Artificial Intelligence
CS-401

Chapter # 03

Solving Problems by Searching

Dr. Hafeez Ur Rehman
(Email: hafeez.urrehman@nu.edu.pk)

Chapter's Outline

- Problem Solving Agents
- Well Defined Problems and Solutions
- Example Problems
- Searching for Solutions
  - Infrastructure of Search Algorithms
  - Performance Metric
- **Uninformed Search Strategies**
- Informed (Heuristic) Search Strategies

Search Algorithms

- Uninformed Blind search
  1. Breadth-first
  2. depth-first
  3. uniform cost
  4. Iterative deepening depth-first
  5. Bidirectional
- Informed Heuristic search
  - Greedy search, A*, Hill climbing, Local Searches
- Important concepts:
  1. Completeness
  2. Optimality
  3. Time complexity
  4. Space complexity

Search Strategies

- A search strategy is defined by picking the **order of node expansion**

- Strategies are **evaluated** along the following four dimensions:
  1. **Completeness**: does it always find a solution if one exists?
  2. **Optimality**: does it always find a least-cost solution?
  3. **Time complexity**: number of nodes generated
  4. **Space complexity**: maximum number of nodes in memory

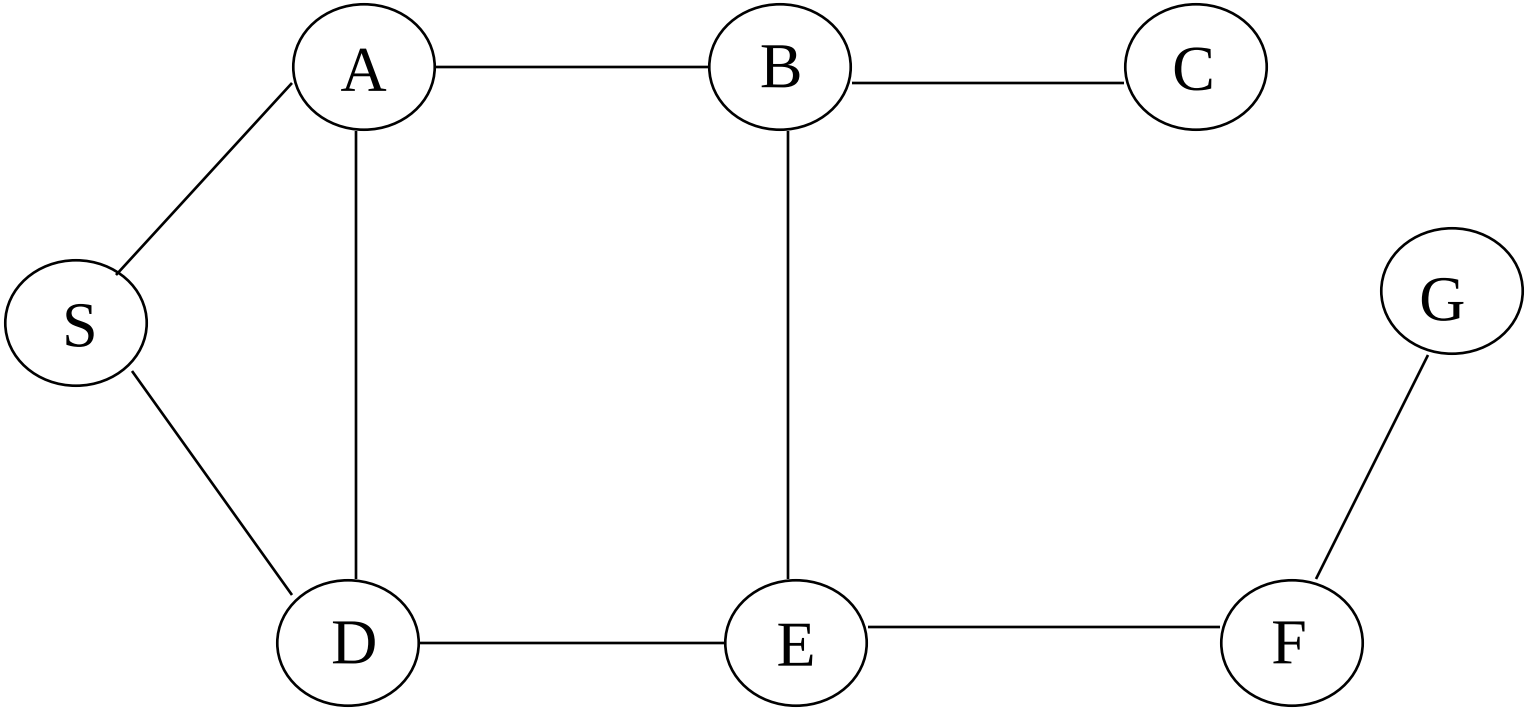
- Time and space complexity are measured in terms of
  - ***b:*** maximum branching factor of the search tree
  - ***d:*** depth of the least-cost solution
  - ***m*:** maximum depth of the state space (may be ∞)

Breadth-First Search (BFS)

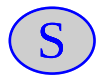
- Expand shallowest unexpanded node
- **Fringe**: nodes waiting in a queue to be explored, also called OPEN
- Implementation:
  - For BFS, *fringe* is a first-in-first-out (FIFO) queue
  - new successors go at end of the queue
- Repeated states?
  - Simple strategy: do not add parent of a node as a leaf

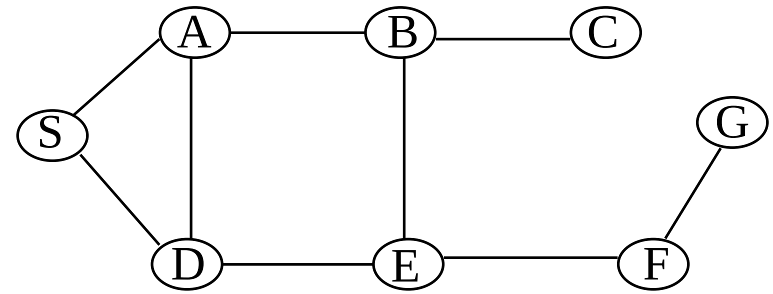
Example: Map Navigation

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



BFS Search Tree





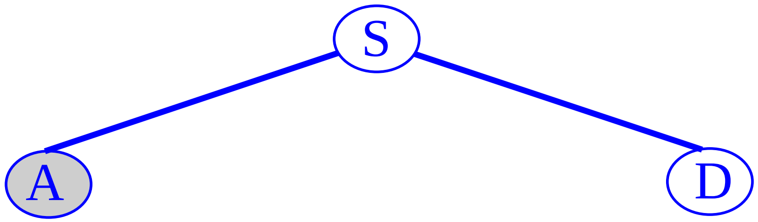
Queue = {S}

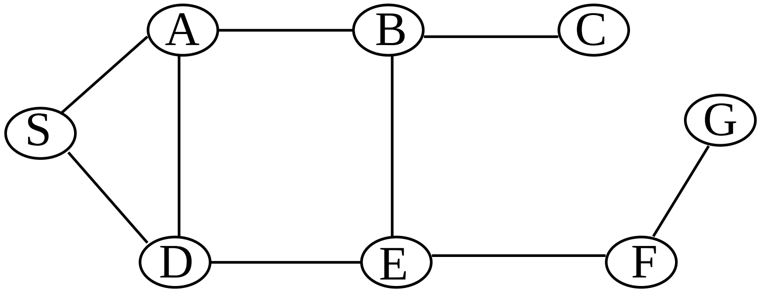
Select S

Goal(S) = true?

