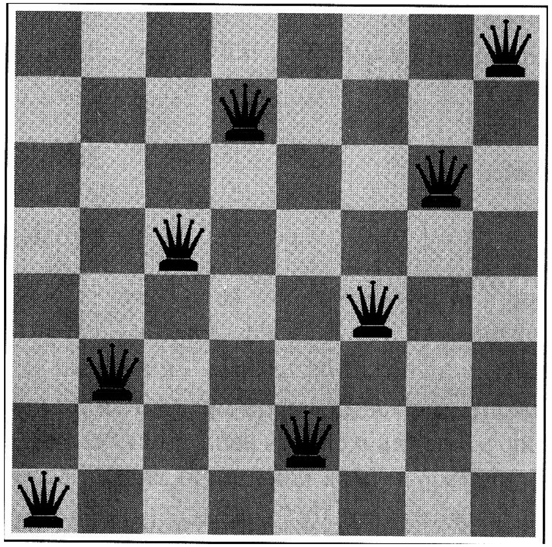
Artificial Intelligence

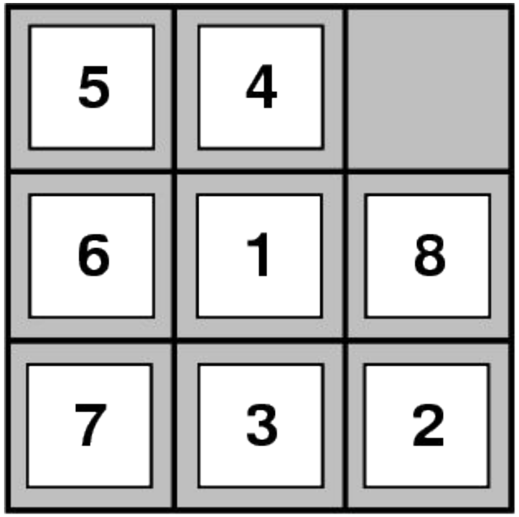
Lecture:
Problem Solving using Search - (Single agent search)
Uninformed Search

Example: 8-queens



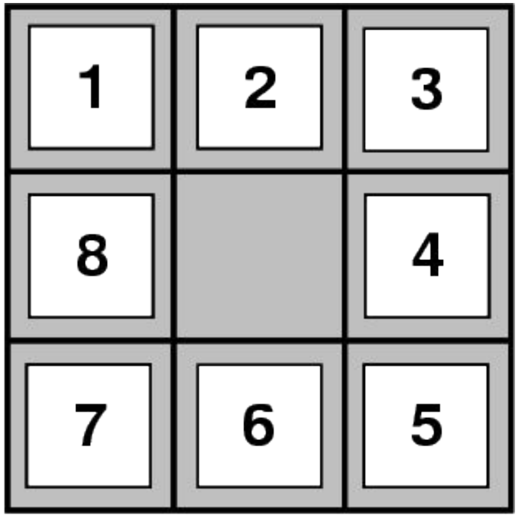
- State: any arrangement of up to 8 queens on the board
- Operation: add a queen (incremental), move a queen (fix-it)
- Initial state: no queens on board
- Goal state: 8 queens, with no queen is attacked
- Solution Path: The set of operations that allowed you to get to the The board that you see above at the indicated positions.

Example: 8-puzzle



5	4	
6	1	8
7	3	2

Start State

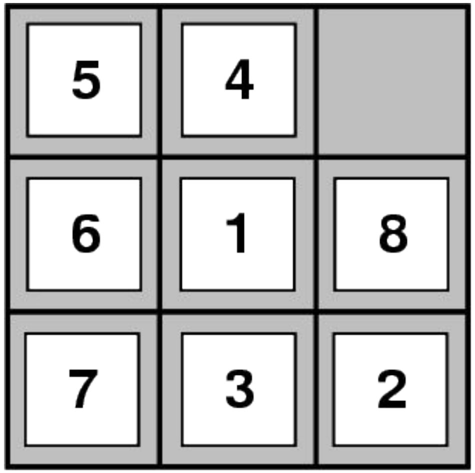


1	2	3
8		4
7	6	5

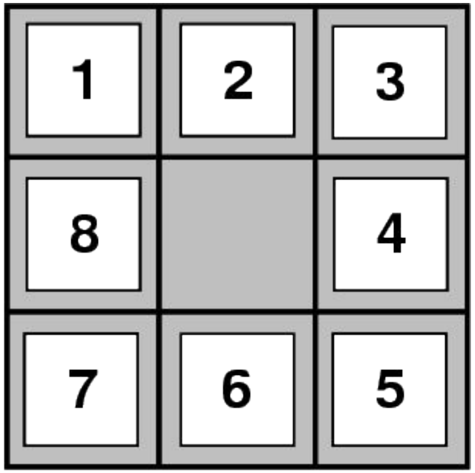
Goal State

- State:
- Operators:
- Goal test:
- Solution path:

Example: 8-puzzle



Start State



Goal State

- **Operators**: moving blank left, right, up, down (ignore jamming)
- **Goal**: goal state
- **State**: integer location of tiles (ignore intermediate locations)
- Solution: move 4 tile to blank, move 1 tile blank, etc.

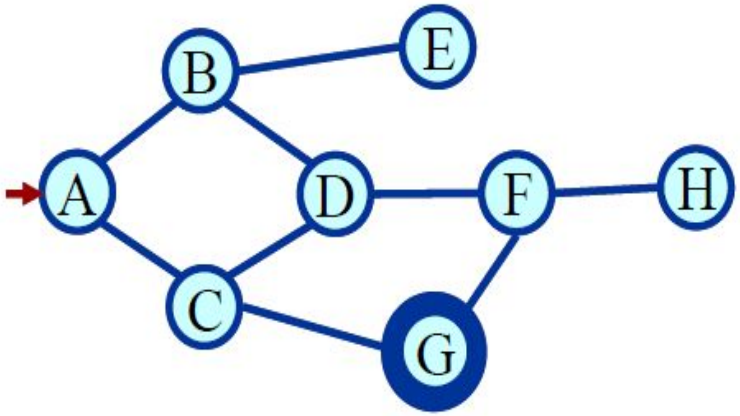
Search

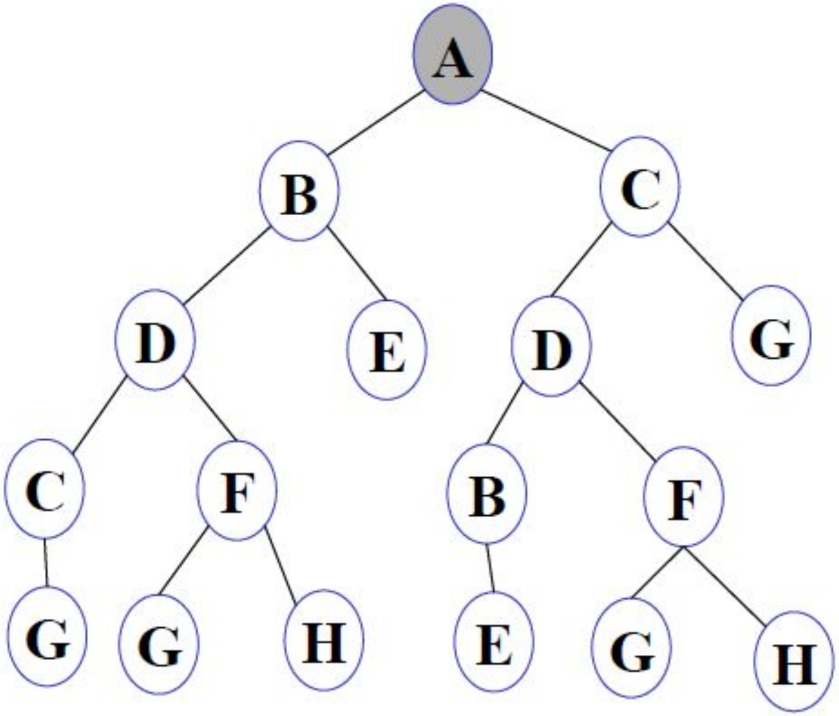
- What is Search?

