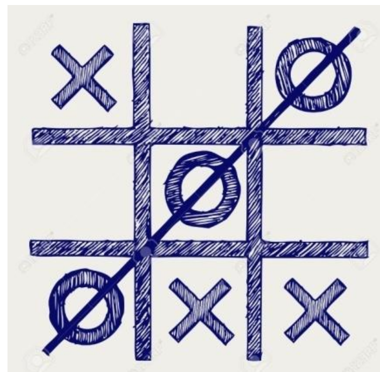
CSC 422

Class #3

Problem Solving as Search

Why searching in AI?



May be you all are familiar with the board game Tic-Tac-Toe.

As an AI student, you want design an intelligent agent, which can play this game with you as an opponent.

Why searching in AI?

Intelligent agent should know the current state of the Tic-Tac-Toe board.

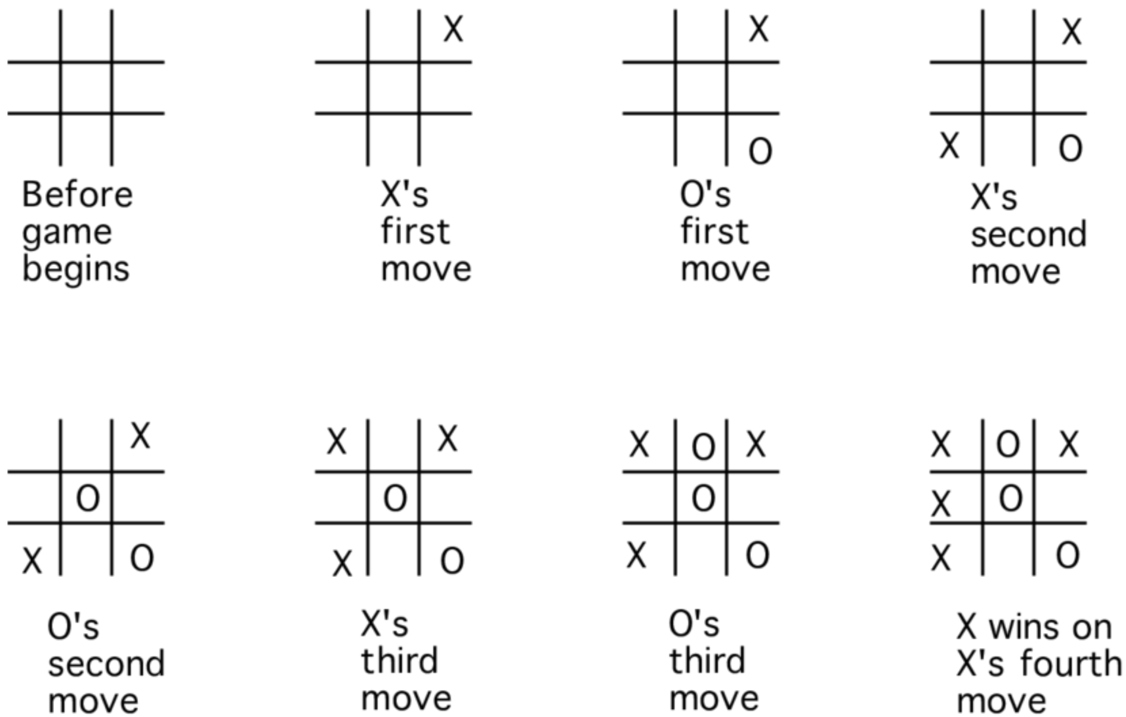


Fig: Some states of Tic-Tac-Toe board.

Why searching in AI?

Intelligent agent should explore the next possible movement and search for the best move to win the game.

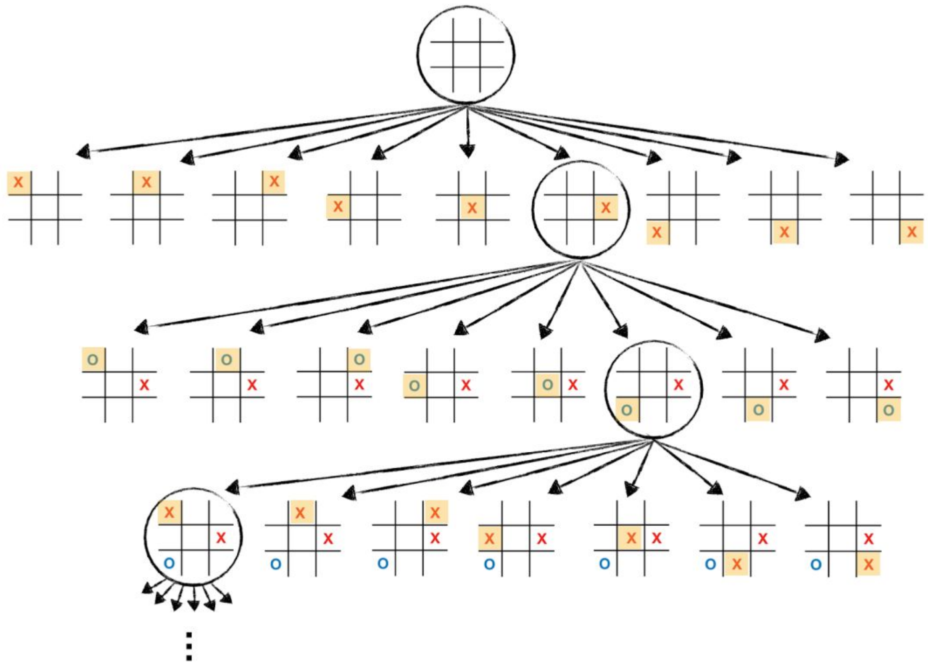


Fig: Game search tree for Tic-Tac-Toe board (partial view)

SEARCH TECHNIQUES

- Search techniques
  - Blind (uninformed Search)
    - Depth first Search (DFS)
    - Breadth first Search (BFS)
  - Heuristic (Informed Search)
    - Hill climbing Search
    - Best-First Search
    - Greedy Search
    - A* search

Other blind search strategies are:

- Depth-limited search (Extended DFS)
- Iterative-deepening search (Extended DFS)
- Uniform cost search
- Bi-directional search

Uninformed Vs Informed Search

Uninformed search: Use only the information available in the problem definition. Example: breadth-first, depth-first, depth limited, iterative deepening, uniform cost and bidirectional search

Informed search: Use domain knowledge or heuristic to choose the best move. Example. Greedy best-first, A*, IDA*, and beam search

Additional Note:
optimization in which the search is to find an optimal value of an objective function: hill climbing, simulated annealing, genetic algorithms, Ant Colony Optimization

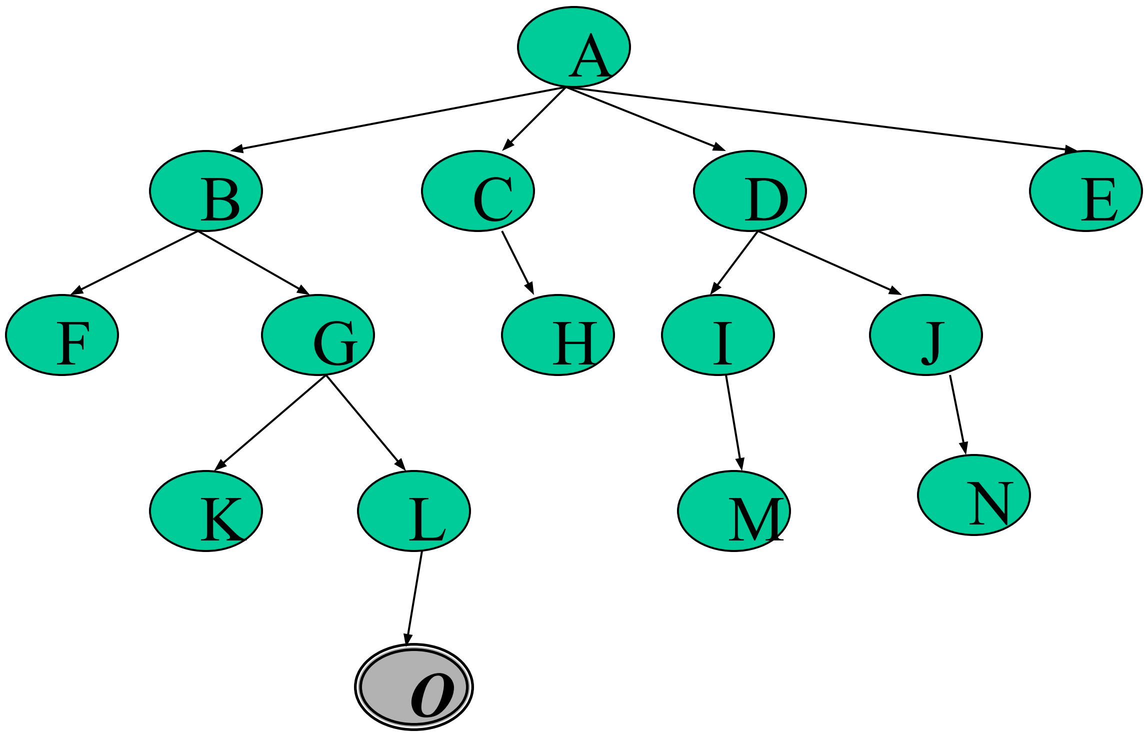
Game playing, an adversarial search: minimax algorithm, alpha-beta pruning

Uninformed Search

Breadth First Search

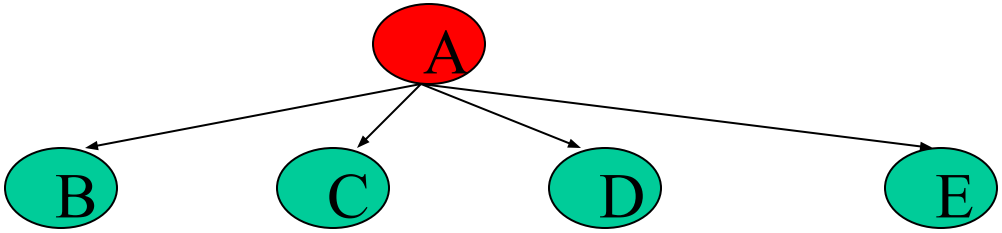
- Application1:

  Given the following state space (tree search), give the sequence of visited nodes when using BFS (assume that the node ***O*** is the goal state):



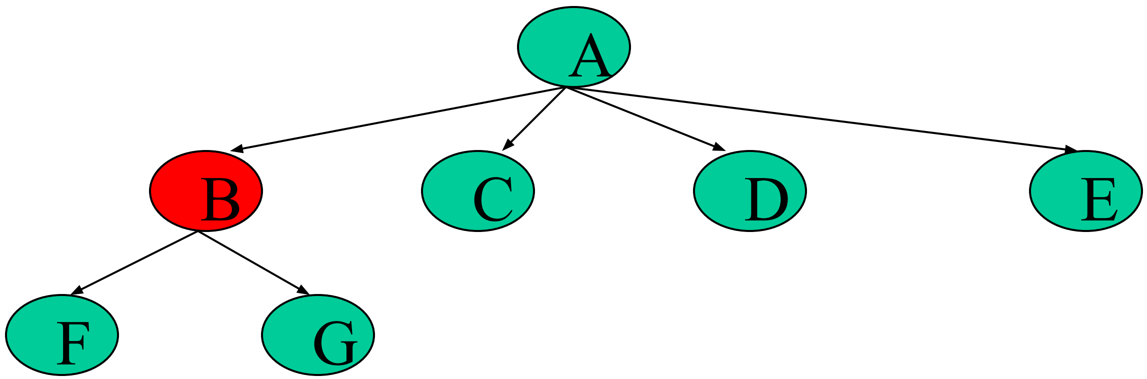
Breadth First Search

- A,



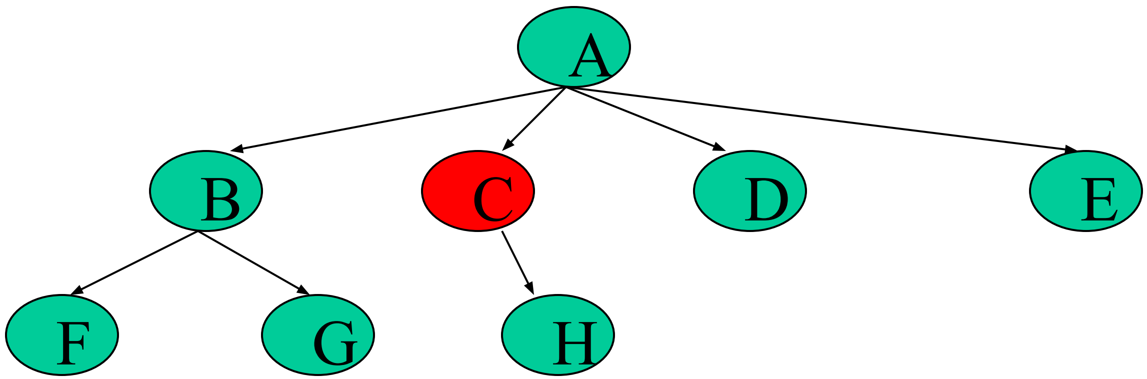
Breadth First Search

- A,
- B,



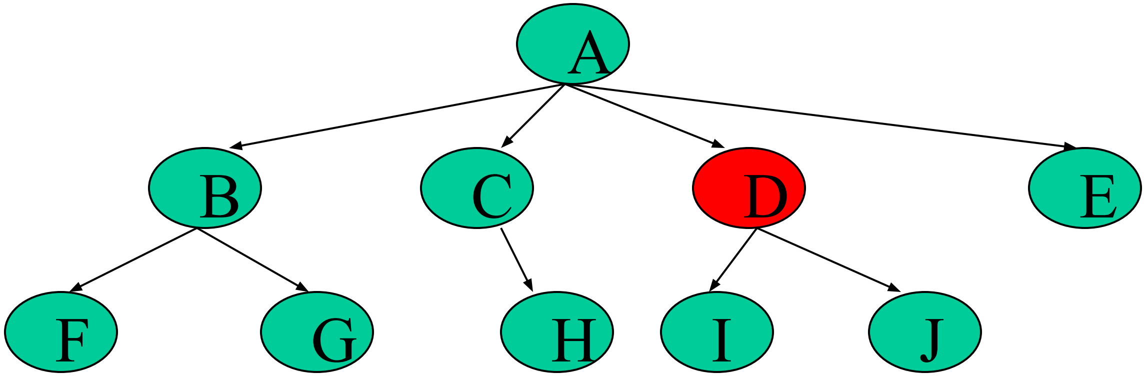
Breadth First Search

- A,
- B,C



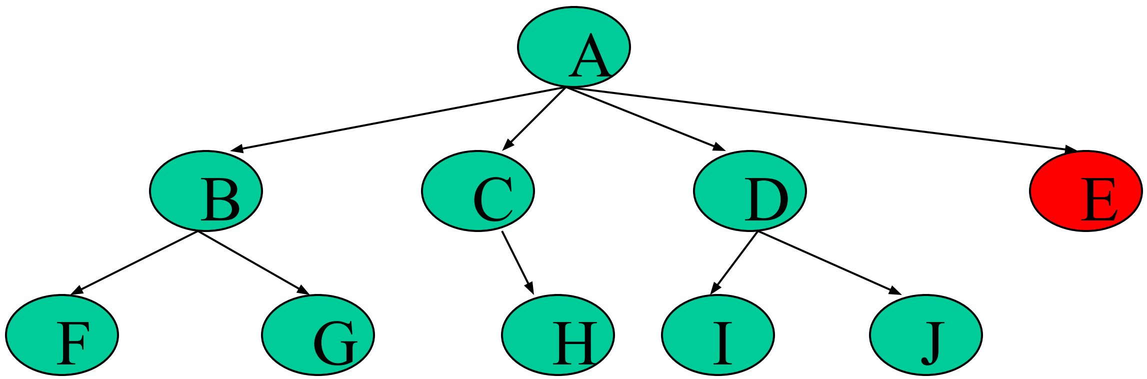
Breadth First Search

- A,
- B,C,D



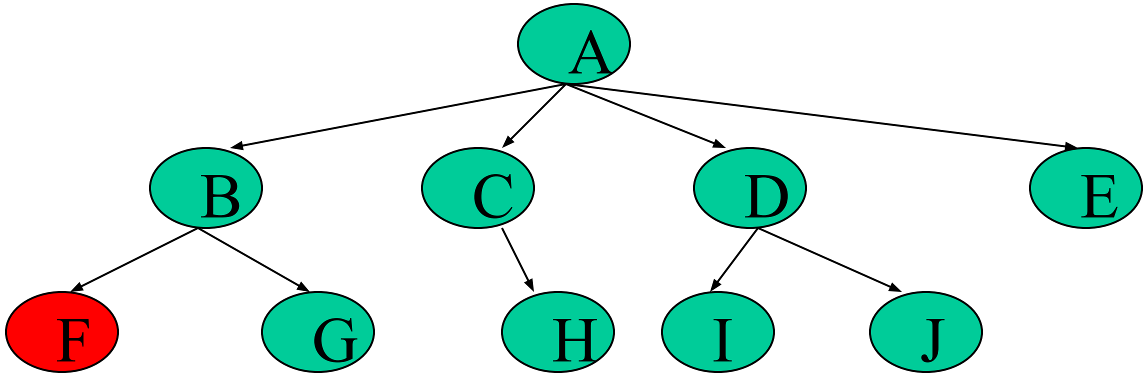
Breadth First Search

- A,
- B,C,D,E



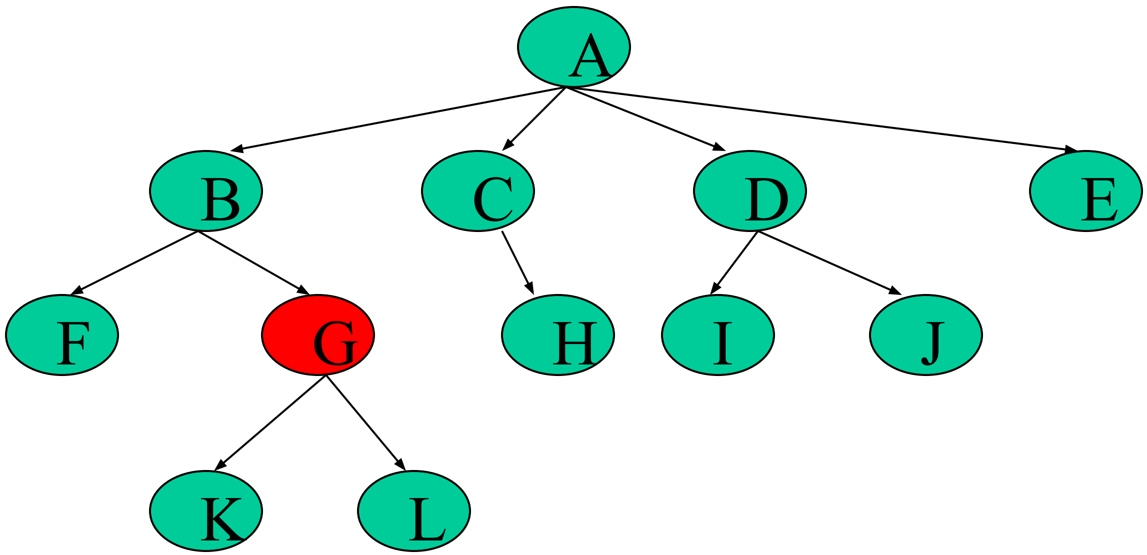
Breadth First Search

- A,
- B,C,D,E,
- F,



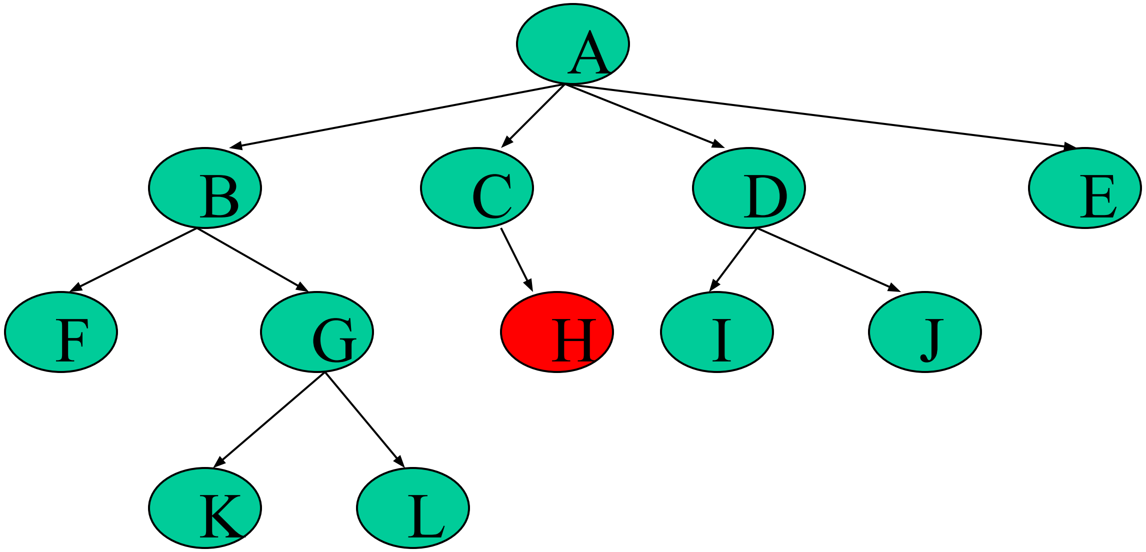
Breadth First Search

- A,
- B,C,D,E,
- F,G



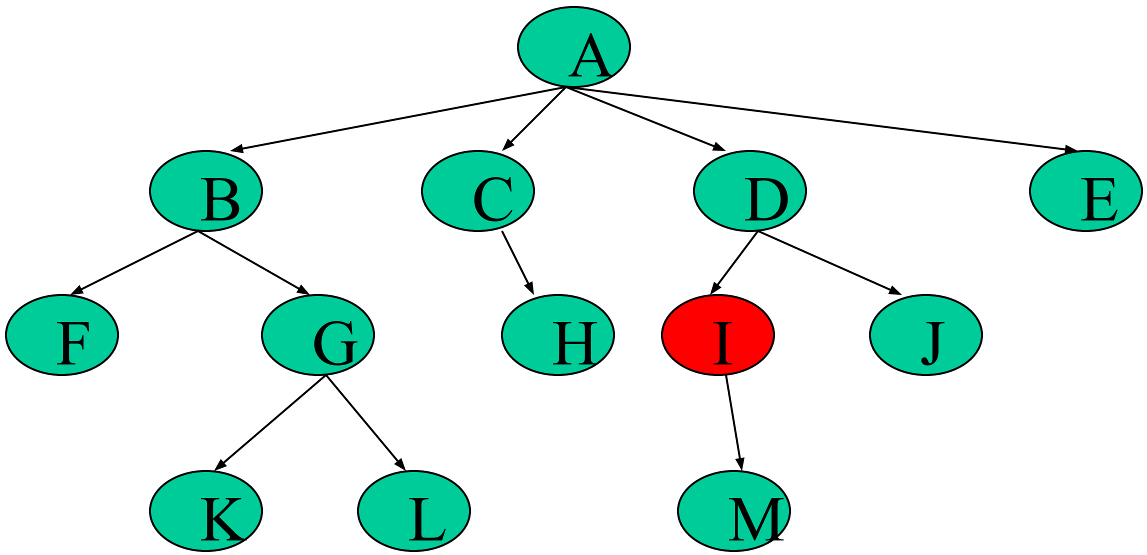
Breadth First Search

- A,
- B,C,D,E,
- F,G,H



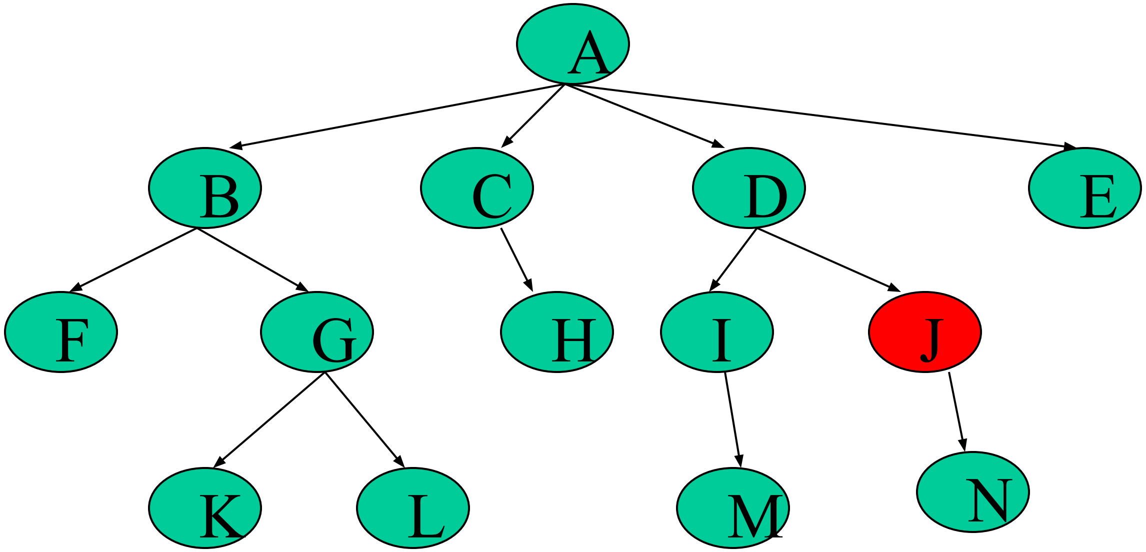
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I



