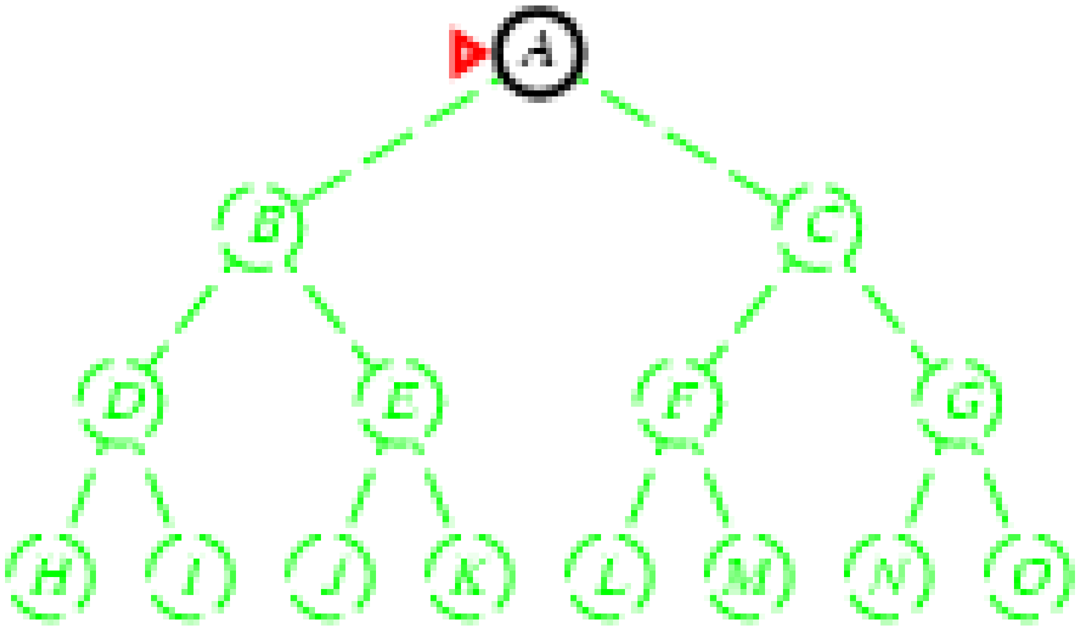
Artificial Intelligence

3.4: Solving Problem by Searching

Depth-first search

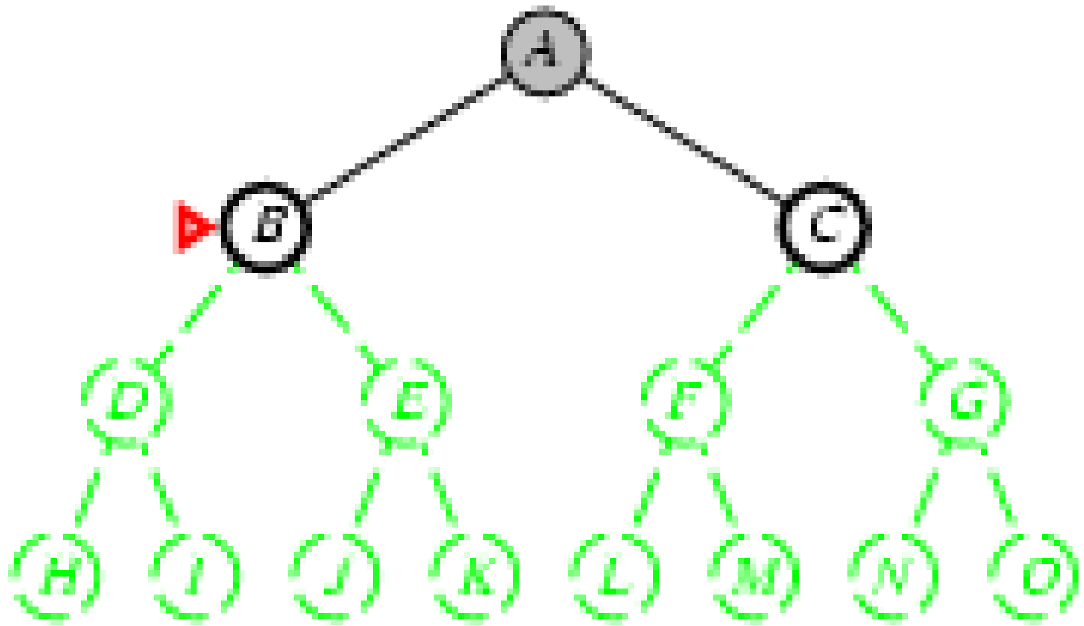
- Expand deepest unexpanded node in the current fringe
- LIFO-Stack
- Implementation:



Depth-first search

- Expand deepest unexpanded node

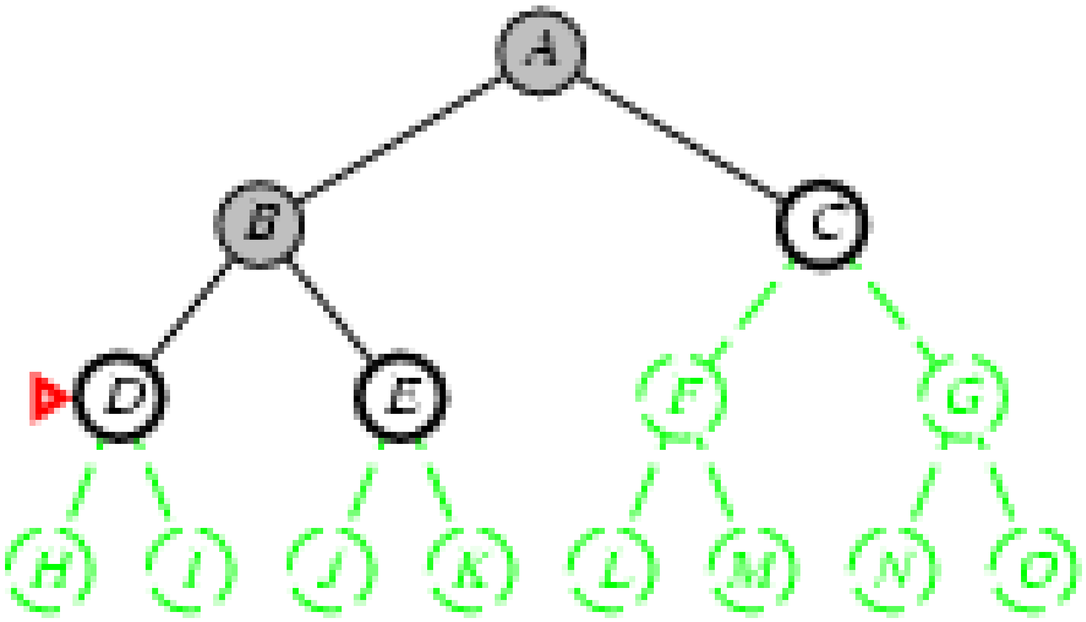
- Implementation:



