Uninformed Search (Ch. 3-3.4)

Search algorithm

For the next few searches we use:
(without the red stuff for trees)

```
function tree-search(root-node)
  fringe ← successors(root-node)
  explored ← empty
  while ( notempty(fringe) )
      {node ← remove-first(fringe)
        state ← state(node)
        if goal-test(state) return solution(node)
        explored ← insert(node,explored)
        fringe ← insert-all(successors(node),fringe,  if node not in explored)
      }
  return failure
end tree-search
```

Search algorithm

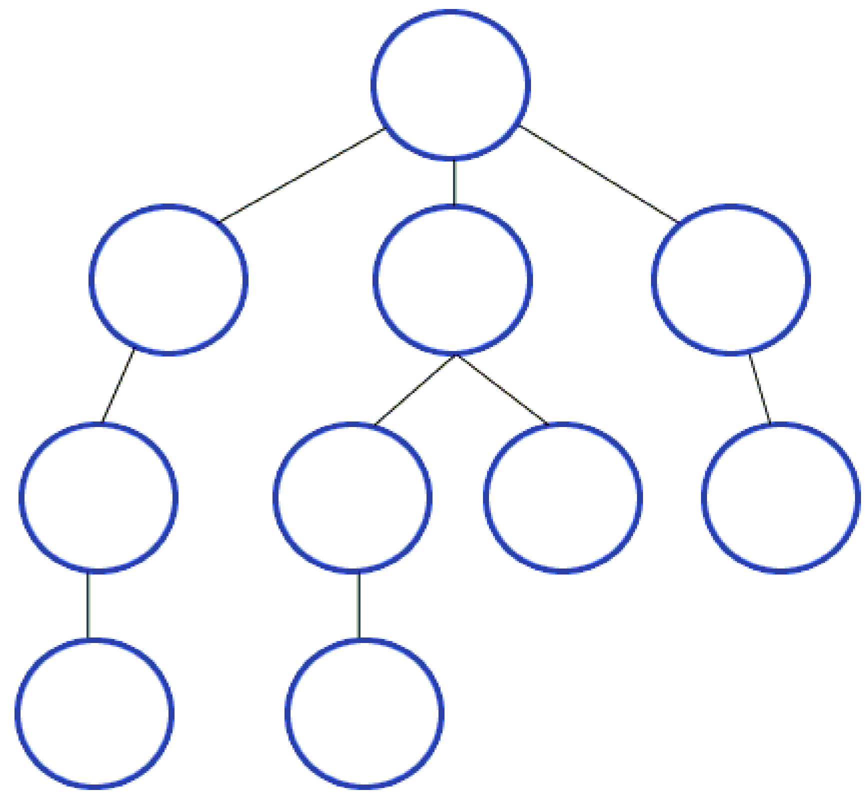
The search algorithms metrics/criteria:
1. Completeness (does it terminate with a valid solution)
2. Optimality (is the answer the best solution)
3. Time (in big-O notation)
4. Space (big-O)

b = maximum branching factor
d = minimum depth of a goal
m = maximum depth of tree (lowest leaf)