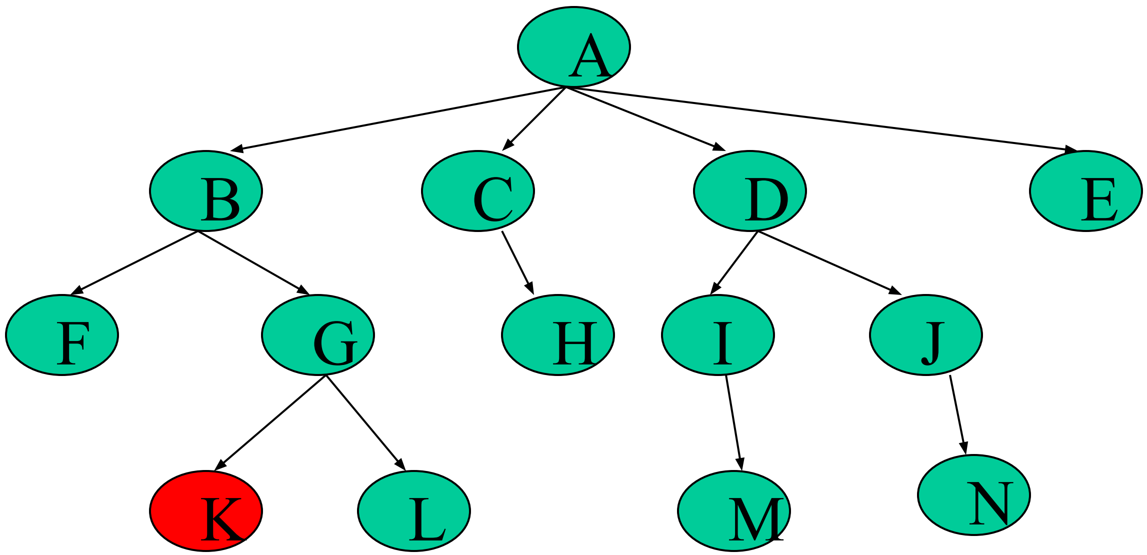
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,



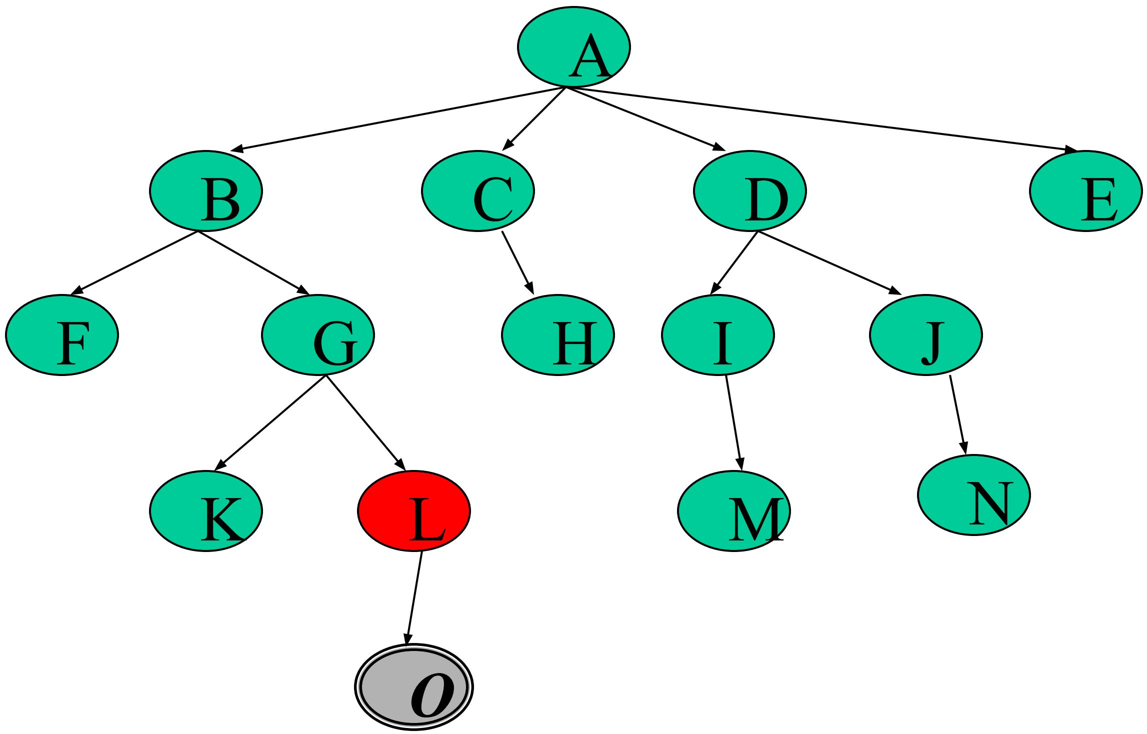
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,
- K,



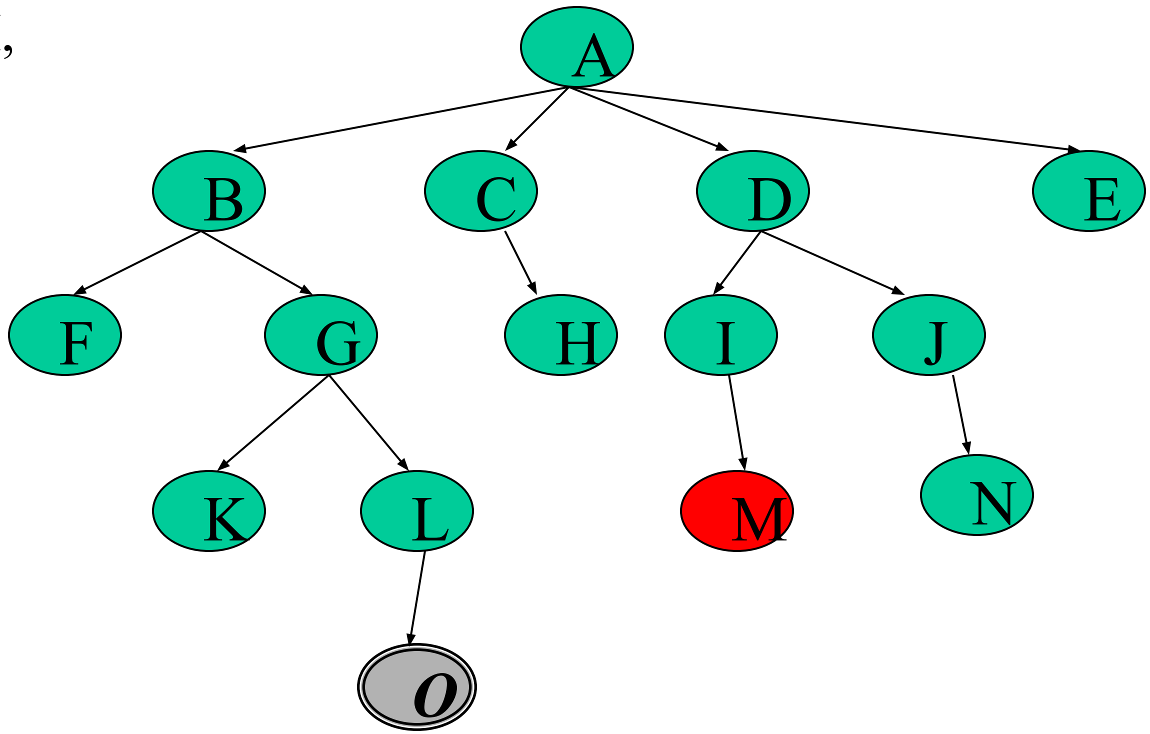
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,
- K,L



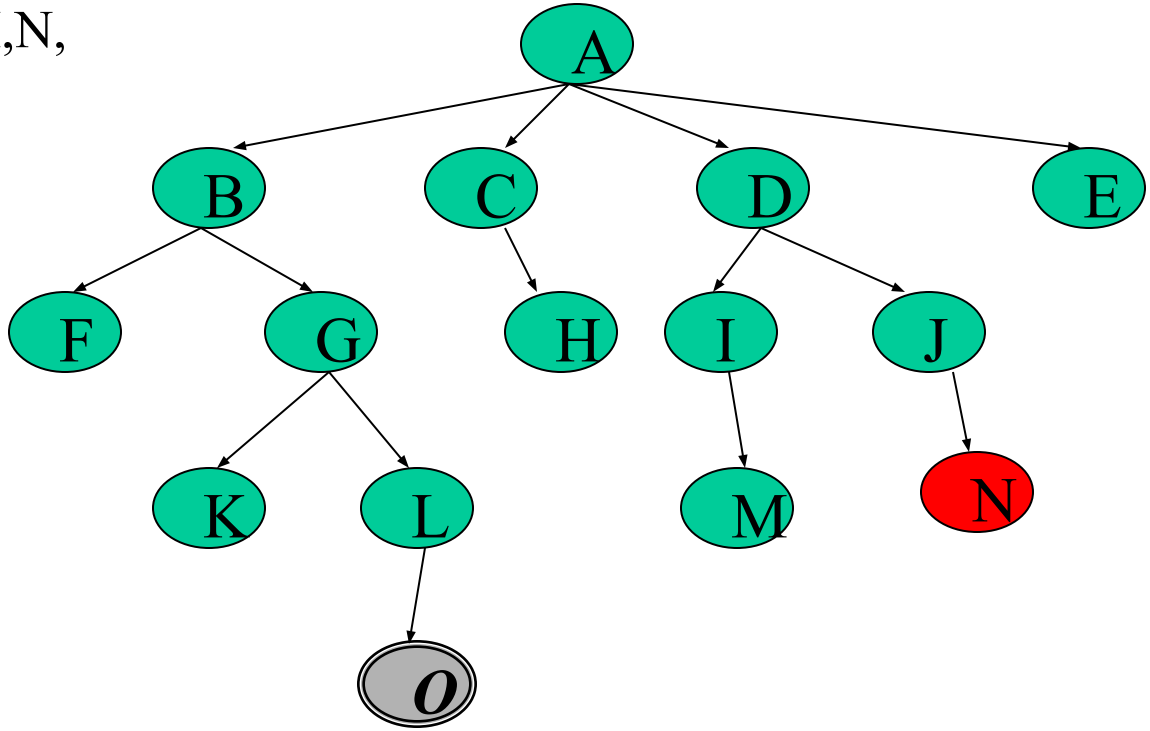
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,
- K,L, M,



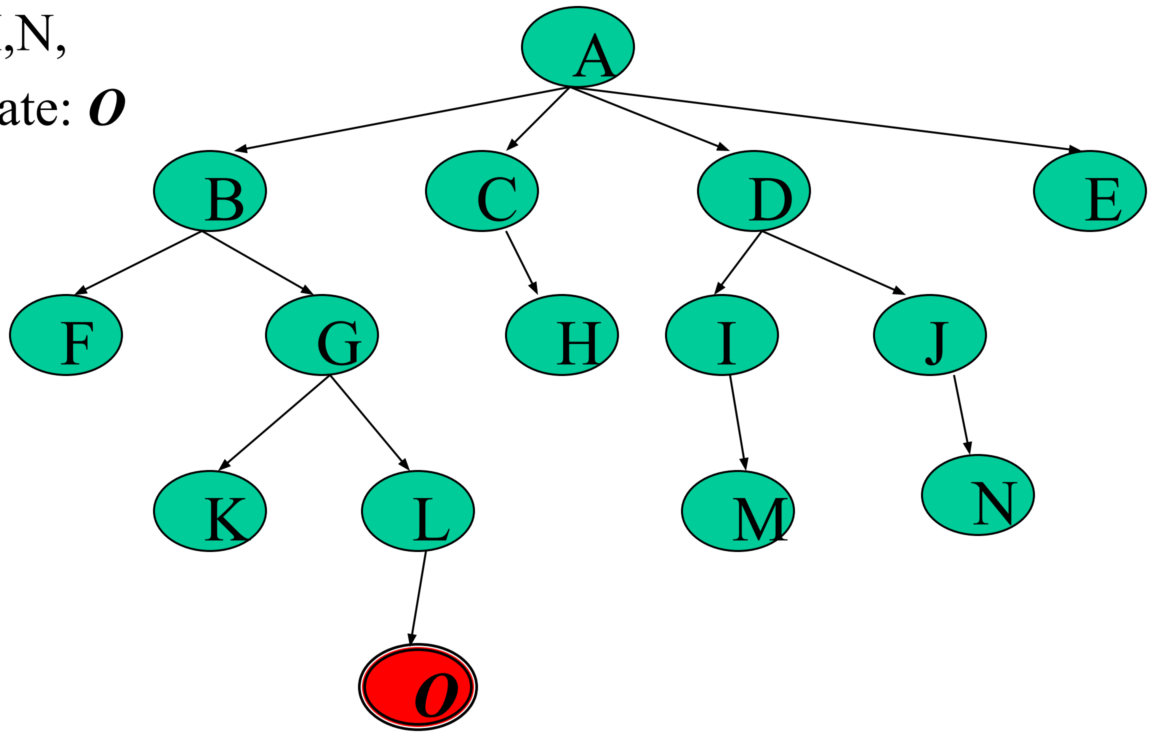
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,
- K,L, M,N,



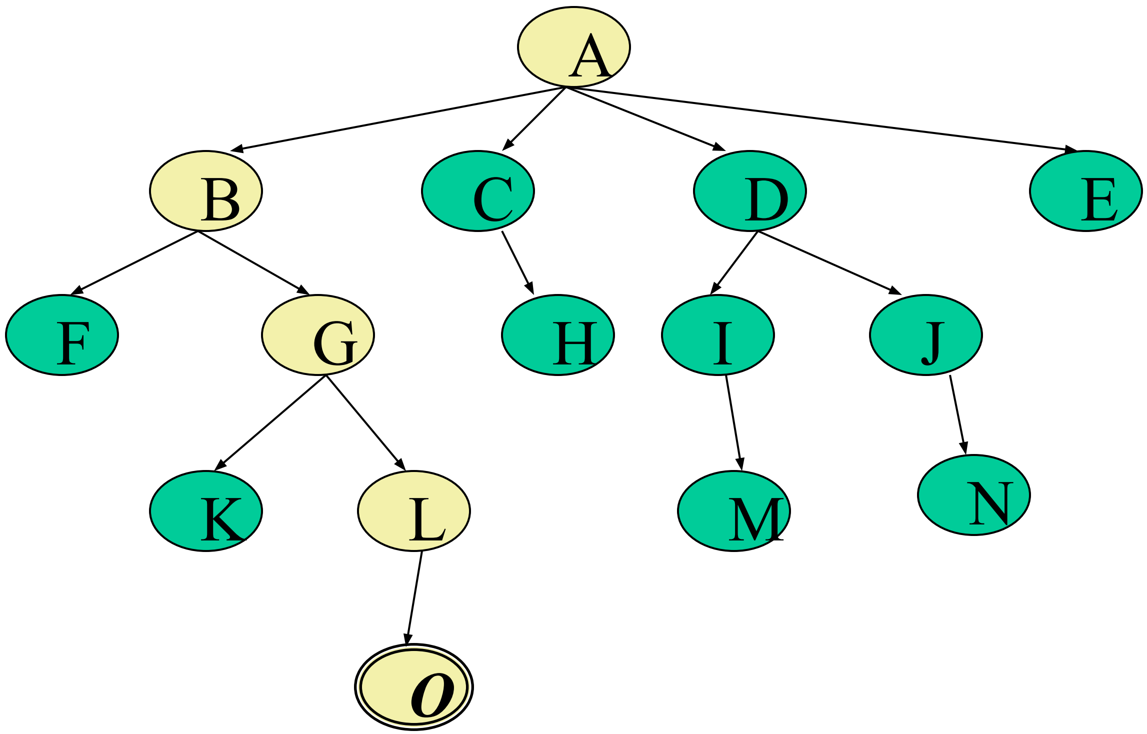
Breadth First Search

- A,
- B,C,D,E,
- F,G,H,I,J,
- K,L, M,N,
- Goal state: ***O***



Breadth First Search

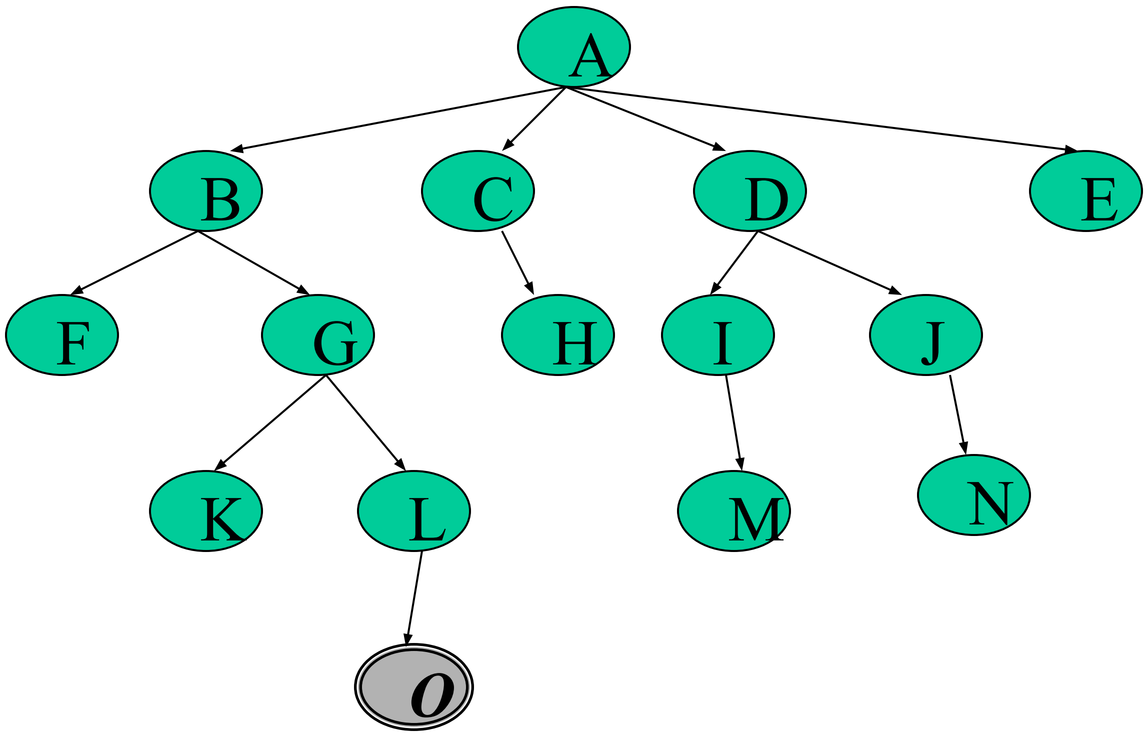
- The returned solution is the sequence of operators in the path:
  A, B, G, L, O



Depth First Search (DFS)

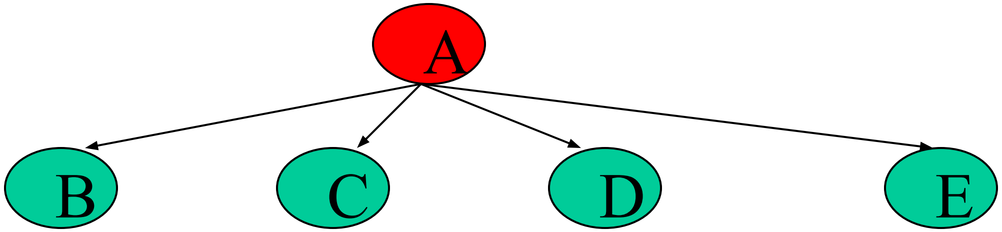
- Application2:

  Given the following state space (tree search), give the sequence of visited nodes when using DFS (assume that the node ***O*** is the goal state):