Depth-first search

- Expand deepest unexpanded node

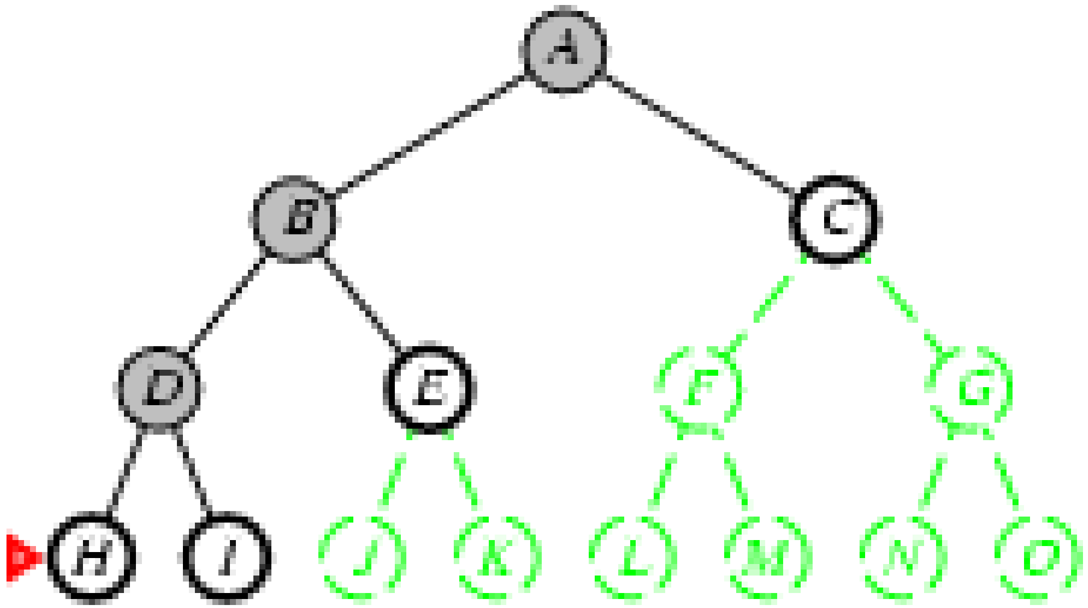
- Implementation:



Depth-first search

- Expand deepest unexpanded node

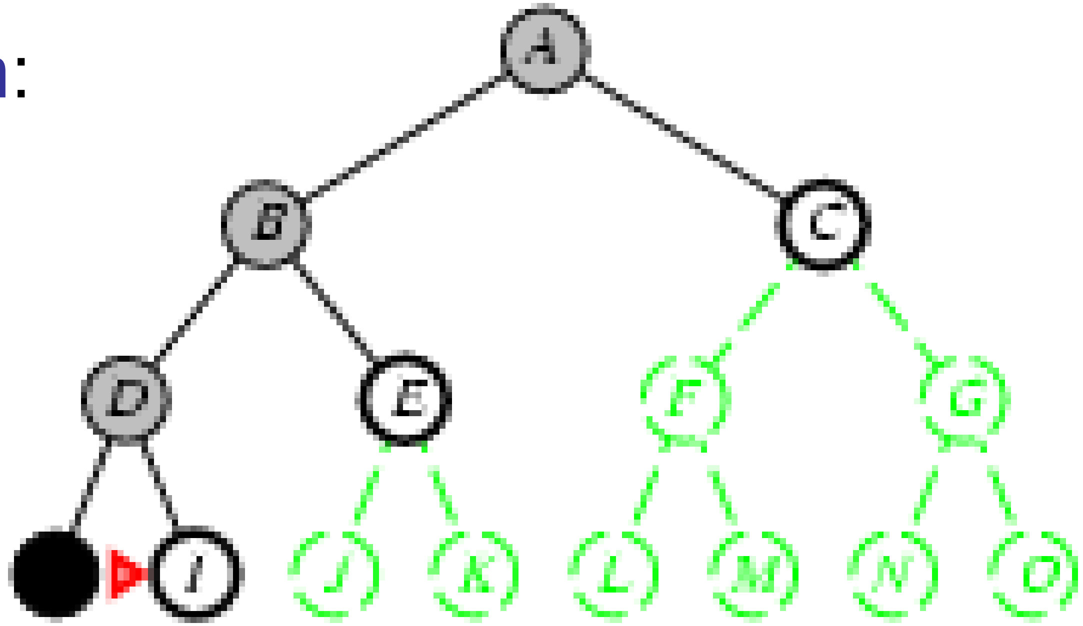
- Implementation:



Depth-first search

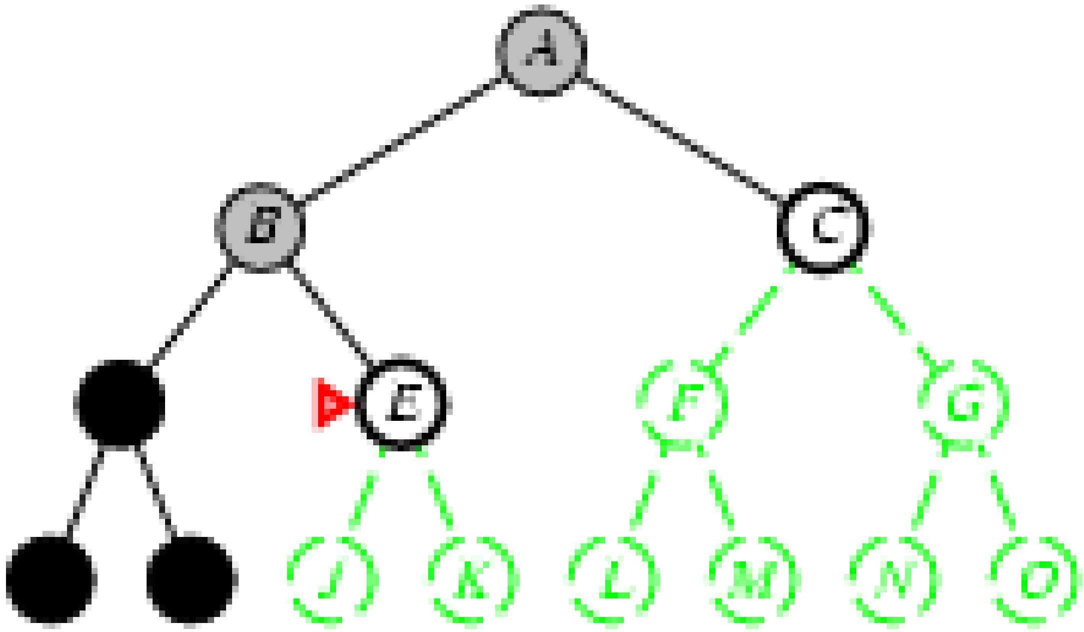
- Expand deepest unexpanded node

- Implementation:



Depth-first search

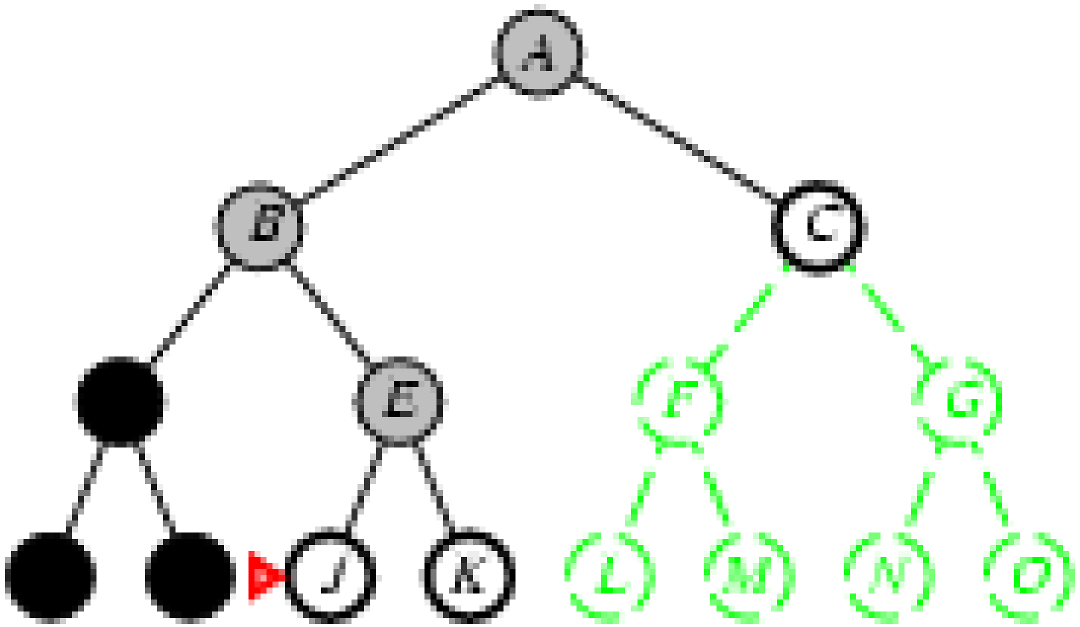
- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO queue, i.e., put successors at front



Depth-first search

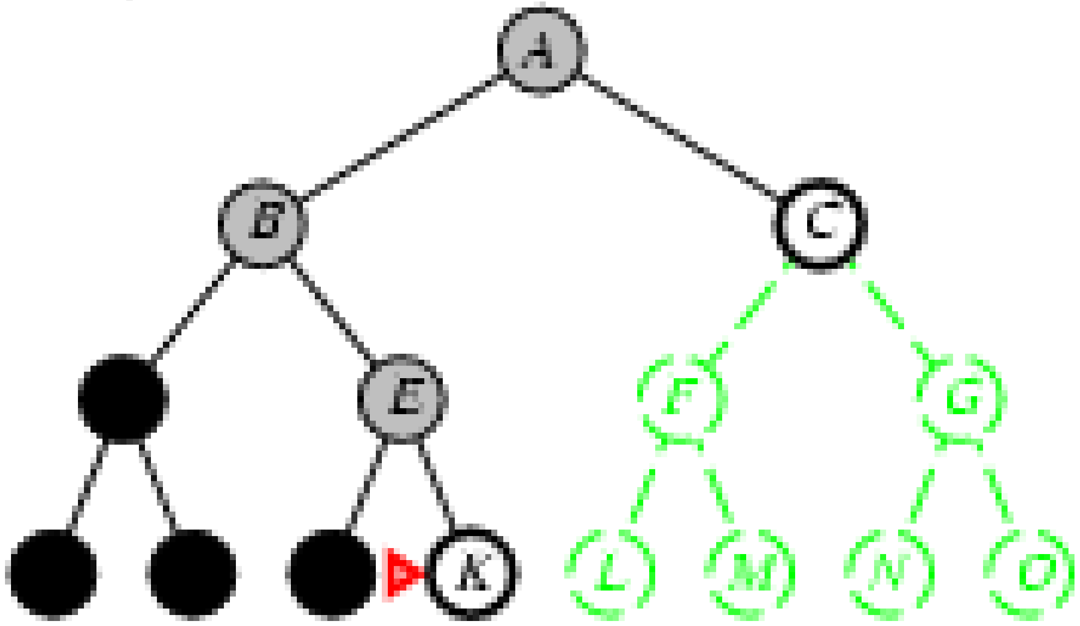
- Expand deepest unexpanded node

- Implementation:



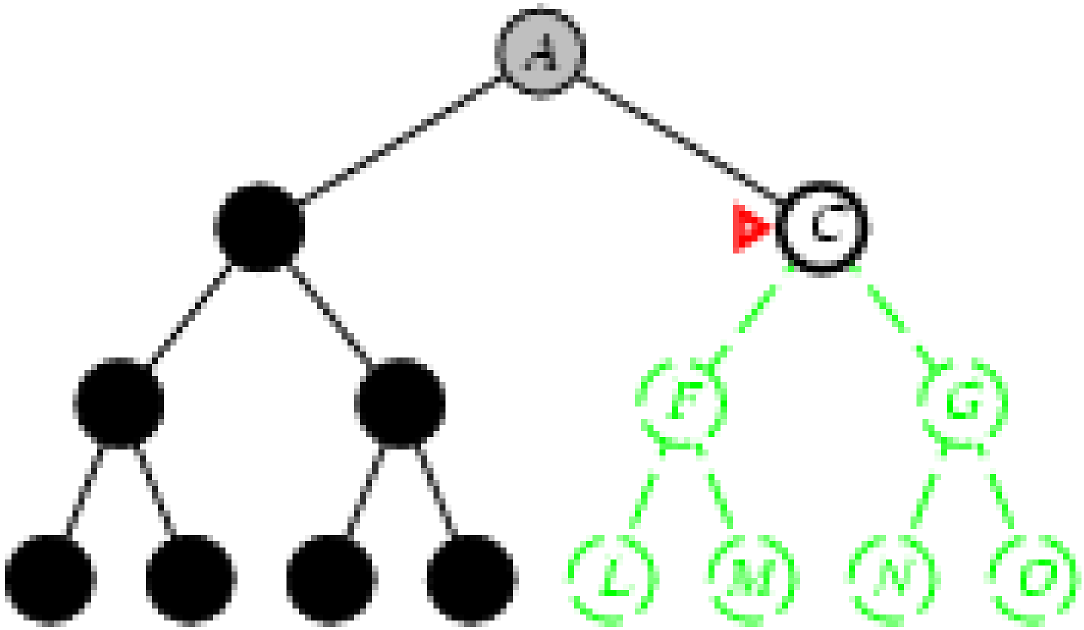
Depth-first search

- Expand deepest unexpanded node
- Implementation:



Depth-first search

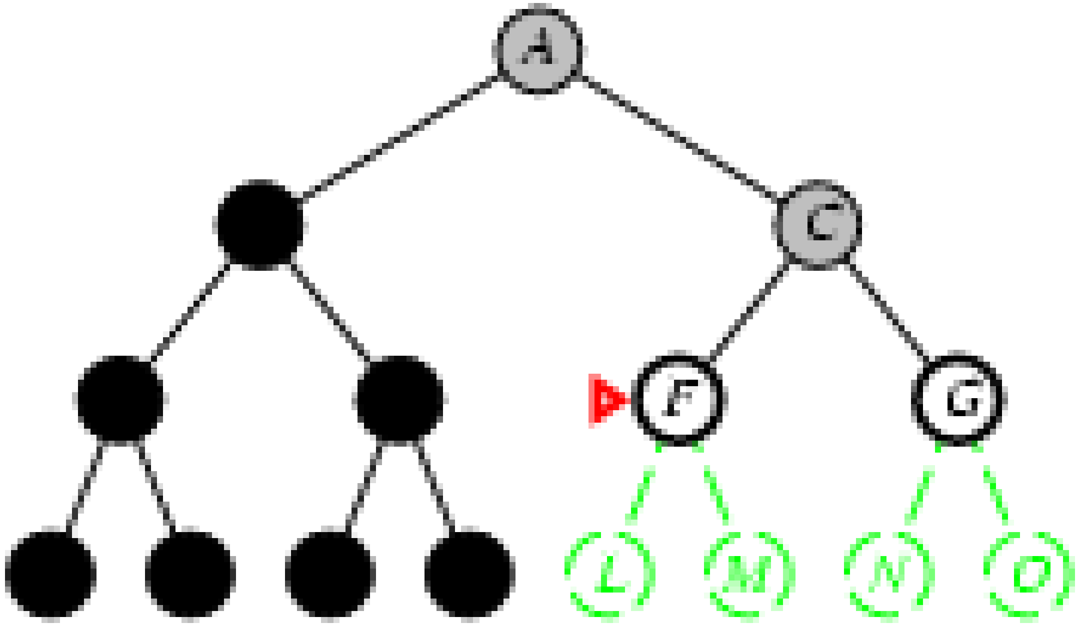
- Expand deepest unexpanded node
- Implementation:



Depth-first search

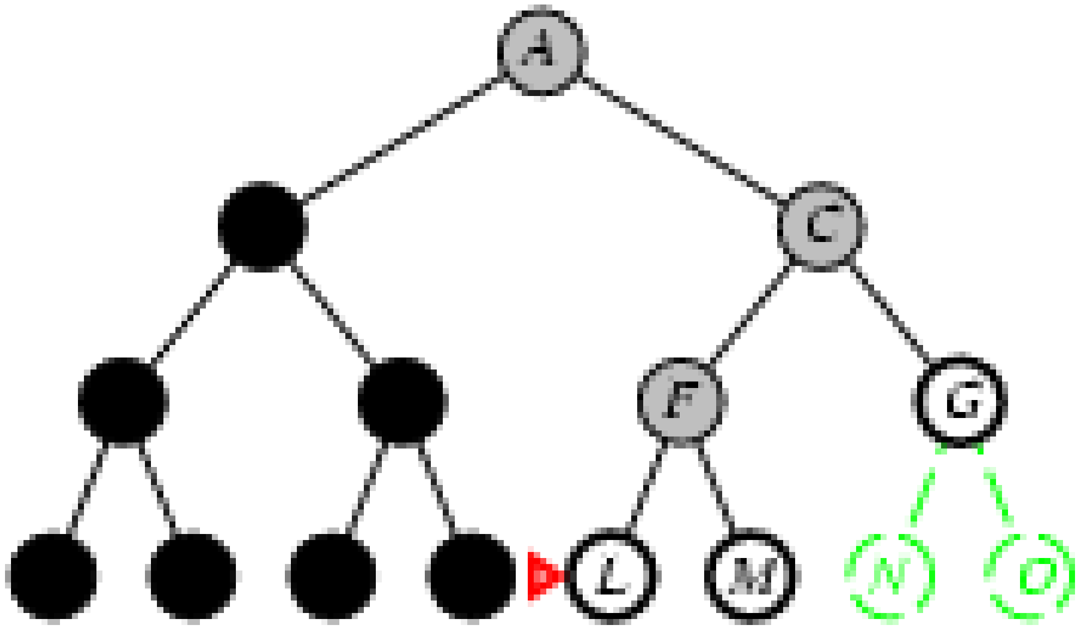
- Expand deepest unexpanded node

- Implementation



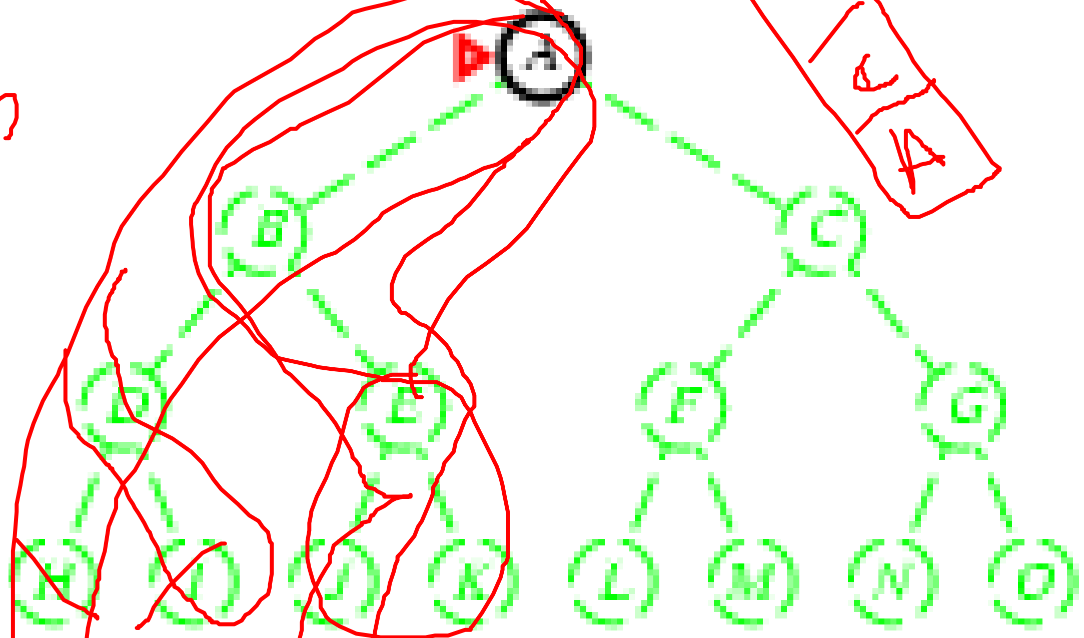
Depth-first search

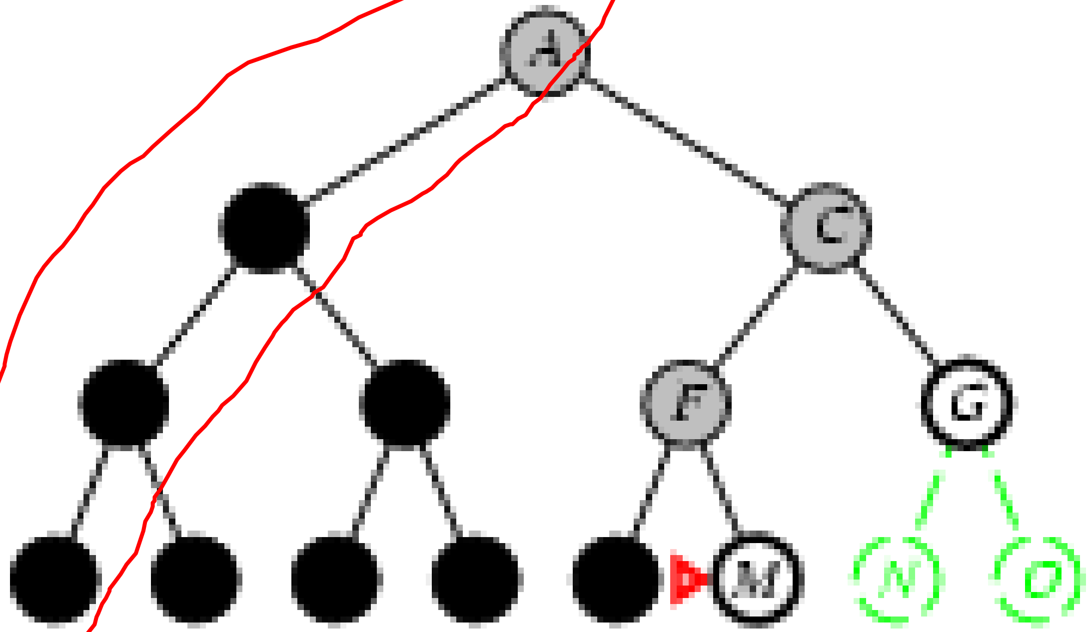
- Expand deepest unexpanded node



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *fringe* = LIFO Stack, i.e., put successors at front





DFS: Evaluation

- DFS Graph version is complete, and **Tree Version is incomplete**
- Why?
- Time Complexity: $0(b^m)$
- If m (maximal depth) is much larger than d(depth of shallowest solution) – time is terrible
  - If there exist multiple solutions, DFS is faster than BFS

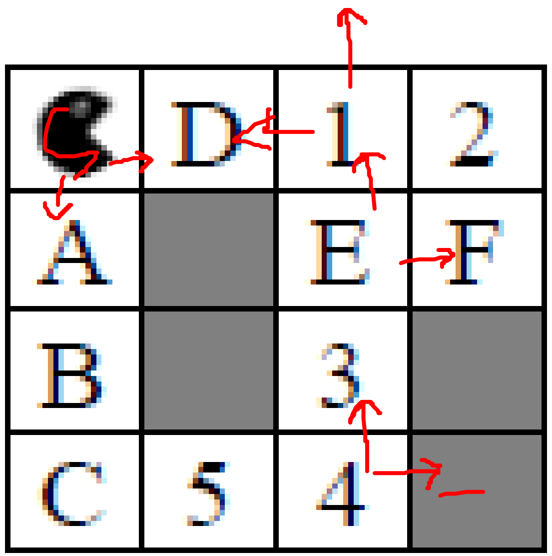
- Space Complexity: 0(bm) (in Tree Variant)
- **Non-Optimal:** it might give a solution with a higher cost

Properties of depth-first search

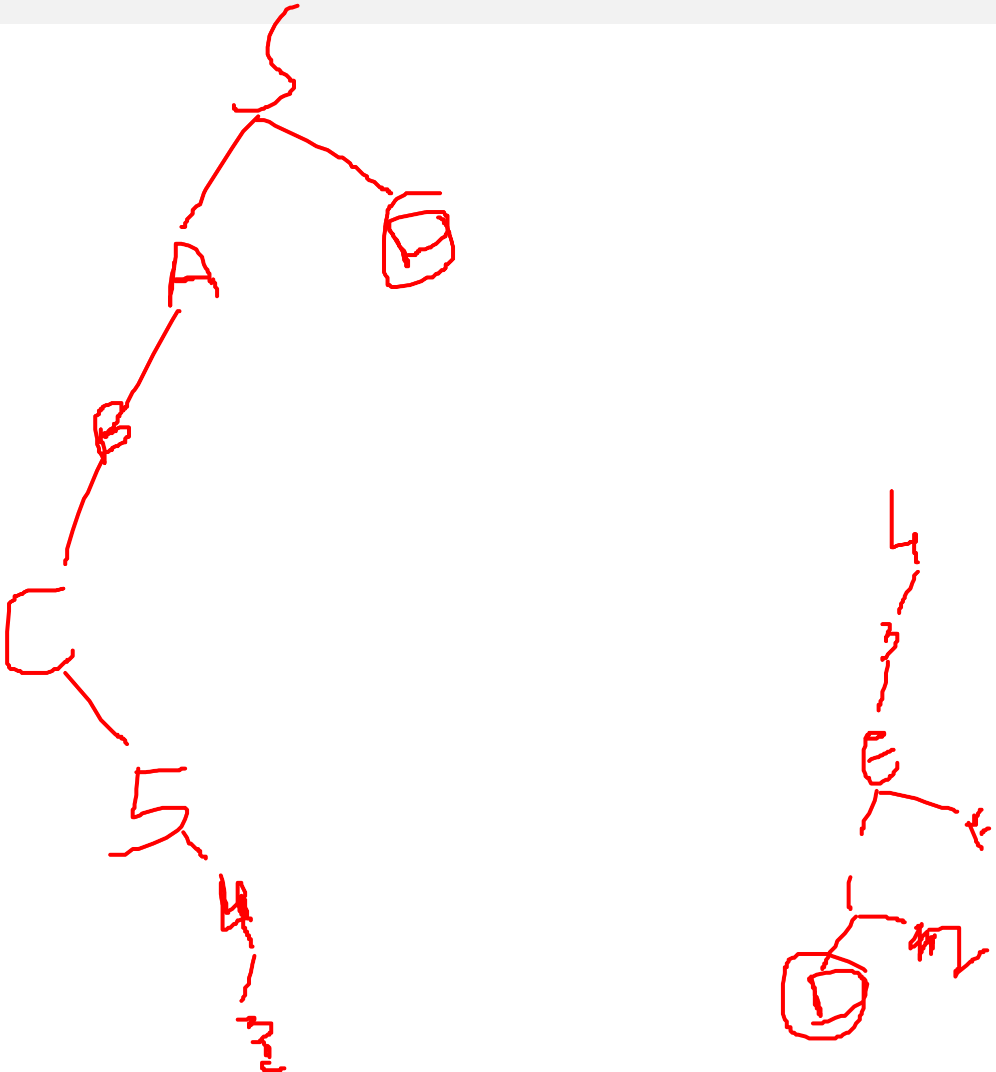
- Complete? No: fails in infinite-depth spaces,
- Time? $O(b^m)$: terrible if m is much larger than d