If not, Expand(S)

BFS Search Tree





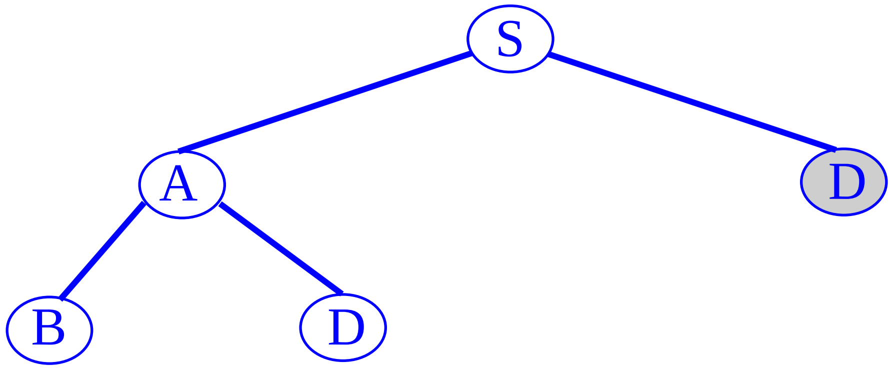
Queue = {A, D}

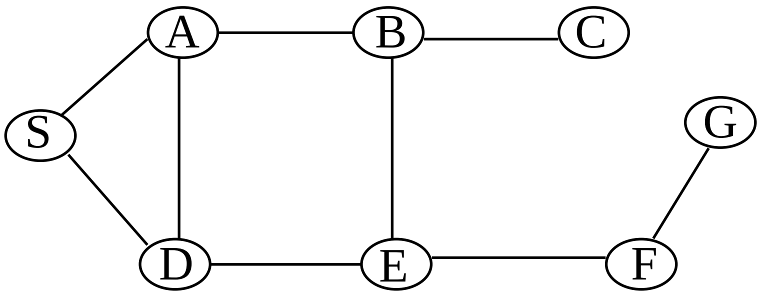
Select A

Goal(A) = true?

If not, Expand(A)

BFS Search Tree





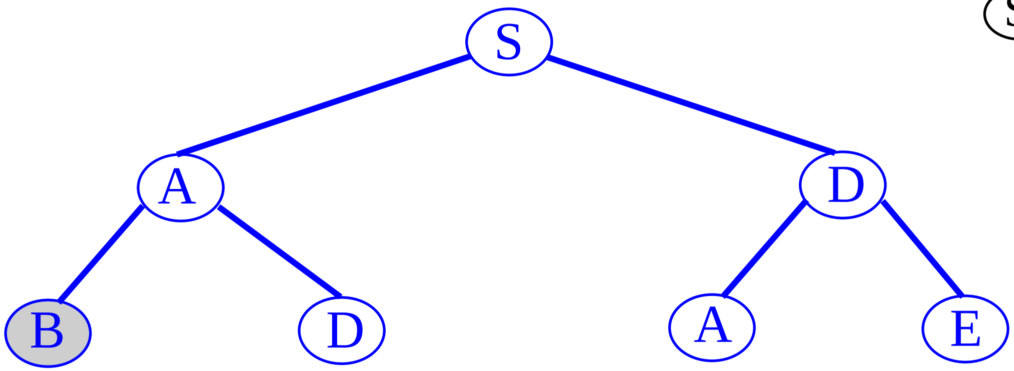
Queue = {D, B, D}

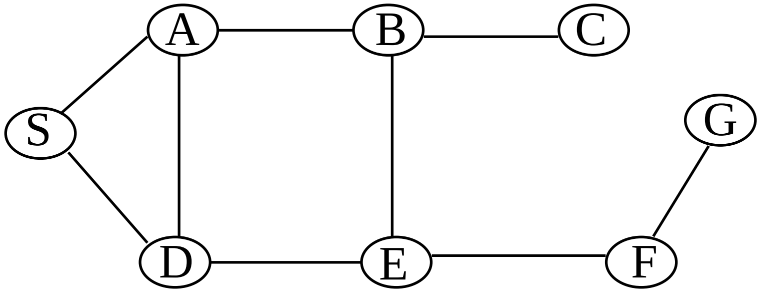
Select D

Goal(D) = true?

If not, expand(D)

BFS Search Tree

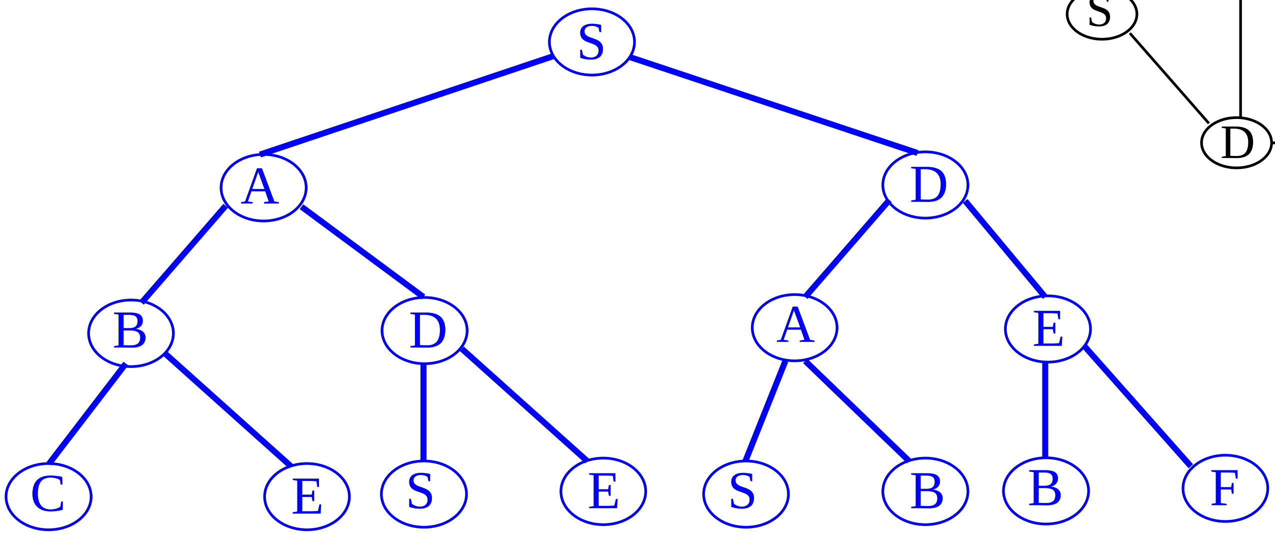


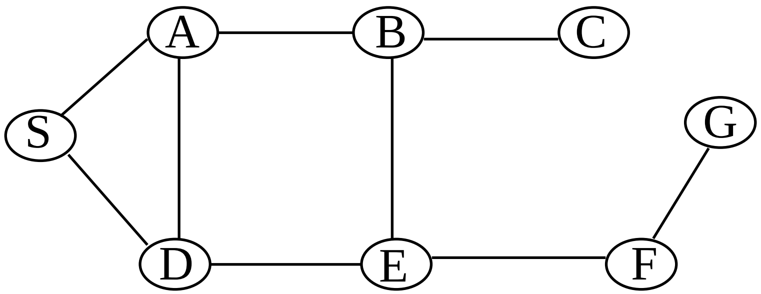


Queue = {B, D, A, E}

Select B
etc.

