Uninformed Search

Unit 2: Uninformed

Search Strategy

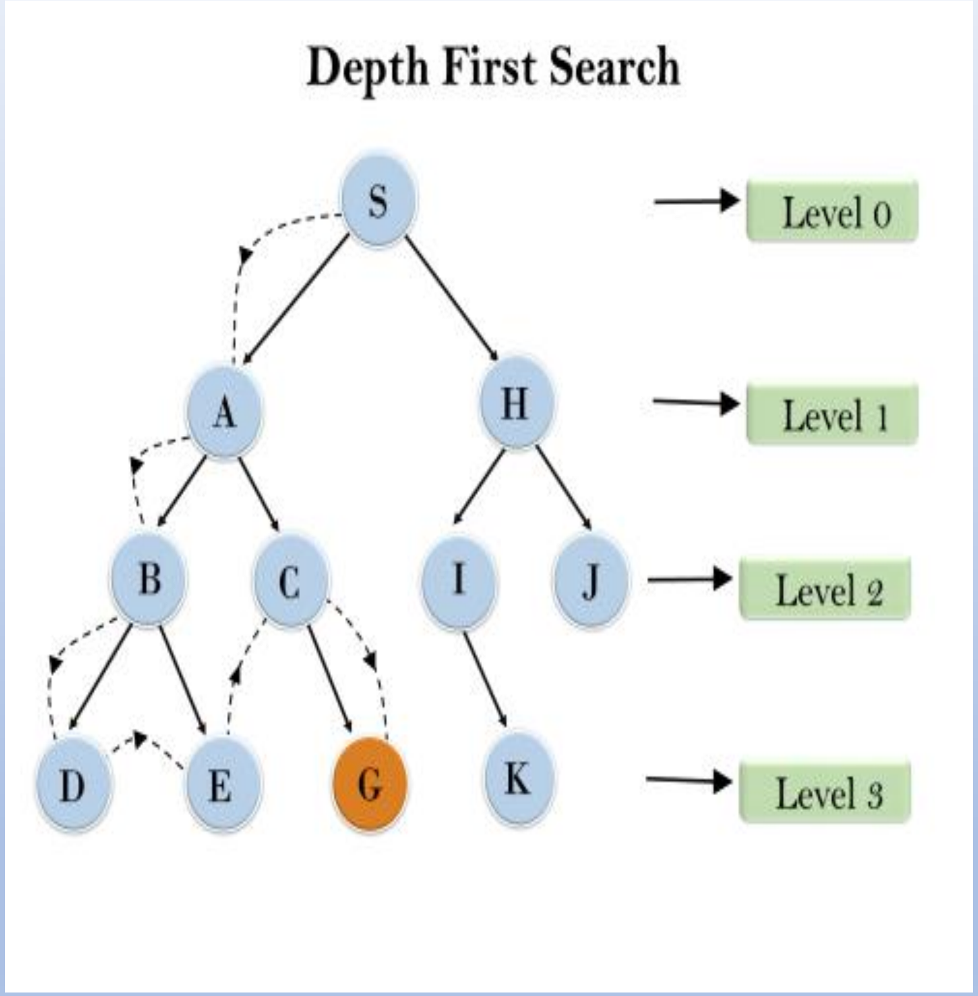


Table of Content

- Implicit and Explicit Search
- General Search tree algorithm
- Performance measuring criterion of an algorithm
- Uninformed Search tree
- Breadth first Search(BFS)
- Depth First Search(DFS)

Search

Searching through a state space involves the following:

- A set of states
- Operators and their cost
- Start state
- A test to check for goal state

Explicit and Implicit Search Tree

- Explicit Search tree:
  - ➢ Explicit representation of node set and edges
  - ➢ Store in memory

- Implicit Search tree:
  - ➢ Graph is too large to generate and store in memory
  - ➢ Nodes are generated as they are explored.