m: maximum depth of any node, d=depth of the shallowest node

- Space? $O(bm)$, i.e., linear space! (bm+1)
- Optimal? No



Initial State



Depth-first Search: When it is appropriate?

Appropriate

- Space is restricted (complex state representation e.g., robotics)
- There are many solutions, perhaps with long path lengths, particularly for the case in which all paths lead to a solution.

Inappropriate

- Cycles
- There are shallow solutions

Why DFS need to be studied and understood?

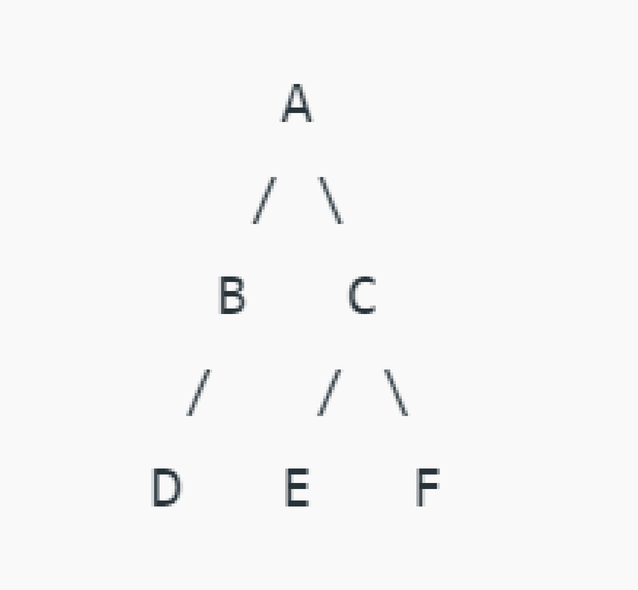
- It is simple enough to allow you to learn the basic aspects of searching (When compared with breadth first)

- It is the basis for a number of more sophisticated / useful search algorithms

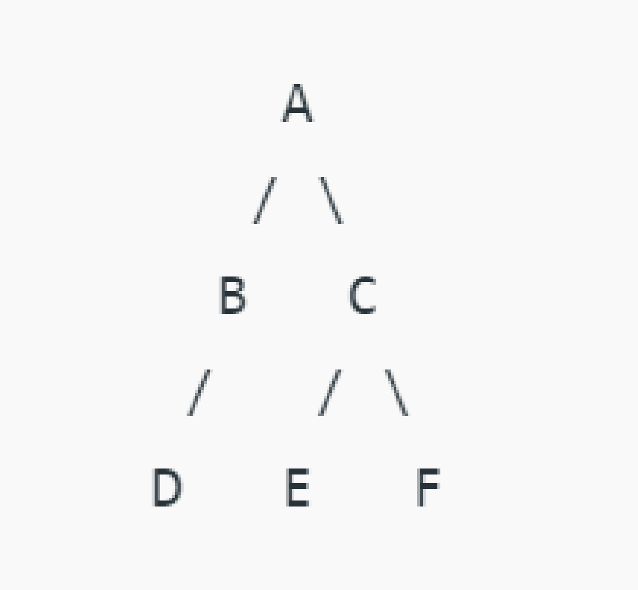
Variant of DFS: Backtracking

- Less Memory usage
- Only one successor generated at a time
- Each partially expanded node remembers which successor to generate next
- Memory : O(m) instead of O(bm)
- Example (N-queen problem, incremental approach)