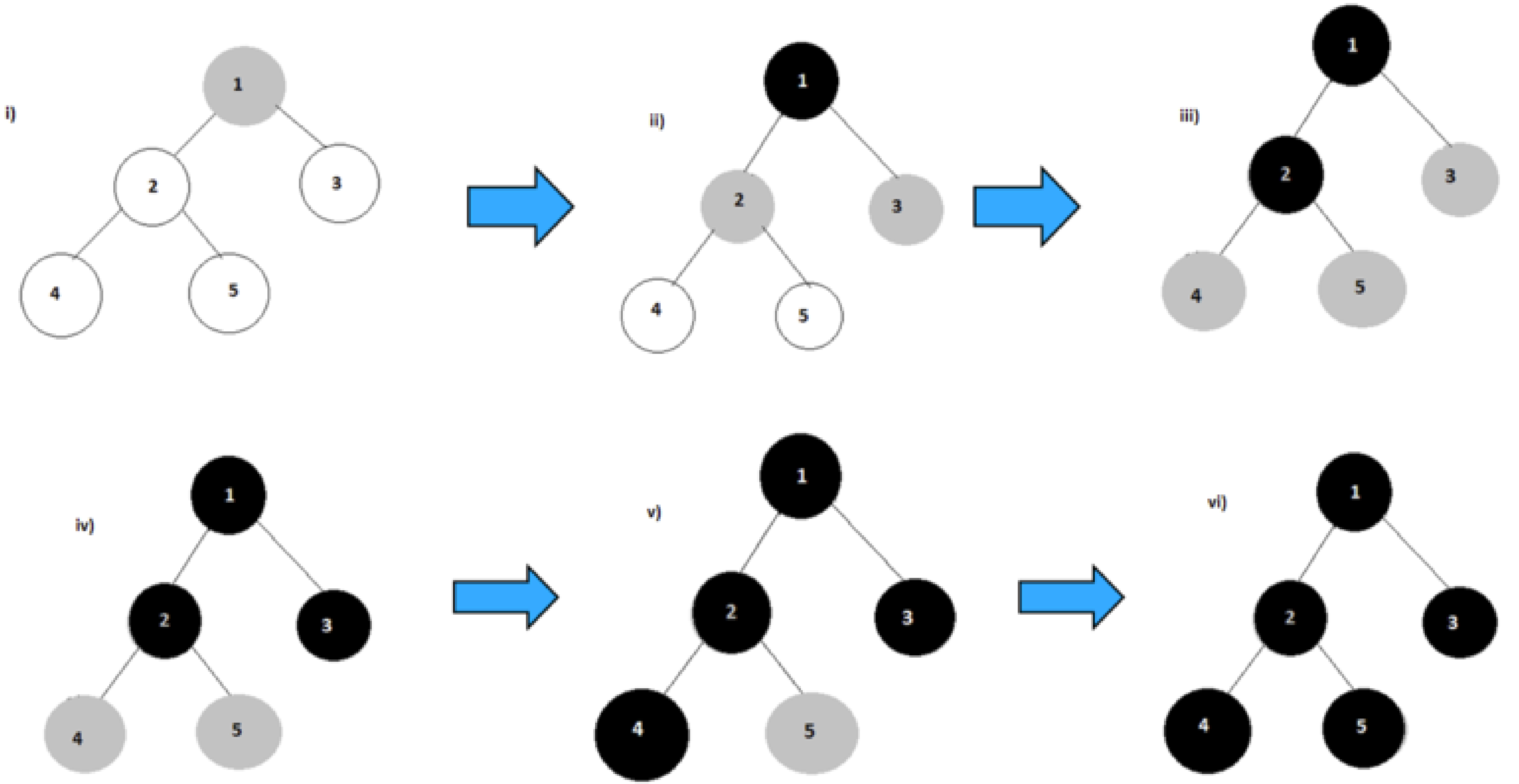
Breadth first search

<u>Breadth first search</u> checks all states which are reached with the fewest actions first

(i.e. will check all states that can be reached by a single action from the start, next all states that can be reached by two actions, then three...)



Breadth first search



(see: https://www.youtube.com/watch?v=5UfMU9TsoEM)
(see: https://www.youtube.com/watch?v=nI0dT288VLs)

Breadth first search

BFS can be implemented by using a simple FIFO (first in, first out) queue to track the fringe/frontier/unexplored nodes

Metrics for BFS:

Complete (i.e. guaranteed to find solution if exists)

Non-optimal (unless uniform path cost)

Time complexity = $O(b^d)$

Space complexity = $O(b^d)$

Breadth first search

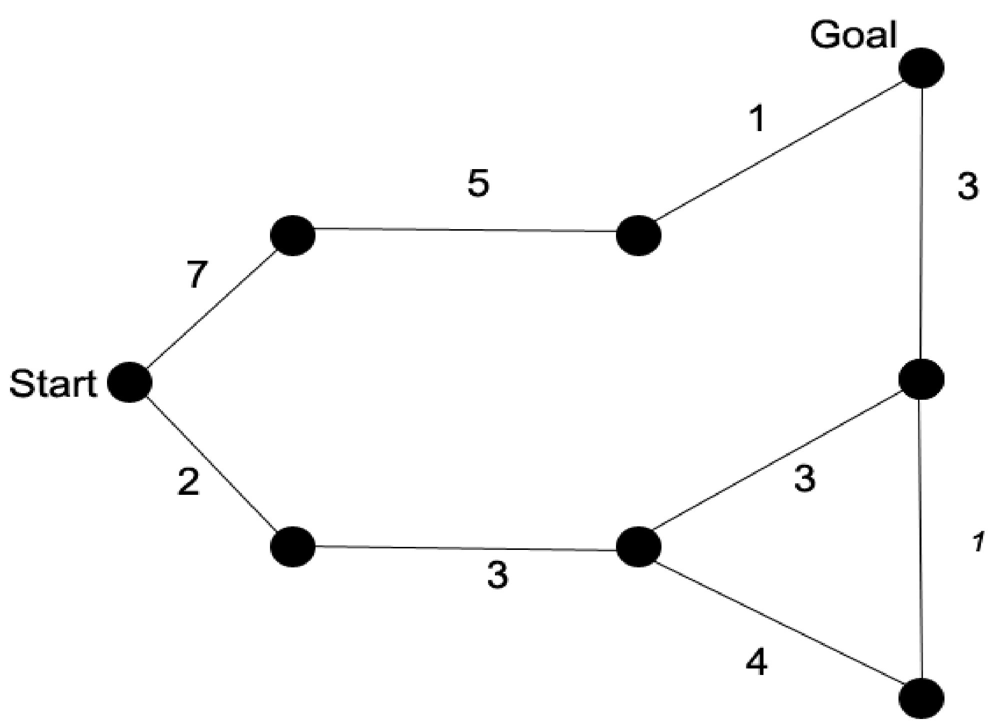
Exponential problems are not very fun:

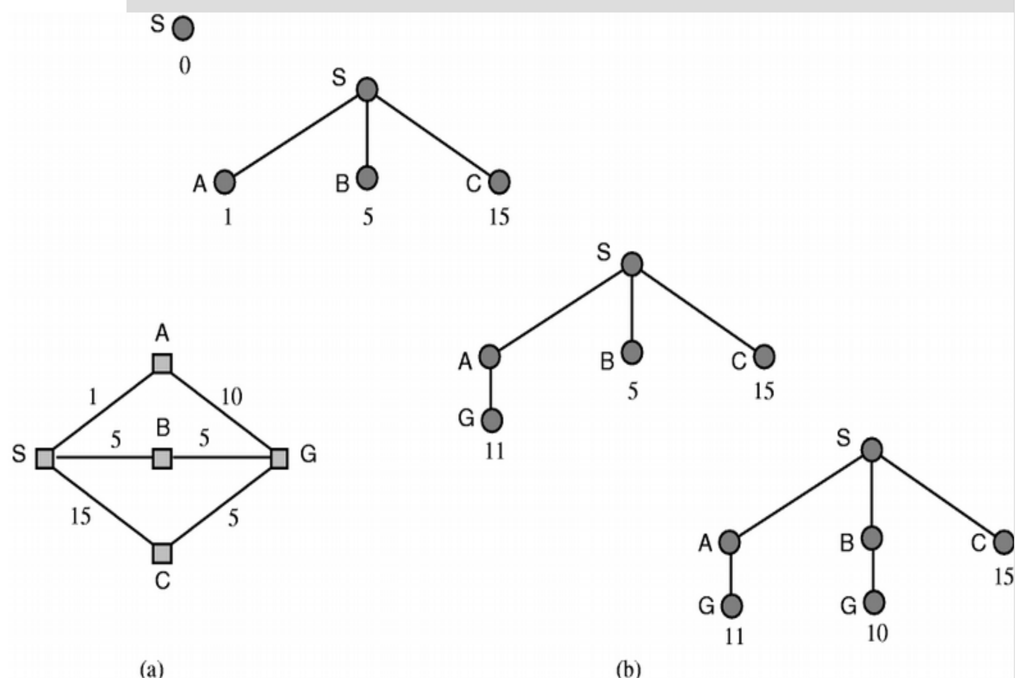
Depth	Nodes	Time	Memory
2	110	.11 milliseconds	107 kilobytes
4	11,110	11 milliseconds	10.6 megabytes
6	10^6	1.1 seconds	1 gigabyte
8	10^8	2 minutes	103 gigabytes
10	10^{10}	3 hours	10 terabytes
12	10^{12}	13 days	1 petabyte
14	10^{14}	3.5 years	99 petabytes
16	10^{16}	350 years	10 exabytes

This is BFS with b=10 (branching factor),
can compute 1 million nodes/sec,
nodes take up 1 KB each

Uniform-cost search

Uniform-cost search also does a queue, but uses a priority queue based on the cost (the lowest cost node is chosen to be explored)





Uniform-cost search

The only modification is when exploring a node we cannot disregard it if it has already been explored by another node

We might have found a shorter path and thus need to update the cost on that node

We also do not terminate when we find a goal, but instead when the goal has the lowest cost in the queue.