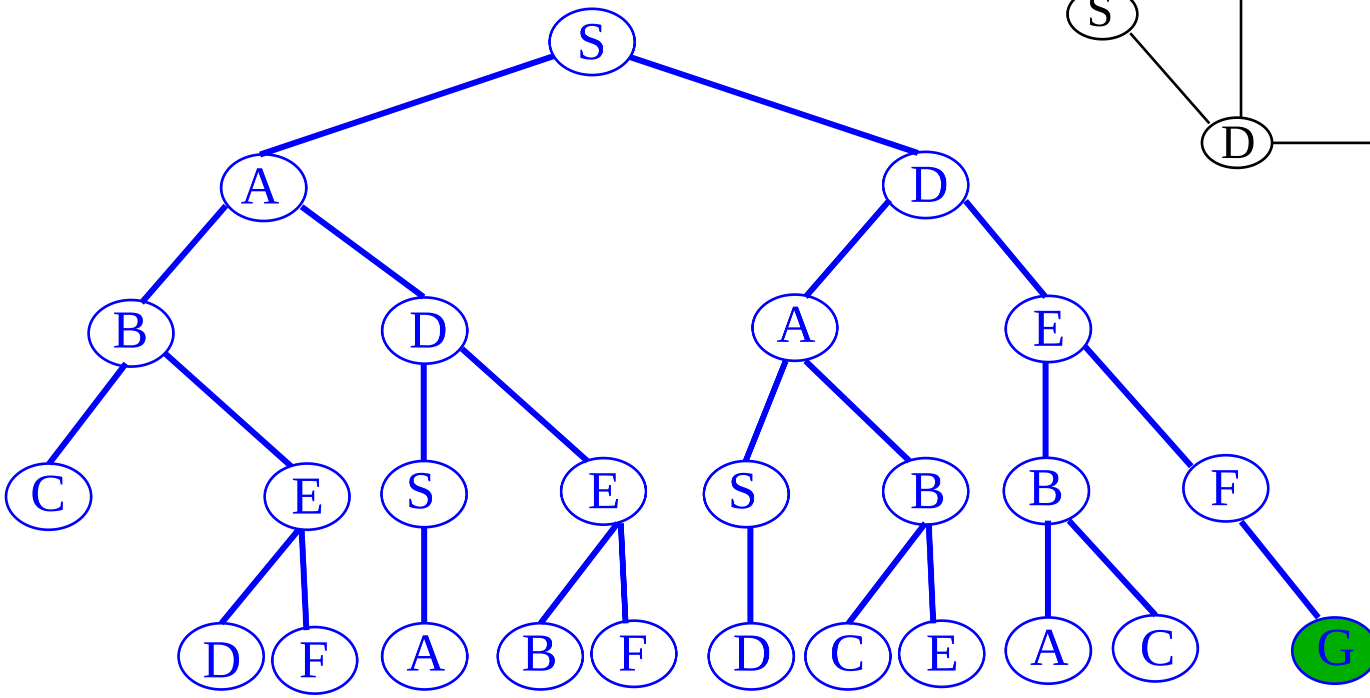
BFS Search Tree

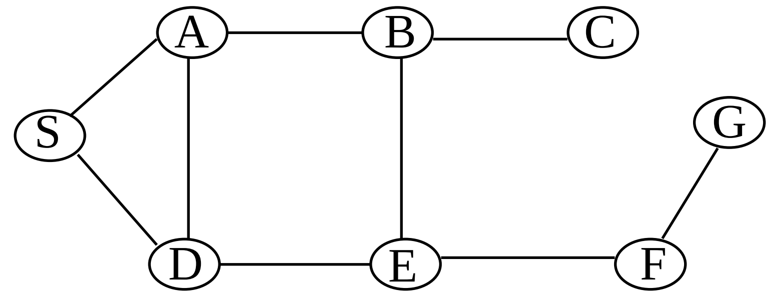




Level 3
Queue = {C, E, S, E, S, B, B, F}

BFS Search Tree





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

Breadth-First Search (BFS) Properties

- Complete?
  Yes

- Optimal?
  Only if path-cost = **non-decreasing function of depth**

- Time complexity
  $O(b^d)$

- Space complexity
  $O(b^d)$

- Main practical drawback?
  exponential time and **space** complexity

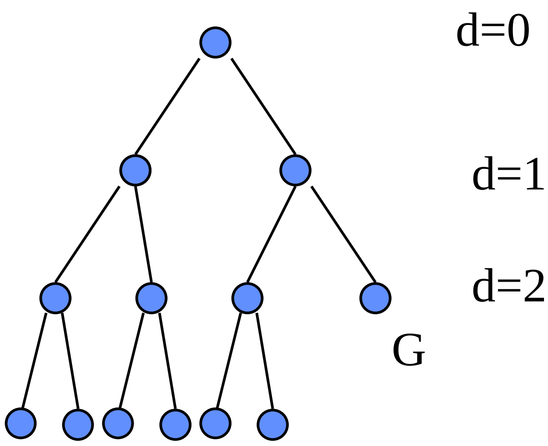
Complexity of Breadth-First Search

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

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

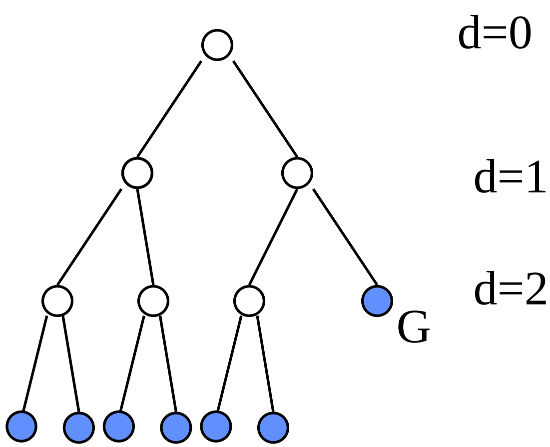
    $= \mathbf{O\ (b^{d+1})}$
  - If goal test is applied **before expansion** then complexity will be:

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



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

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



Examples of Time and Memory Requirements for Breadth-First Search

Assuming b=10,   10,000 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

Breadth-First Search (BFS) Algorithm

Important for Implementation

```
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)
```

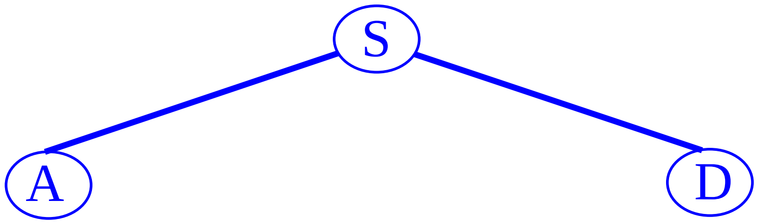
Depth-First Search (DFS)

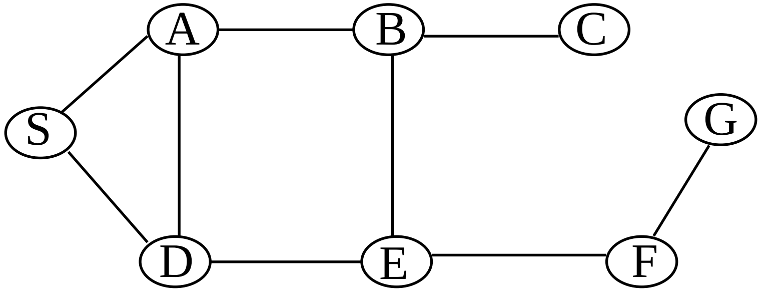
- Expand **deepest** unexpanded node

- Implementation:
  - For DFS, *fringe* is a **Last-in-first-out (LIFO)** queue
  - new successors go at beginning of the queue

- **Repeated nodes**?
  - Simple strategy: Do not add a state in the fringe if that state is already on the path from the root to the current node