- Formulate appropriate problems in optimization and planning (sequence of actions to achieve a goal) as search tasks: initial state, operators, goal test, path cost

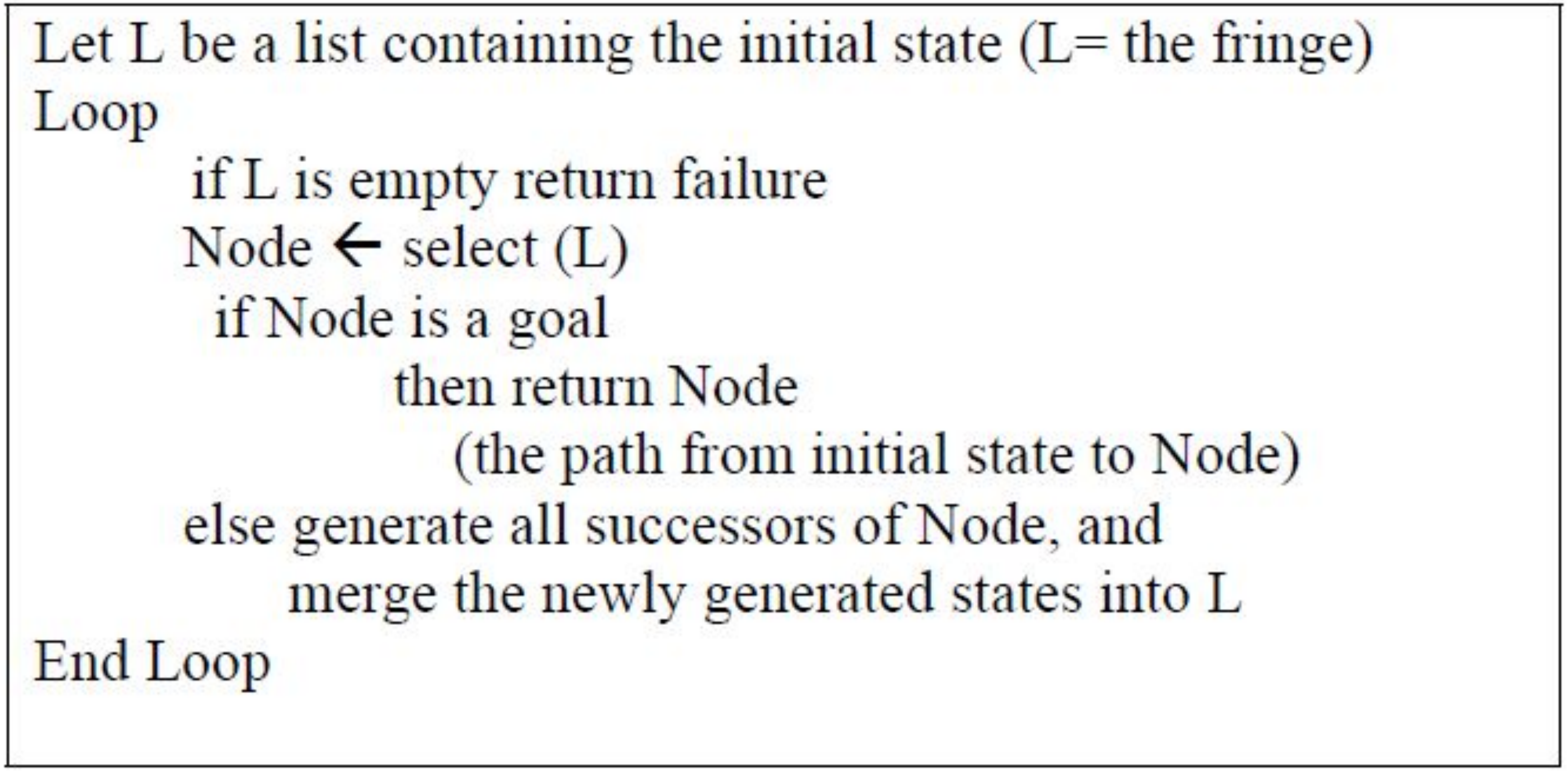
Search Tree

List all Possible Path
Eliminate Cycles from paths
Result: Search Tree





The basic search algorithm



```
Let L be a list containing the initial state (L= the fringe)
Loop
    if L is empty return failure
    Node ← select (L)
    if Node is a goal
        then return Node
            (the path from initial state to Node)
    else generate all successors of Node, and
        merge the newly generated states into L
End Loop
```

Evaluating Search Strategies

- **Completeness**
  - Guarantees finding a solution whenever one exists
- **Time complexity**
  - How long (worst or average case) does it take to find a solution? Usually measured in terms of the number of nodes expanded
- **Space complexity**
  - How much space is used by the algorithm? Usually measured in terms of the maximum size of the “nodes” list during the search
- **Optimality/Admissibility**
  - If a solution is found, is it guaranteed to be an optimal one? That is, is it the one with minimum cost?

Problem solving

- We want:
  - To automatically solve a problem

- We need:
  - A representation of the problem
  - Algorithms that use some strategy to solve the problem defined in that representation

Problem representation

- General:
  - **State space**: a problem is divided into a set of resolution steps from the initial state to the goal state
  - **Reduction to sub-problems**: a problem is arranged into a hierarchy of sub-problems
- Specific:
  - Game resolution
  - Constraints satisfaction

States

- A problem is defined by its elements and their relations.

- In each instant of the resolution of a problem, those elements have specific descriptors (How to select them?) and relations.

- A **state** is a representation of those elements in a given moment.

- Two special states are defined:
  - **Initial state** (starting point)
  - **Final state** (goal state)

State modification: successor function

- A successor function is needed to move between different states.
- A **successor function** is a description of possible actions, a set of operators. It is a transformation function on a state representation, which convert it into another state.
- The successor function defines a relation of accessibility among states.
- Representation of the successor function:
  - Conditions of applicability
  - Transformation function

State space

- The **state space** is the set of all states reachable from the initial state.
- It forms a graph (or map) in which the nodes are states and the arcs between nodes are actions.
- A **path** in the state space is a sequence of states connected by a sequence of actions.
- The solution of the problem is part of the map formed by the state space.

Problem solution

- A **solution** in the state space is a path from the initial state to a goal state or, sometimes, just a goal state.
- **Path/solution cost**: function that assigns a numeric cost to each path, the cost of applying the operators to the states
- Solution quality is measured by the path cost function, and an **optimal solution** has the lowest path cost among all solutions.
- Solutions: any, an **optimal one**, all. Cost is important depending on the problem and the type of solution sought.

Problem description

- Components:
  - State space (explicitly or implicitly defined)
  - Initial state
  - Goal state (or the conditions it has to fulfill)
  - Available actions (operators to change state)
  - Restrictions (e.g., cost)
  - Elements of the domain which are relevant to the problem (e.g., incomplete knowledge of the starting point)
  - Type of solution:
    - Sequence of operators or goal state
    - Any, an optimal one (cost definition needed), all

Uninformed vs. informed search

- **Uninformed search strategies**
  - Also known as "blind search," uninformed search strategies use **no information about the likely "direction" of the goal node(s)**
  - Uninformed search methods: Breadth-first, depth-first, depth-limited, uniform-cost, depth-first iterative deepening, bidirectional
- **Informed search strategies**
  - Also known as "**heuristic search**" informed search strategies ***use information about the domain to*** (try to) (usually) head in the general direction of the goal node(s)
  - Informed search methods: Hill climbing, best-first, greedy search, beam search, A, A*