BFS vs DFS

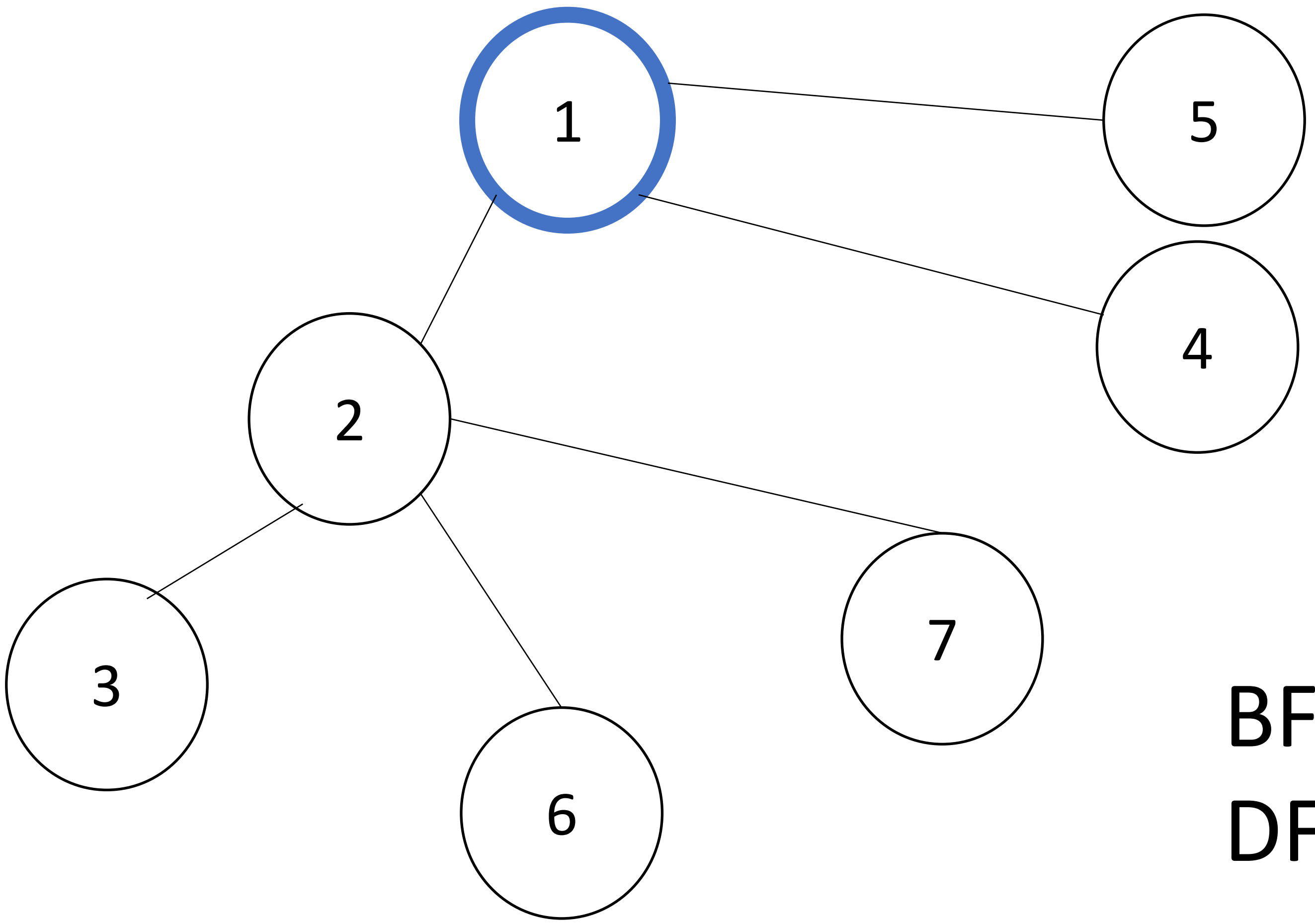


A, B, C, D, E, F



A, B, D, C, E, F

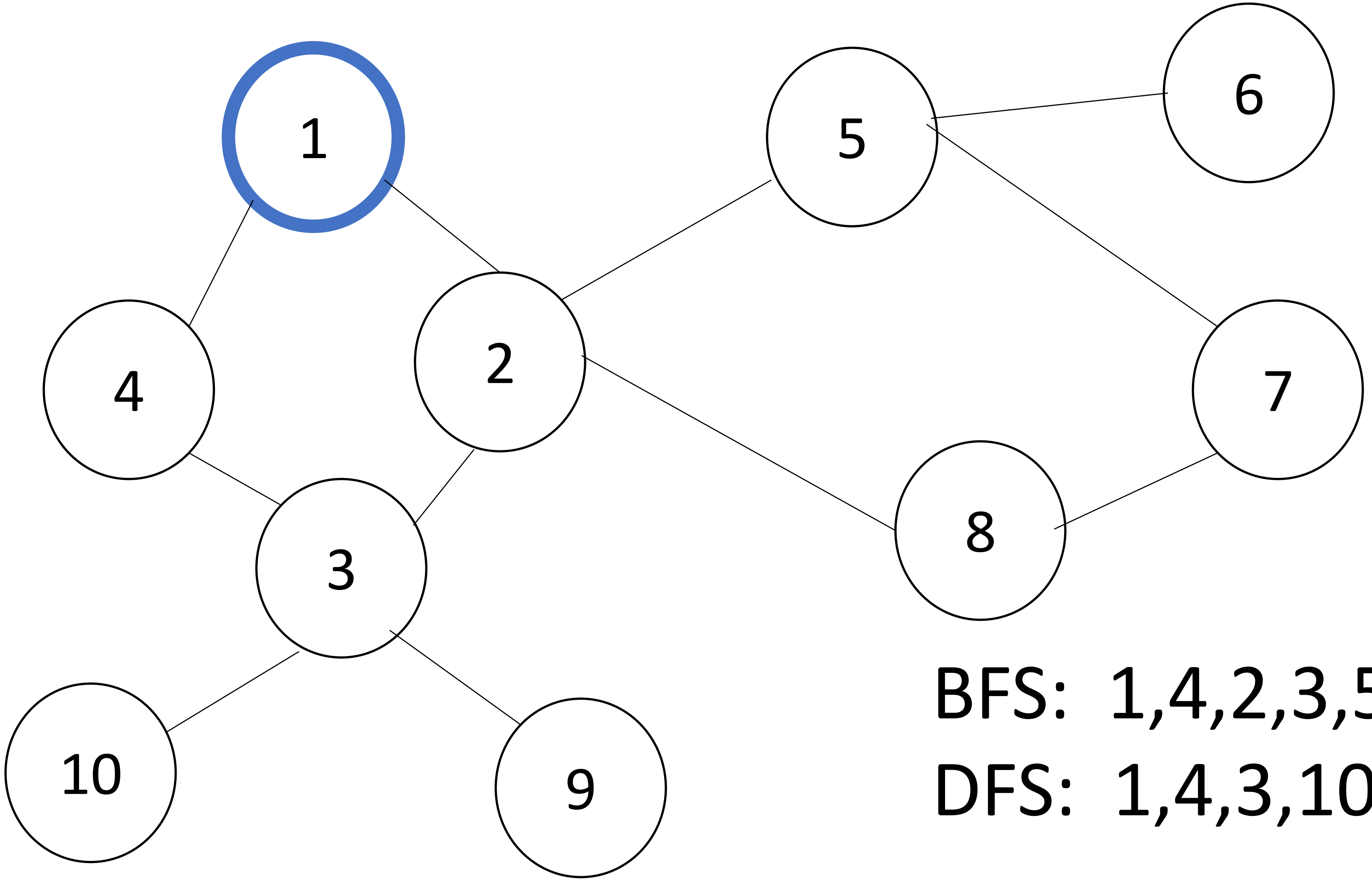
Apply BFS and DFS on following Graph



BFS:

DFS:

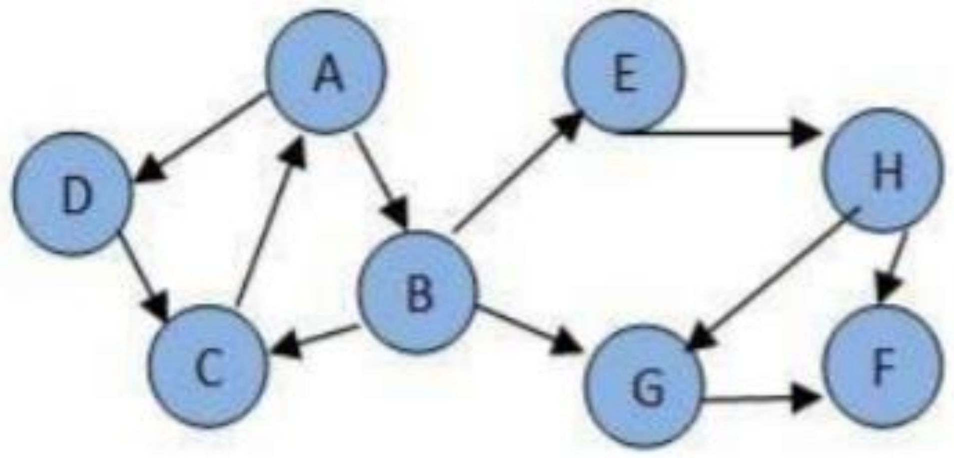
Apply BFS and DFS on following Graph



BFS: 1,4,2,3,5,8,10,9,7,6

DFS: 1,4,3,10,9,2,8,7,5,6

BFS vs DFS



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

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

Activity

- Web crawling
- Social network analysis
    - Finding similarity of two people on the social network