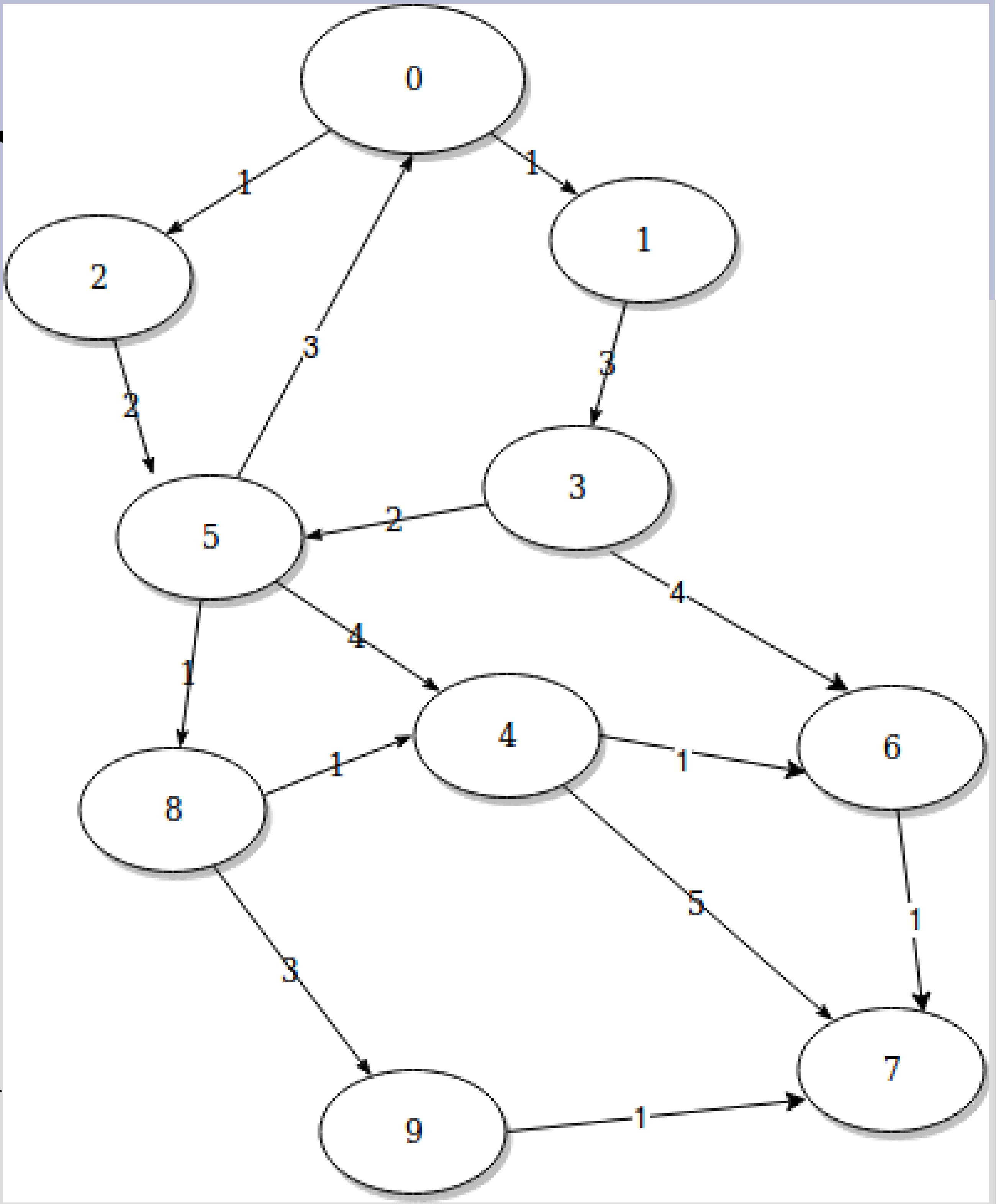
Unifor

Try it yourself!

Run uniform-cost search with:

Initial = Node 0
Goal = Node 7

(Note: this graph is directed)



Uniform-cost search

UCS is..