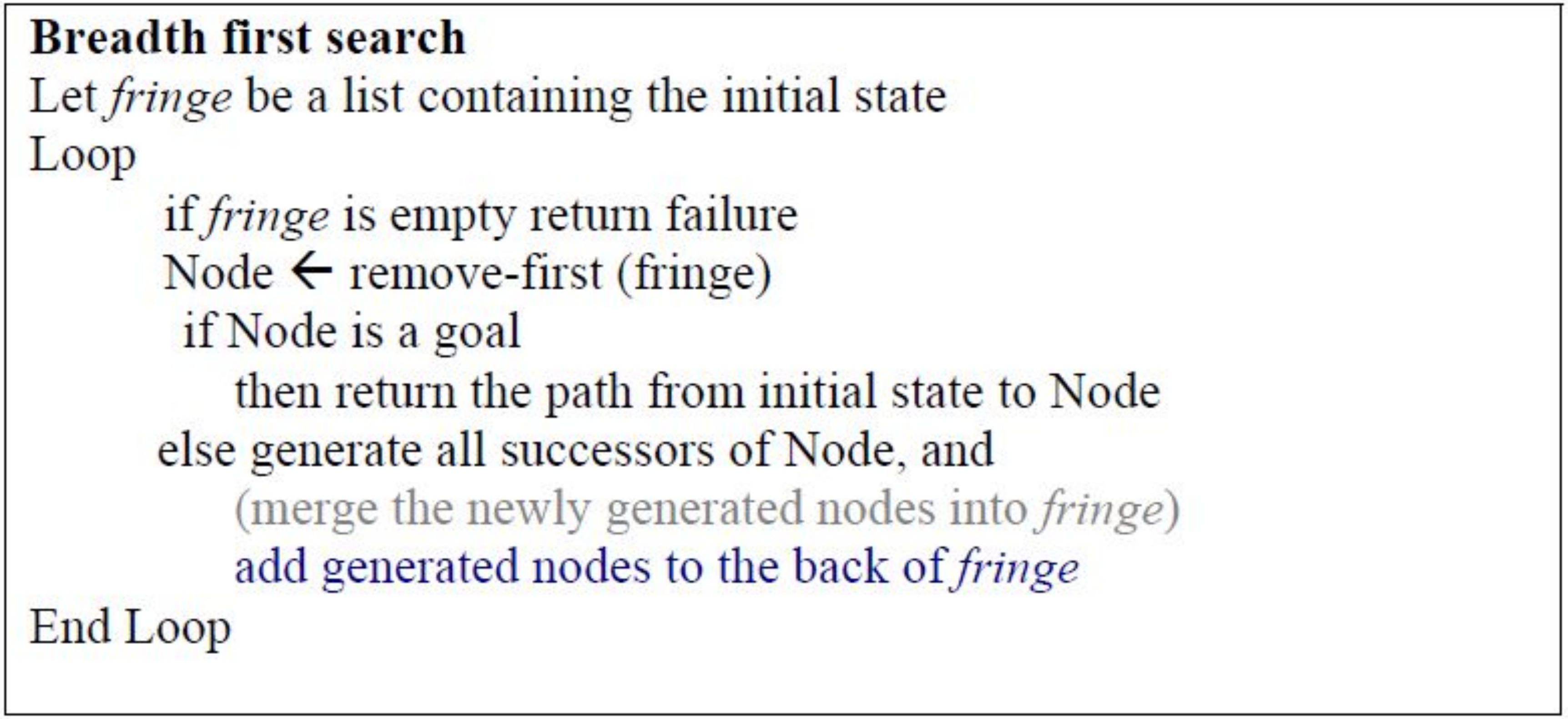
Uninformed	Informed
1. Search without information	1. Search with information
2. No Knowledge	2. use knowledge to find steps to solution
3. Time Consuming	3. quick solution
4. More complexity - time and space	4. less complexity
5. BFS, DFS	5. A*, Best First, AO*

Breadth-First Search (BFS)

- Uninformed Search technique
- FIFO (Queue)
- Expand shallowest unexpanded node
- Level search
- Fringe: nodes waiting in a queue to be explored, also called OPEN
- Optimal - shortest path
- Implementation:
  - For BFS, *fringe* is a first-in-first-out (FIFO) queue
  - new successors go at end of the queue

- Repeated states?
  - Simple strategy: do not add parent of a node as a leaf

BFS Algorithm



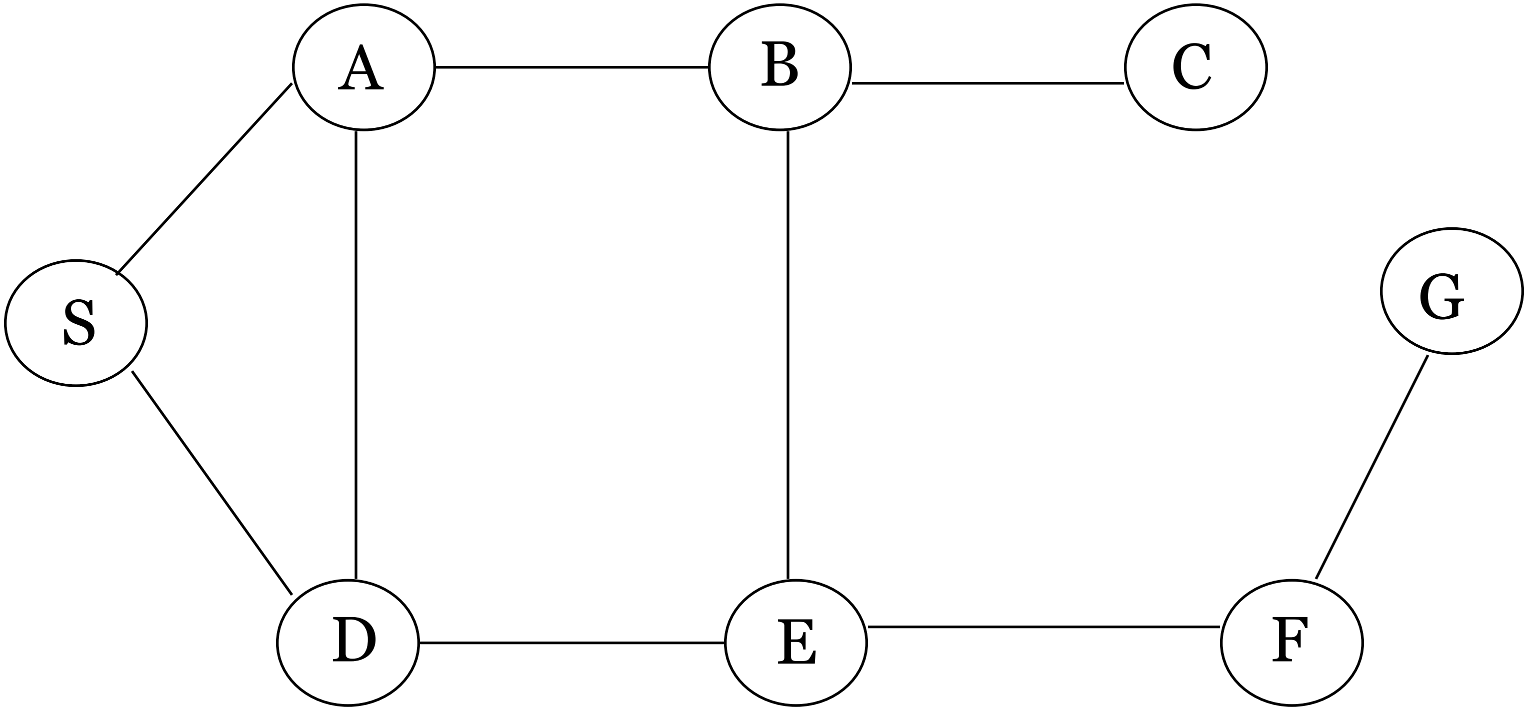
Breadth first search

```
Let fringe be a list containing the initial state
Loop
        if fringe is empty return failure
        Node ← remove-first (fringe)
         if Node is a goal
            then return the path from initial state to Node
        else generate all successors of Node, and
            (merge the newly generated nodes into fringe)
            add generated nodes to the back of fringe
End Loop
```

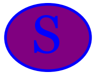
Example: Map Navigation

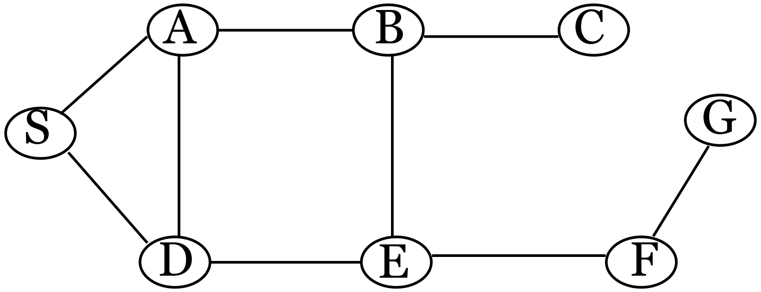
State Space:

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



BFS Search Tree





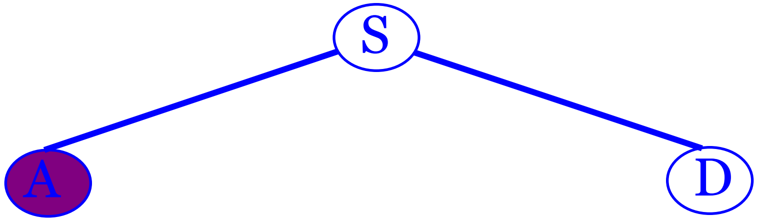
Queue = {S}

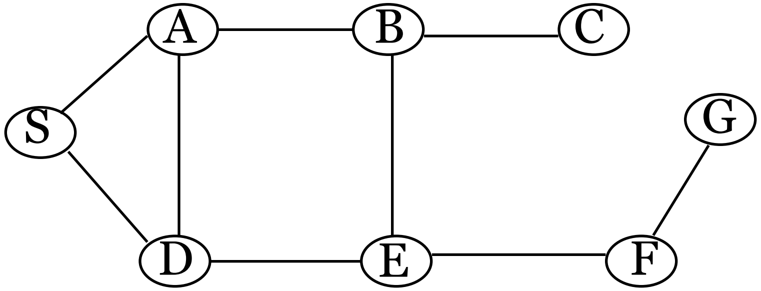
Select S

Goal(S) = true?