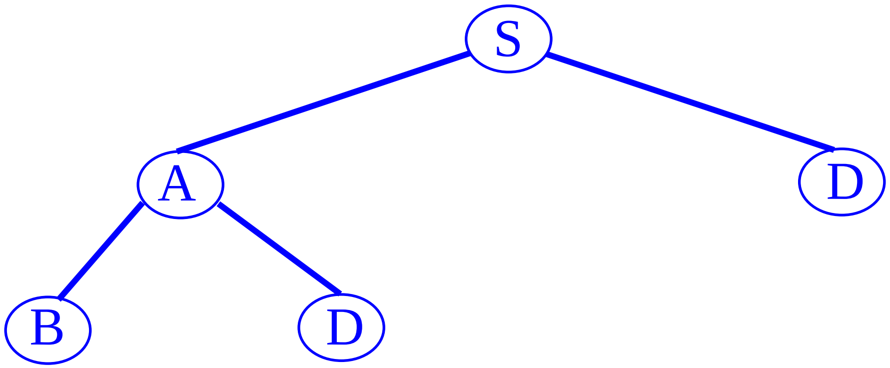
DFS Search Tree

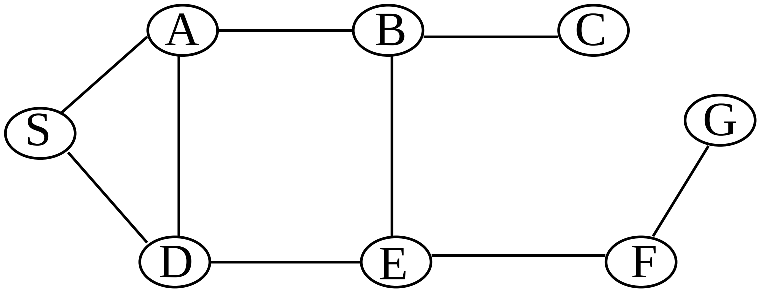




Queue = {A,D}

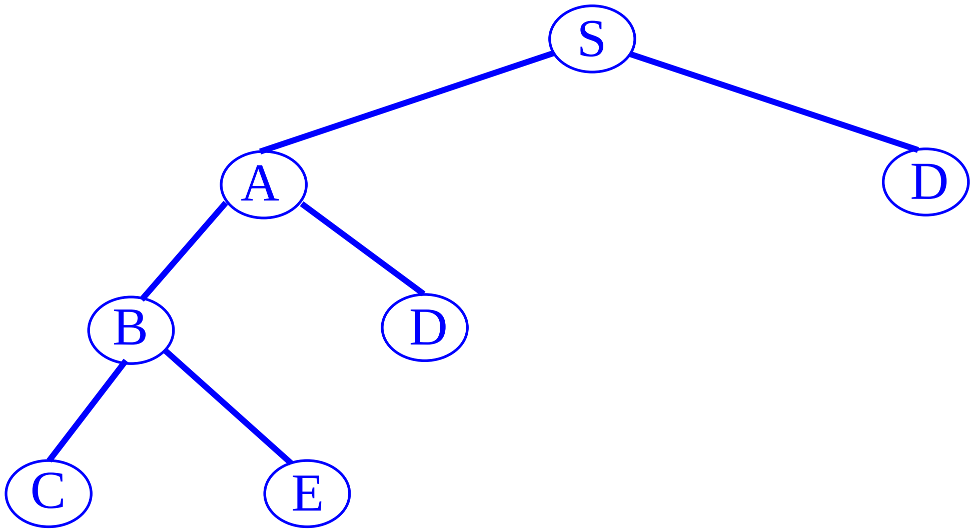
DFS Search Tree

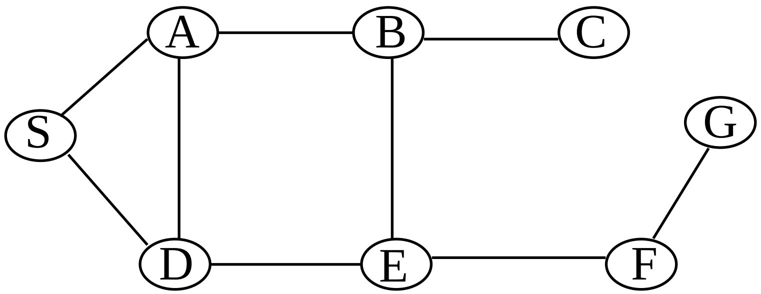




Queue = {B,D,D}

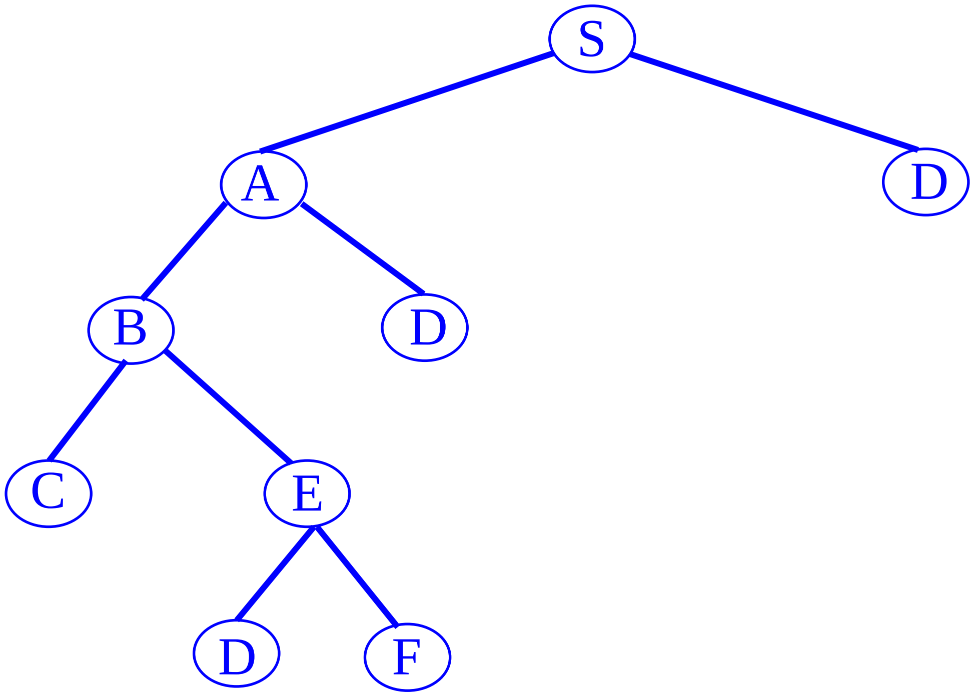
DFS Search Tree

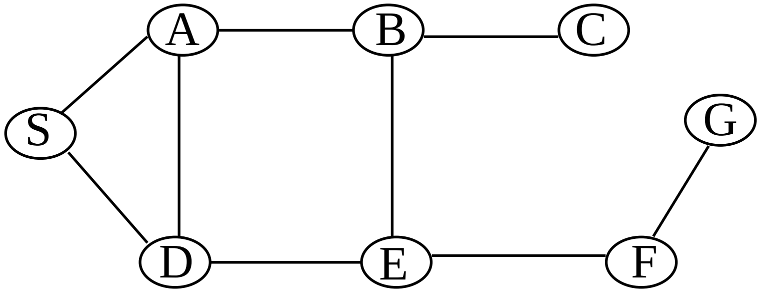




Queue = {C,E,D,D}

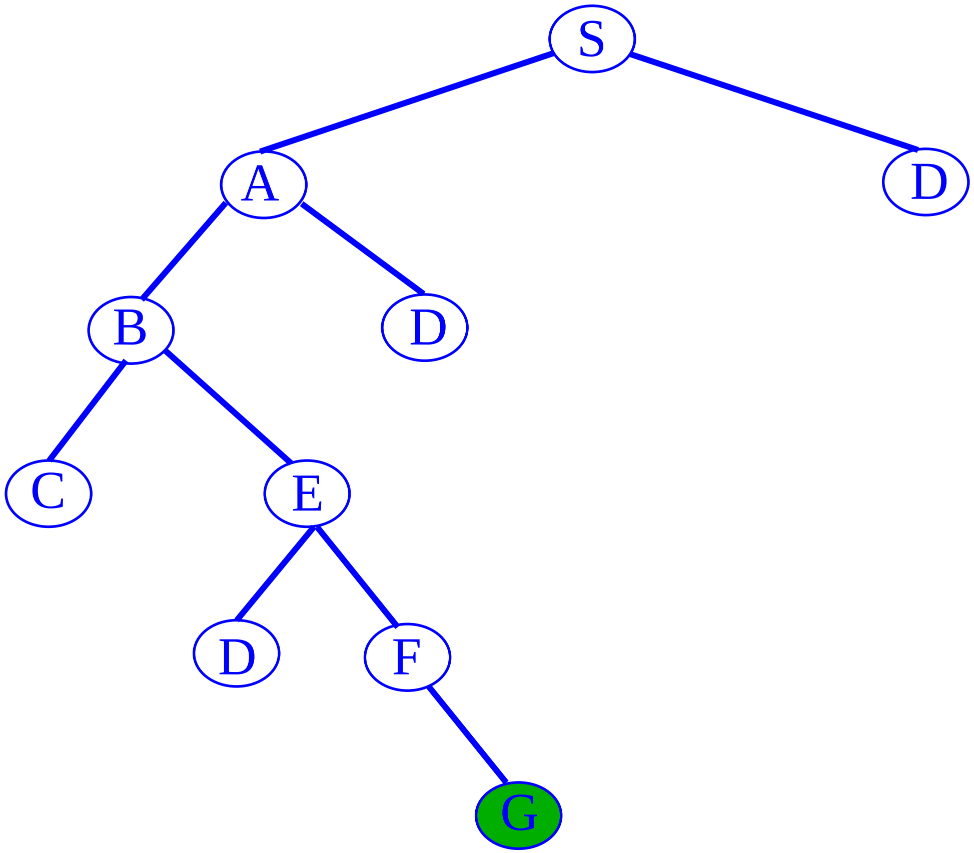
DFS Search Tree

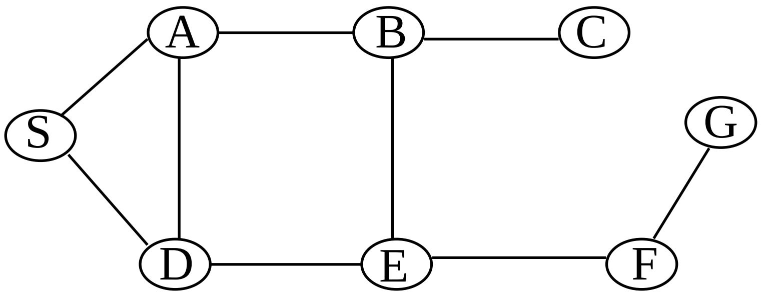