General Search tree algorithm

Function tree_search(problem,strategy) return a solution or failure

Initialize the search tree using the initial state

Loop do

- If there are no candidates for expansion then return failure
- Choose a leaf node for expansion according to strategy
- If node contains a goal state then return the corresponding solution
- Else expand the node and add the resulting nodes to the search tree

Measuring Problem Solving Performance

- Completeness: algorithm guaranted find a solution when there is one.
- Optimality: does the strategy find optimal solution
- Time Complexity: how long does it takes to find solution
- Space Complexity: how much memory is needed to perform a search

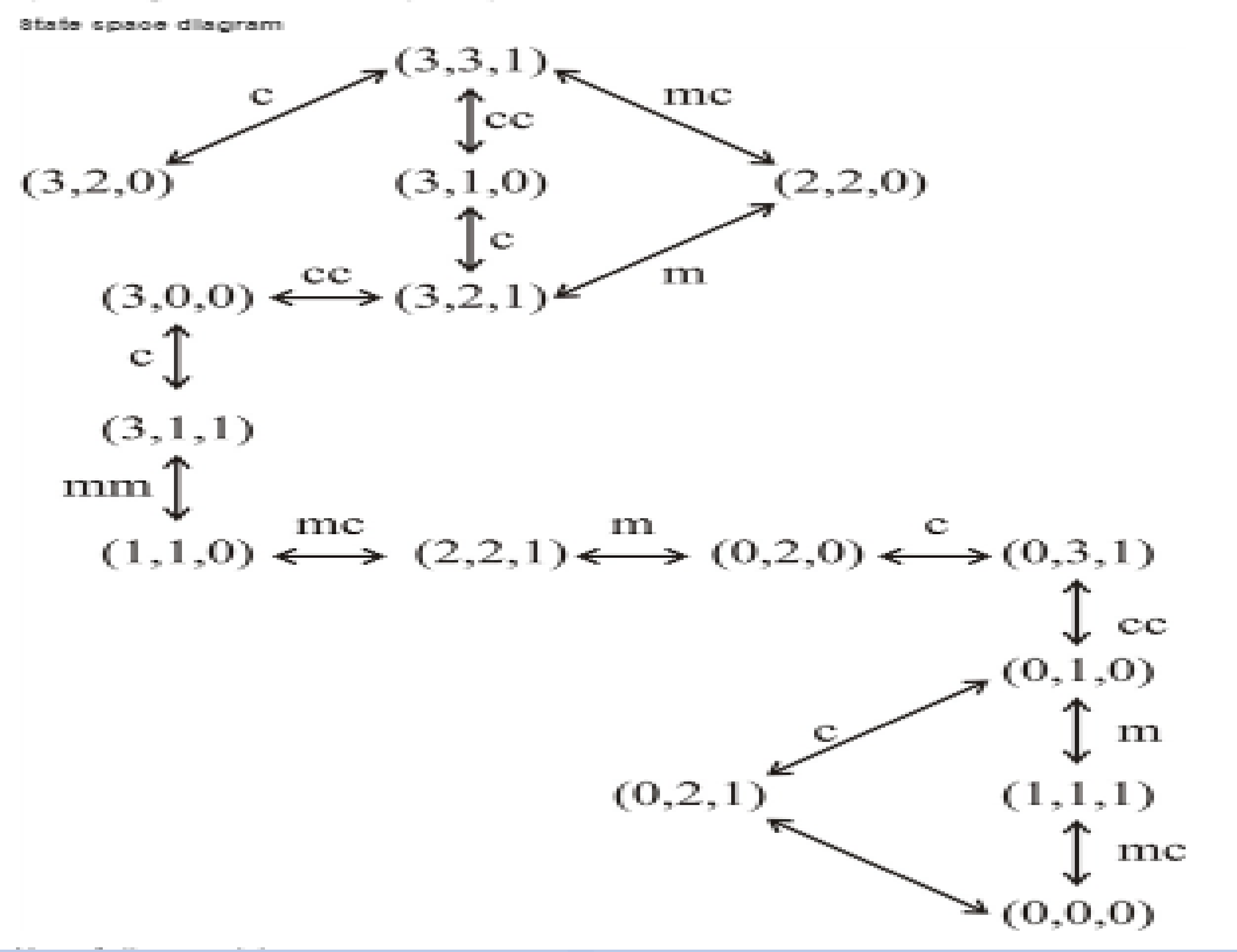
Uninformed Search or Blind Search

- No clue whether one non goal state is better than any other
- Don't know if your current exploration is likely to be fruitful.
- Search is blind
- Example:
  - Breadth First Search
  - Depth First Search
  - Iterative Deepening Search
  - Bi-directional Search

Search Tree Terminology

- The search start with initial state or root node
- The OPEN set contains the generated nodes.
- The algorithm picks one node from OPEN, expand it and keep it in CLOSED
- A solution to the search problem is a sequence of operators.
- Search tree may be infinite because of loops even the number of states is small

Search Tree: Missionaries and Cannibals problem



Breadth First Search (BFS)

Let OPEN/fringe be a list containing the initial state

Loop

if OPEN/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,

add generated nodes to the back of fringe

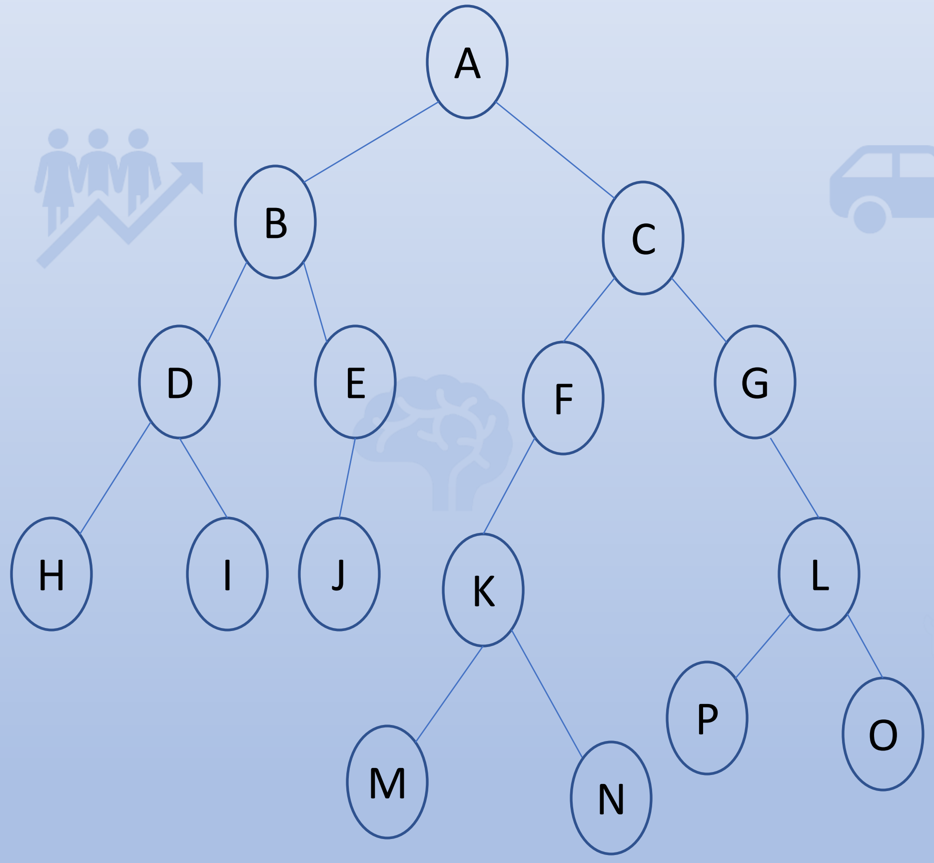
End Loop

Breadth First Search (BFS)

Start State: A

Goal State: G

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



Depth First Search (DFS)

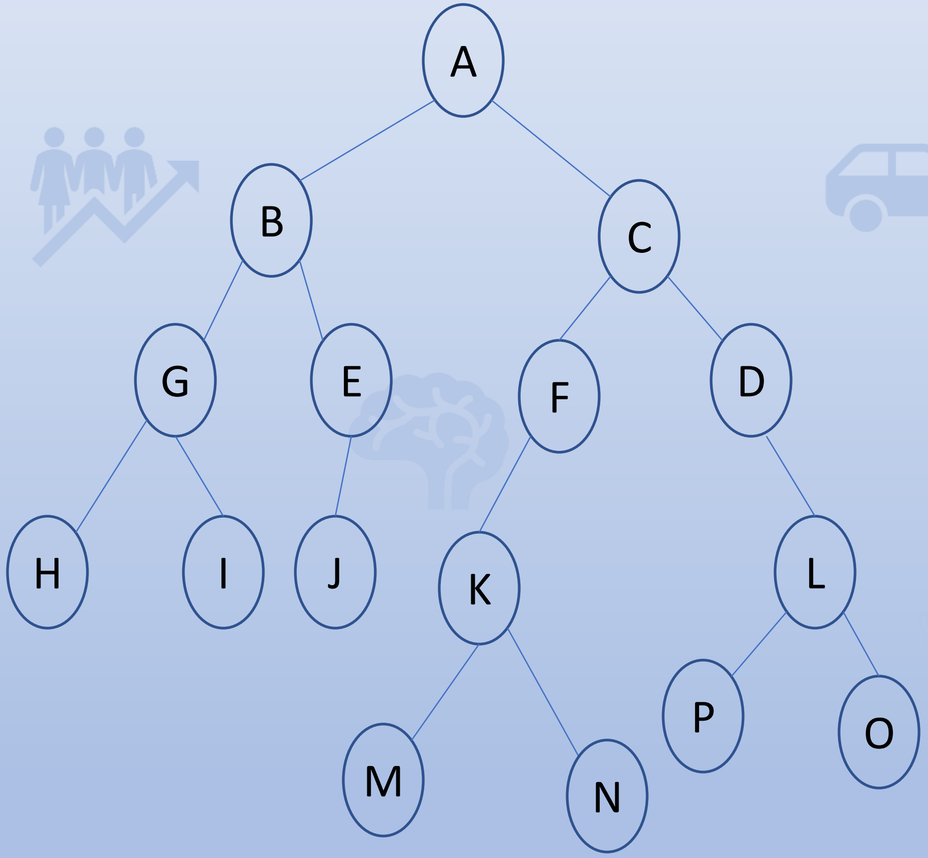
```
Let OPEN/fringe be a list containing the initial state
 Loop
        if OPEN/fringe is empty return failure
        Node <-remove-Last (fringe)
        if Node is a goal
                 then return the path from initial state to Node
        else
                generate all successors of Node,
                add generated nodes to the back of fringe
 End Loop
```

Depth First Search (BFS)

Start State: A

Goal State: G

OPEN	CLOSED
A	
B,C	A
B,F,D	A,C
B,F,L	A,C,D
B,F,P,O	A,C,D,L
B,F,P	A,C,D,L,O
B,F	A,C,D,L,O,P
B,K	A,C,D,L,O,P,F



Time Complexity of BFS

- Progress of the search is a simple complete binary search tree.
- How much memory it will take to execute
- Every state has b child(say)
- Root of the search tree generate b nodes at the first level.
- b^2 nodes generated at second level.
- Let the solution of the problem has a path length of d
- The maximum numbers of nodes expanded before finding a solution is
- $1+b+b^2+b^3+......+ b^d$

Time Complexity of BFS

- The time and space required for BFS is O(b^d)
- The following figure represent the time and space requirement for BFS with branching factor b=10

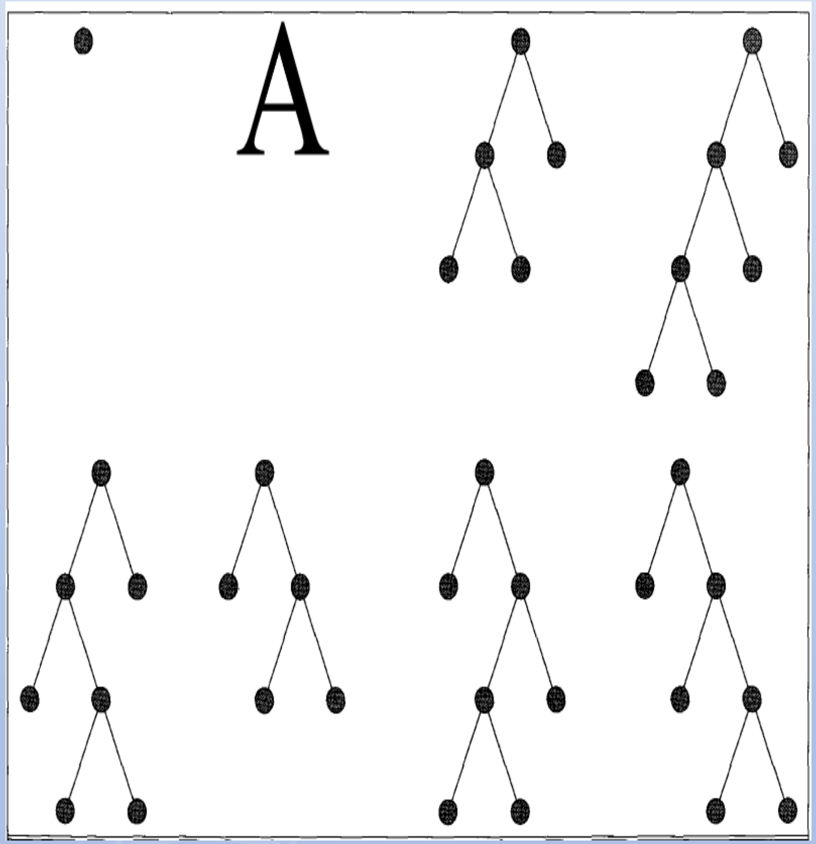
Depth	Nodes	Time	Memory
2	1100	.11 seconds	1 megabyte
4	111,100	11 seconds	106 megabytes
6	10^7	19 minutes	10 gigabytes
8	10^9	31 hours	1 terabytes
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

Figure 3.11 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 10,000 nodes/second; 1000 bytes/node.

Memory requirement is a bigger problem than execution time

Time and Space Complexity of DFS

- Always expand one of the node at the deepest level of the tree
- The search go back and expand nodes at shallower level when search hits at the dead end.
- Modest memory requirement O(bm)
- Time complexity O(b^m), m is the maximum depth.



Advantages of BFS

Advantages of BFS

- ➢BFS will not get trapped exploring a blind alley.
- ➢It is complete. If there is a solution, then BFS is guaranteed to find it.

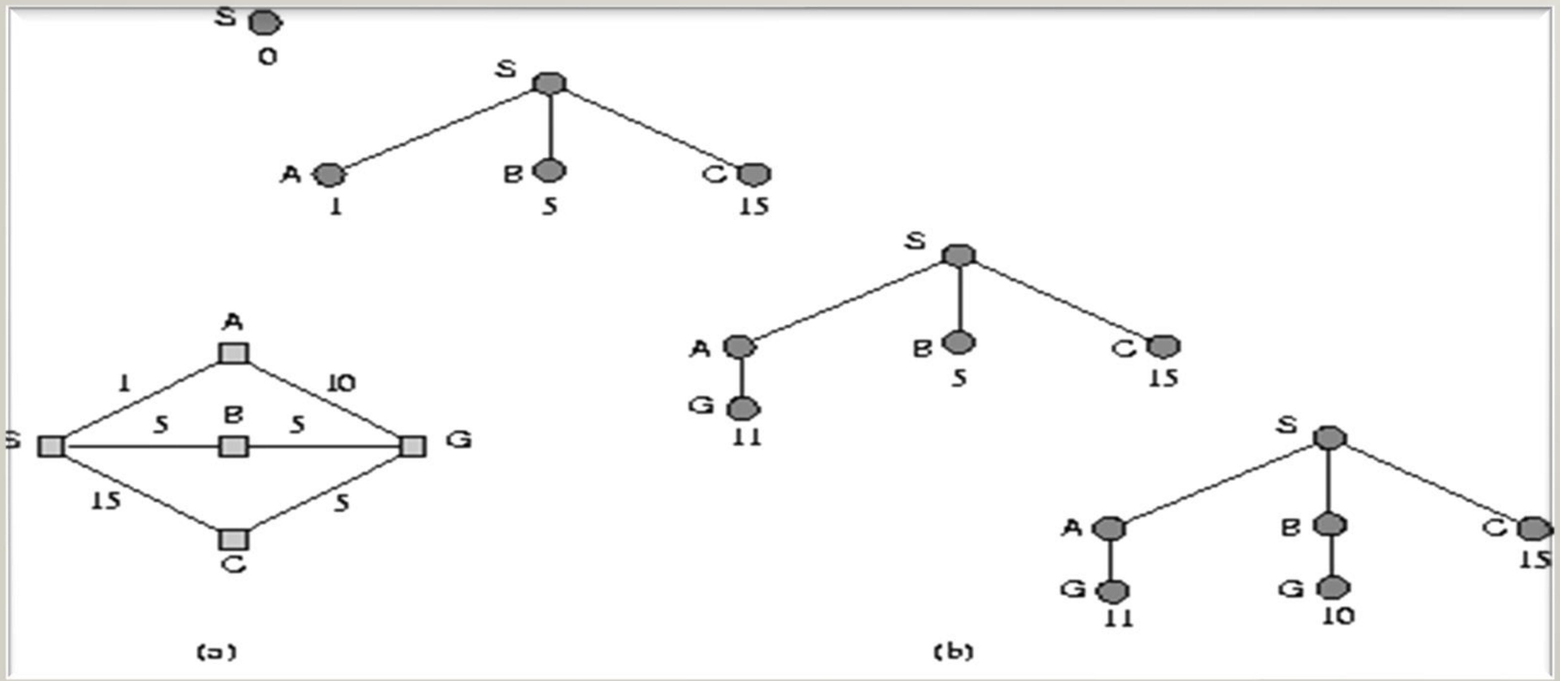
Advantages of DFS

- ➢Requires less memory since only the nodes on the current path are stored.
- ➢Depth first search may find a solution without examining much of the search space at all.

Uniform Cost Search

- BFS find the shallowest goal state.
- Goal may not be the least cost solution for a general path cost function.
- Uniform Cost Search is a modified version of BFS, always expand the lowest cost node on the fringe
- Guarantees to find out cheapest solution.
- The path cost should be nondecreasing.

Uniform Cost Search



Depth Limited Search

- Depth limited search avoid the pitfalls of DFS by imposing a cut-off on the maximum depth of a path.
- Is complete but not optimal
- If we choose depth limit too small, then is not even complete.
- The time and space complexity is similar to DFS
- Space complexity is O(bl), where l is the depth limit.
- Time complexity is O(b^l)

Iterative Deepening

- Introduced by Korf , Stickel Tyson 1985
- Utilize the linear memory requirements of depth-first search and ensure to find out goal node at minimal depth.
- Successive depth first searches are conducted.
- Each depth bound increase by 1.
- The number of nodes expanded is not more than DFS.

Iterative Deepening

IterativeDeepening(T,S,G)

For l=0 to infinity

DepthlimitedSearch(T,S,G,l)

- Overhead expansion is not large
- It search only twice of as long as complete BFS
- Time complexity is still O(b^d)
- Space complexity is O(bd)

Bidirectional Search

- Simultaneously search from the initial state S and backward from the goal
- It stops when two search meets at the middle.
- The solution will be found in$O\left(2 * b^{\frac{d}{2}}\right) = O(b^{\frac{d}{2}})$
- Applicable when all operators are reversable.

Comparing Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l > d$	Yes	Yes

Thank You