If not, Expand(S)

BFS Search Tree





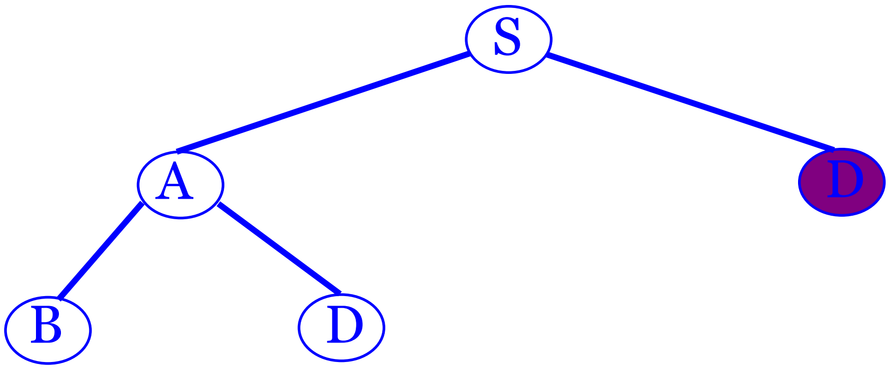
Queue = {A, D}

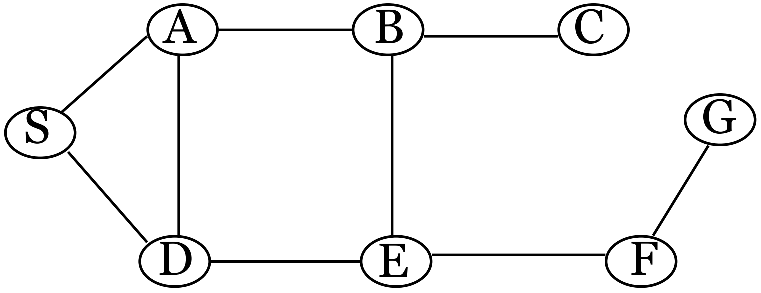
Select A

Goal(A) = true?

If not, Expand(A)

BFS Search Tree





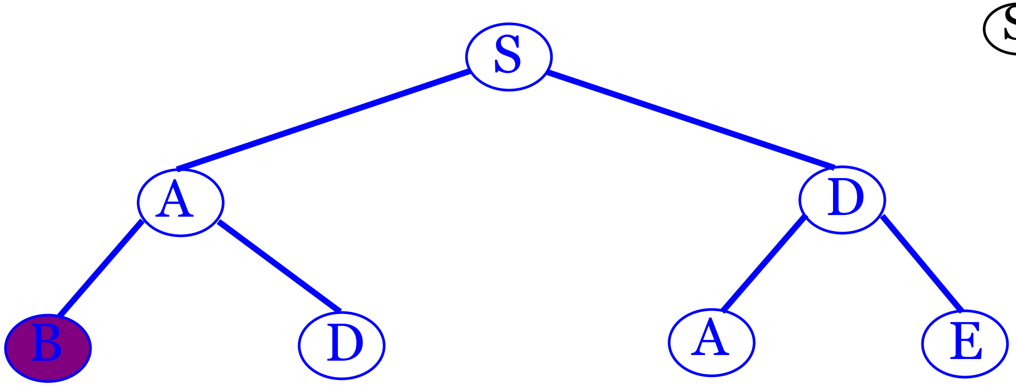
Queue = {D, B, D}

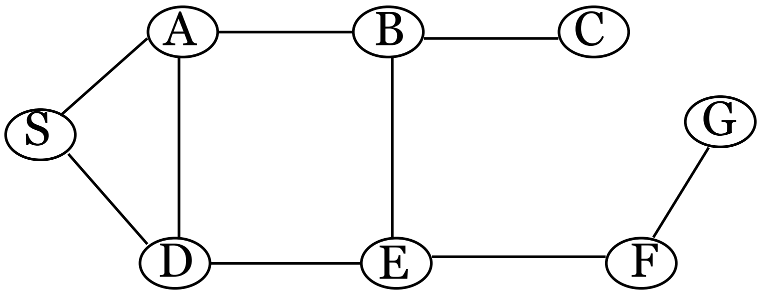
Select D

Goal(D) = true?

If not, expand(D)

BFS Search Tree

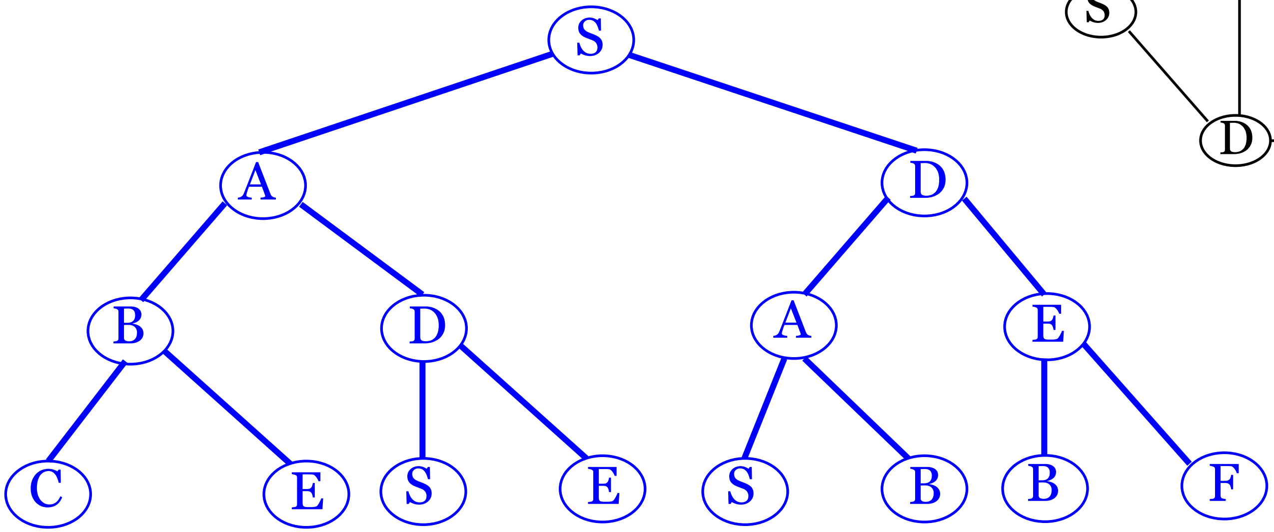


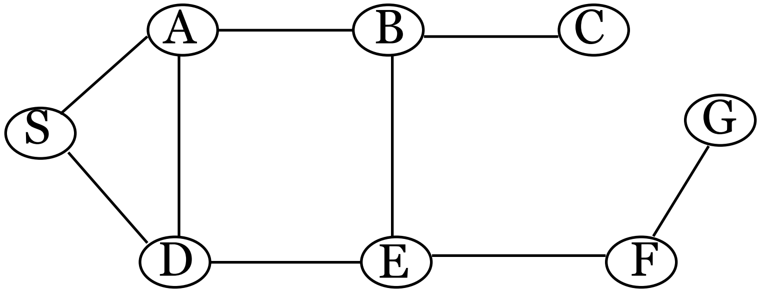


Queue = {B, D, A, E}

Select B
etc.