Queue = {D,F,D,D}

DFS Search Tree

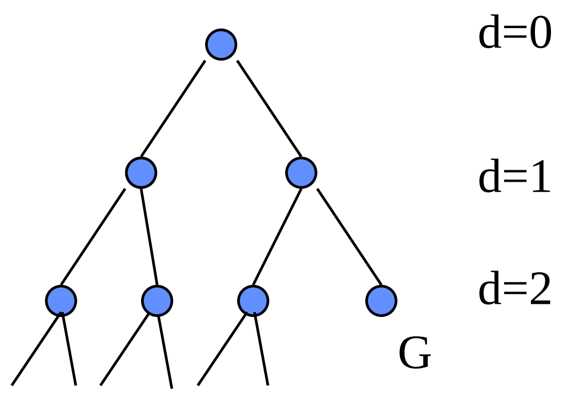




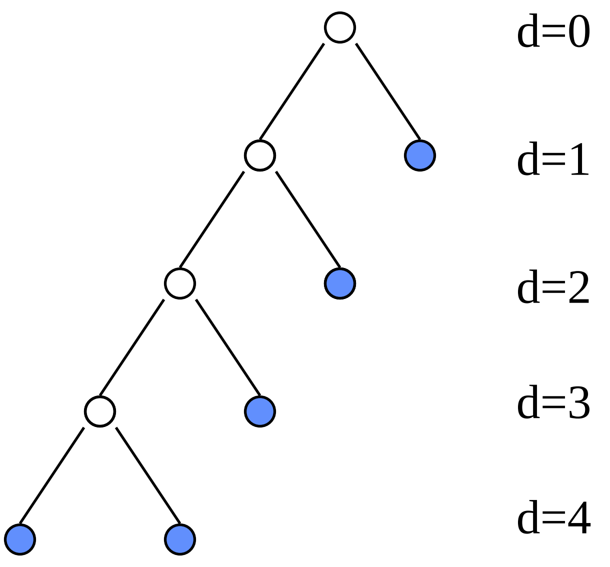
Queue = {G,D,D}

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, 10,000 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?
  - Not complete **if tree has unbounded depth**
- Optimal?
  - **No**
- Time complexity?
  - **Exponential**
- Space complexity?
  - **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**