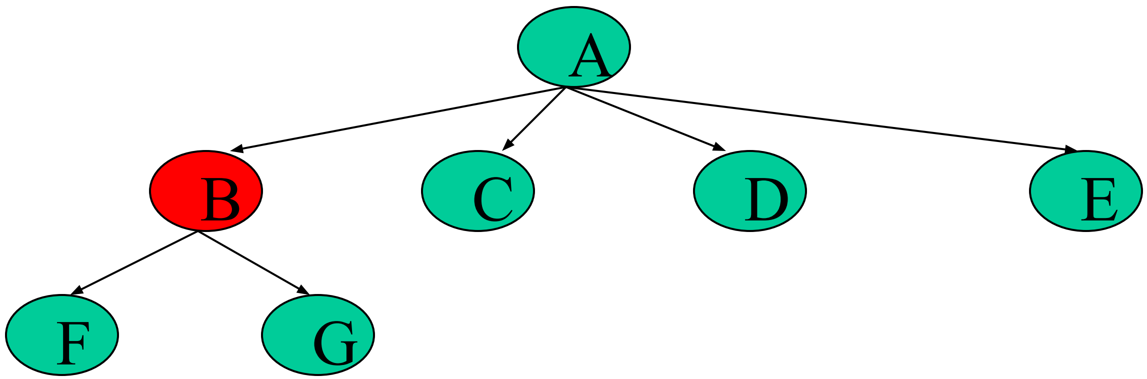
Depth First Search

- A,



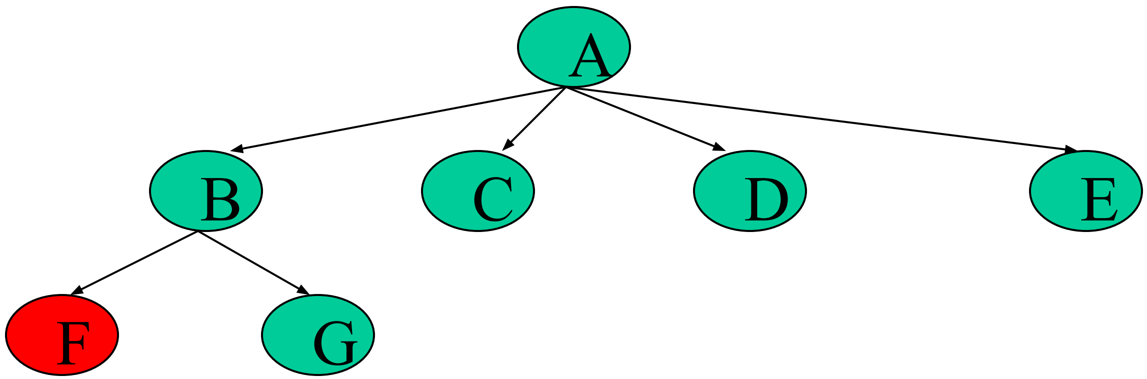
Depth First Search

- A,B,



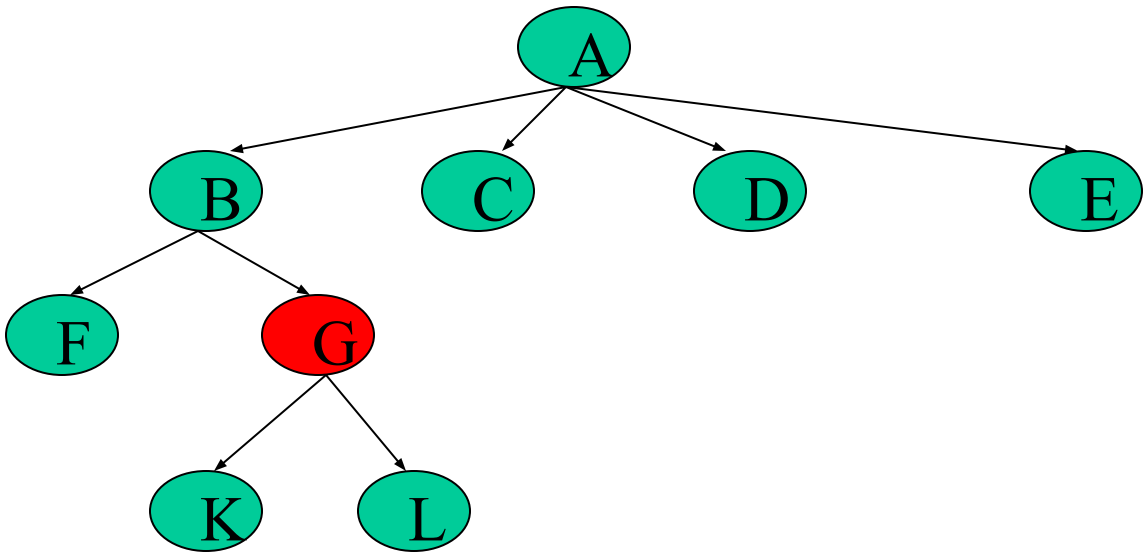
Depth First Search

- A,B,F,



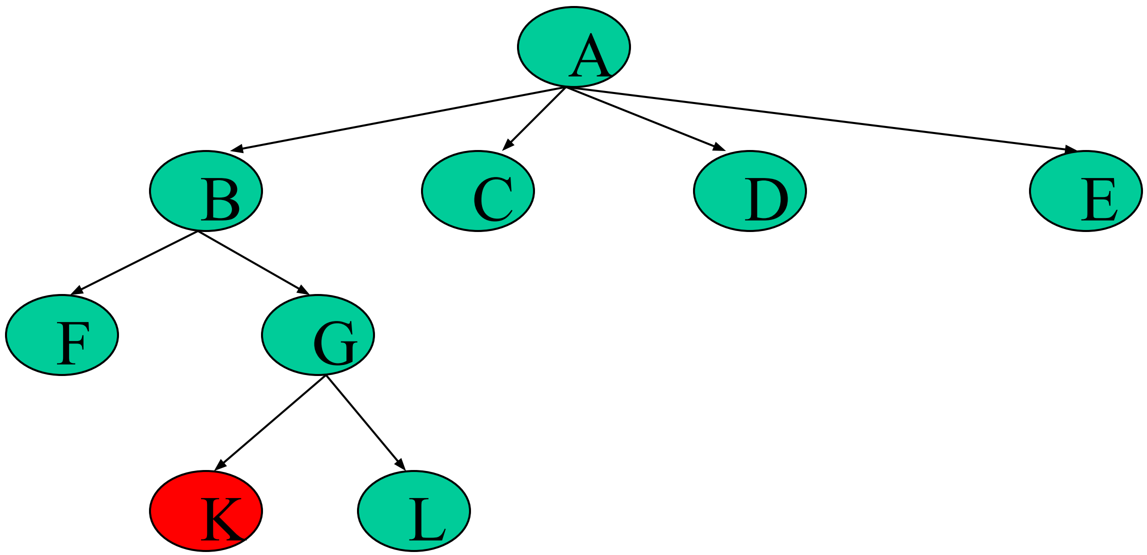
Depth First Search

- A,B,F,
- G,



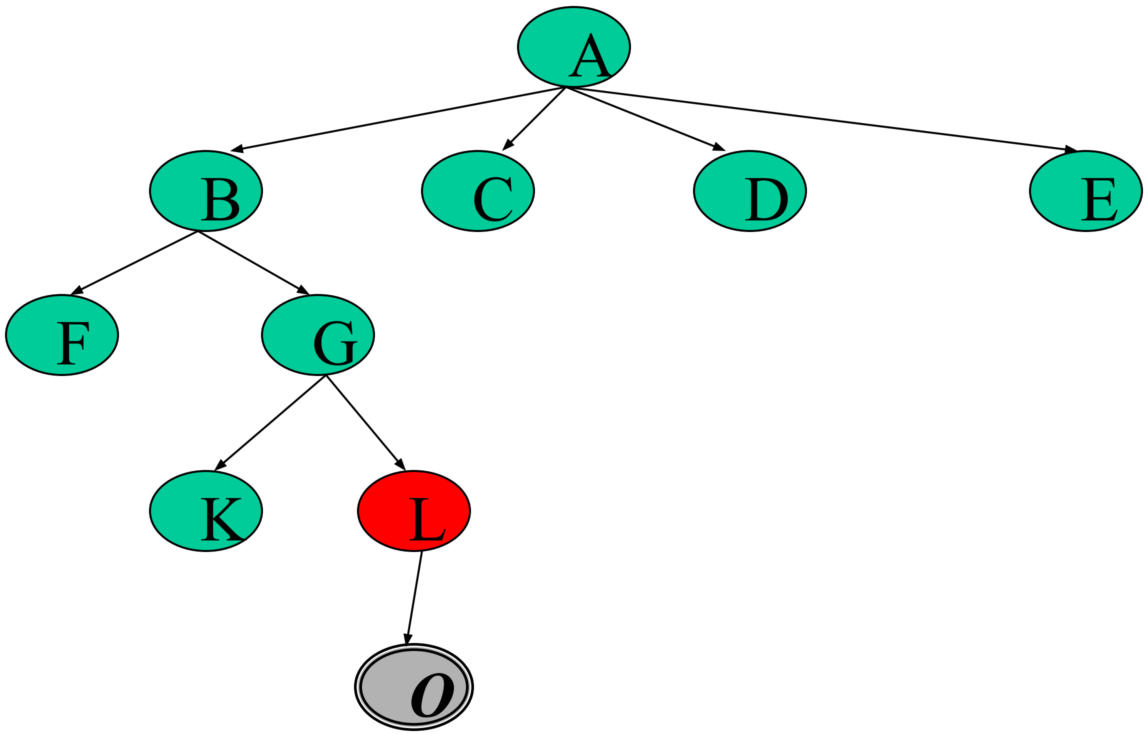
Depth First Search

- A,B,F,
- G,K,



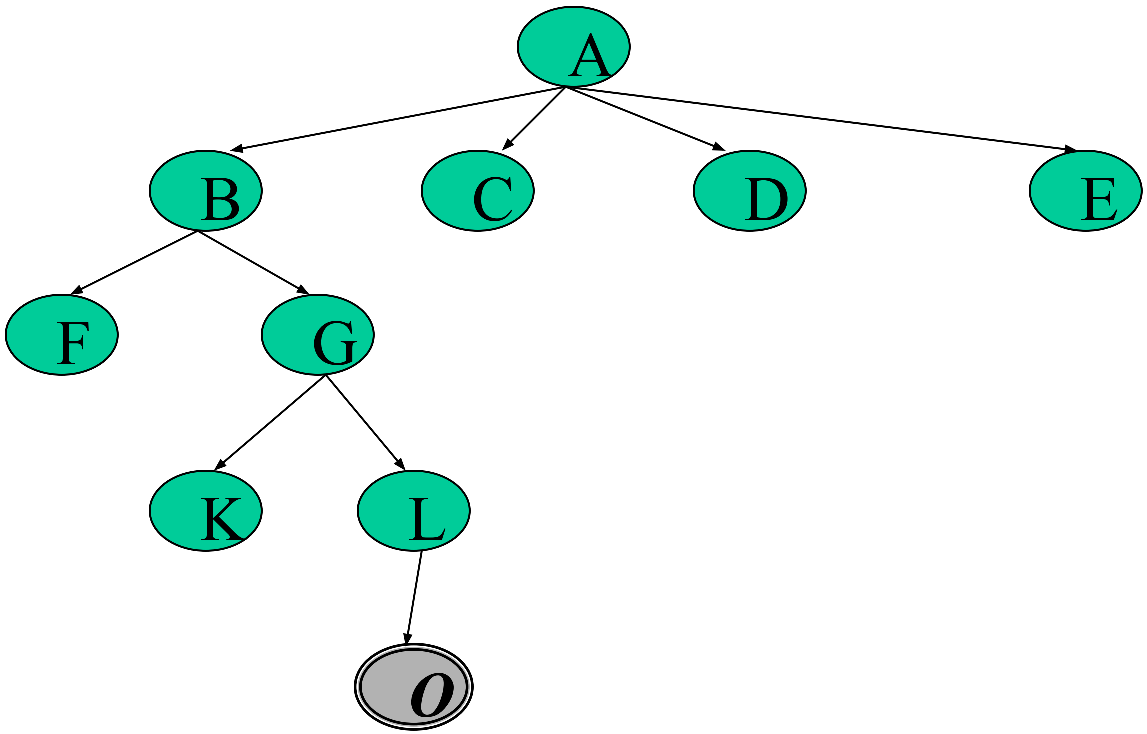
Depth First Search

- A,B,F,
- G,K,
- L,



Depth First Search

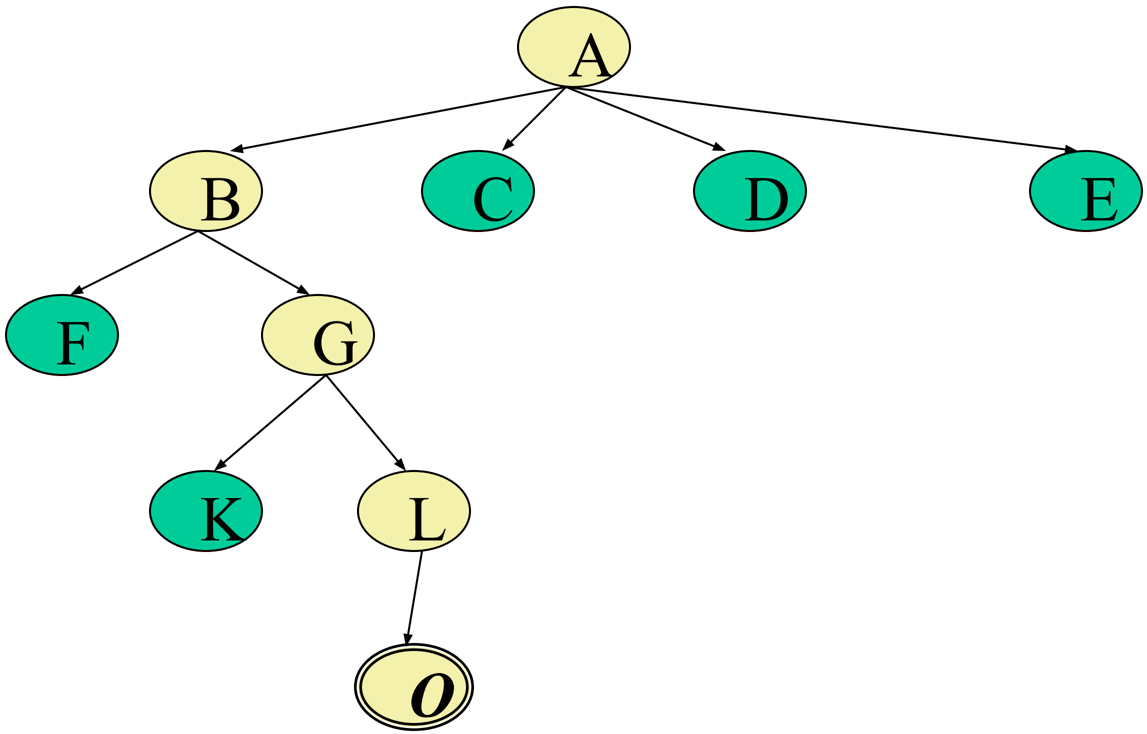
- A,B,F,
- G,K,
- L, *O: Goal State*



Depth First Search

The returned solution is the sequence of operators in the path:

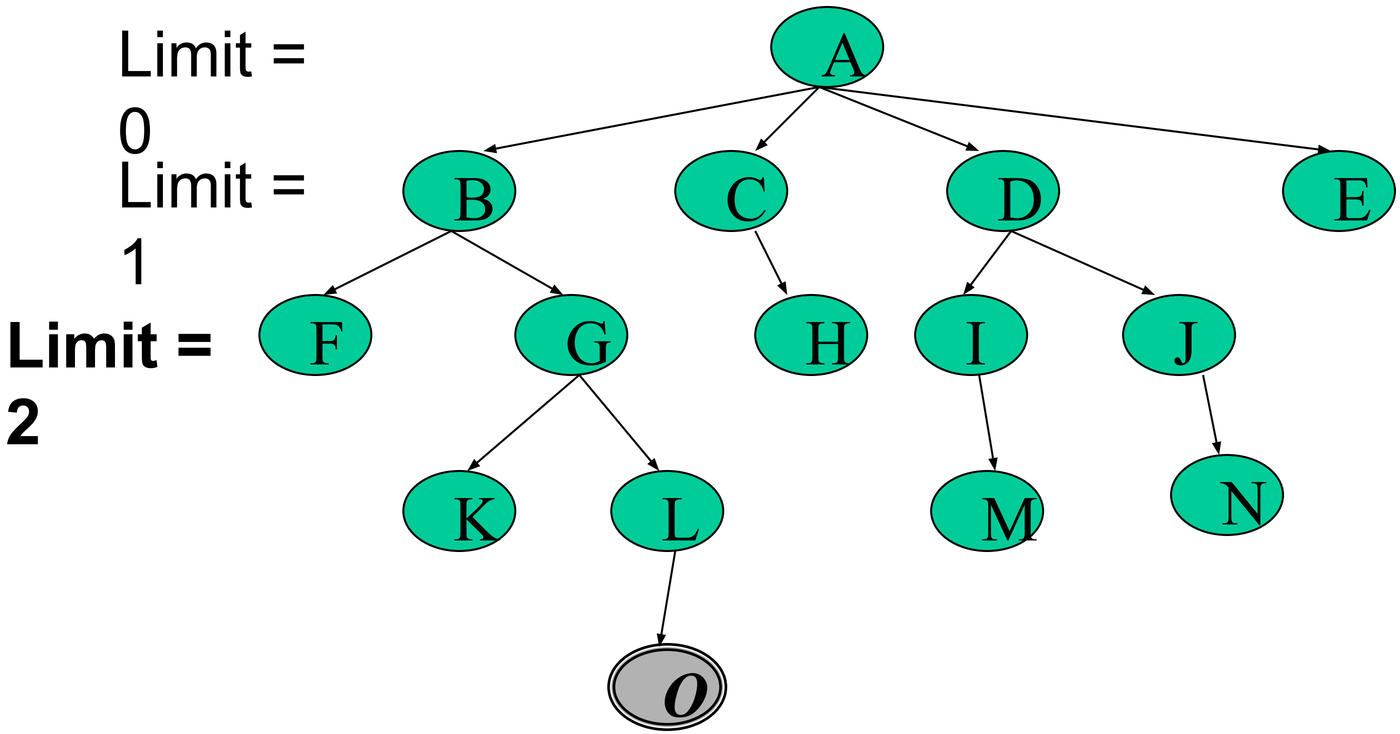
A, B, G, L, O



Depth-Limited Search (DLS)

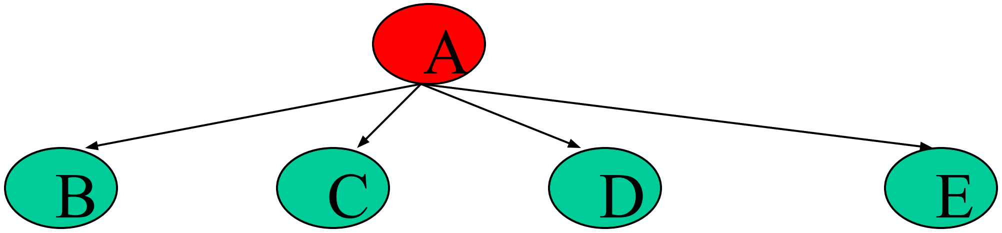
- Application3:

  Given the following state space (tree search), give the sequence of visited nodes when using DLS (Limit = 2):



Depth-Limited Search (DLS)

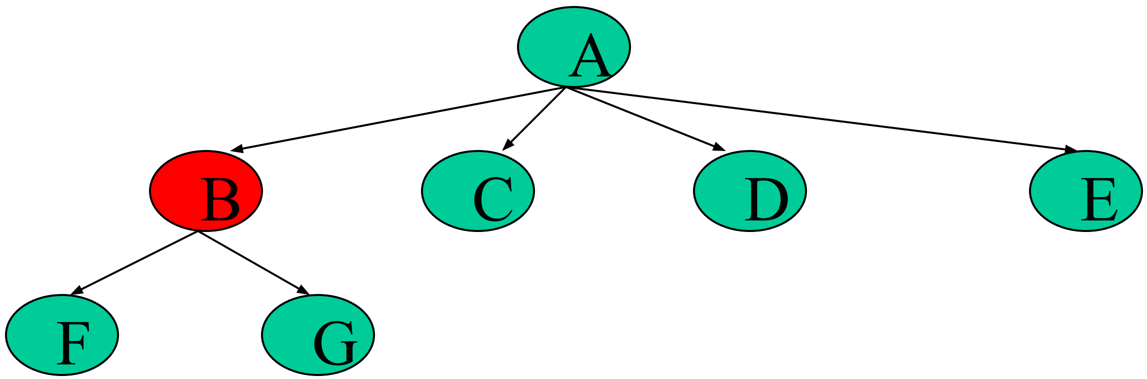
- A,



Limit = 2

Depth-Limited Search (DLS)

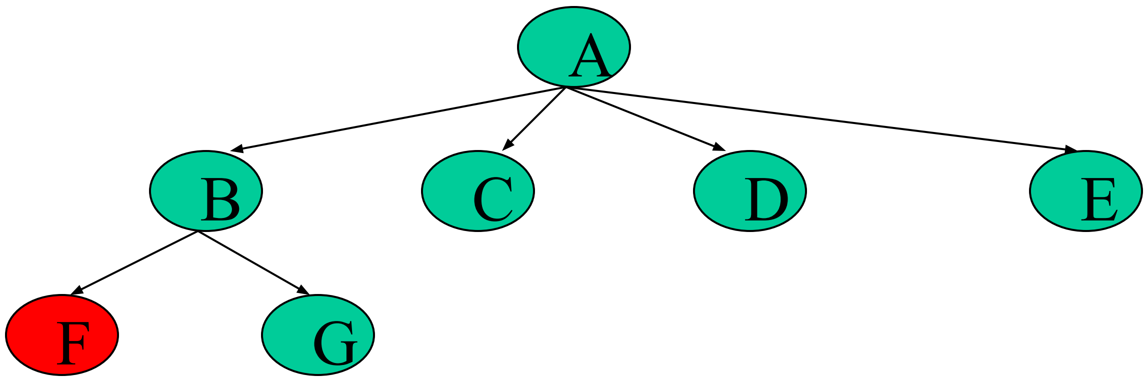
- A,B,



Limit = 2

Depth-Limited Search (DLS)

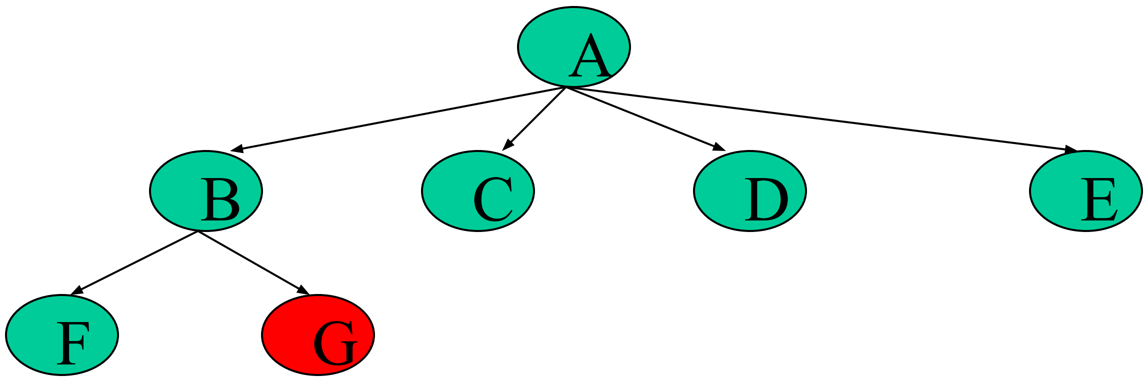
- A,B,F,



Limit = 2

Depth-Limited Search (DLS)

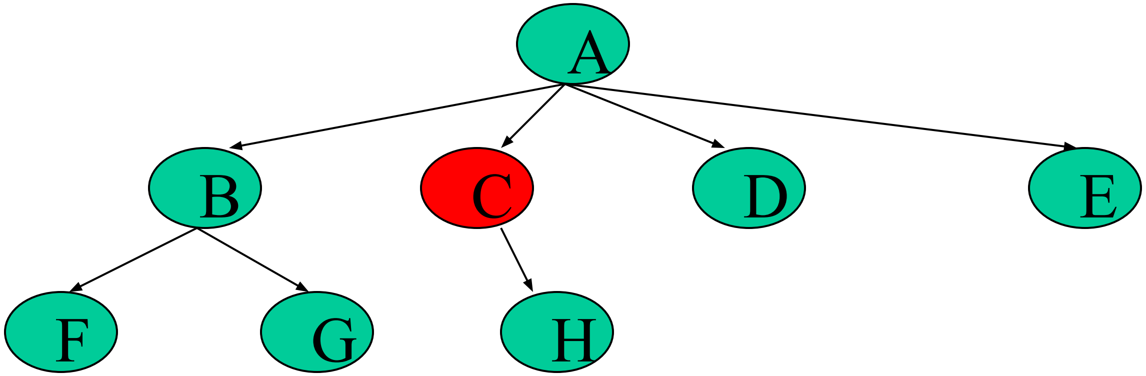
- A,B,F,
- G,



Limit = 2

Depth-Limited Search (DLS)

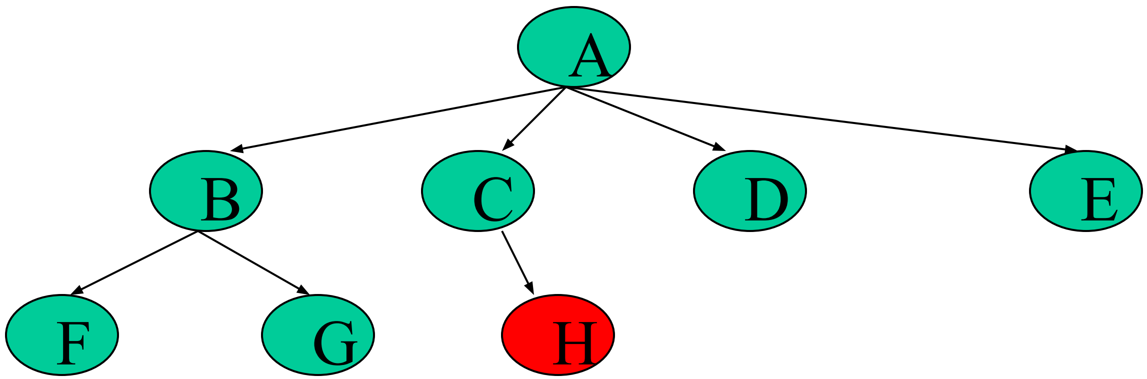
- A,B,F,
- G,
- C,



Limit = 2

Depth-Limited Search (DLS)