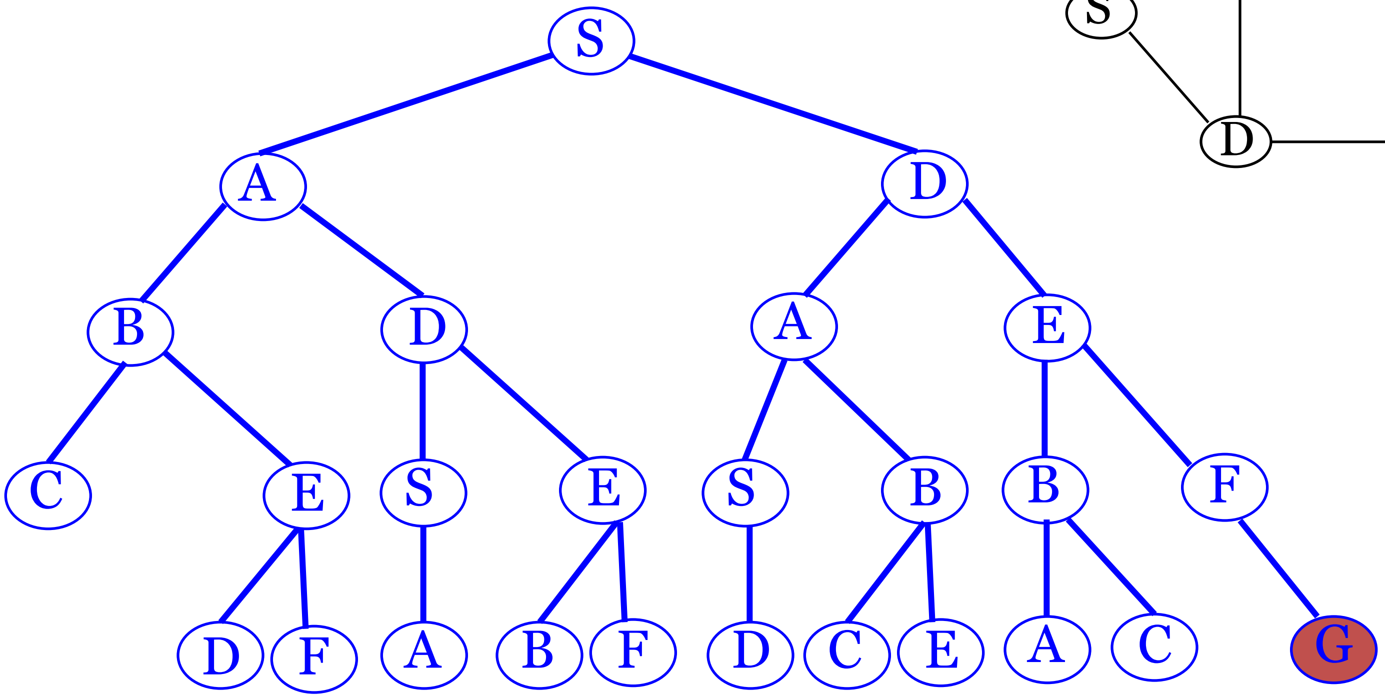
BFS Search Tree

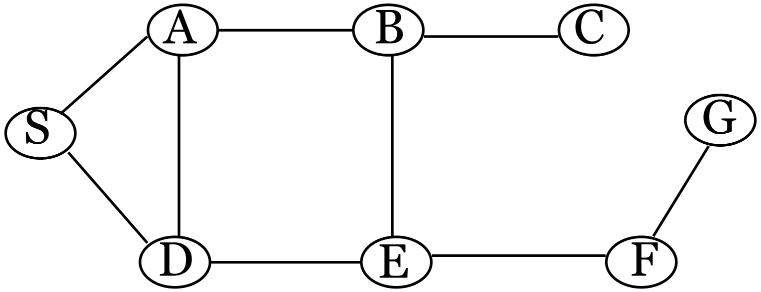




Level 3
Queue = {C, E, S, E, S, B, B, F}

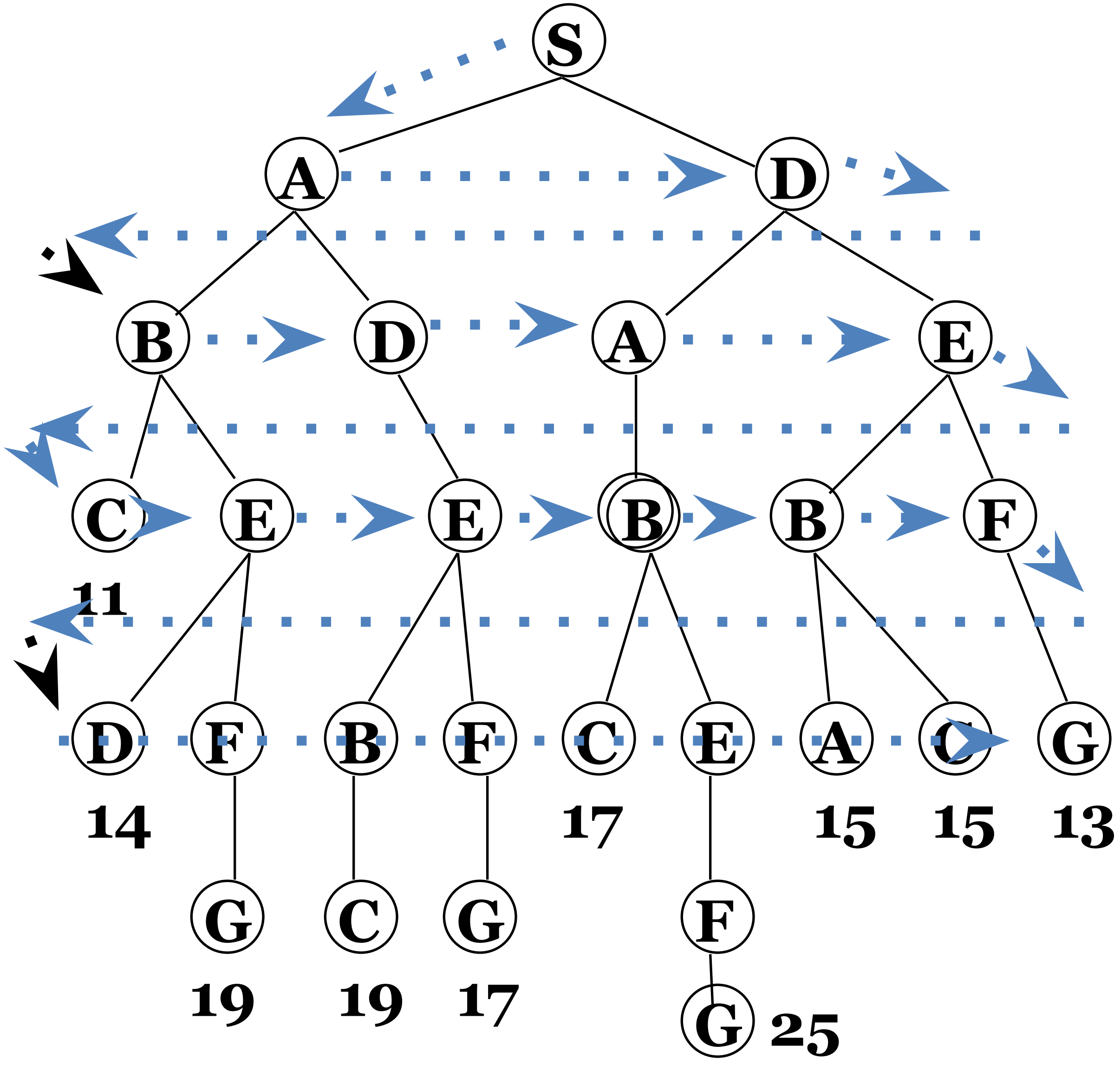
BFS Search Tree





Level 4
Expand queue until G is at front
Select G
Goal(G) = true

Another Breath-first search



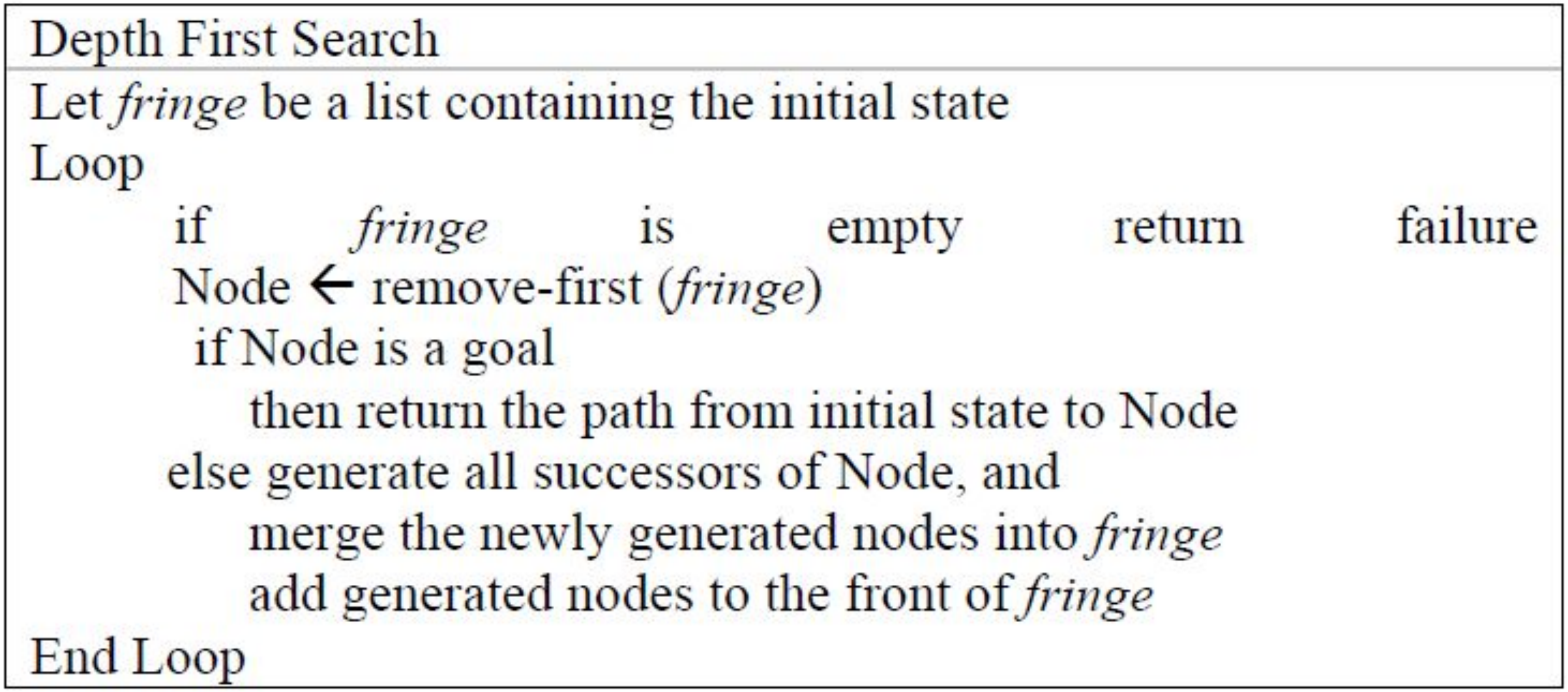
Depth-First Search (DFS)

- Uninformed Search
- Stack (LIFO)
- Expand deepest unexpanded node
- LIFO
- Deepest Node
- Incomplete ---> 1. Loop 2. search space infinite i.e. unlimited depth

Non Optimal
Time complexity

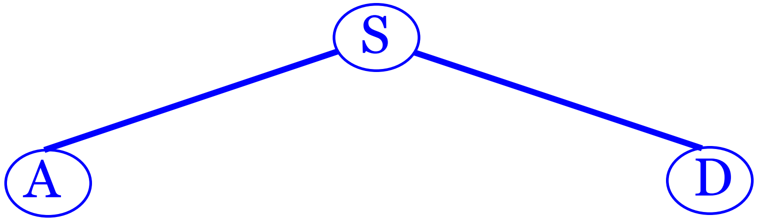
- Implementation:
  - For DFS, *fringe* is a LIFO queue
  - new successors go at beginning of the queue
- Repeated nodes?
  - Simple strategy: Do not add a state as a leaf if that state is on the path from the root to the current node

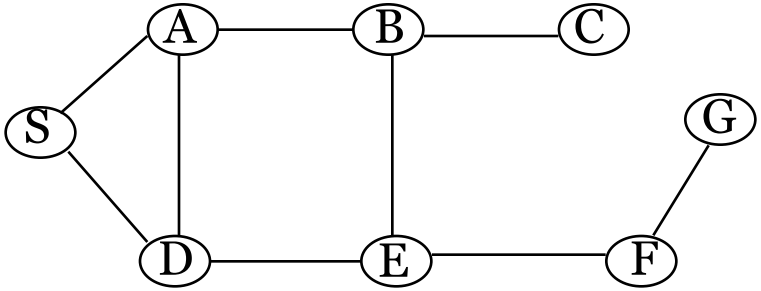
DFS Algorithm



```
Depth First Search
Let *fringe* be a list containing the initial state
Loop
        if fringe is empty return failure
        Node ← remove-first (fringe)
         if Node is a goal
            then return the path from initial state to Node
        else generate all successors of Node, and
            merge the newly generated nodes into fringe
            add generated nodes to the front of fringe
End Loop
```

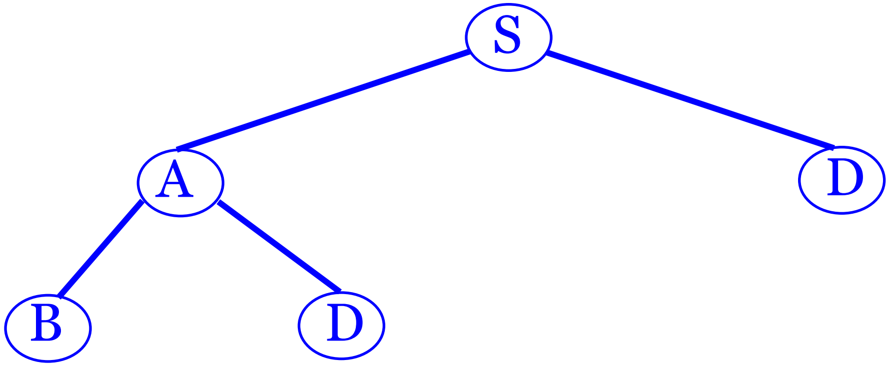
DFS Search Tree

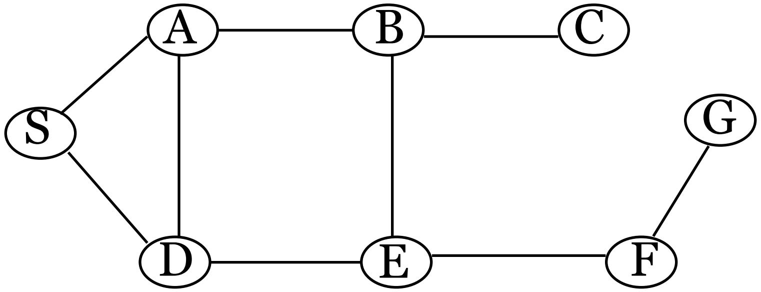




Queue = {A,D}

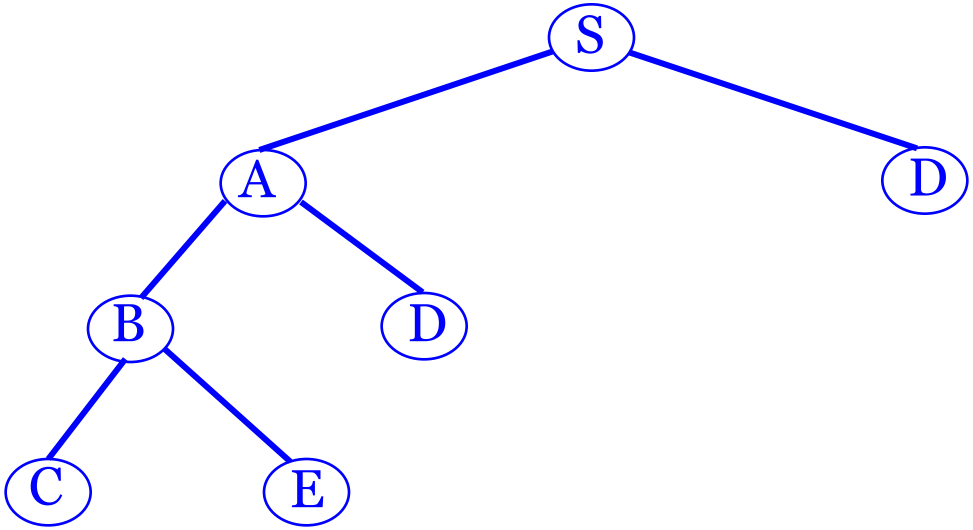
DFS Search Tree

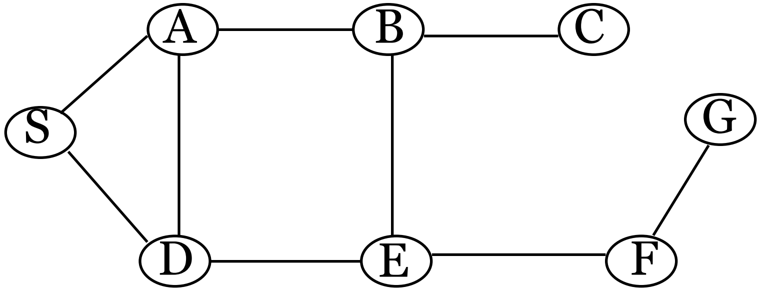




Queue = {B,D,D}

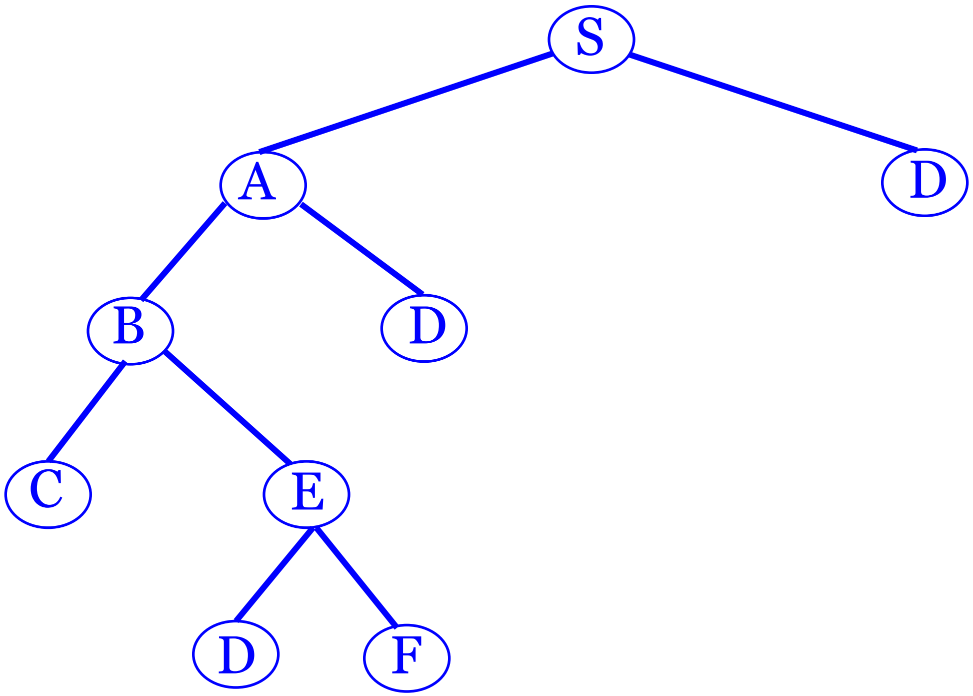
DFS Search Tree

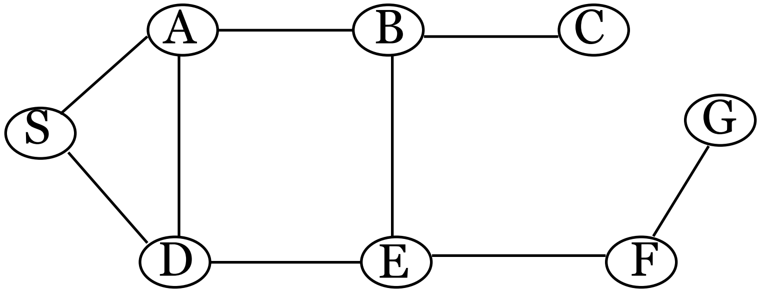




Queue = {C,E,D,D}

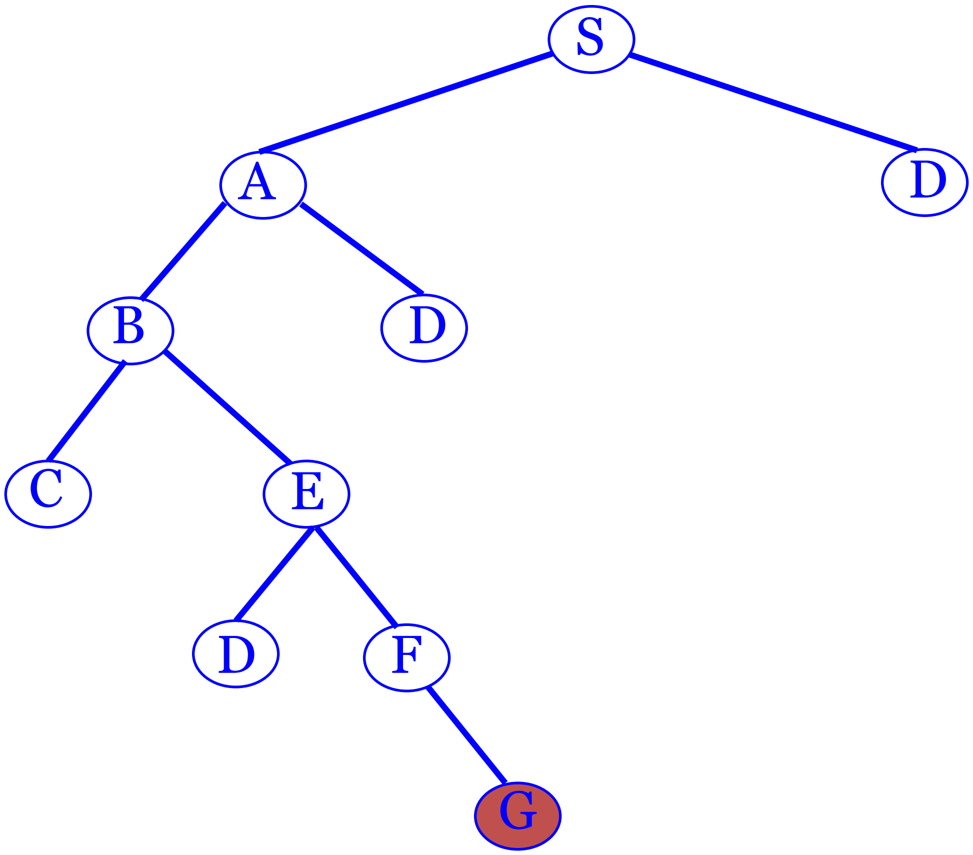
DFS Search Tree

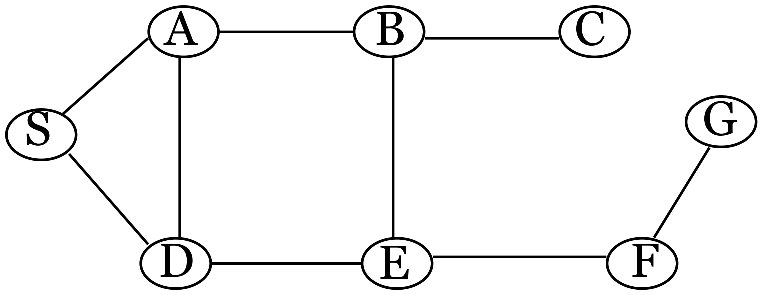




Queue = {D,F,D,D}

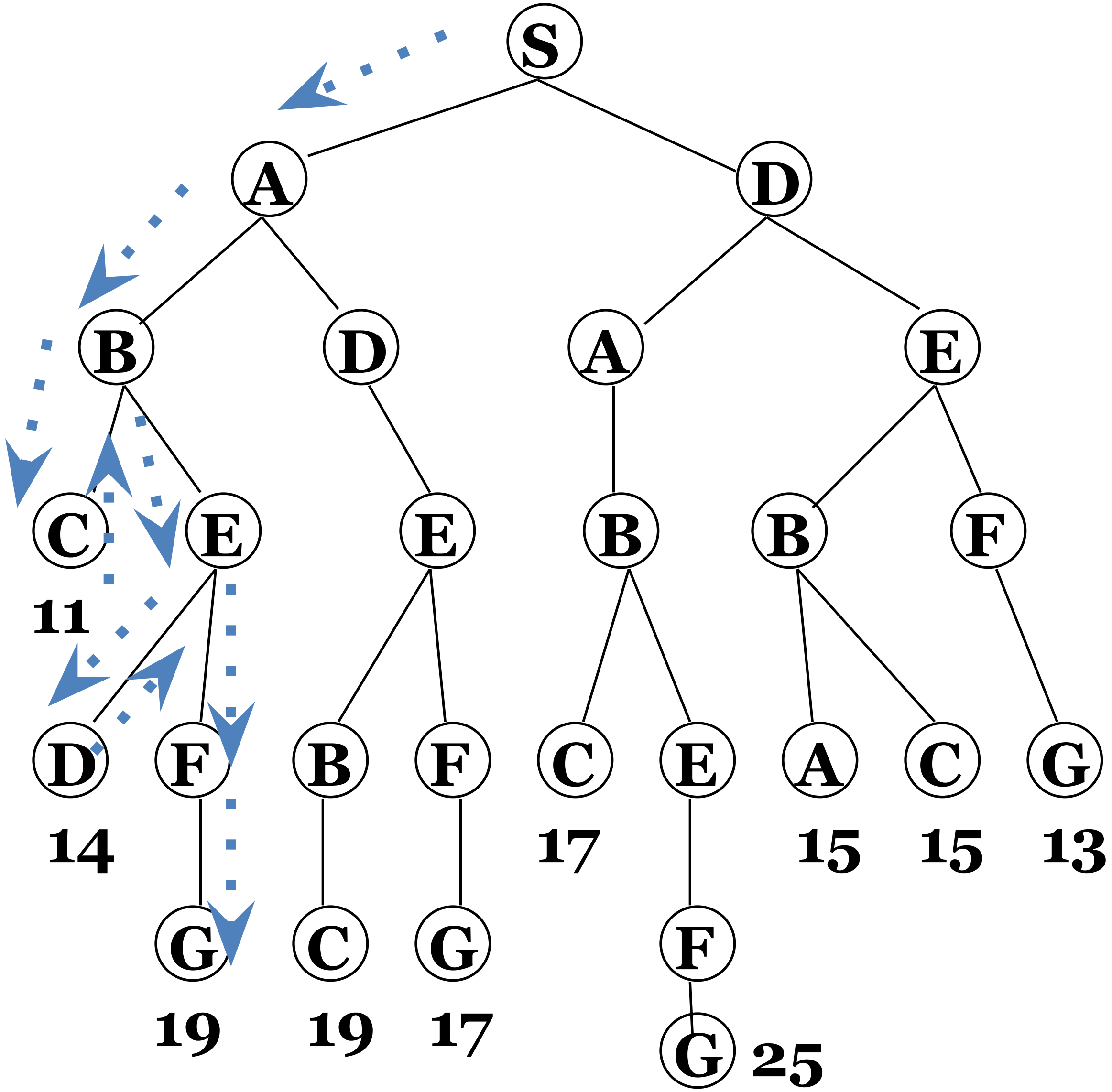
DFS Search Tree





Queue = {G,D,D}

Depth-first search



Evaluation of Search Algorithms

- **Completeness**
  - does it always find a solution if one exists?
- **Optimality**
  - does it always find a least-cost (or min depth) solution?
- **Time complexity**
  - number of nodes generated (worst case)
- **Space complexity**
  - number of nodes in memory (worst case)

- **Time and space complexity are measured in terms of**
  - *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 (BFS) Properties

- **Complete? Yes**
- **Optimal? Yes**, for the shortest path
- **Time complexity?** $O(b^d)$

  $$1 + b + b^2 + \dots + b^d = O(b^d)$$

  exponential in the depth of the solution *d*
- **Space complexity?** $O(b^d)$

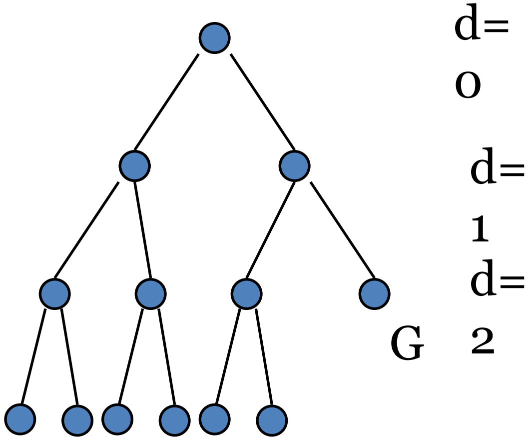
  same as time - every node is kept in the memory
- **Main practical drawback?** exponential space complexity

Complexity of Breadth-First Search

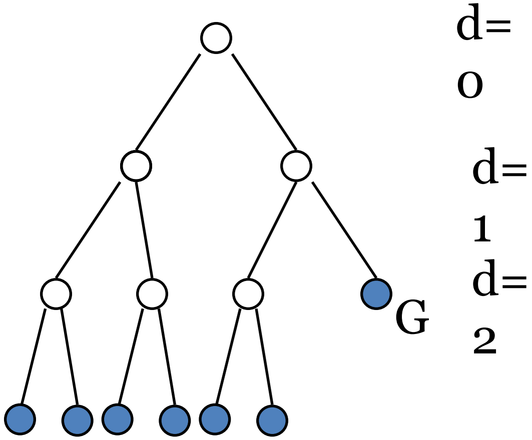
- **Time Complexity**
  - assume (worst case) that there is 1 goal leaf at the RHS at depth d
  - so BFS will generate

$$= b + b^2 + \ ..... + b^d + b^{d+1} - b$$

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



- **Space Complexity**
  - how many nodes can be in the queue (worst-case)?
  - at depth d there are b^{d+1} unexpanded nodes in the Q = $\mathbf{O\ (b^{d+1})}$



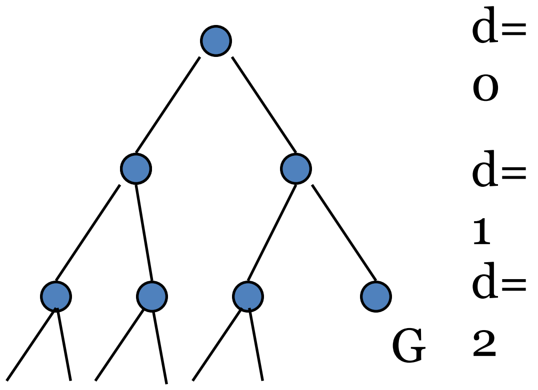
Examples of Time and Memory Requirements for Breadth-First Search

Assuming b=10, 10000 nodes/sec, 1kbyte/node

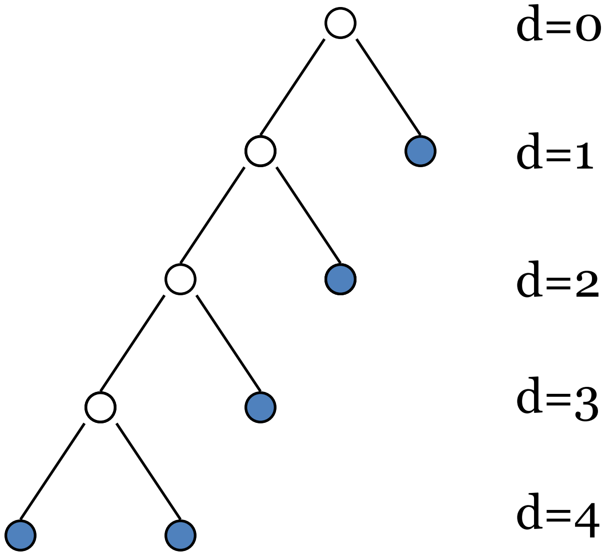
Depth of Solution	Nodes Generated	Time	Memory
2	1100	0.11 seconds	1 MB
4	111,100	11 seconds	106 MB
8	10^9	31 hours	1 TB
12	10^{13}	35 years	10 PB

What is the Complexity of Depth-First Search?

- Time Complexity
  - maximum tree depth = m
  - assume (worst case) that there is 1 goal leaf at the RHS at depth d so DFS will generate **O (b^m)**



- Space Complexity
  - how many nodes can be in the queue (worst-case)?
  - at depth m we have b nodes
  - and b-1 nodes at earlier depths
  - total = b + (m-1)*(b-1) = **O(bm)**



Examples of Time and Memory Requirements for Depth-First Search

Assuming b=10, m = 12, 10000 nodes/sec, 1kbyte/node

Depth of Solution	Nodes Generated	Time	Memory
2	10^{12}	3 years	120kb
4	10^{12}	3 years	120kb
8	10^{12}	3 years	120kb
12	10^{12}	3 years	120kb

Depth-First Search (DFS) Properties

- **Complete?**
  - **No**. Not complete if tree has unbounded depth
- **Optimal?**
  - **No**. Solution found first may not be the shortest possible
- **Time complexity?**

  $O(b^m)$

  - **Exponential** **exponential in the maximum depth of the search tree *m***
- **Space complexity?**

  $O(bm)$

  linear in the maximum depth of the search tree *m*

  - **Linear**

Comparing DFS and BFS

- **Time complexity**: same, but
  - In the worst-case BFS is always better than DFS
  - Sometime, on the average DFS is better if:
    - many goals, no loops and no infinite paths

- BFS is much worse memory-wise
    - DFS is linear space
    - BFS may store the whole search space.

- In general
    - BFS is better if goal is not deep, if infinite paths, if many loops, if small search space
    - DFS is better if many goals, not many loops,
    - DFS is much better in terms of memory

Thank You!

Any Questions?