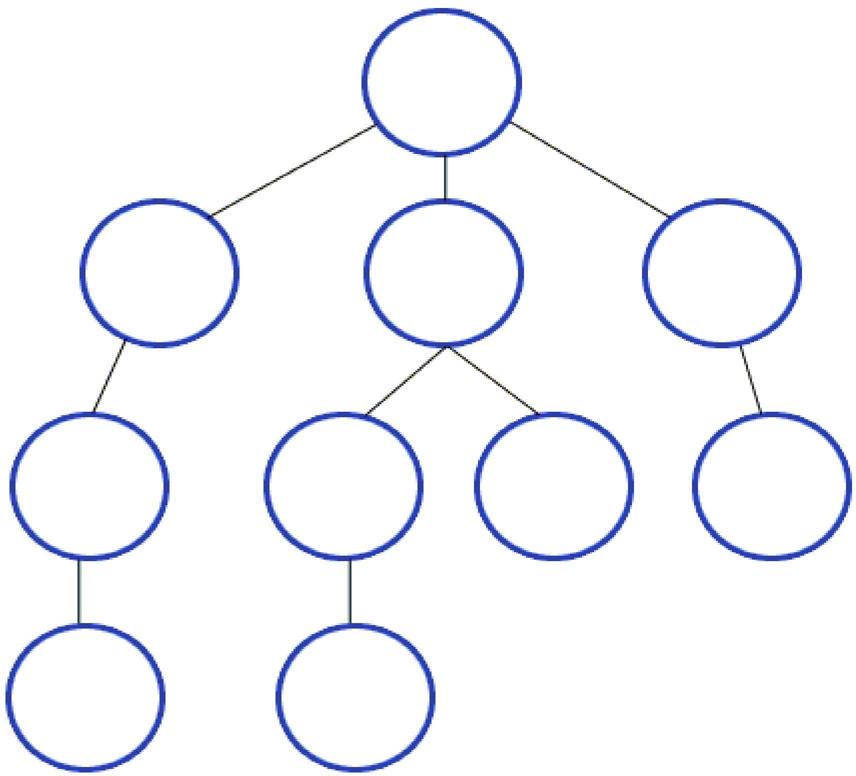
1. Complete (if costs strictly greater than 0)
2. Optimal

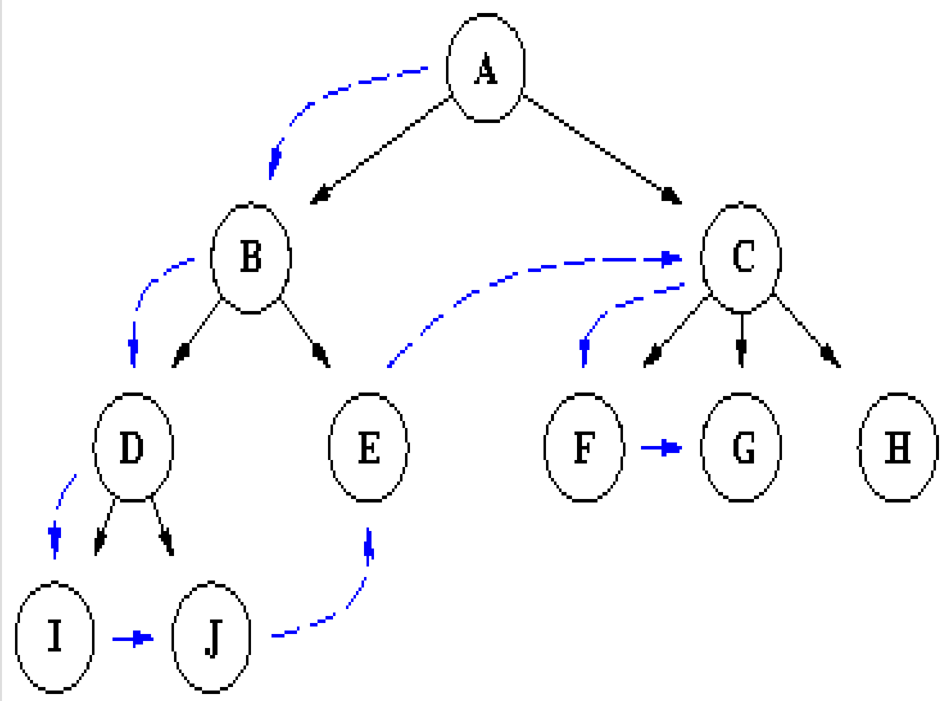
However....

3&4. Time complexity = space complexity = $O(b^{1+C^*/\min(\text{edge cost})})$, where C* cost of optimal solution (much worse than BFS)

Depth first search

DFS is same as BFS except with a FILO (or LIFO) instead of a FIFO queue





Depth first search

Metrics:

1. Might not terminate (not complete) (e.g. in vacuum world, if first expand is action L)
2. Non-optimal (just... no)
3. Time complexity = $O(b^m)$
4. Space complexity = O(b*m)

Only way this is better than BFS is the space complexity...