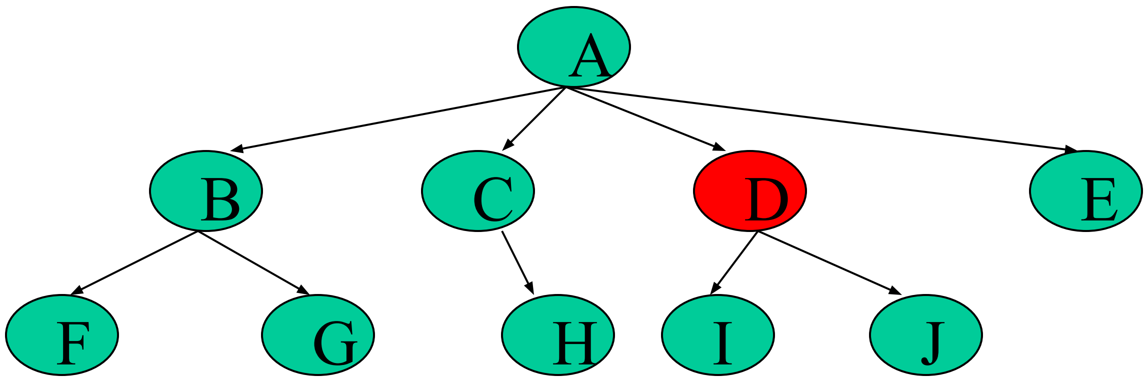
- A,B,F,
- G,
- C,H,



Limit = 2

Depth-Limited Search (DLS)

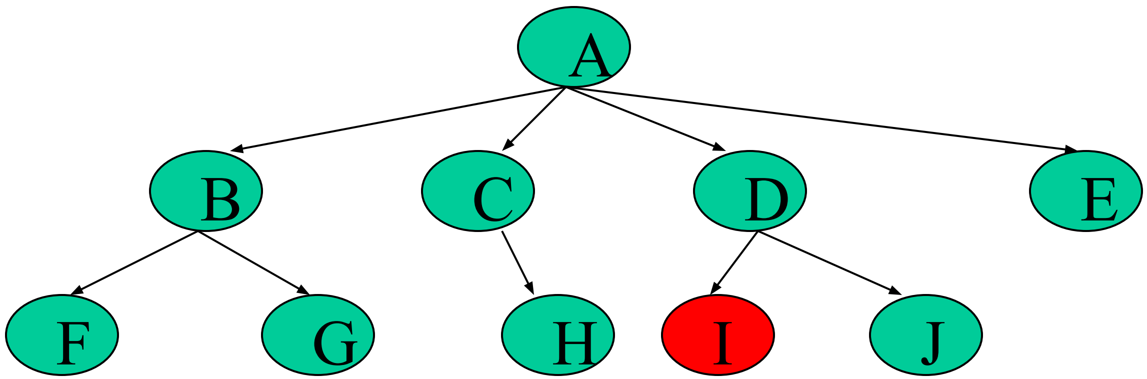
- A,B,F,
- G,
- C,H,
- D,



Limit = 2

Depth-Limited Search (DLS)

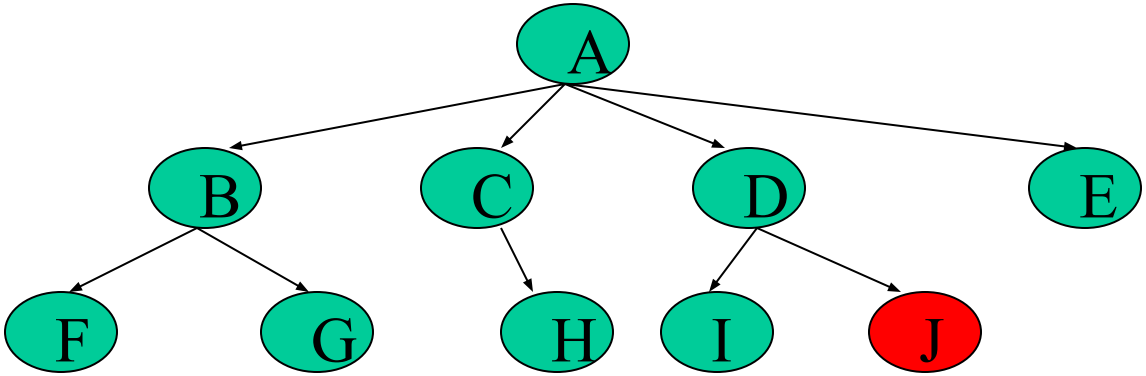
- A,B,F,
- G,
- C,H,
- D,I



Limit = 2

Depth-Limited Search (DLS)

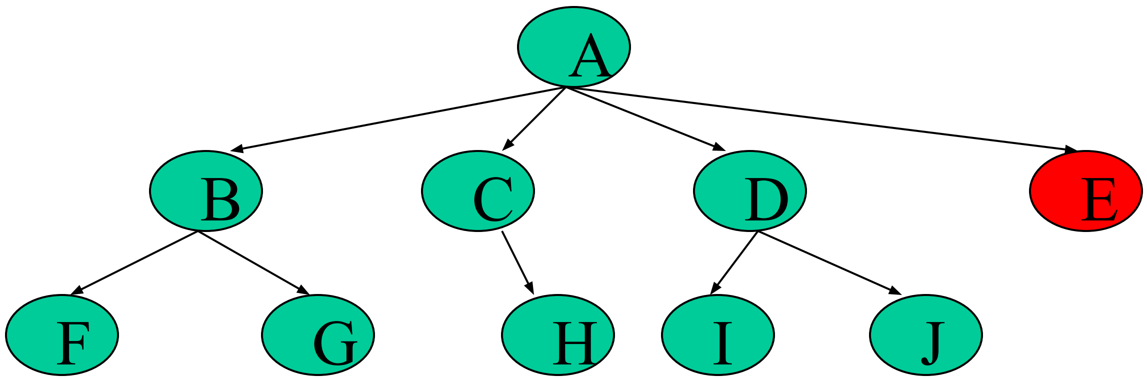
- A,B,F,
- G,
- C,H,
- D,I
- J,



Limit = 2

Depth-Limited Search (DLS)

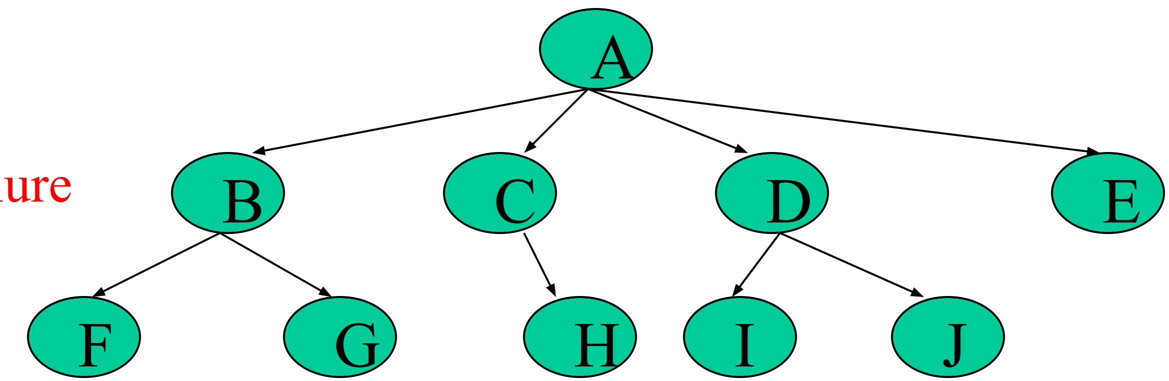
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E



Limit = 2

Depth-Limited Search (DLS)

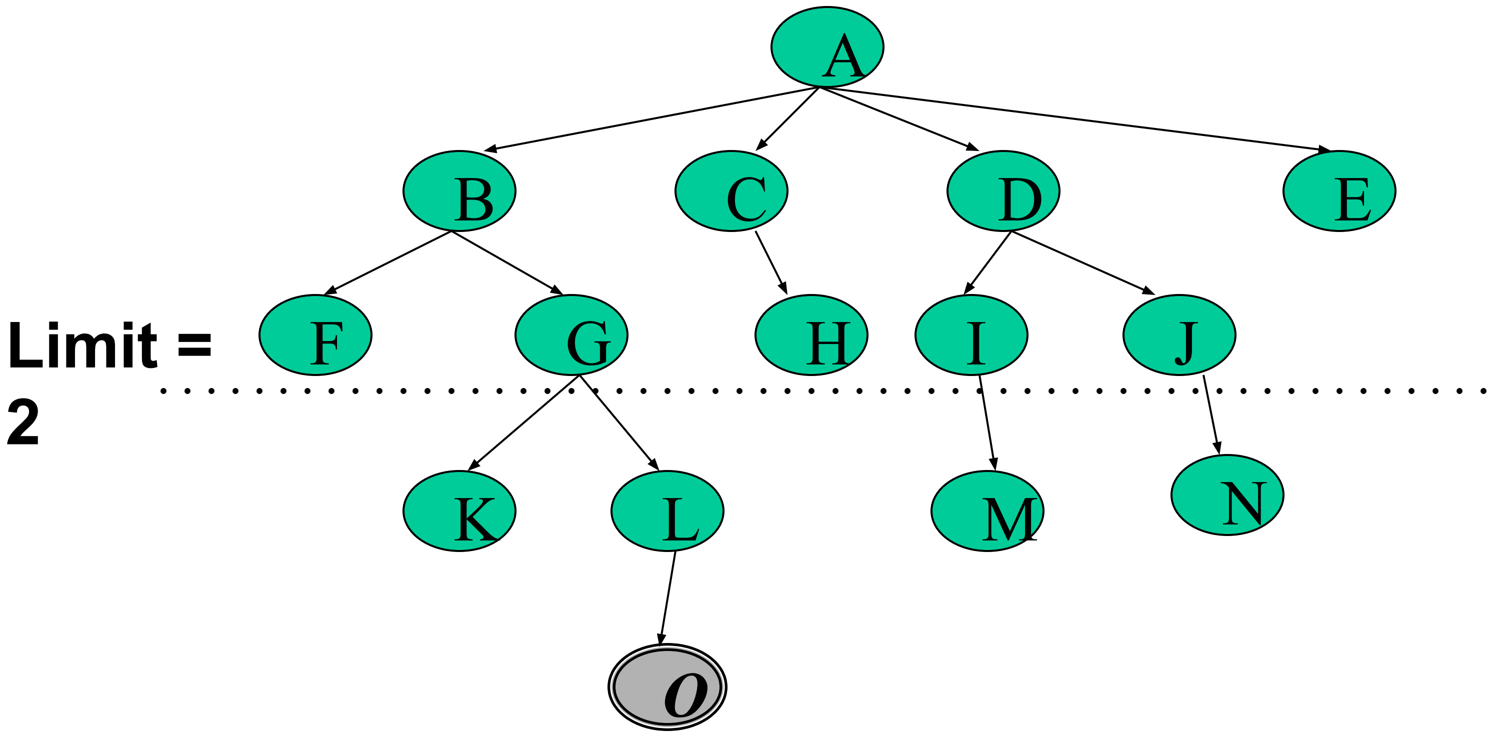
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E, Failure



Limit = 2

Depth-Limited Search (DLS)

- DLS algorithm returns Failure (no solution)
- The reason is that the goal is beyond the limit (Limit =2): the goal depth is (d=4)



Iterative Deepening Search (IDS)

- Application4:

  Given the following state space (tree search), give the sequence of visited nodes when using IDS:

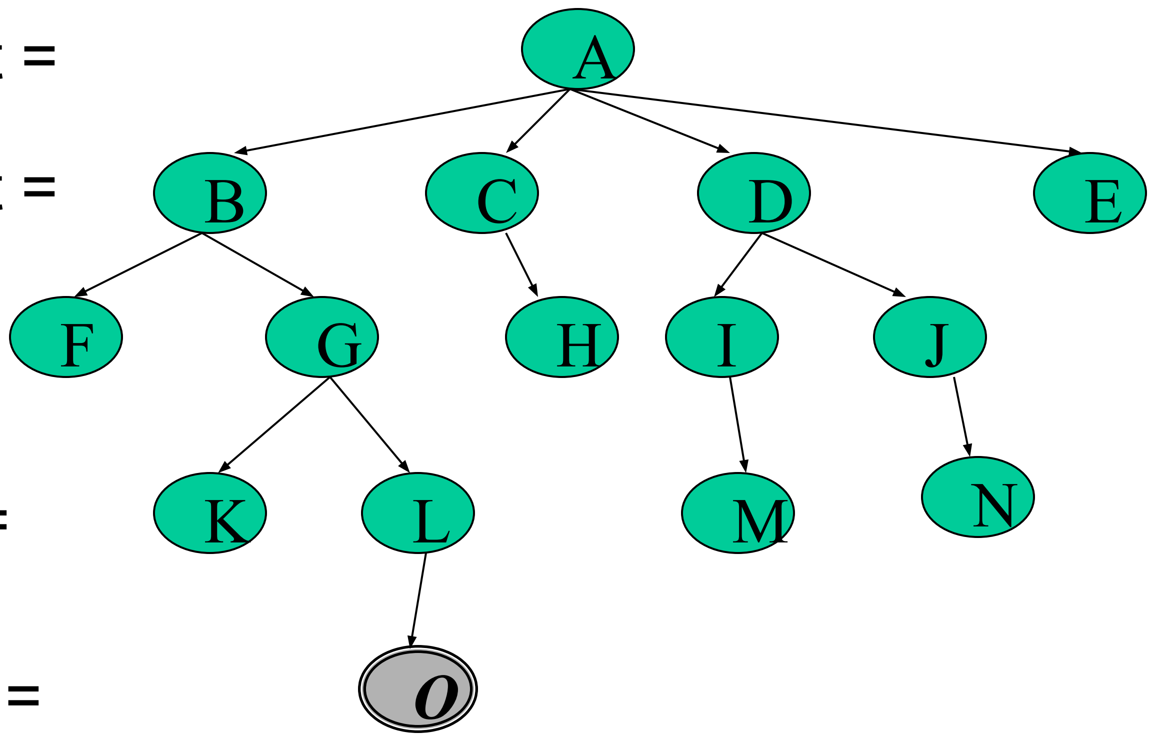
Limit = 0

Limit = 1

Limit = 2

Limit = 3

Limit = 4



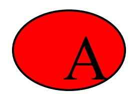
Iterative Deepening Search (IDS)

DLS with bound = 0

Iterative Deepening Search (IDS)

- A,

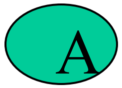
Limit = 0



Iterative Deepening Search (IDS)

- A, Failure

Limit = 0



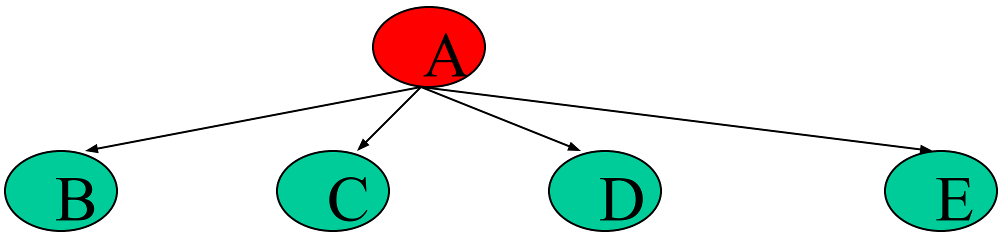
Iterative Deepening Search (IDS)

DLS with bound = 1

Iterative Deepening Search (IDS)

- A,

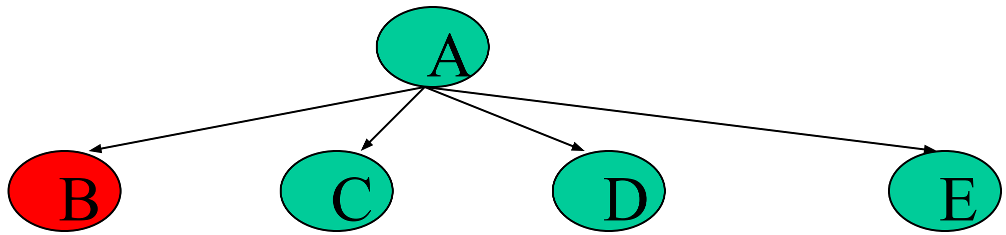
Limit = 1



Iterative Deepening Search (IDS)

- A,B,

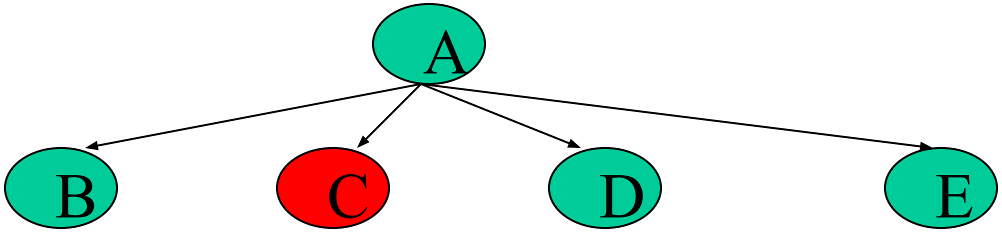
Limit = 1



Iterative Deepening Search (IDS)

- A,B,
- C,

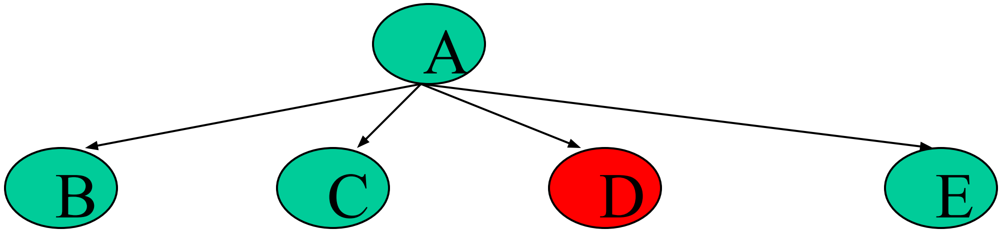
Limit = 1



Iterative Deepening Search (IDS)

- A,B,
- C,
- D,

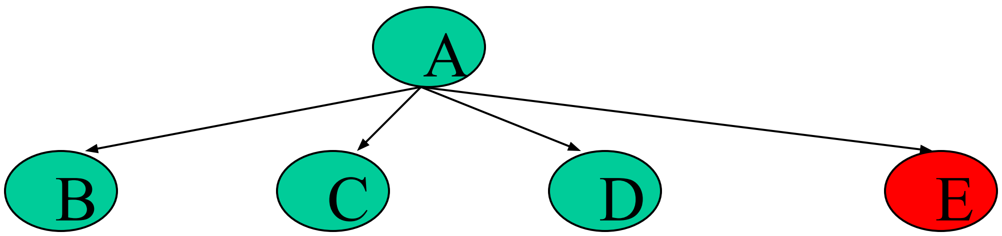
Limit = 1



Iterative Deepening Search (IDS)

- A,B
- C,
- D,
- E,

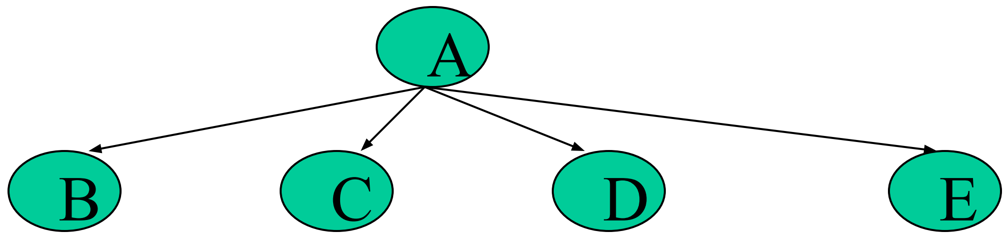
Limit = 1



Iterative Deepening Search (IDS)

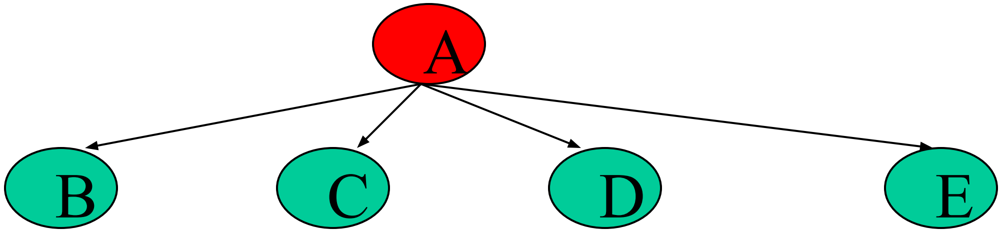
- A,B,
- C,
- D,
- E, Failure

Limit = 1



Iterative Deepening Search (IDS)

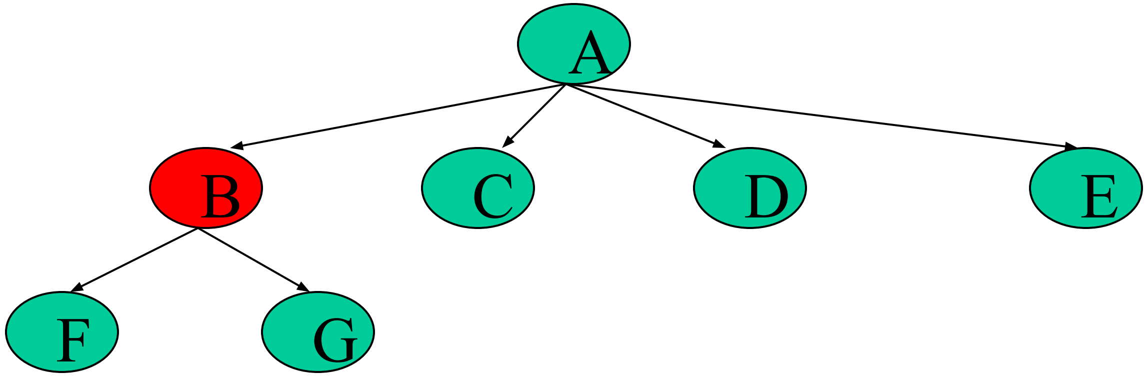
- A,



Limit = 2

Iterative Deepening Search (IDS)

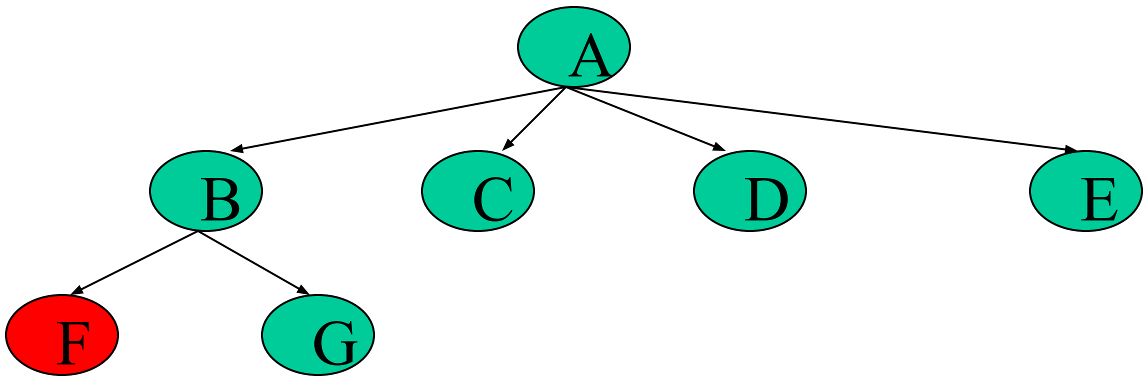
- A,B,



Limit = 2

Iterative Deepening Search (IDS)

- A,B,F,

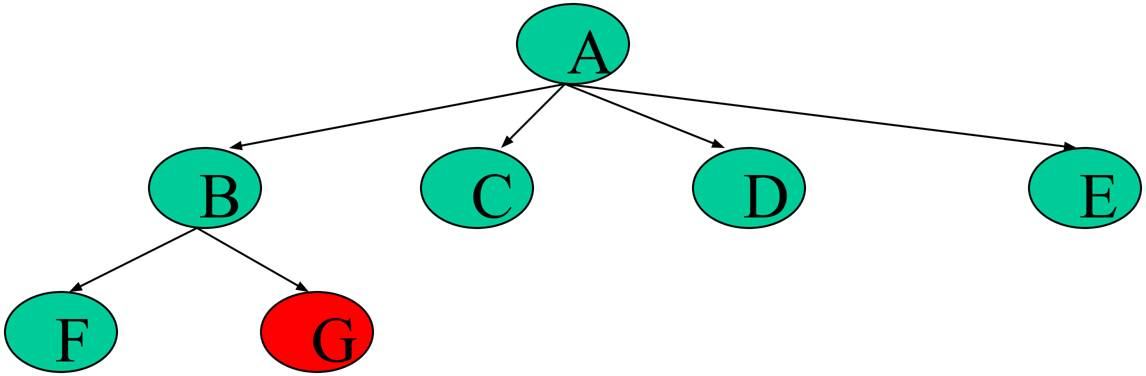


Limit = 2

Iterative Deepening Search (IDS)

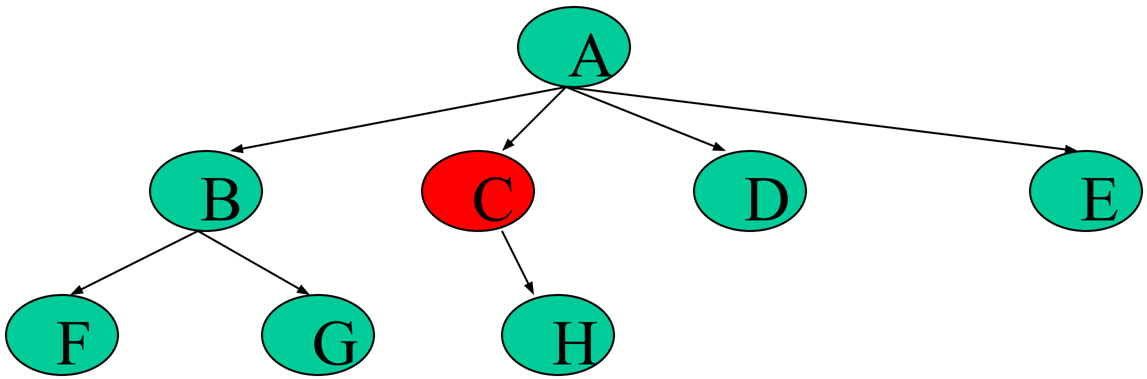
- A,B,F,
- G,

Limit = 2



Iterative Deepening Search (IDS)

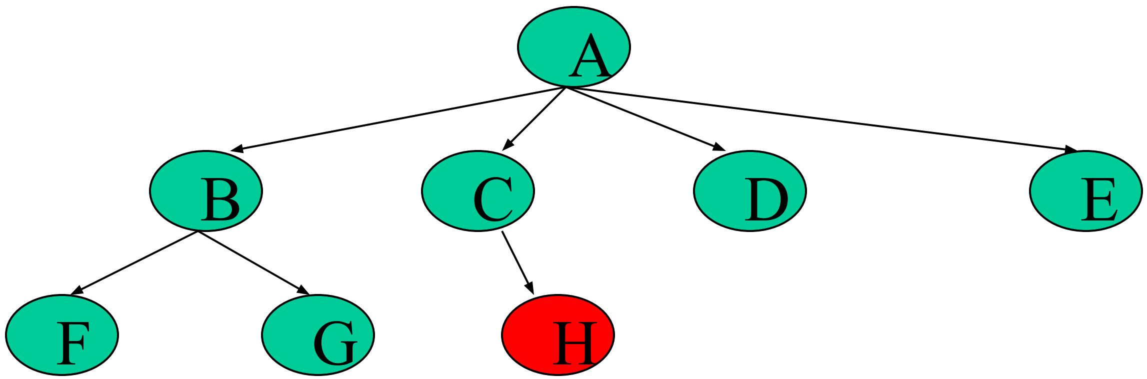
- A,B,F,
- G,
- C,



Limit = 2

Iterative Deepening Search (IDS)

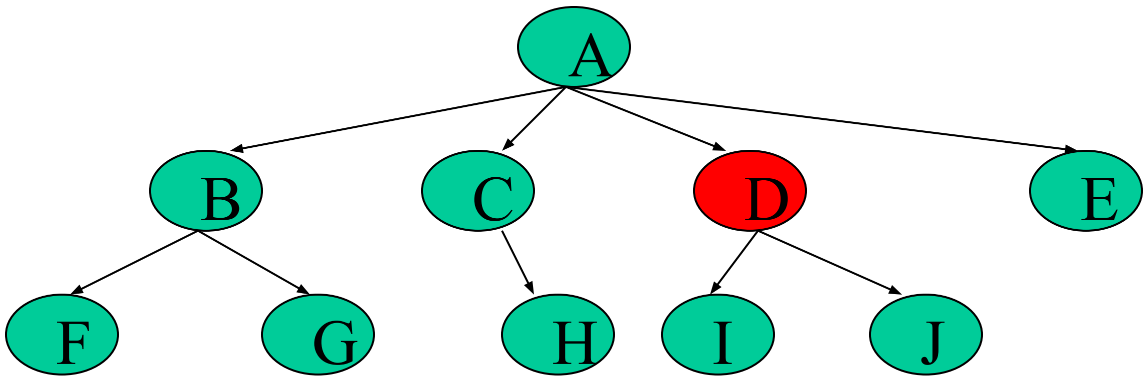
- A,B,F,
- G,
- C,H,



Limit = 2

Iterative Deepening Search (IDS)

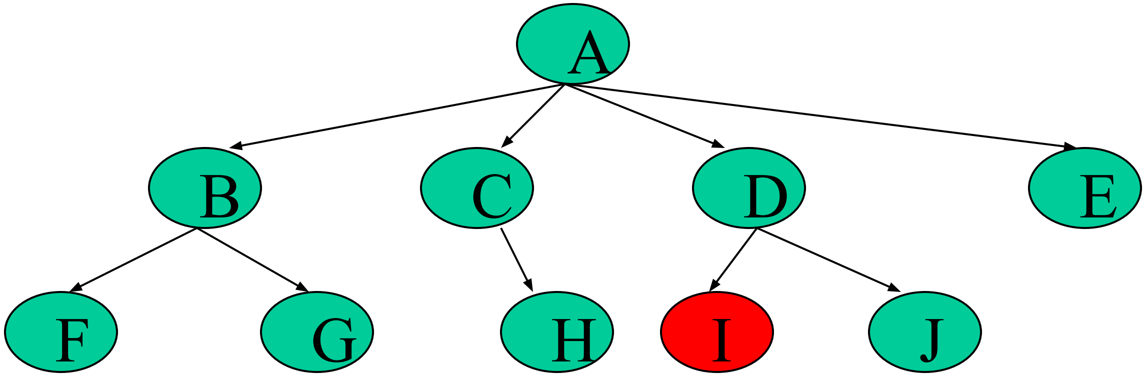
- A,B,F,
- G,
- C,H,
- D,



Limit = 2

Iterative Deepening Search (IDS)

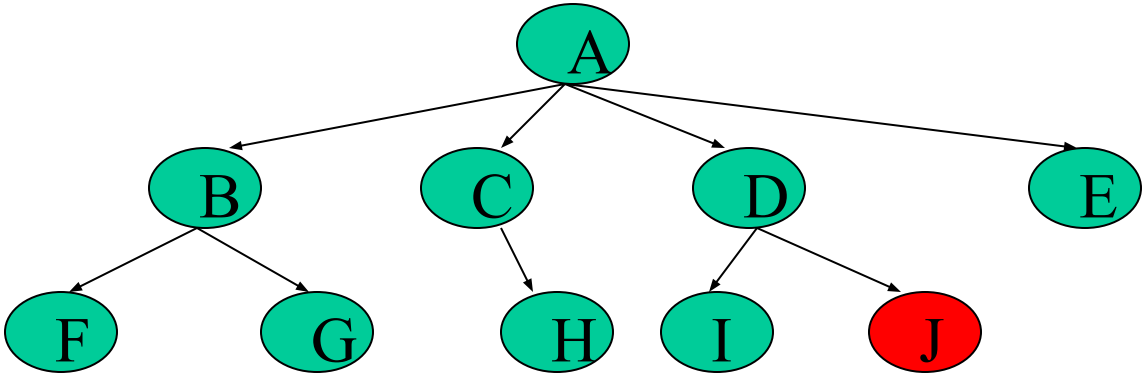
- A,B,F,
- G,
- C,H,
- D,I



Limit = 2

Iterative Deepening Search (IDS)

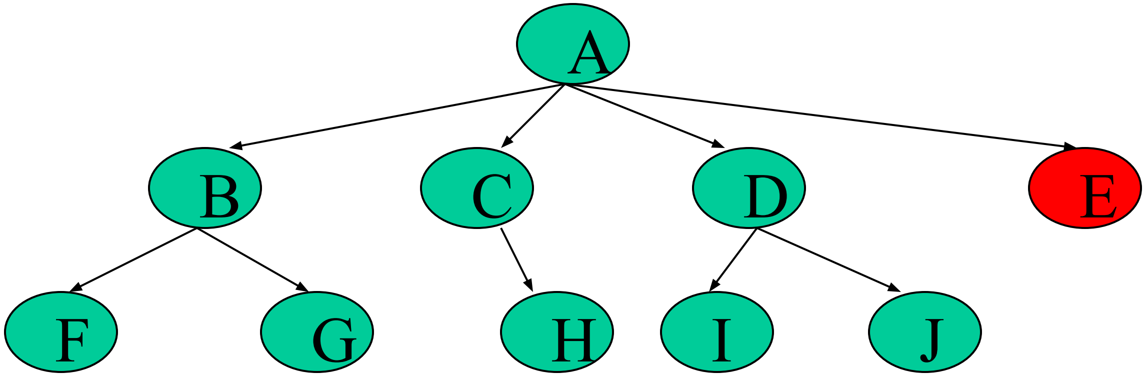
- A,B,F,
- G,
- C,H,
- D,I
- J,



Limit = 2

Iterative Deepening Search (IDS)

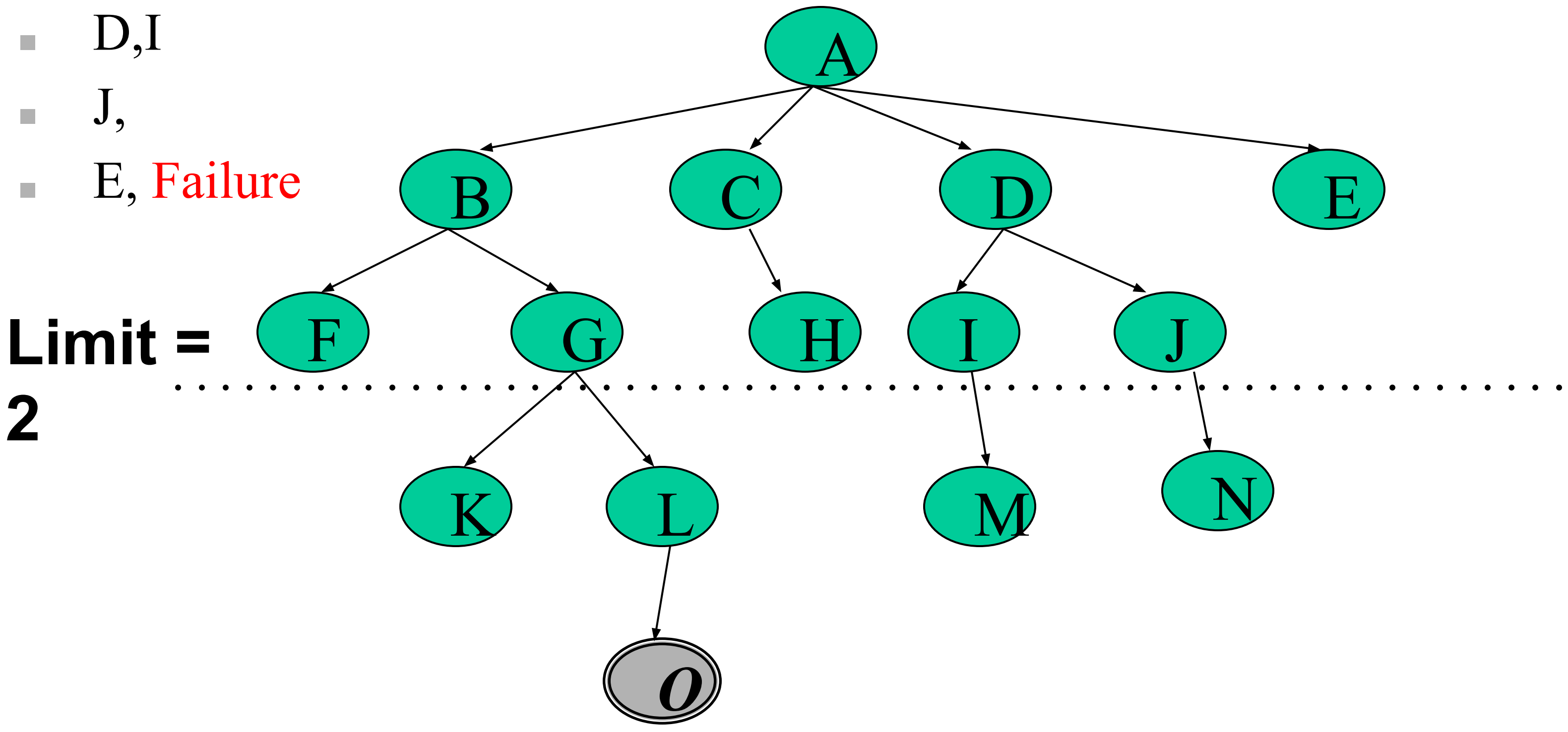
- A,B,F,
- G,
- C,H,
- D,I
- J,
- E



Limit = 2

Iterative Deepening Search (IDS)

- A,B,F,
- G,
- C,H,
- D,I
- J,
- E, Failure

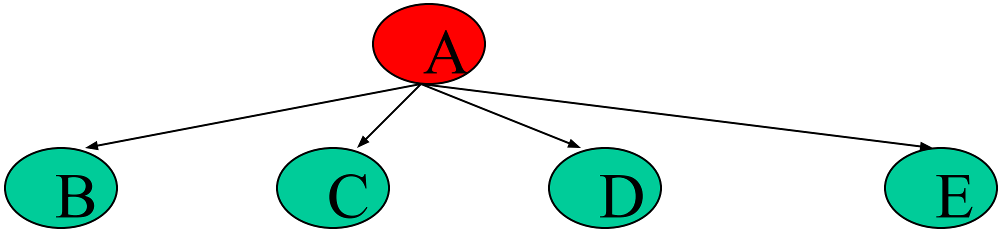


Iterative Deepening Search (IDS)

DLS with bound = 3

Iterative Deepening Search (IDS)

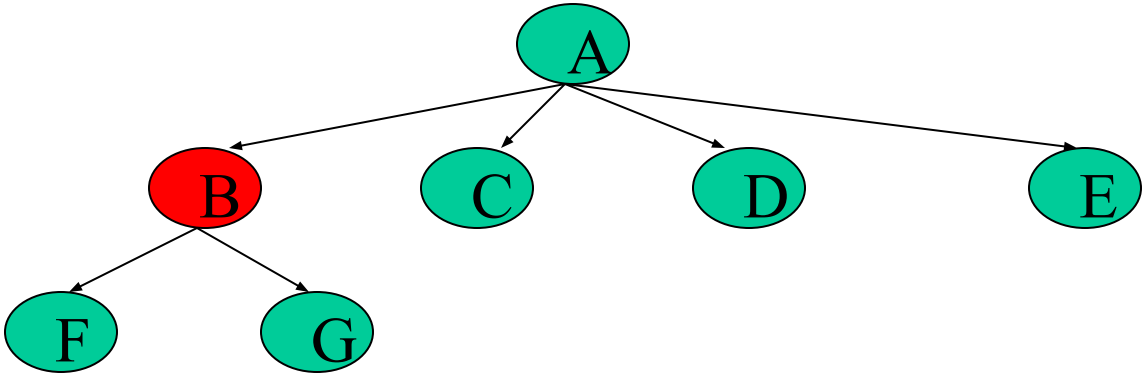
- A,



Limit = 3

Iterative Deepening Search (IDS)

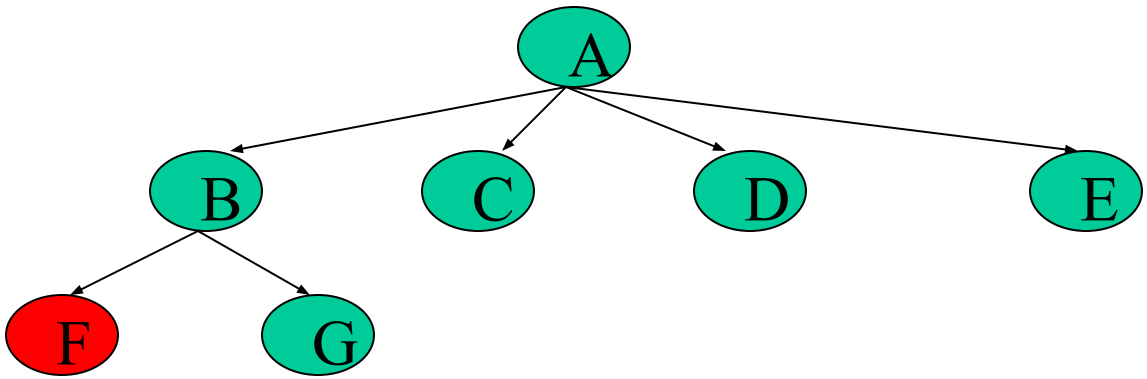
- A,B,



Limit = 3

Iterative Deepening Search (IDS)

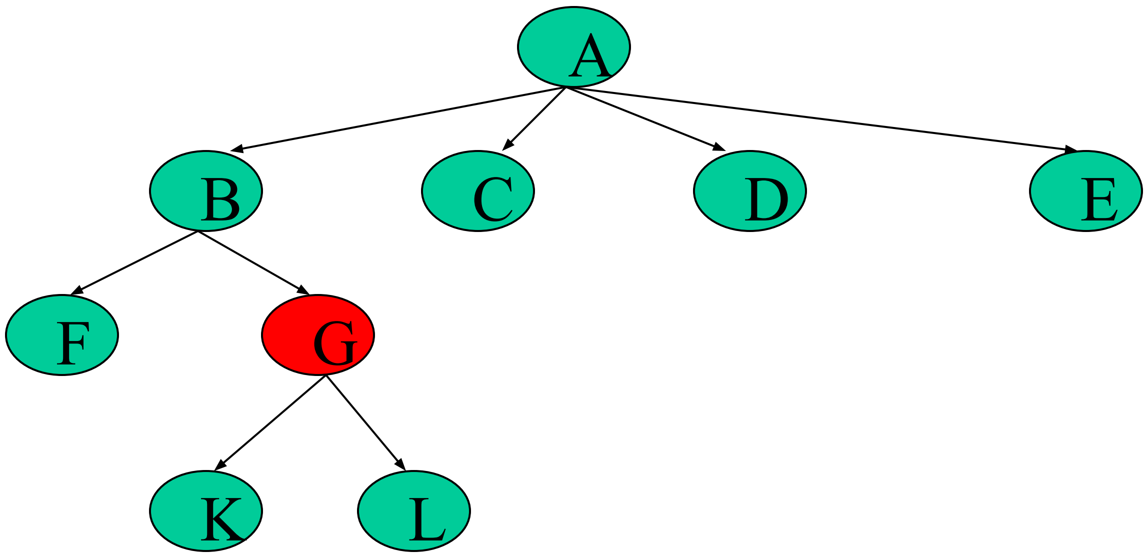
- A,B,F,



Limit = 3

Iterative Deepening Search (IDS)

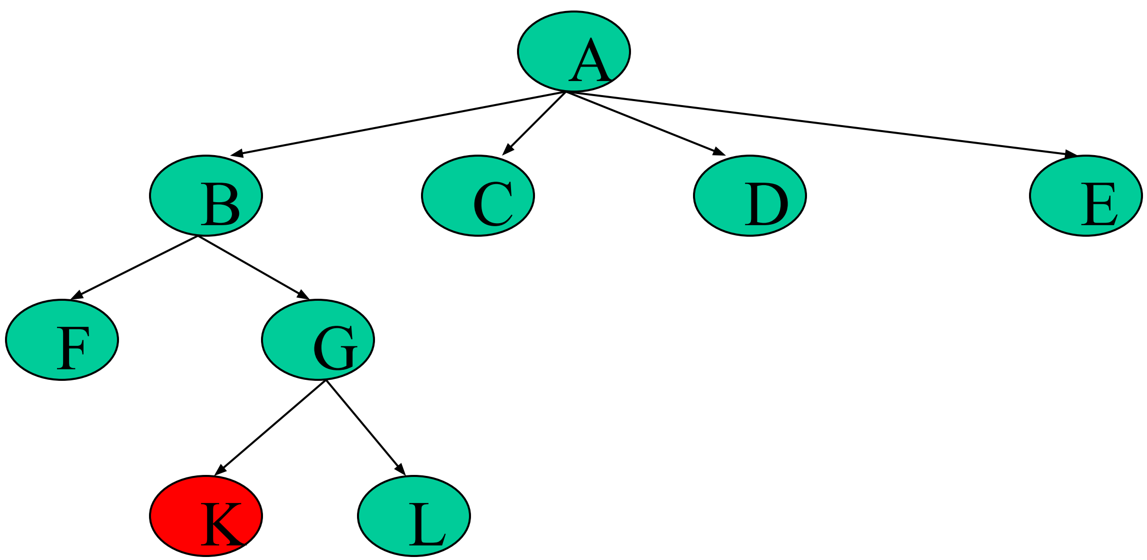
- A,B,F,
- G,



Limit = 3

Iterative Deepening Search (IDS)

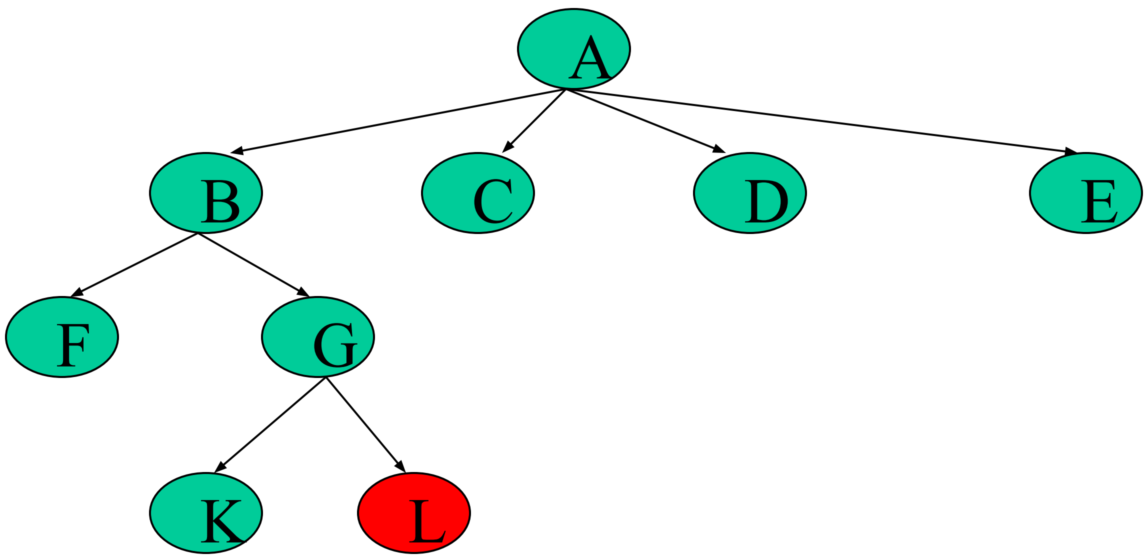
- A,B,F,
- G,K,



Limit = 3

Iterative Deepening Search (IDS)

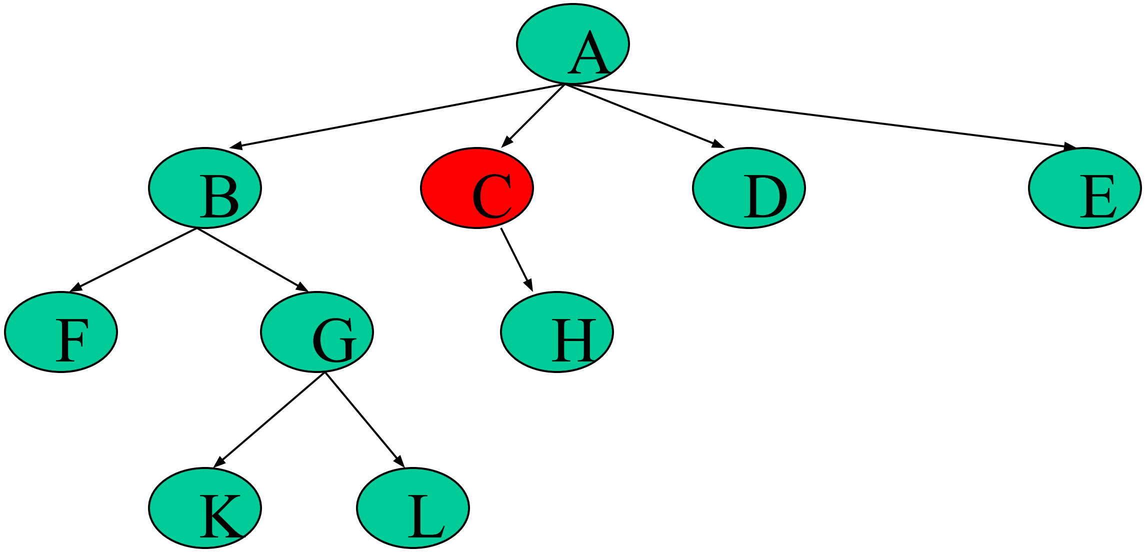
- A,B,F,
- G,K,
- L,



Limit = 3

Iterative Deepening Search (IDS)

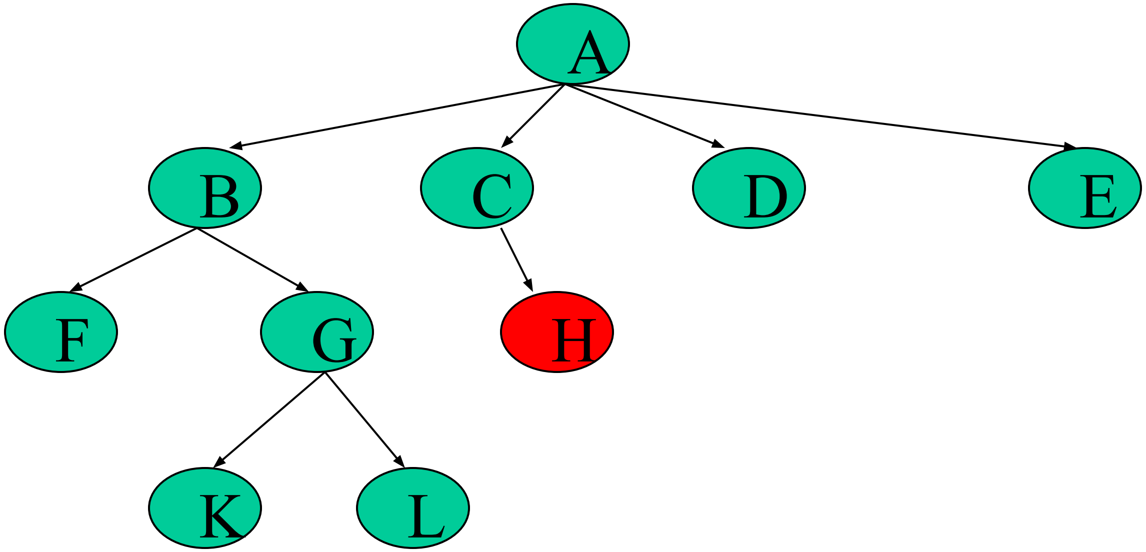
- A,B,F,
- G,K,
- L,
- C,



Limit = 3

Iterative Deepening Search (IDS)

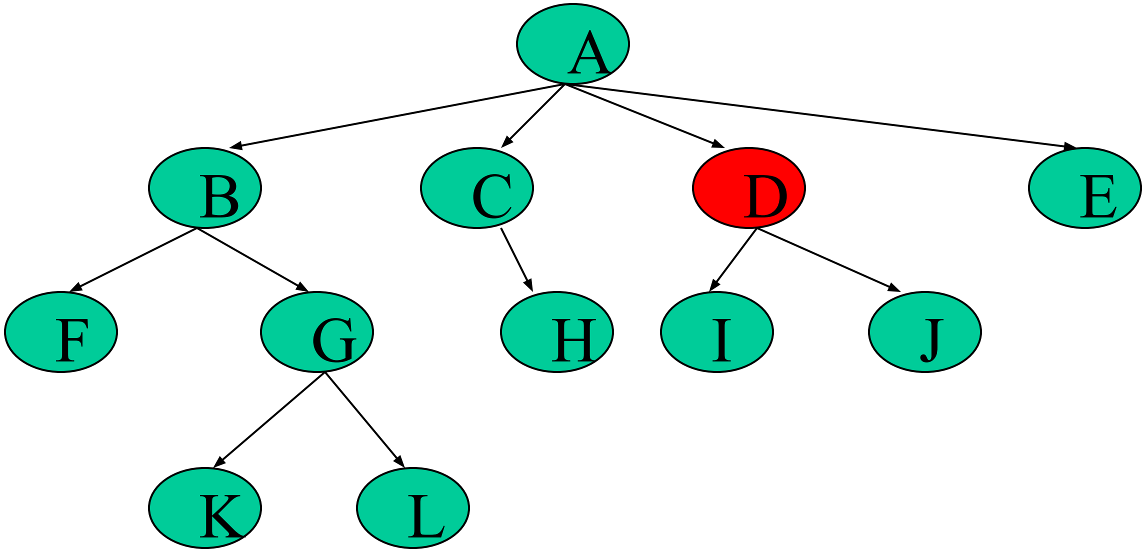
- A,B,F,
- G,K,
- L,
- C,H,



Limit = 3

Iterative Deepening Search (IDS)

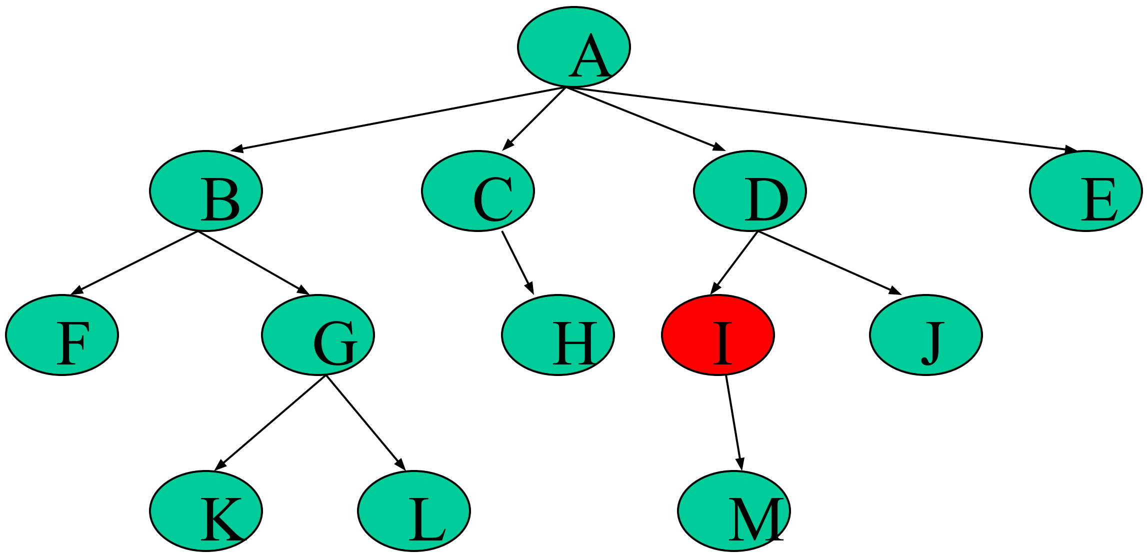
- A,B,F,
- G,K,
- L,
- C,H,
- D,



Limit = 3

Iterative Deepening Search (IDS)

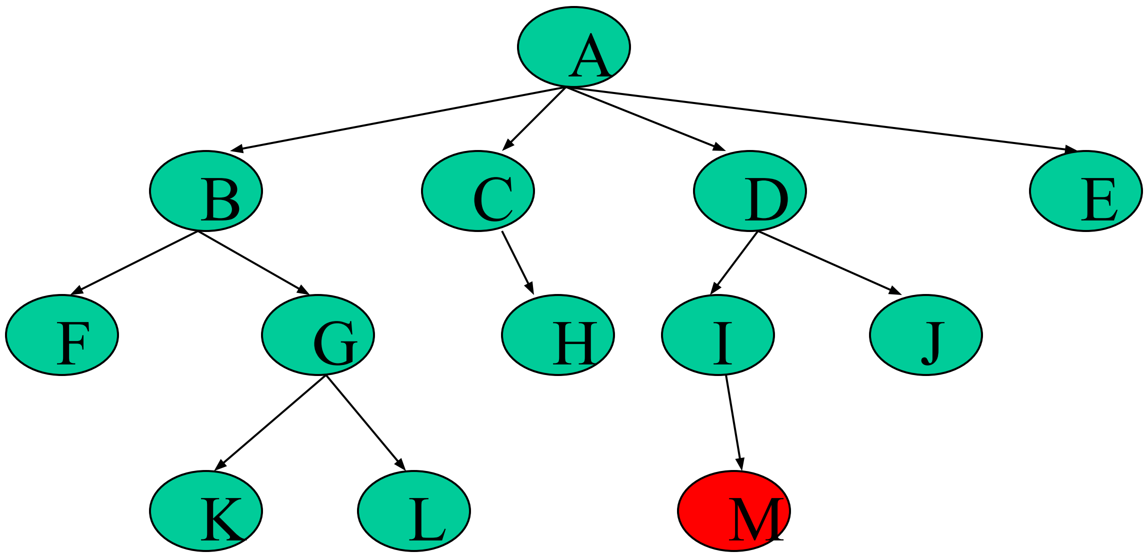
- A,B,F,
- G,K,
- L,
- C,H,
- D,I,



Limit = 3

Iterative Deepening Search (IDS)

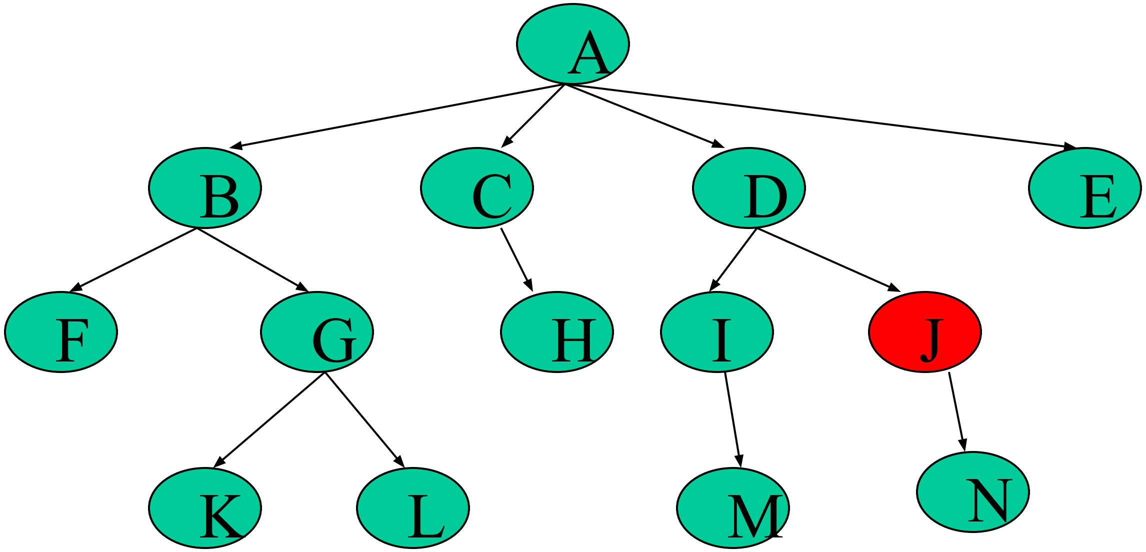
- A,B,F,
- G,K,
- L,
- C,H,
- D,I,M,



Limit = 3

Iterative Deepening Search (IDS)

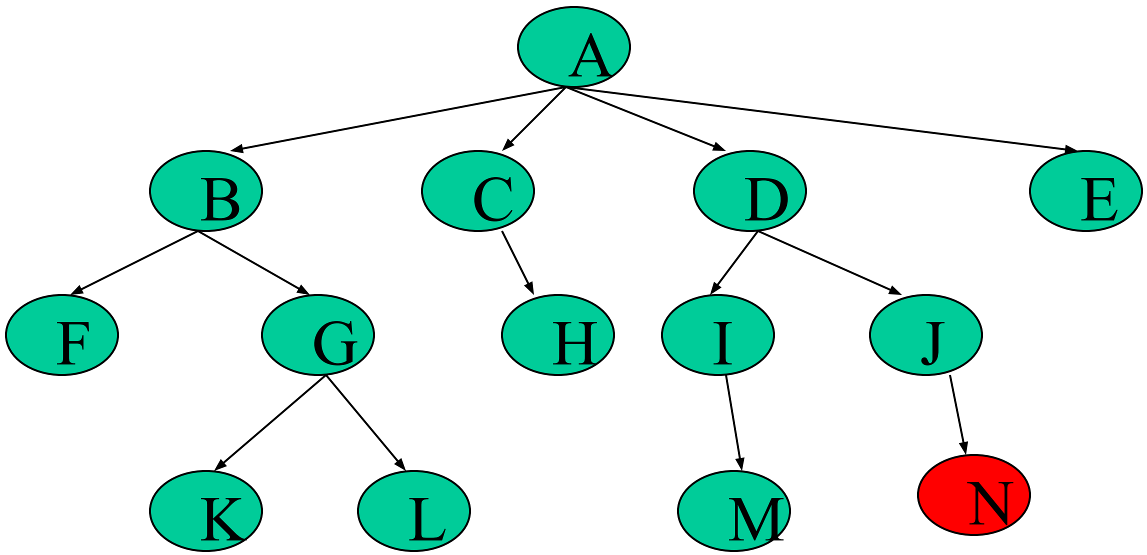
- A,B,F,
- G,K,
- L,
- C,H,
- D,I,M,
- J,



Limit = 3

Iterative Deepening Search (IDS)

- A,B,F,
- G,K,
- L,
- C,H,
- D,I,M,
- J,N,

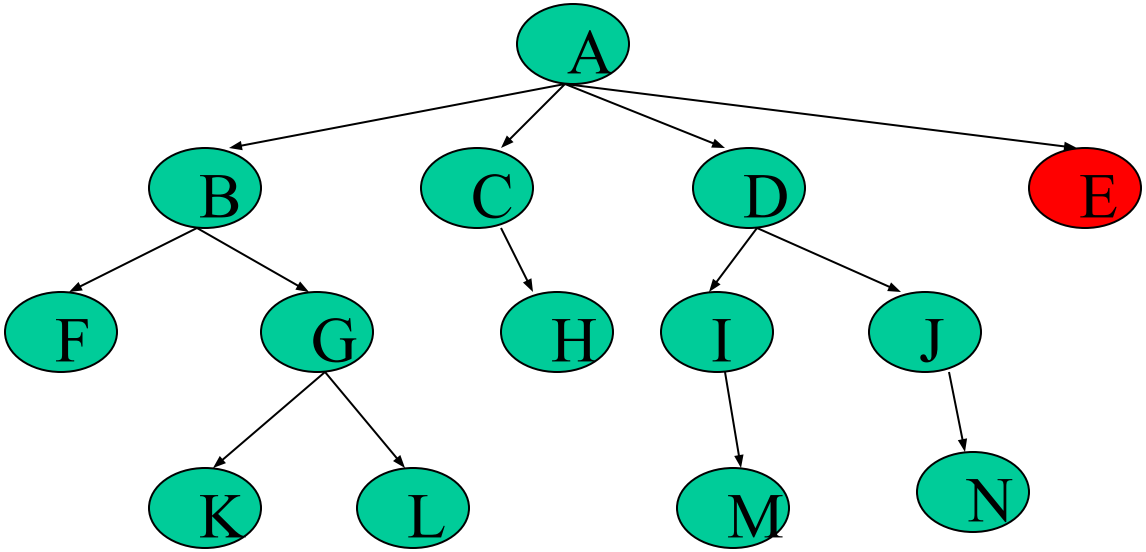


Limit = 3

Iterative Deepening Search (IDS)

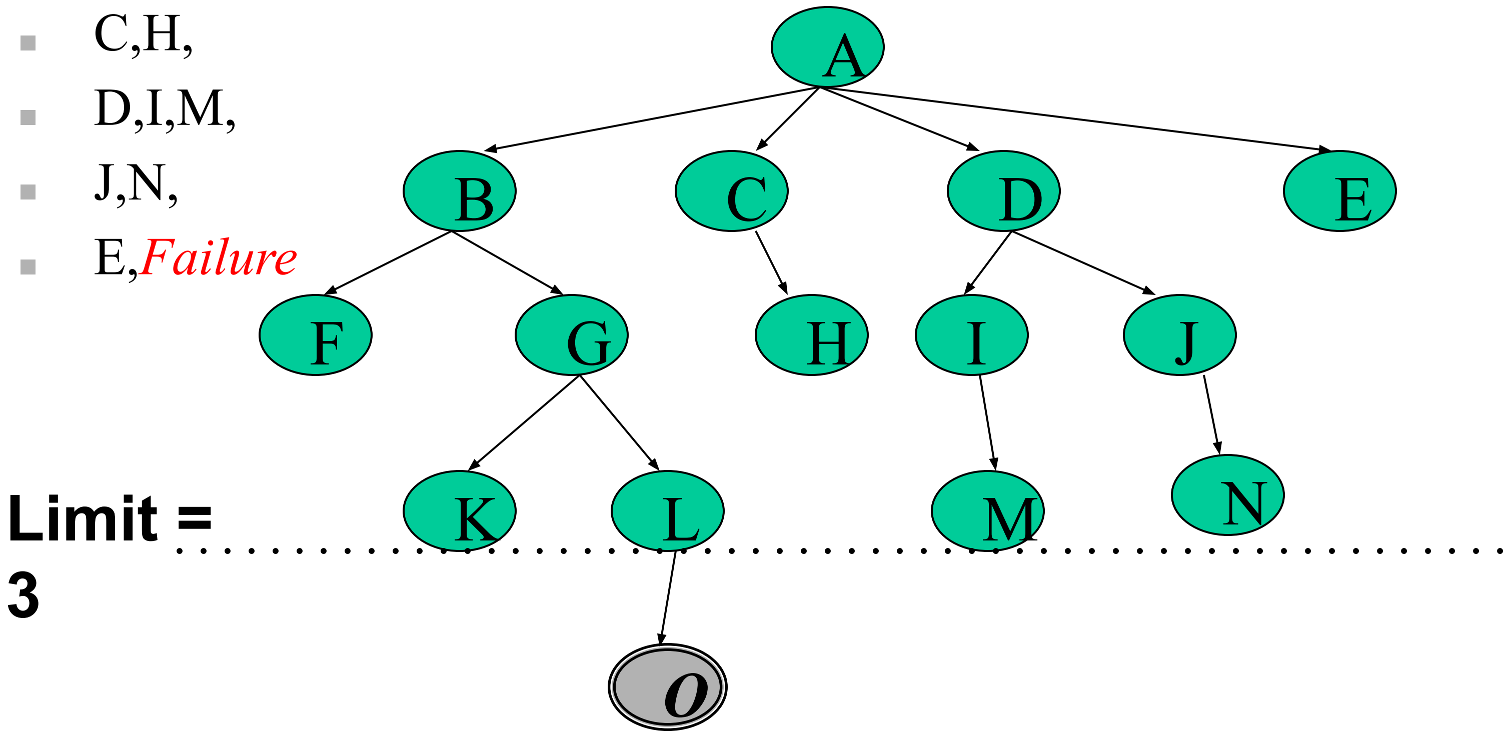
- A,B,F,
- G,K,
- L,
- C,H,
- D,I,M,
- J,N,
- E,

Limit = 3



Iterative Deepening Search (IDS)

- A,B,F,
- G,K,
- L,
- C,H,
- D,I,M,
- J,N,
- E,*Failure*

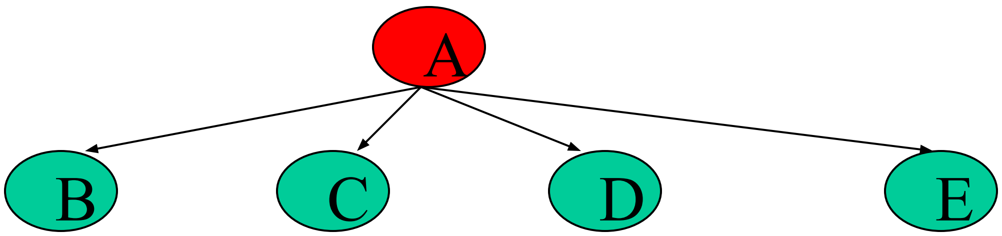


Iterative Deepening Search (IDS)

DLS with bound = 4

Iterative Deepening Search (IDS)

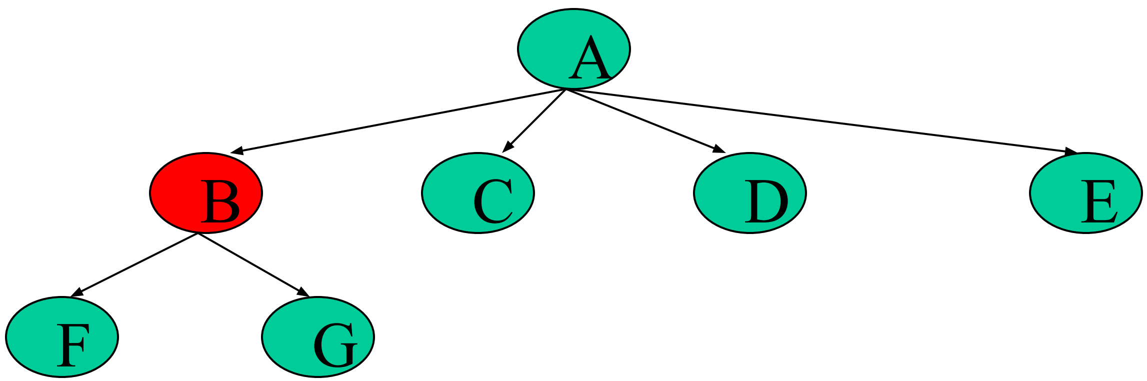
- A,



Limit = 4

Iterative Deepening Search (IDS)

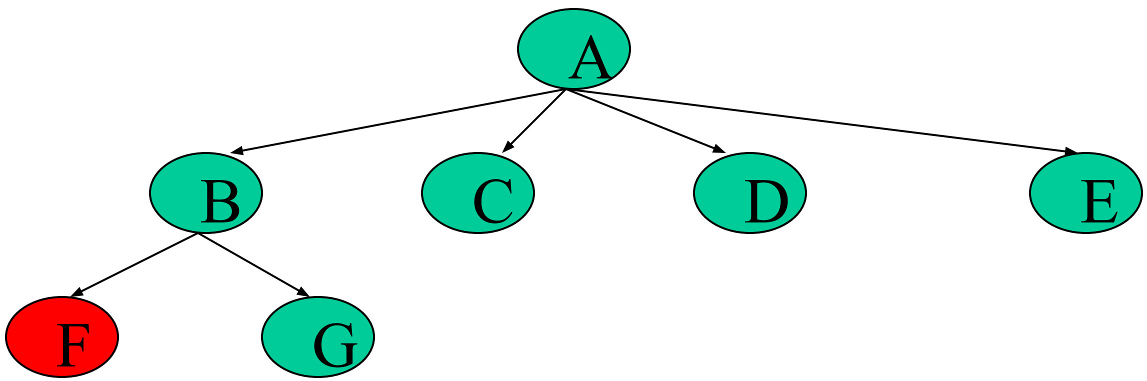
- A,B,



Limit = 4

Iterative Deepening Search (IDS)

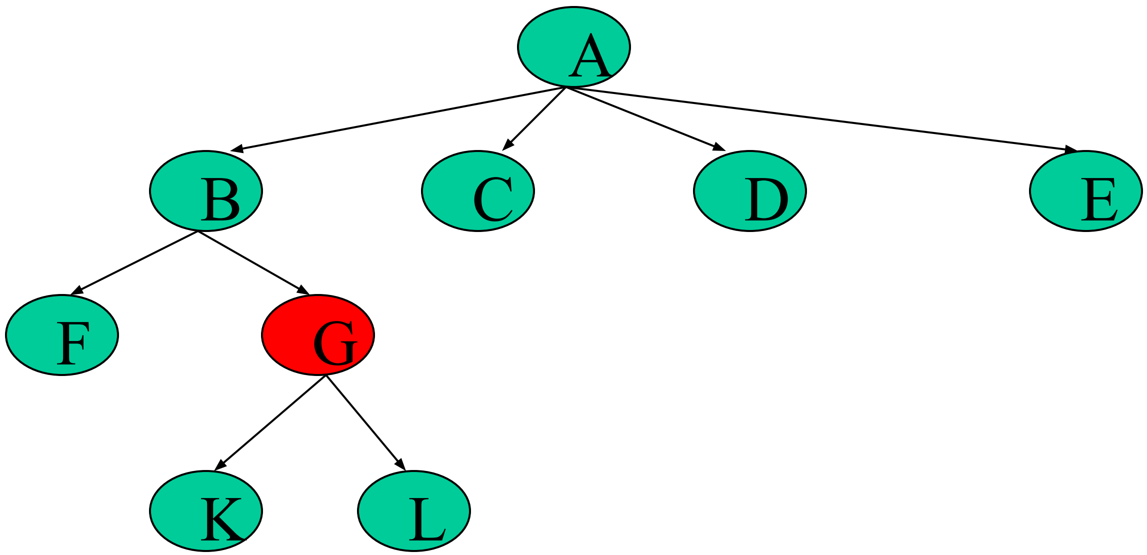
- A,B,F,



Limit = 4

Iterative Deepening Search (IDS)

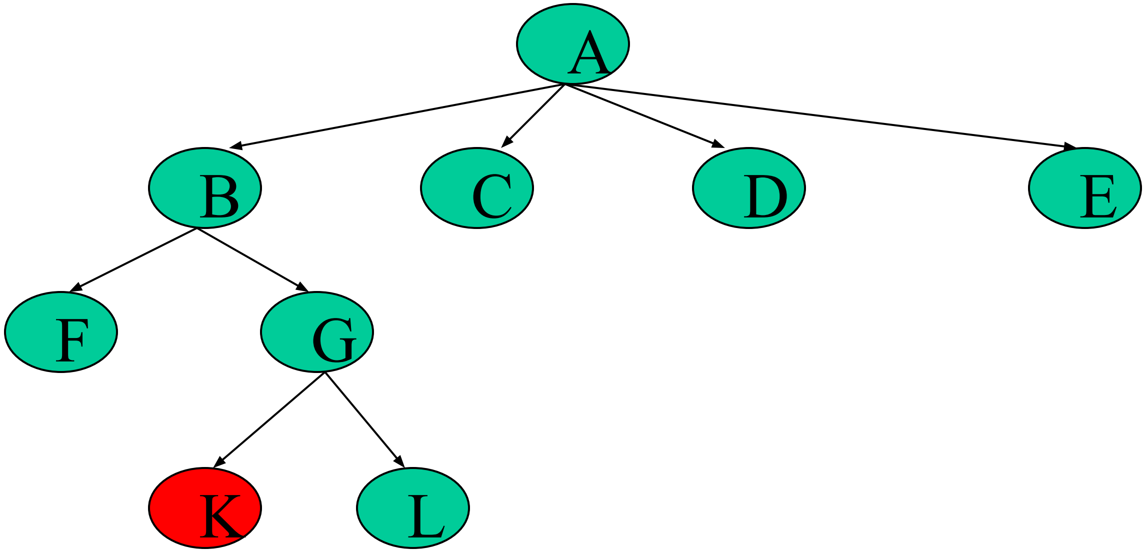
- A,B,F,
- G,



Limit = 4

Iterative Deepening Search (IDS)

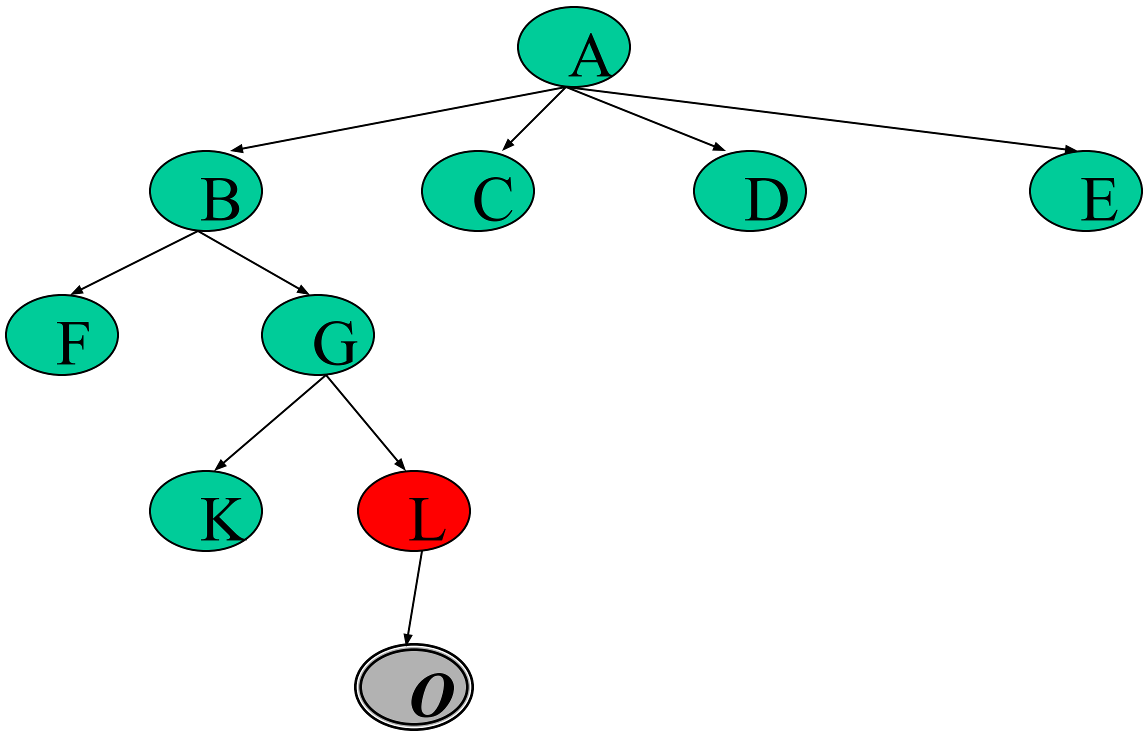
- A,B,F,
- G,K,



Limit = 4

Iterative Deepening Search (IDS)

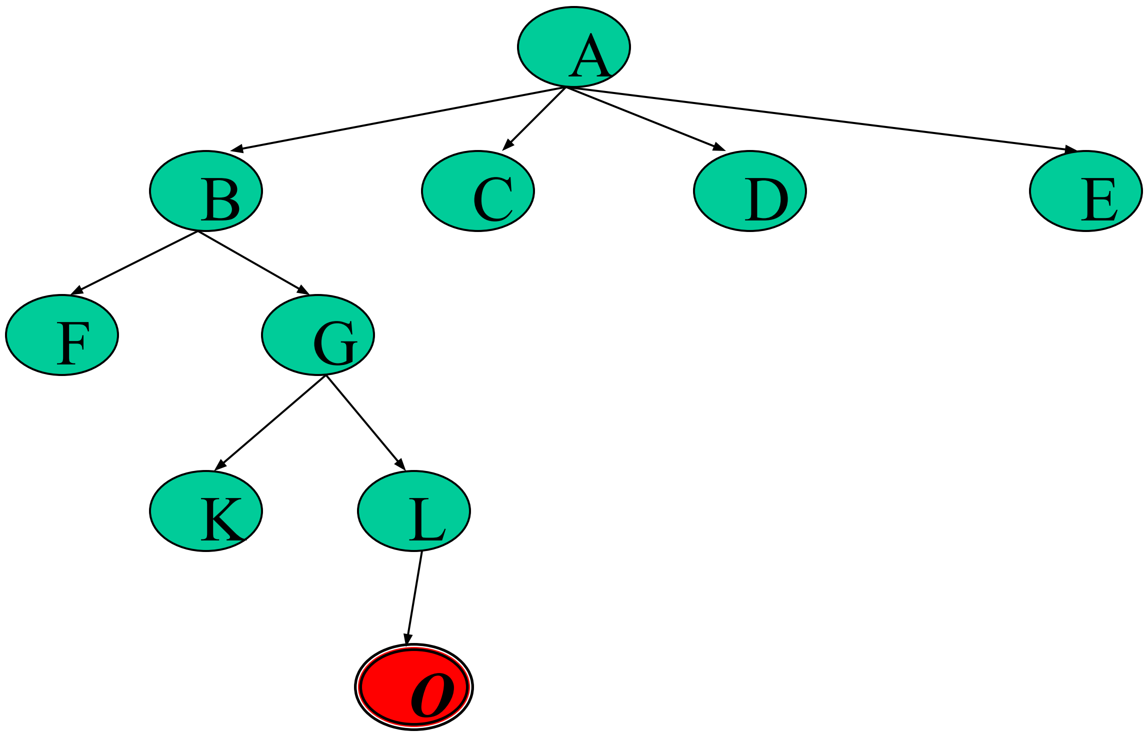
- A,B,F,
- G,K,
- L,



Limit = 4

Iterative Deepening Search (IDS)

- A,B,F,
- G,K,
- L, *O*: *Goal State*

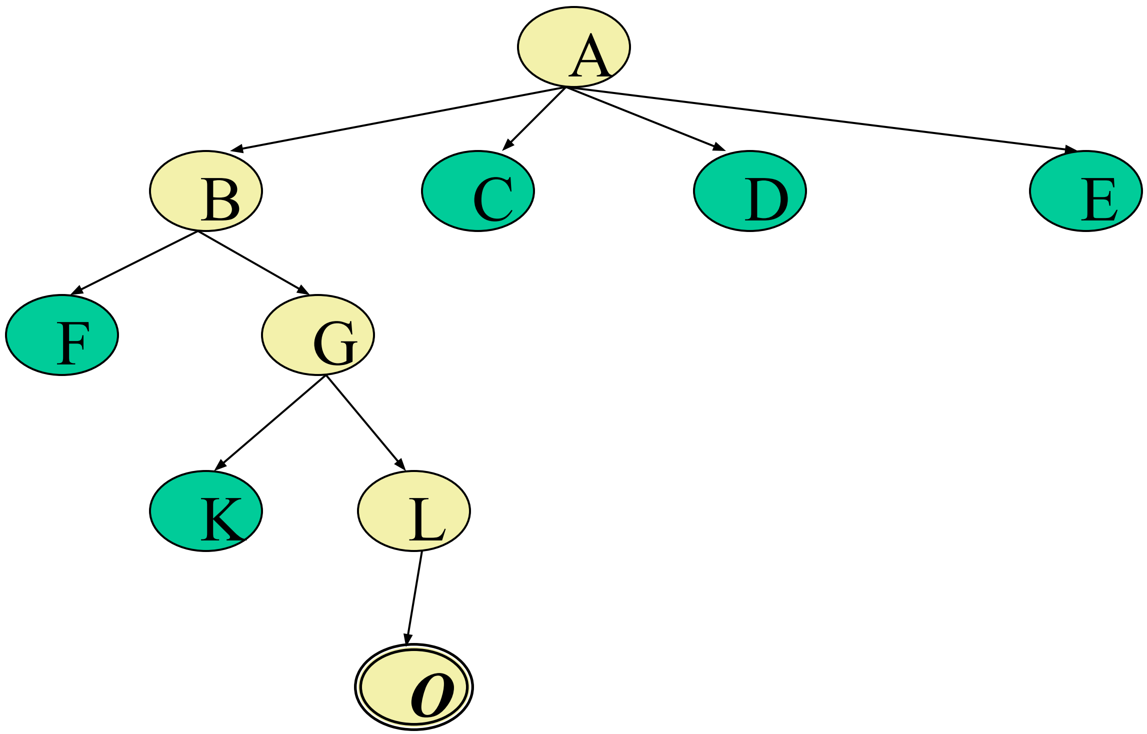


Limit = 4

Iterative Deepening Search (IDS)

The returned solution is the sequence of operators in the path:

A, B, G, L, O



Uniform Cost Search (UCS)

Main idea: Uniform-cost Search: Expand node with smallest path cost g(n).

- **Implementation**:

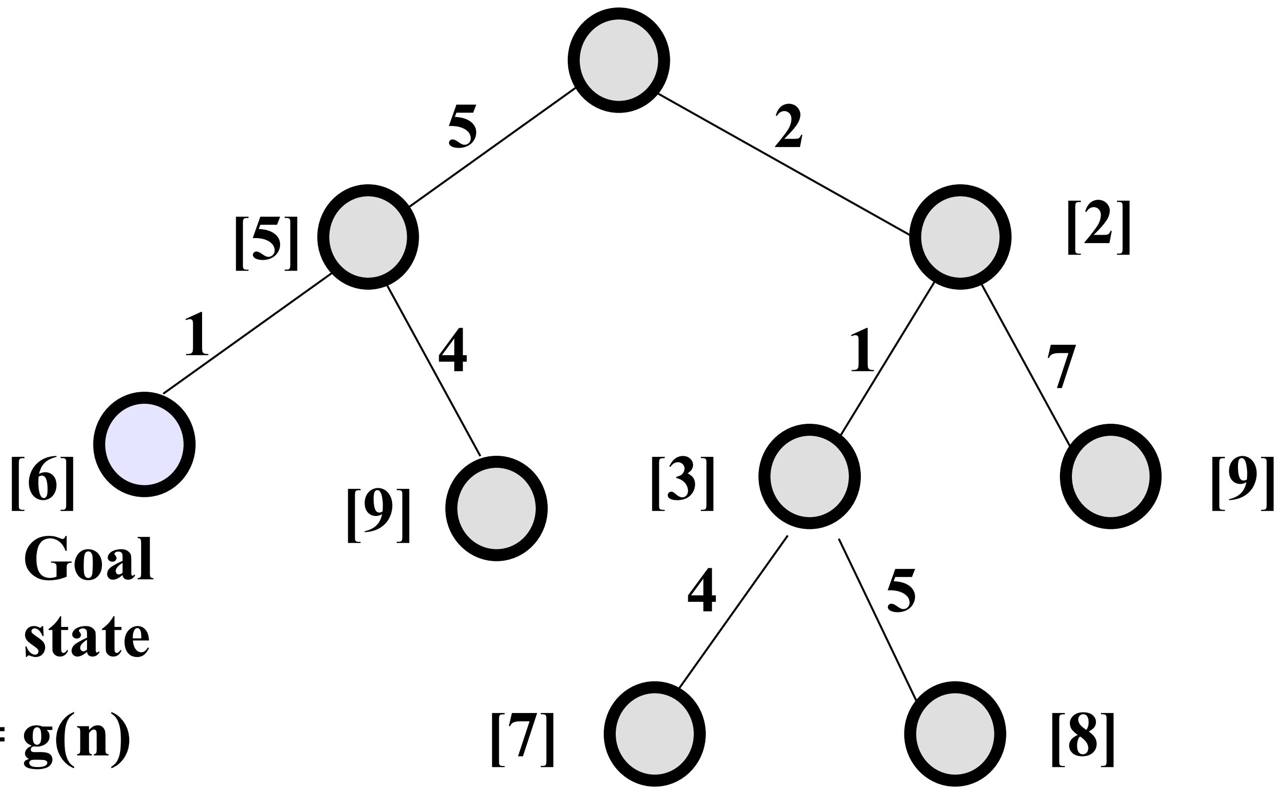
  Enqueue nodes in order of cost g(n).

  QUEUING-FN:- insert in order of increasing path cost.

  Enqueue new node at the appropriate position in the queue so that we dequeue the cheapest node.

- Complete? Yes.
- Optimal? Yes, if path cost is nondecreasing function of depth
- Time Complexity: $O(b^d)$
- Space Complexity: $O(b^d)$, note that every node in the fringe keep in the queue.

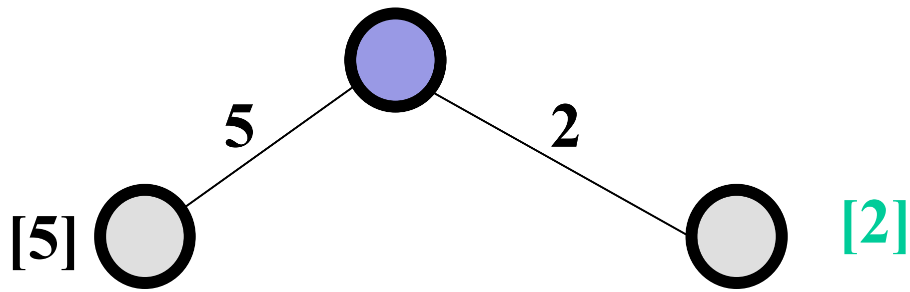
Uniform Cost Search (UCS)



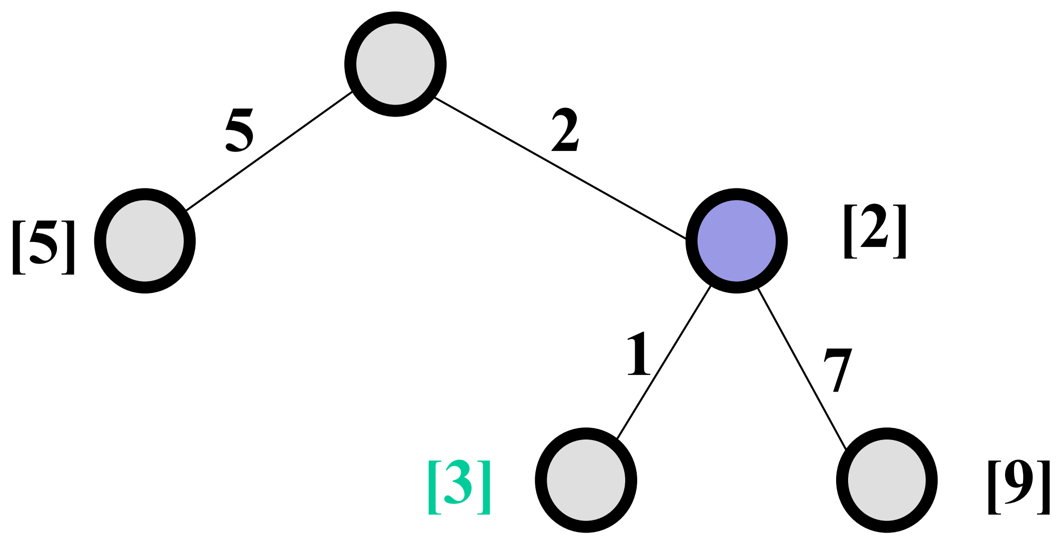
[x] = g(n)

path cost of node n

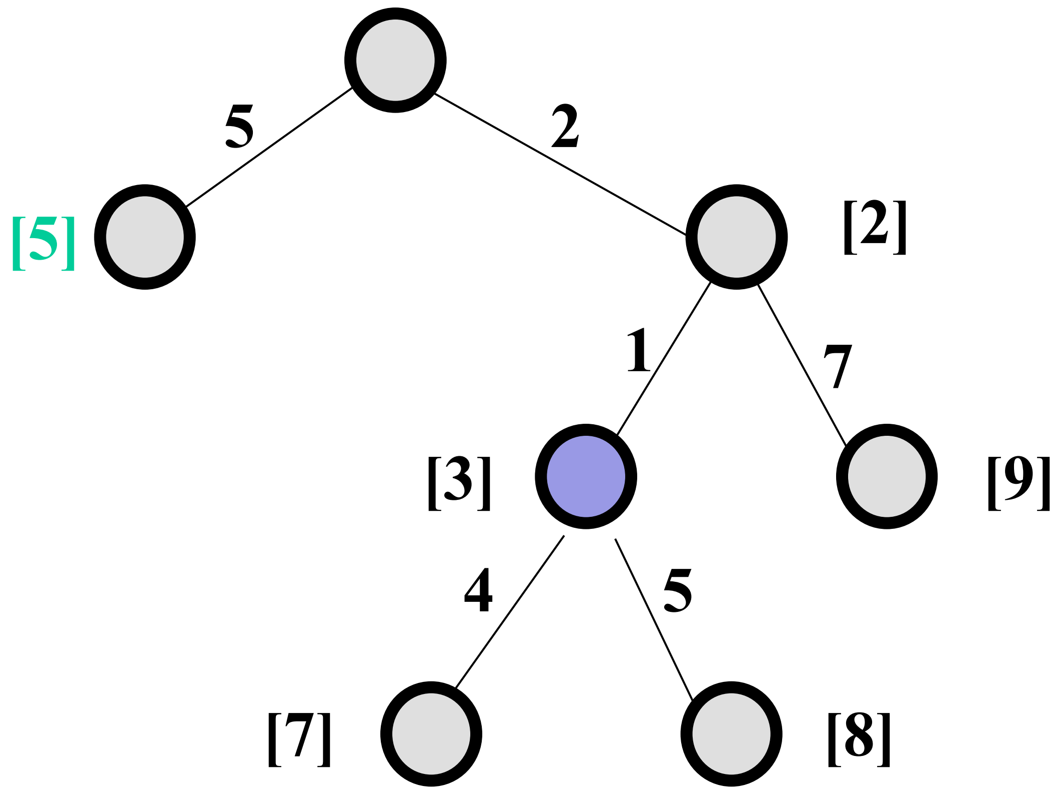
Uniform Cost Search (UCS)



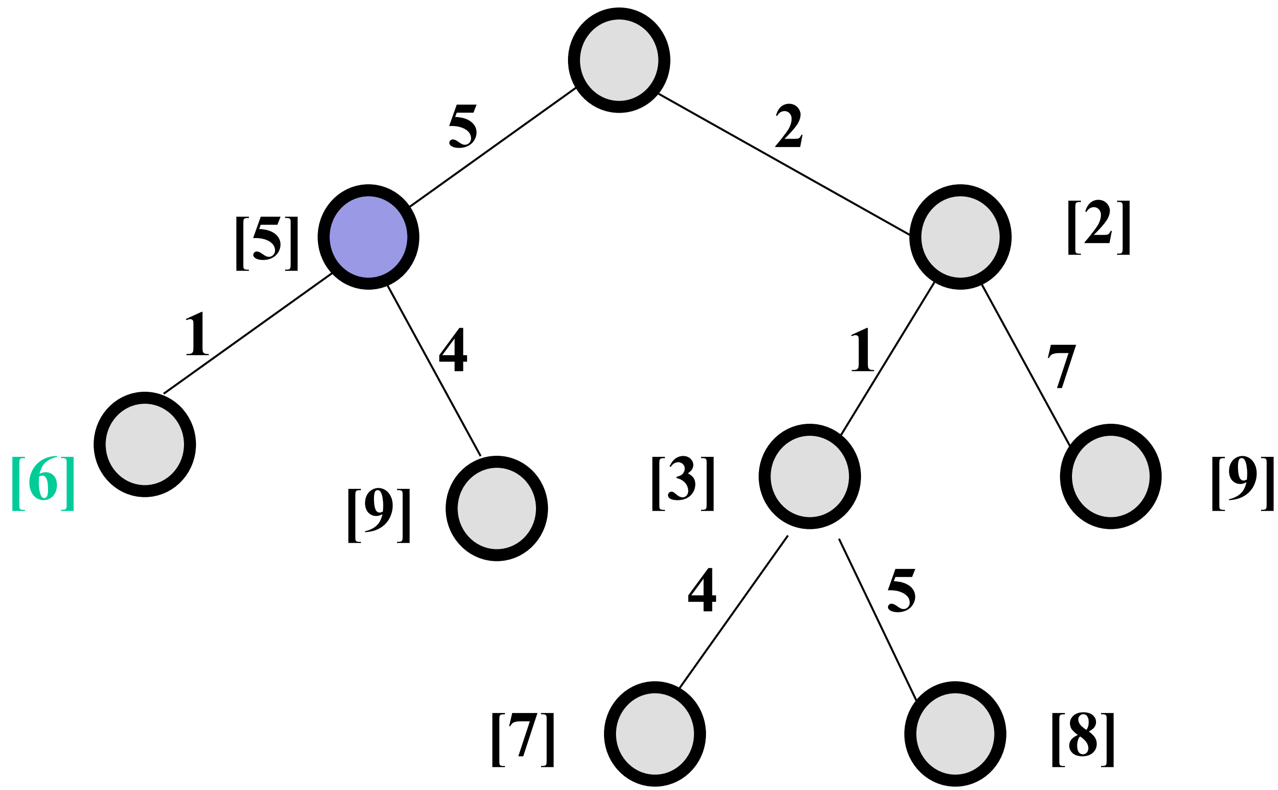
Uniform Cost Search (UCS)



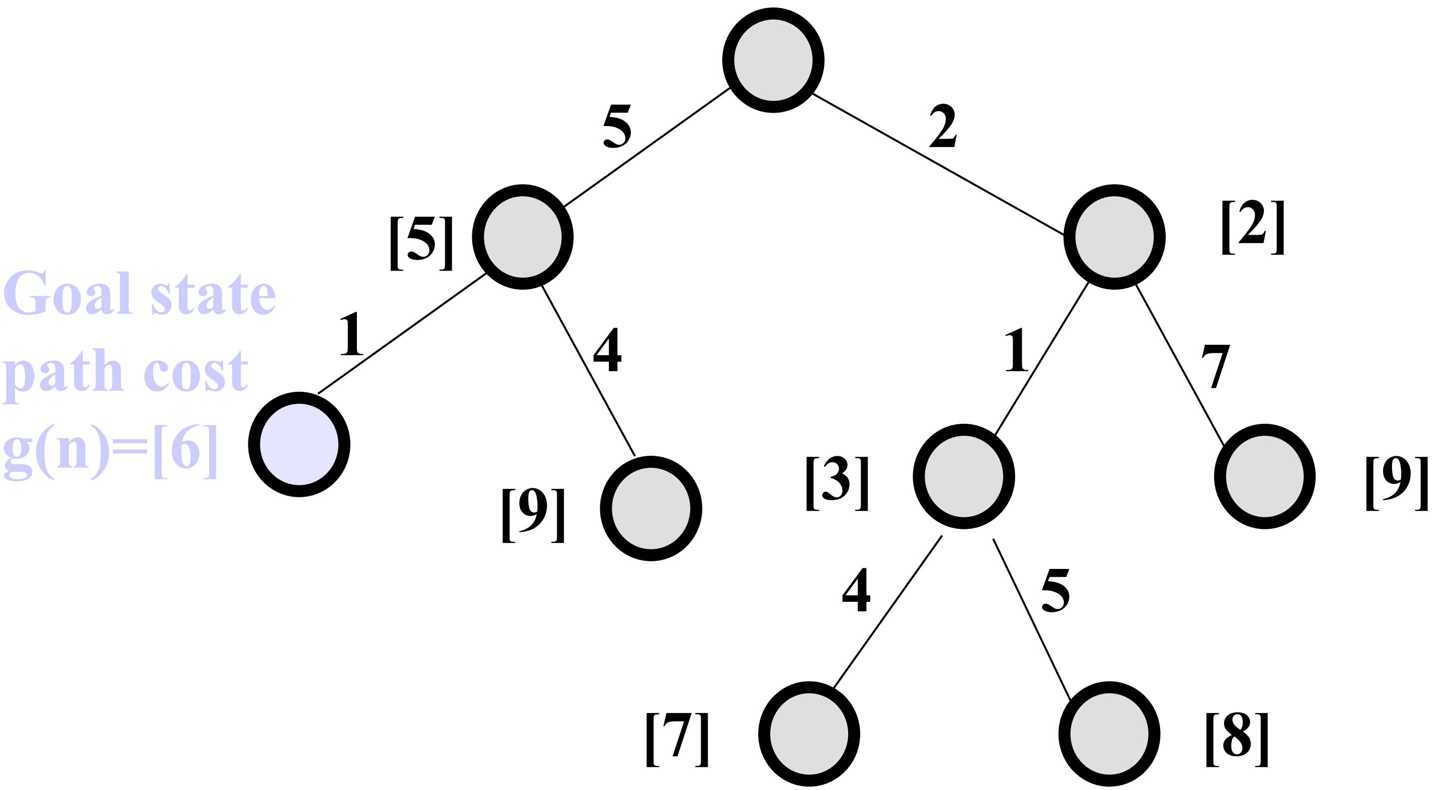
Uniform Cost Search (UCS)



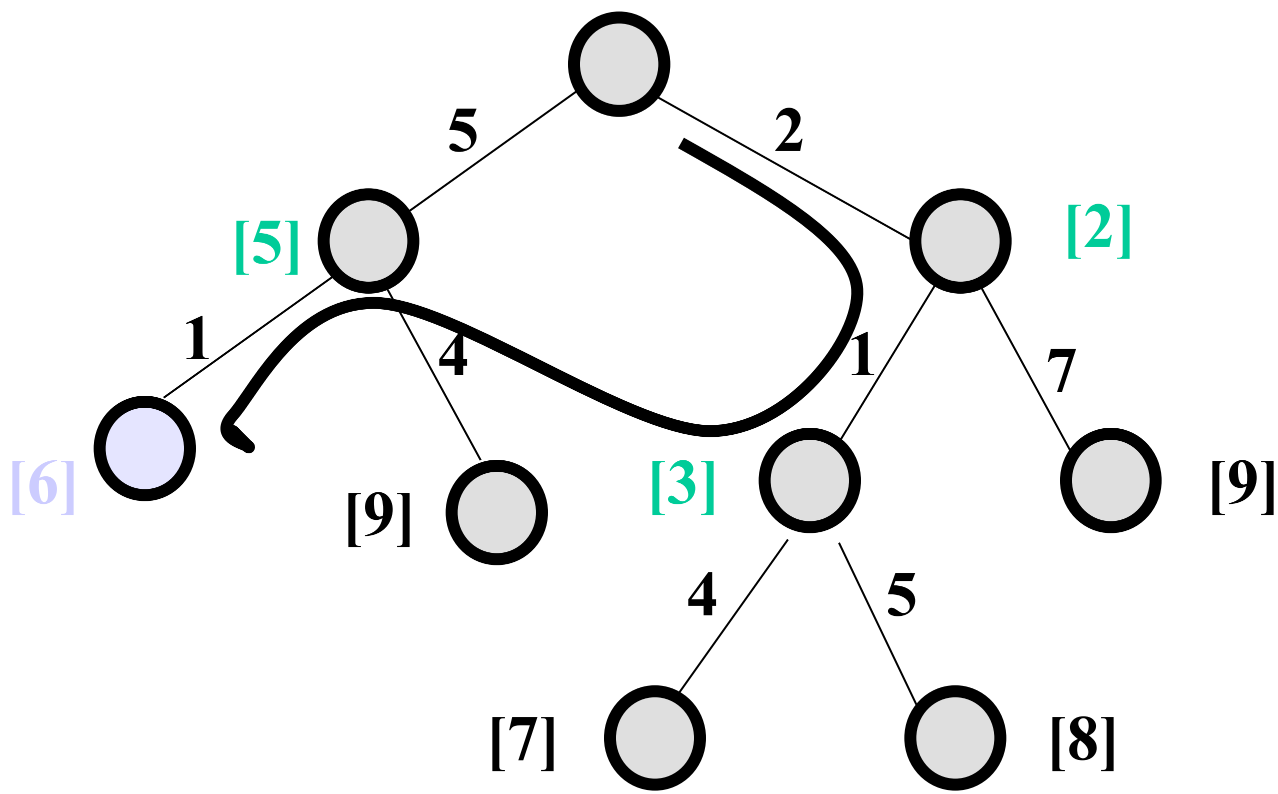
Uniform Cost Search (UCS)



Uniform Cost Search (UCS)



Uniform Cost Search (UCS)



Uniform-cost search

Breadth-first is only optimal if step costs is increasing with depth (e.g. constant). Can we guarantee optimality for any step cost?

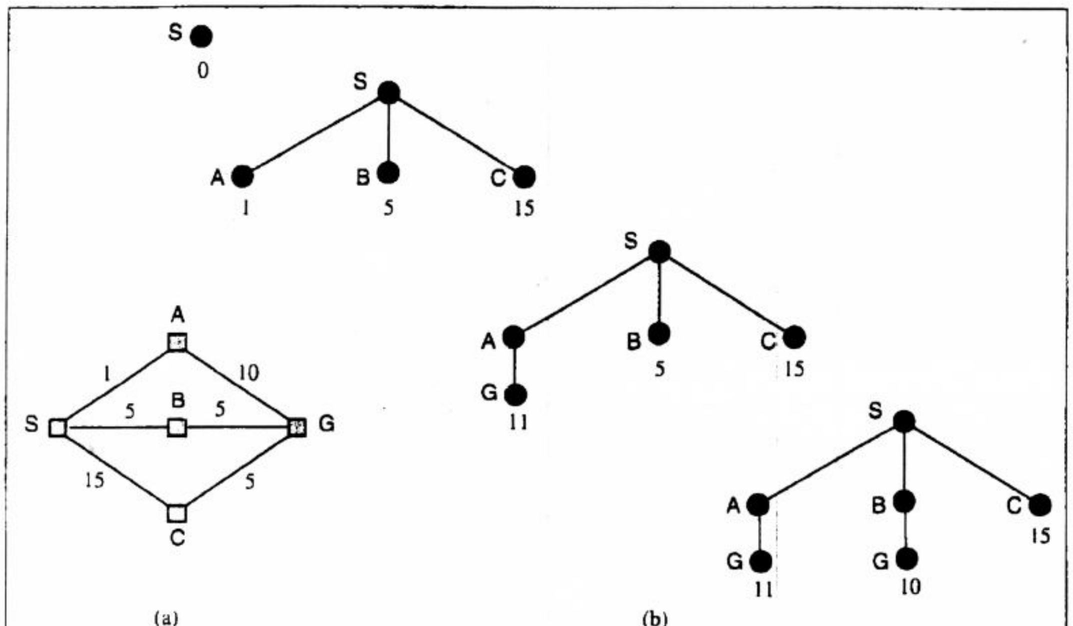
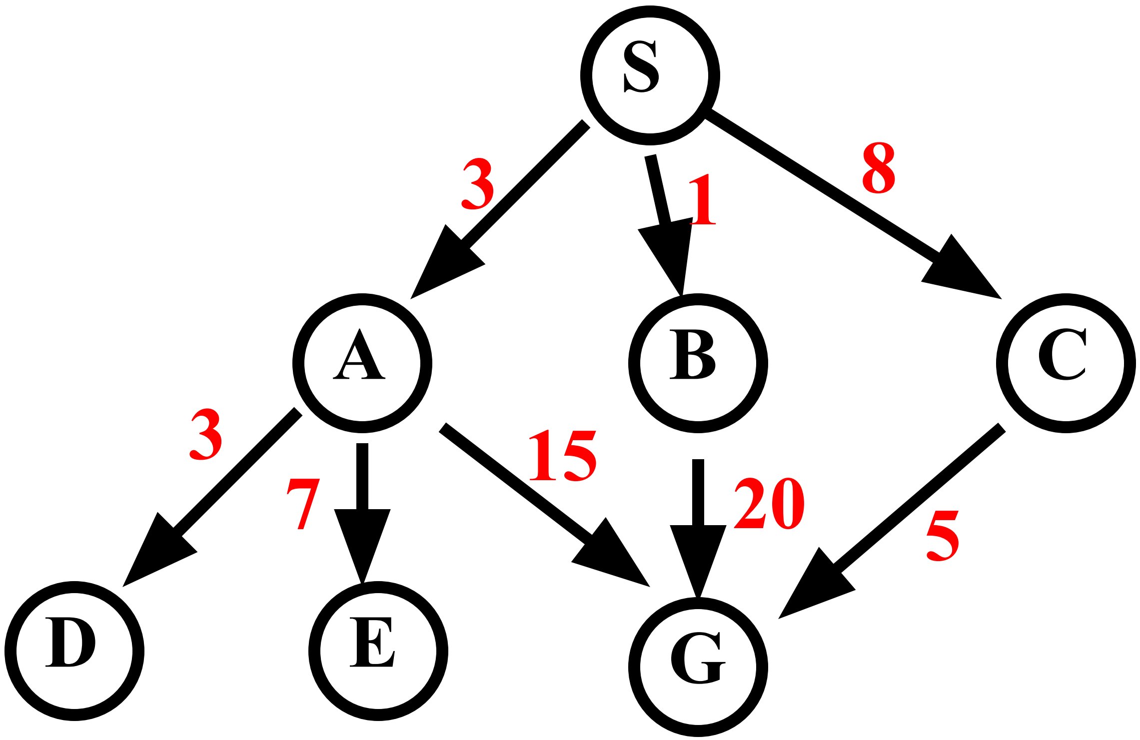
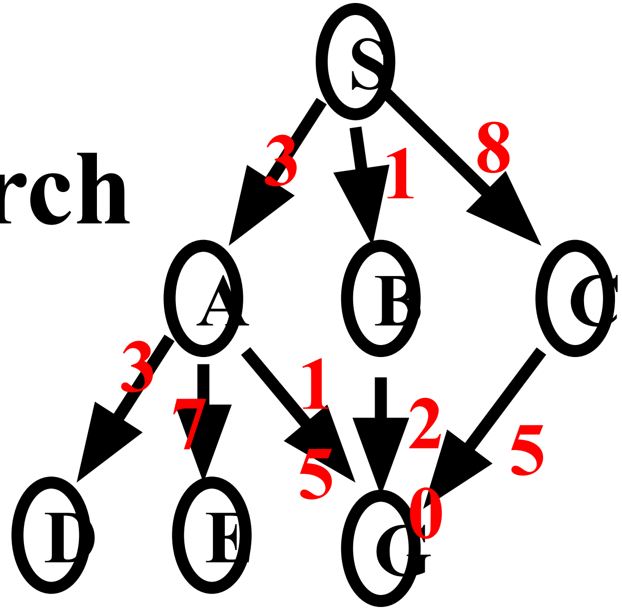


Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Example for Illustrating Search Strategies



Depth-First Search

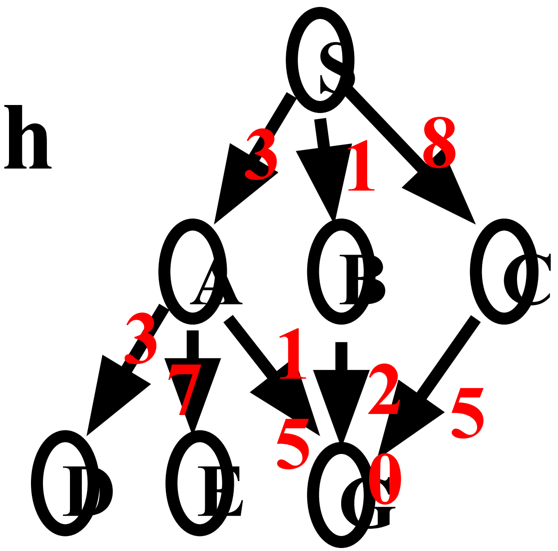


Expanded node	**Nodes list**
	{ S^0 }
S^0	{ A^3 B^1 C^8 }
A^3	{ D^6 E^{10} G^{18} B^1 C^8 }
D^6	{ E^{10} G^{18} B^1 C^8 }
E^{10}	{ G^{18} B^1 C^8 }
G^{18}	{ B^1 C^8 }

Solution path found is S A G, cost 18

Number of nodes expanded (including goal node) = 5

Breadth-First Search

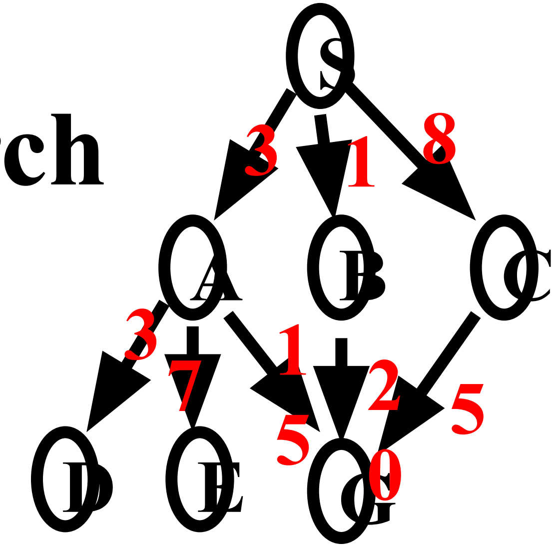


Expanded node	Nodes list
	$\{ S^0 \}$
S^0	$\{ A^3\ B^1\ C^8 \}$
A^3	$\{ B^1\ C^8\ D^6\ E^{10}\ G^{18} \}$
B^1	$\{ C^8\ D^6\ E^{10}\ G^{18}\ G^{21} \}$
C^8	$\{ D^6\ E^{10}\ G^{18}\ G^{21}\ G^{13} \}$
D^6	$\{ E^{10}\ G^{18}\ G^{21}\ G^{13} \}$
E^{10}	$\{ G^{18}\ G^{21}\ G^{13} \}$
G^{18}	$\{ G^{21}\ G^{13} \}$

Solution path found is S A G , cost 18

Number of nodes expanded (including goal node) = 7

Uniform-Cost Search



Expanded node	**Nodes list**
	$\{ S^0 \}$
S^0	$\{ B^1\ A^3\ C^8 \}$
B^1	$\{ A^3\ C^8\ G^{21} \}$
A^3	$\{ D^6\ C^8\ E^{10}\ G^{18} \}$
D^6	$\{ C^8\ E^{10}\ G^{18} \}$
C^8	$\{ E^{10}\ G^{13} \}$
E^{10}	$\{ G^{13} \}$
G^{13}	$\{\}$

Solution path found is S C G, cost 13

Number of nodes expanded (including goal node) = 7

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?

What Criteria are used to Compare different search techniques ?

As we are going to consider different techniques to search the problem space, we need to consider what criteria we will use to compare them.

- **Completeness**: Is the technique guaranteed to find an answer (if there is one).
- **Optimality/Admissibility** : does it always find a least-cost solution?
  - an admissible algorithm will find a solution with minimum cost
- **Time Complexity**: How long does it take to find a solution.
- **Space Complexity**: How much memory does it take to find a solution.

Time and Space Complexity ?

Time and space complexity are measured in terms of:

- The average number of new nodes we create when expanding a new node is the (effective) branching factor **b**.
- The (maximum) branching factor **b** is defined as the maximum nodes created when a new node is expanded.
- The length of a path to a goal is the depth **d**.
- The maximum length of any path in the state space **m**.

Properties of breadth-first search

- Complete? Yes it always reaches goal (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d+(b^{d+1}-b)) = O(b^{d+1})$
  (this is the number of nodes we generate)
- Space? $O(b^{d+1})$ (keeps every node in memory, either in fringe or on a path to fringe).
- Optimal? Yes (if we guarantee that deeper solutions are less optimal, e.g. step-cost=1).

- Space is the bigger problem (more than time)

Properties of depth-first search

- Complete? No: fails in infinite-depth spaces

  Can modify to avoid repeated states along path
- Time? $O(b^m)$ with m=maximum depth
- terrible if m is much larger than d
  - but if solutions are dense, may be much faster than breadth-first
- 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)

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\, b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 or increasing function of depth.

Uniform-cost search

Implementation: *fringe* = queue ordered by path cost
Equivalent to breadth-first if all step costs all equal.

Complete? Yes, if step cost $\geq \varepsilon$
(otherwise it can get stuck in infinite loops)

Time? # of nodes with *path cost* $\leq$ cost of optimal solution.

Space? # of nodes on paths with path cost $\leq$ cost of optimal solution.

Optimal? Yes, for any step cost.

Bi-Directional Search

Complexity: time and space complexity are: $O(b^{d/2})$

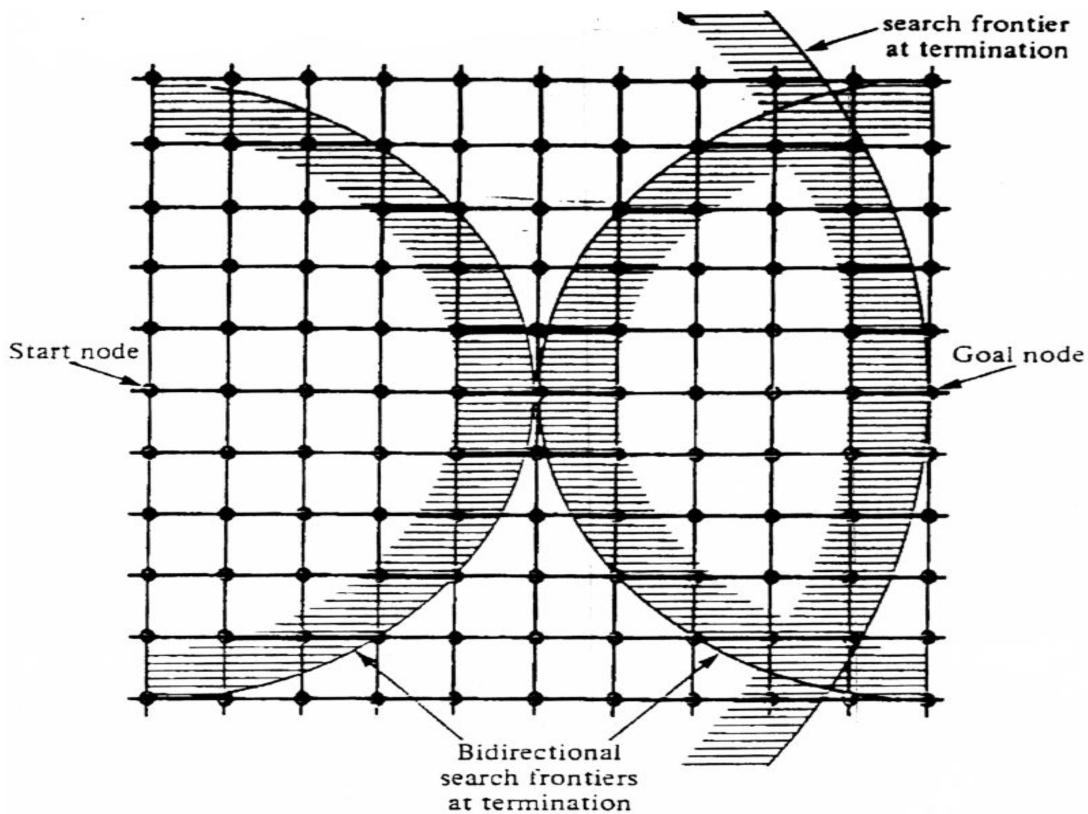


Fig. 2.10 Bidirectional and unidirectional breadth-first searches.

Summary of 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