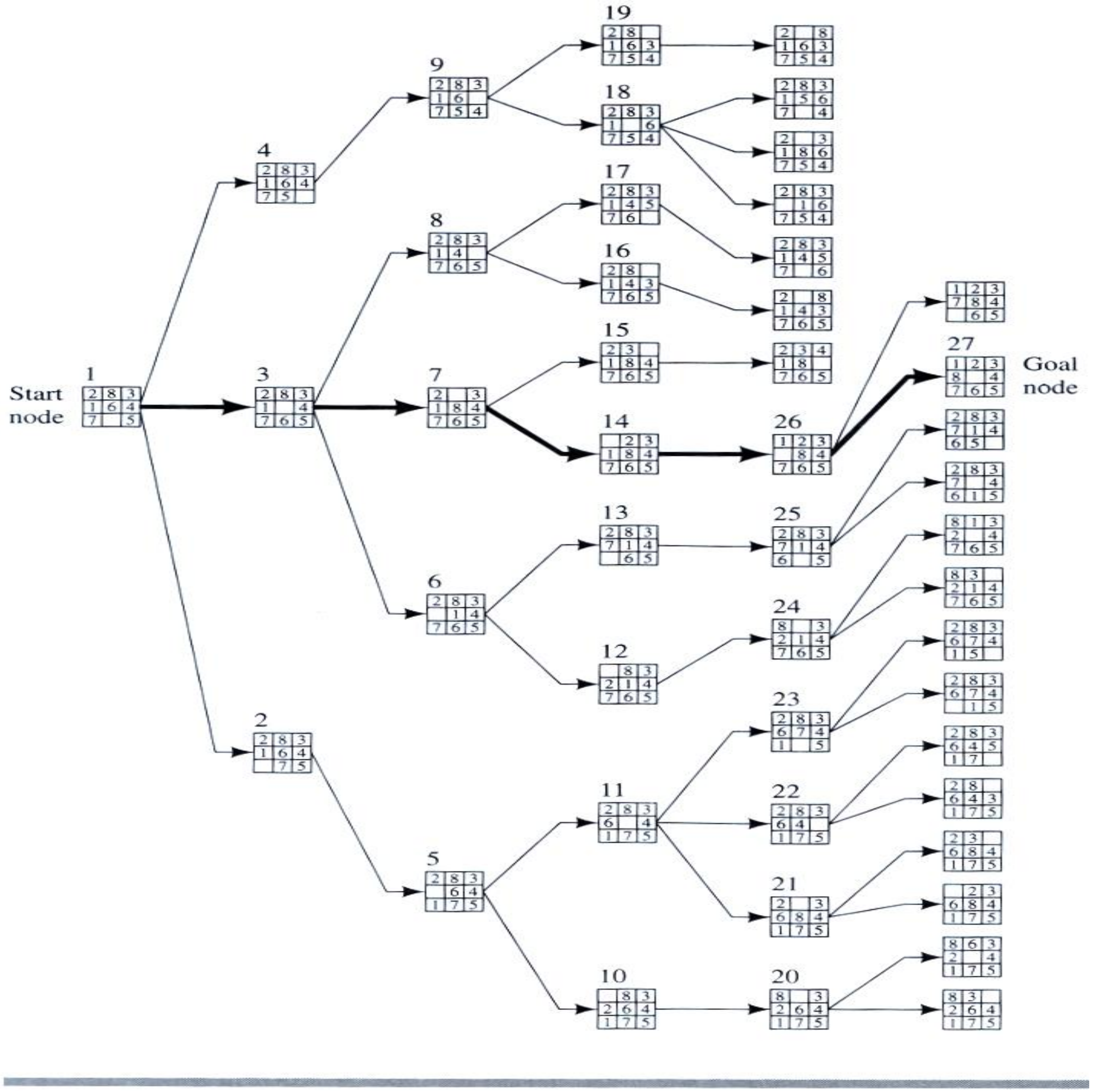
DFS with a depth-limit L

- Standard DFS, but tree is not explored below some depth-limit L

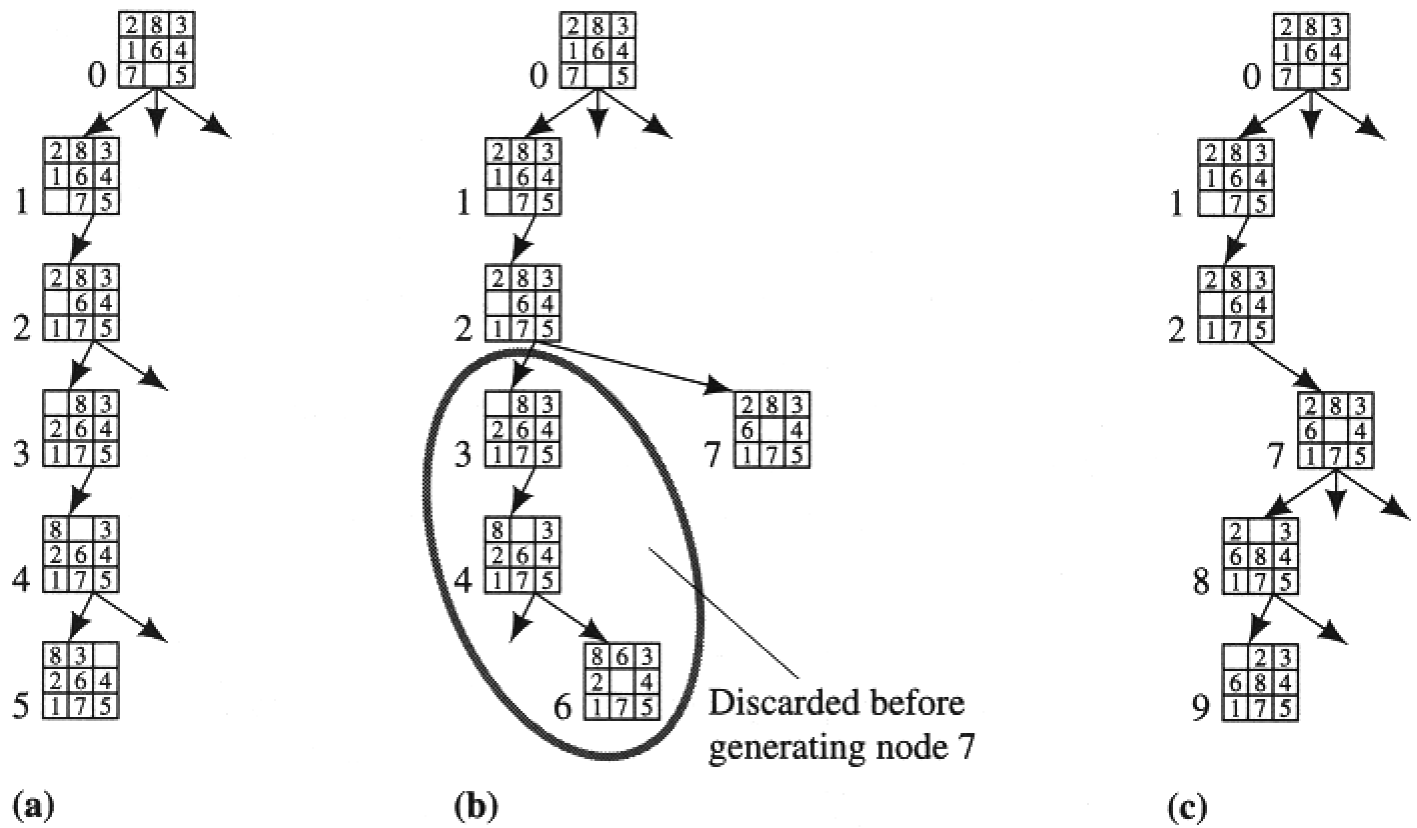
- Solves problem of infinitely deep paths with no solutions
  - But **will be incomplete** if solution is below depth-limit

- Depth-limit **L** can be selected based on **problem knowledge**
  - E.g., diameter of state-space:
    - E.g., max number of steps between 2 cities
  - But typically not known ahead of time in practice ☹

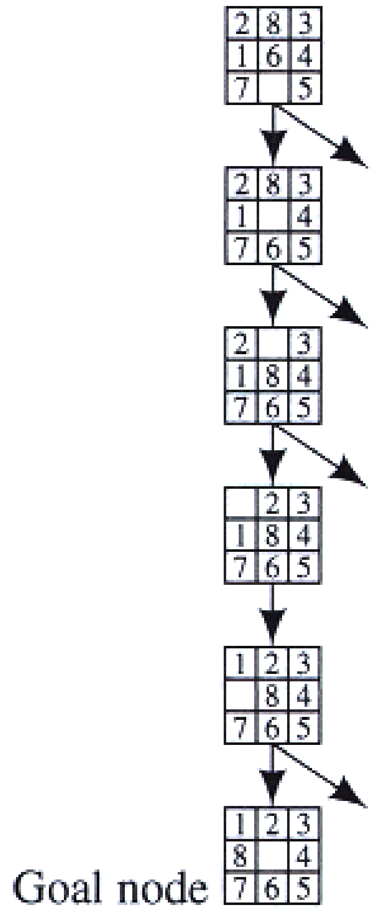
Depth-First Search with a depth-limit, L = 5



Depth-First Search with a depth-limit



Generation of the First Few Nodes in a Depth-First Search



The Graph When the Goal Is Reached in Depth–First Search

Iterative Deepening Search (IDS)

- Run multiple DFS searches with increasing depth-limits

- Iterative deepening search
  - L = 0
  - **While** no solution, **do**

    DFS from initial state S_0 with cutoff L

    If found goal,

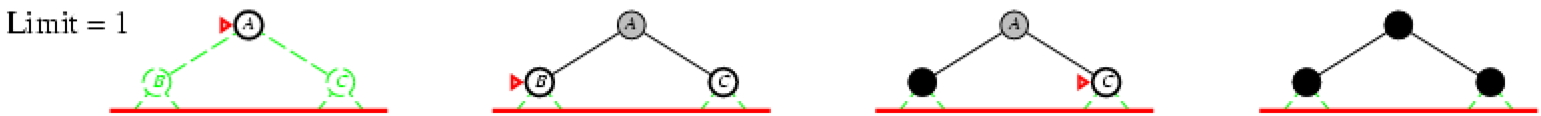
    stop and return solution,

    else, increment depth limit L

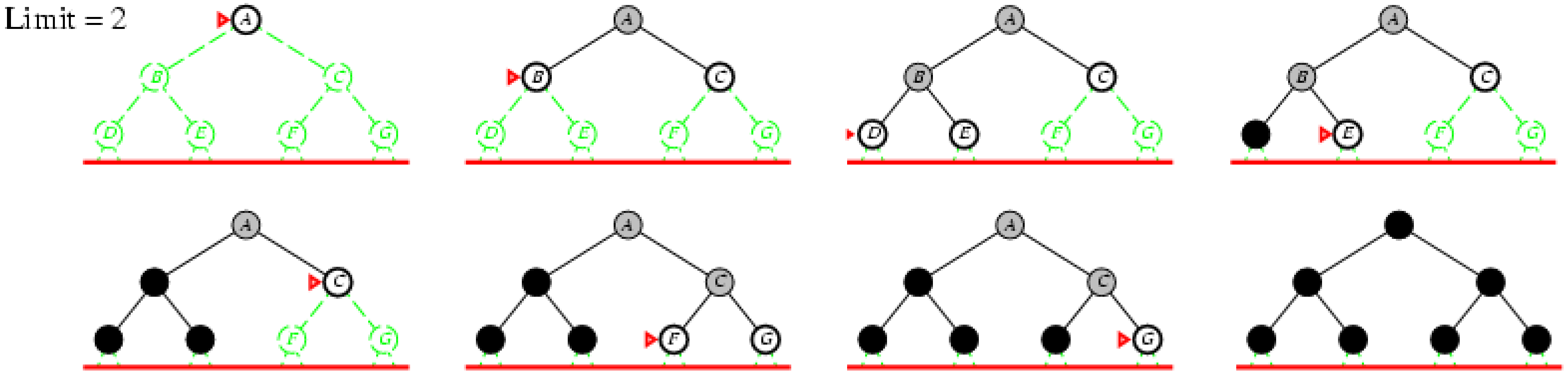
Iterative deepening search *L*=0

Limit = 0

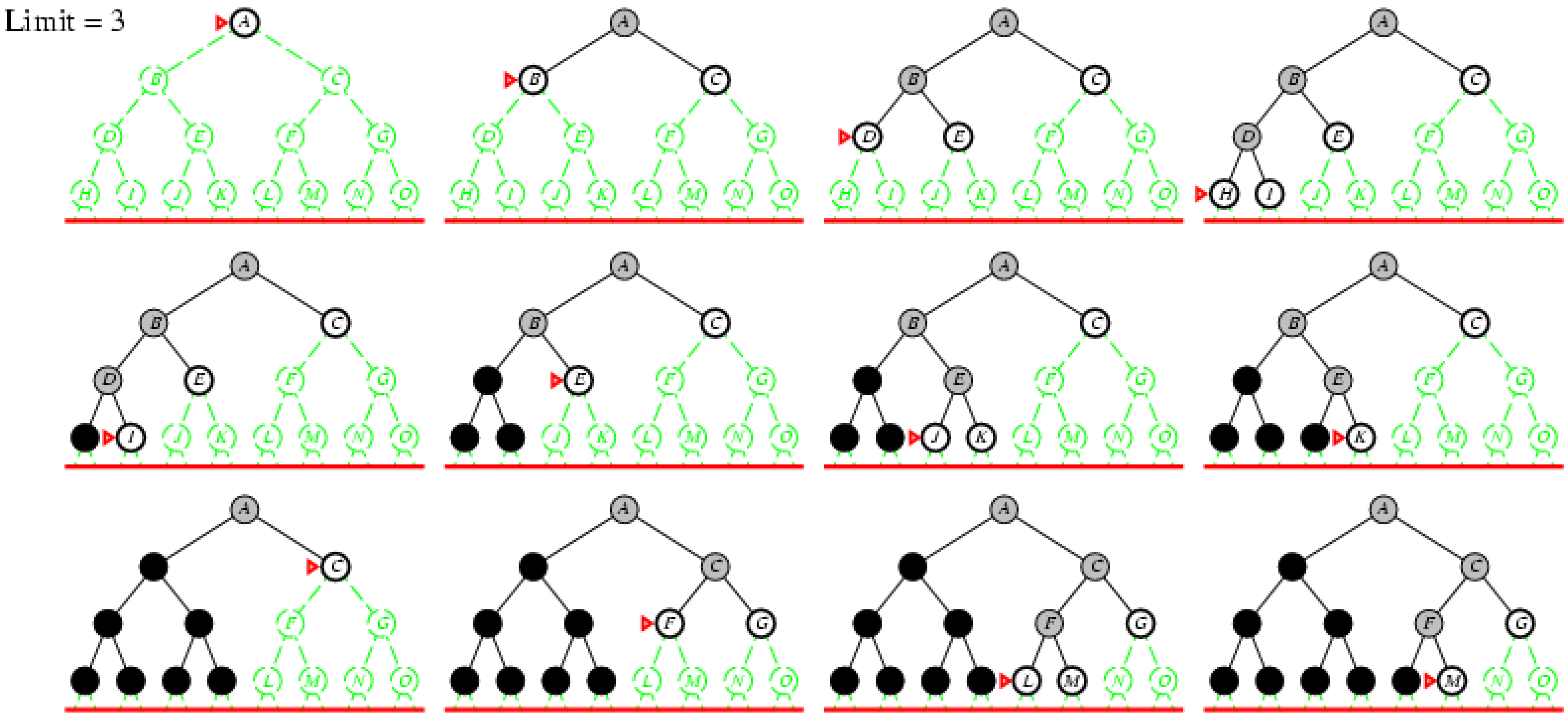
Iterative deepening search *L*=1



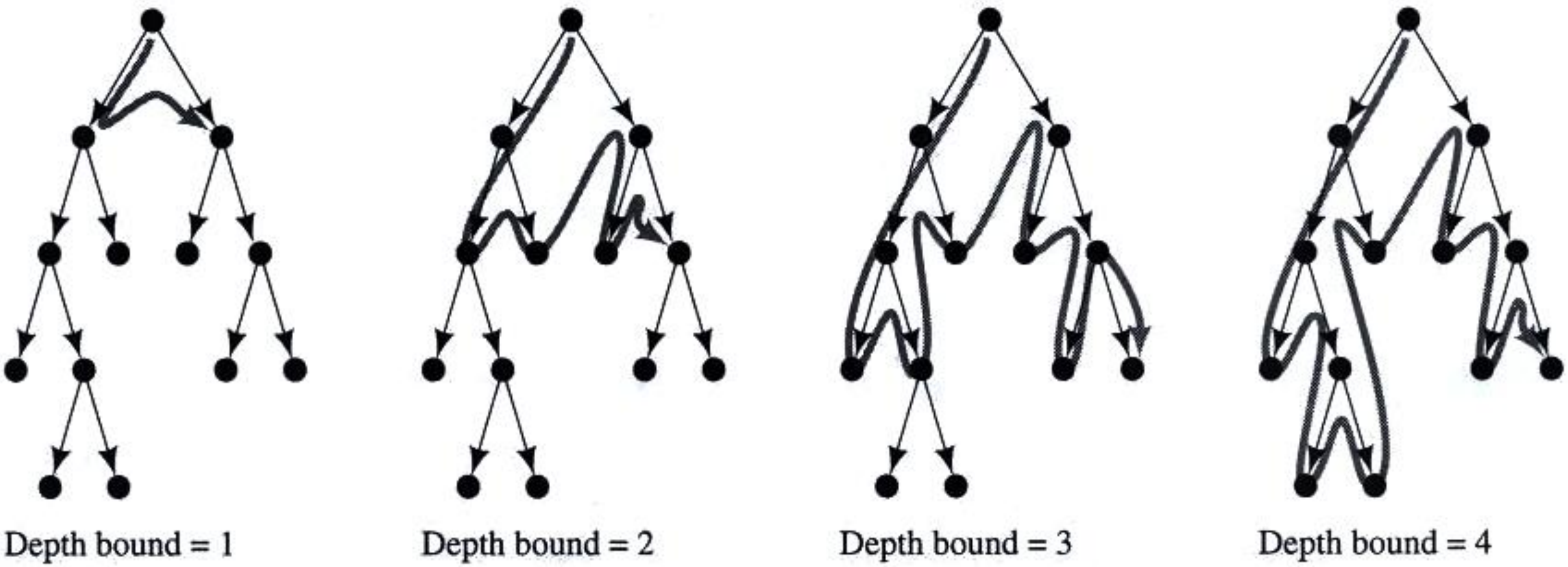
Iterative deepening search $L=2$



Iterative Deepening Search *L*=3



Iterative deepening search



Stages in Iterative-Deepening Search

Properties of Iterative Deepening Search

- Space complexity = **O(bd)**
  - (since its like depth first search run different times, with maximum depth limit d)
- Time Complexity
  - **$(d)\ b + (d-1)\ b^2 + ... + (1)\ b^d = O(b^d)$**
    (i.e., asymptotically the same as BFS or DFS to limited depth d in the worst case)
- **Complete**?
  - **Yes**
- **Optimal**
  - **Only if path cost is a non-decreasing function of depth**
- IDS combines the small memory footprint of DFS, and has the completeness guarantee of BFS

IDS in Practice

- Isn't IDS wasteful?
  - Repeated searches on different iterations

  - **Compare IDS and BFS:**
    - **E.g., b = 10 and d = 5**

    - **N(IDS) ~ $db + (d-1)b^2 + \ldots\ldots b^d = 123{,}450$**

    - **N(BFS) ~ $b + b^2 + \ldots\ldots b^d = 111{,}110$**

    - **Difference is only about 10%**
      - Most of the time is spent at depth d, which is the same amount of time in both algorithms

- In practice, IDS is the **preferred uniform search method** with a **large search space** and **unknown solution depth**

Uniform Cost Search

- Optimality: path found = lowest cost
  - Algorithms so far are only optimal under restricted circumstances

- Let **g(n)** = cost from start state S to node n

- Uniform Cost Search:
  - **Always expand the node on the fringe with minimum cost g(n)**

  - **Note that if costs are equal (or almost equal) will behave similarly to BFS**

The reason why it is called uniform cost search is because at any given point in time, the priority queue is filled with path costs that are mostly uniform.

Uniform Cost Search

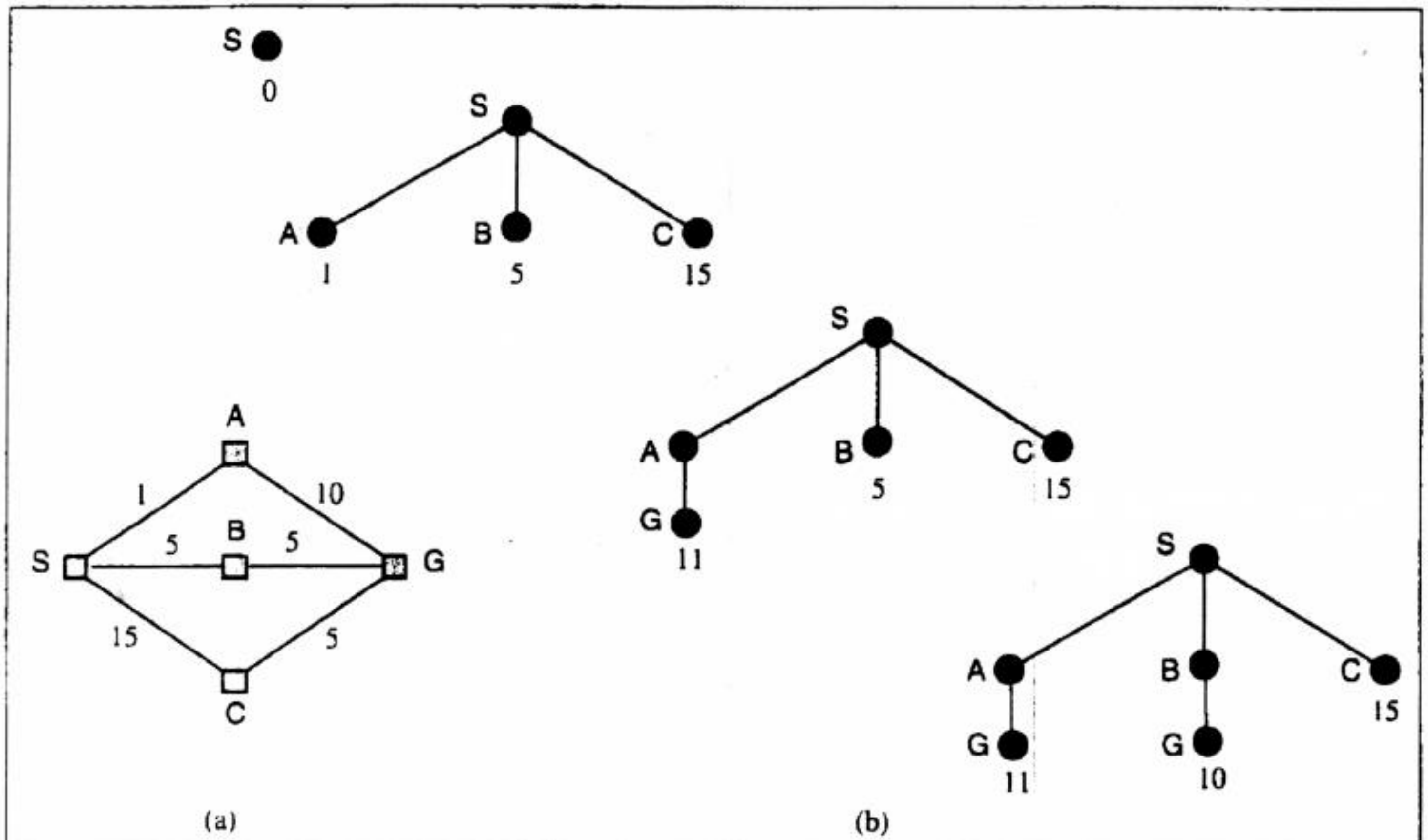


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Optimality of Uniform Cost Search?

- Assume that every step costs at least $\varepsilon > 0$
- **Proof of Completeness**:

  Given that every step will cost more than 0, and assuming a **finite branching factor**, there is a **finite number of expansions** required before the **total path cost** is equal to the **path cost of the goal state**. Hence, we will reach it in a finite number of steps.
- **Proof of Optimality given Completeness**:
  - Assume UCS is not optimal.
  - Then there must be a goal state with path cost smaller than the goal state which was found (invoking completeness)
  - However, this is impossible because UCS would have expanded that node first by definition.
  - Contradiction.

Optimality of Uniform Cost Search?

- Assume that every step costs at least $\varepsilon > 0$
- **Proof of Completeness**:

  Given that every step will cost more than 0, and assuming a **finite branching factor**, there is a **finite number of expansions** required before the **total path cost** is equal to the **path cost of the goal state**. Hence, we will reach it in a finite number of steps.

- **Proof of Optimality given Completeness**:
  - OR Alternatively:
  - Suppose UCS terminates at goal state n with path cost **g(n)** but there exists another goal state **n'** with **g(n') < g(n)**
  - By the graph separation property, there must exist a node **n"** on the frontier that is on the optimal path to **n'** But because **g(n") ≤ g(n') < g(n)**, **n"** should have been expanded first!

Complexity of Uniform Cost

- Let C* be the **cost of the optimal solution**
- Assume that every step costs at least $\varepsilon > 0$
- Worst-case **time** and **space** complexity is:

$$\mathbf{O(\ b^{\ [1\ +\ floor(C^*/\varepsilon)]}\ )}$$

Why?

floor(C*/ε) ~ depth of solution if all costs are approximately equal

- This **can be greater** than **O(b^d)**: the search can explore long paths consisting of small steps before exploring shorter paths consisting of larger steps

Bi-Directional Search

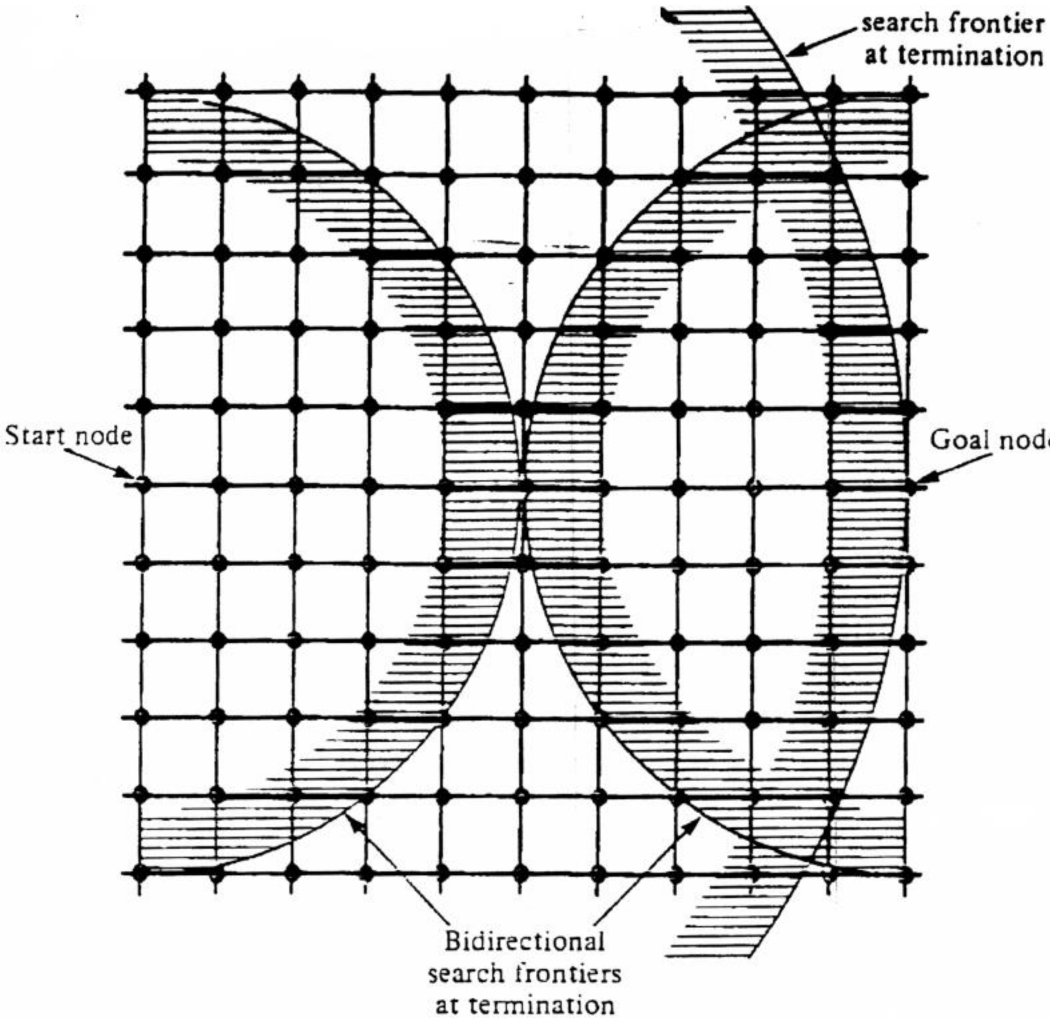


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Bidirectional Search

- Idea
  - simultaneously search forward from S and backwards from G
  - stop when both "meet in the middle"
  - need to keep track of the intersection of 2 open sets of nodes

- What does searching backwards from G mean
  - need a way to specify the predecessors of G
    - this can be difficult,
    - e.g., predecessors of checkmate in chess?
  - **what if there are multiple goal states?**
  - **what if there is only a goal test, no explicit list? E.g. Traveling Salesman Problem**

- Complexity
  - time complexity at best is: $O(2\ b^{(d/2)}) = O(b^{(d/2)})$
  - memory complexity is the same

Comparison of Uninformed Search Algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

Summary

- A review of search
  - a search space consists of states and operators: it is a graph
  - a search tree represents a particular exploration of search space
- There are various strategies for “uninformed search”
  - breadth-first
  - depth-first
  - iterative deepening
  - bidirectional search
  - Uniform cost search
- Various trade-offs among these algorithms
  - “best” algorithm will depend on the nature of the search problem
- Next up – heuristic search methods