Activity

- Web crawling
  - BFS is more likely to find the required topic ur searching
- Social network analysis
  - Finding similarity of two people on the social network
  - DFS is more likley to do better and find similarity between two nodes, where they connect etc.
  - BFS will search all immediate child nodes and then the child of further nodes, so it might get scattered and not too detailed

Uniform-cost search

- Expand least-cost unexpanded node-Cheapest-First Search
- Implementation:
  – *fringe* = queue ordered by path cost
- Equivalent to breadth-first if path cost of all edges is same

Insert the root into the queue

While the queue is not empty

Dequeue the maximum priority element from the queue

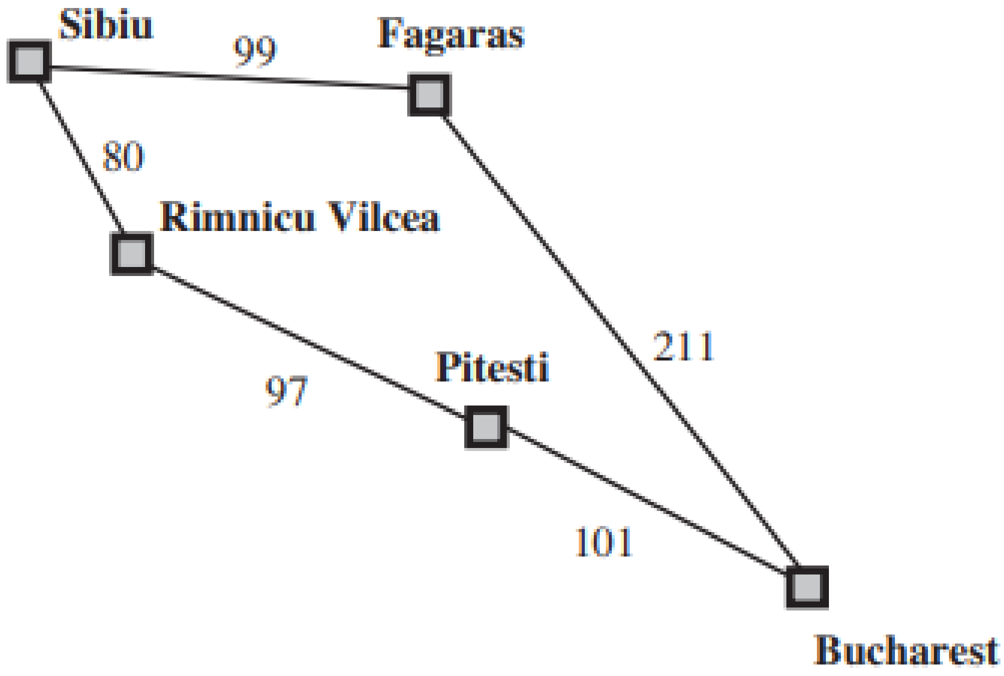
(If priorities are same, alphabetically smaller path is chosen)

If the path is ending in the goal state, print the path and exit

Else

Insert all the children of the dequeued element, with the cumulative costs as priority

Uniform Cost Search



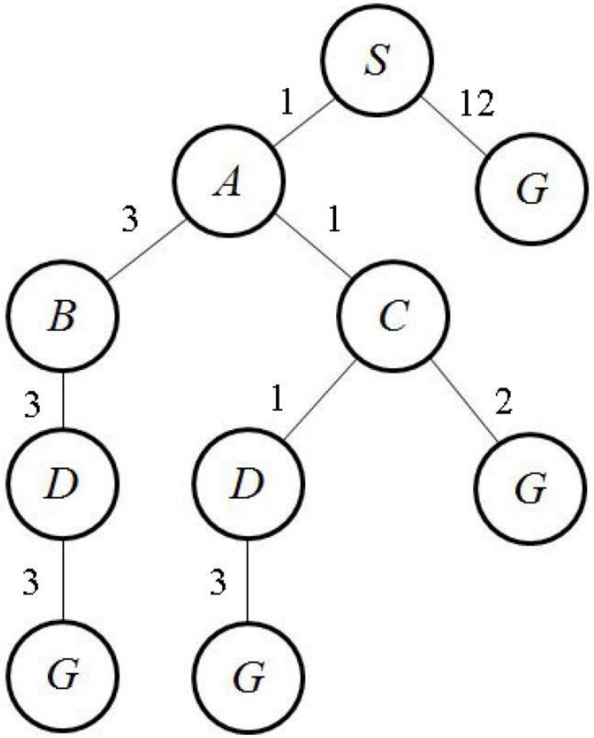
Part of the Romania state space, selected to illustrate uniform-cost search.

The successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively.

The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost 80 + 97=177.

The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost 99+211=310.

Now a goal node has been generated, but uniform-cost search keeps going, choosing Pitesti for expansion and adding a second path to Bucharest with cost 80+97+101= 278.



Initialization: { [S , 0] }

Iteration1: { [S->A , 1] , [S->G , 12] }

Iteration2: { [S->A->C , 2] , [S->A->B , 4] , [S->G , 12] }

Iteration3: { [S->A->C->D , 3] , [S->A->B , 4] , [S->A->C->G , 4] , [S->G , 12] }

Iteration4: { [S->A->B , 4] , [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->G , 12] }

Iteration5: { [S->A->C->G , 4] , [S->A->C->D->G , 6] , [S->A->B->D , 7] , [S->G , 12] }

Iteration6 gives the final output as S->A->C->G.

Uniform-cost Search

Uniform-cost search is guided by path costs rather than depths

- Complete? Yes, if step cost ≥ ε (0 not allowed due to recursion/looping)
- Time? # of nodes with *g(path cost)* ≤ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$ where *C^* is the cost of the optimal solution, ε some small positive constant*
- Space? # of nodes with *g* ≤ cost of optimal solution $O(b^{ceiling(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of *g(n)*

When all step costs are the same, uniform-cost search is similar to breadth-first search, except that the latter stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost