CS 348
Intro to Artificial Intelligence

Day 2
Uninformed Search

(slides shamelessly based on Doug Downey, Sara Sood, and others)

Mike Rubenstein

Today:

- Class business
  - Homework 0 Due April 5 7:00pm
  - Uninformed search (chapter 3.1-3.4)
  - Slides available on canvas
- Homework 1 (uninformed search)
  - Due April 14 7pm

Updated OH

Time	Monday Mudd 3532	Tuesday Tech EG 20	Wednesday Tech EG 20	Thursday Tech EG 20	Friday Mudd 3532
9:30 - 10:00		1			
10:00 - 10:30		1		2	
10:30 - 11:00				3	
11:00 - 11:30	1	2	2	3	1
11:30 - 12:00	1	2	3	3	1
12:00 - 12:30		1	3	3	1
12:30 - 1:00		2	3	3	1
1:00 - 1:30	1	1	4	4	1
1:30 - 2:00	1	1	2	3	1
2:00 - 2:30			3		1
2:30 - 3:00			3		
3:00 - 3:30	1		2		
3:30 - 4:00	1		2	3	
4:00 - 4:30	1	1	4	4	
4:30 - 5:00	1	1	4	4	
5:00 - 5:30		1	4	4	
5:30 - 6:00		1	4	4	
6:00 - 6:30				4	

What is Search?

- A class of techniques for systematically finding or constructing solutions to problems.

- Example technique: generate-and-test
- Example problem: Combination lock
  1. Generate possible solution
     - Random
     - Enumerate all possibilities
  2. Test solution
  3. If solution found then done, else goto 1

Search through a problem space / state space

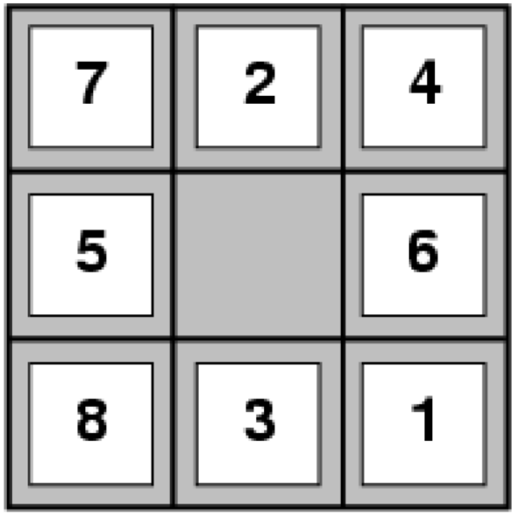
- Input:
  - Set of states
    - Each state describes the current environment
  - Operators
    - Actions the agent can take
  - Start state
    - At start of problem, what does the environment look like
  - Goal state [or a test]
    - Is more than one goal possible?
      - Test for goal
    - What do we want the environment to look like

- Output
  - Goal state
  - Path from start state to a goal state
  - Shortest path?

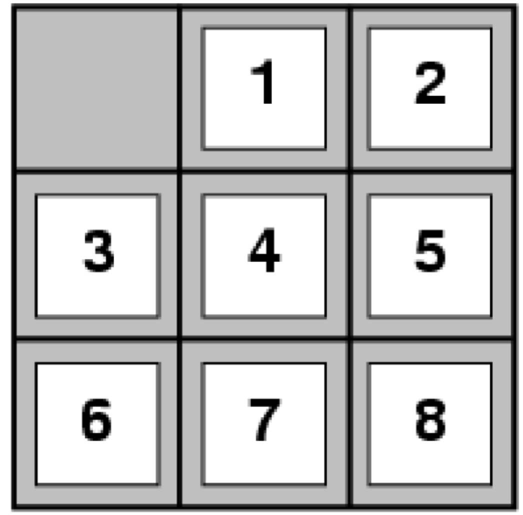
Why is search interesting?

- Many AI problems can be formulated as search problems.
- For example
  - Path planning
  - Games
  - Logic proofs
- Hint: understanding search will be useful for lab 1

Example: sliding puzzle



Start state

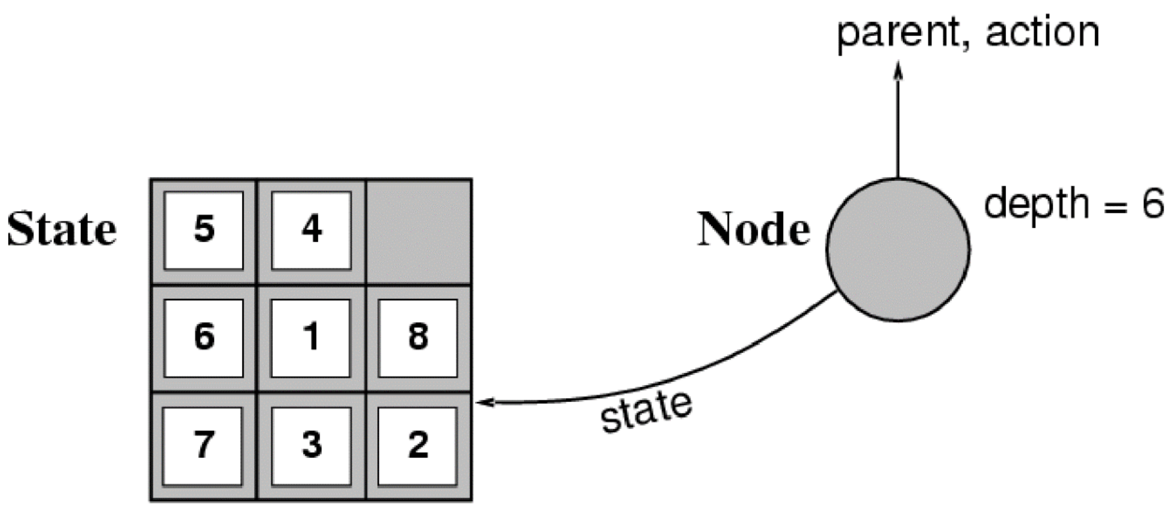


Goal state

- States: location of all tiles (362880 possible)
- Actions: move blank up, down, left, right
- Goal test?

Implementation

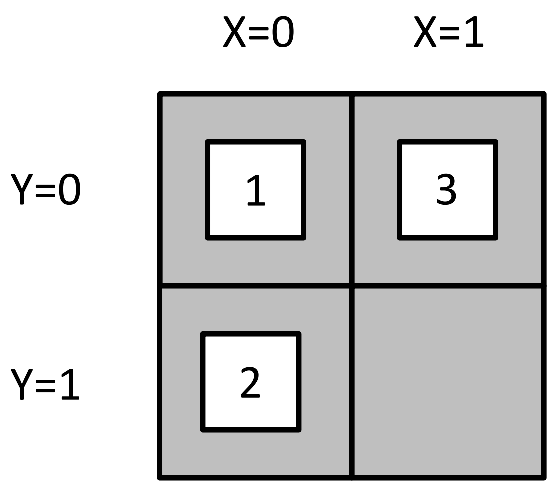
- A <u>state</u> is a representation of a physical configuration
- A <u>node</u> is a data structure which makes up a search tree. It includes State, parent, action, depth



- Expand function: creates new nodes from a parent, filling in the various fields and creating the corresponding state

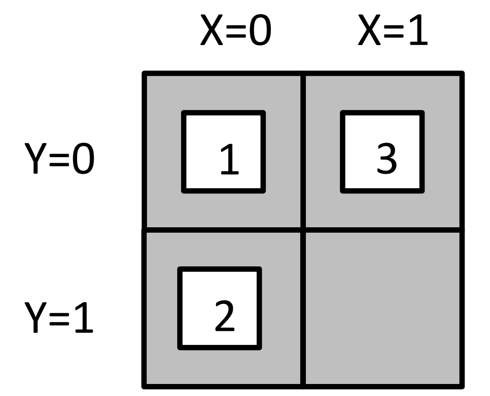
Exercise

- Generate pseudo code for new state from parent and action



- Represent state as 2D array state[x][y]
  - state[0][0]=1,state[0][1]=2,state[1][0]=3,state[1][1]=-1
  - If blank is at Xb,Yb (i.e state[Xb][Yb]=-1)
    - How to modify state array for 4 choices of motion
    - What do we need to be careful with?

Exercise



If move is up and ??

????

If move is down and ??

????

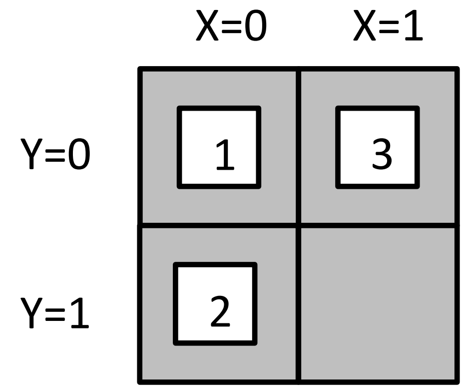
If move is left and ??

????

If move is right and ??

????

Exercise



If move is up and Yb==1

state[Xb][1]=?? state[Xb][0]=??

If move is down and ??

????

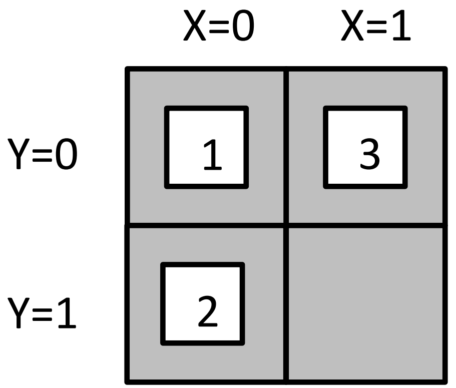
If move is left and ??

????

If move is right and ??

????

Exercise



If move is up and Yb==1

state[Xb][1]=state[Xb][0], state[Xb][0]=-1

If move is down and ??

????

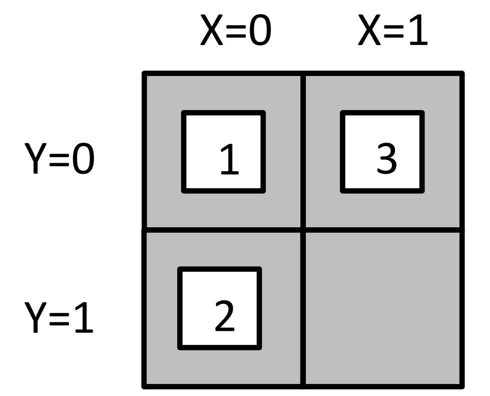
If move is left and ??

????

If move is right and ??

????

Exercise



If move is up and Yb==1

state[Xb][1]=state[Xb][0], state[Xb][0]=-1

If move is down and Yb==0

state[Xb][0]=state[Xb][1], state[Xb][1]=-1

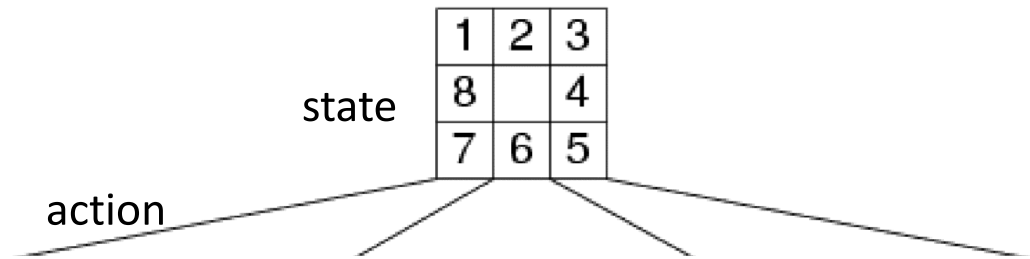
If move is left and Xb==1

state[1][Yb]=state[0][Yb], state[0][Yb]=-1

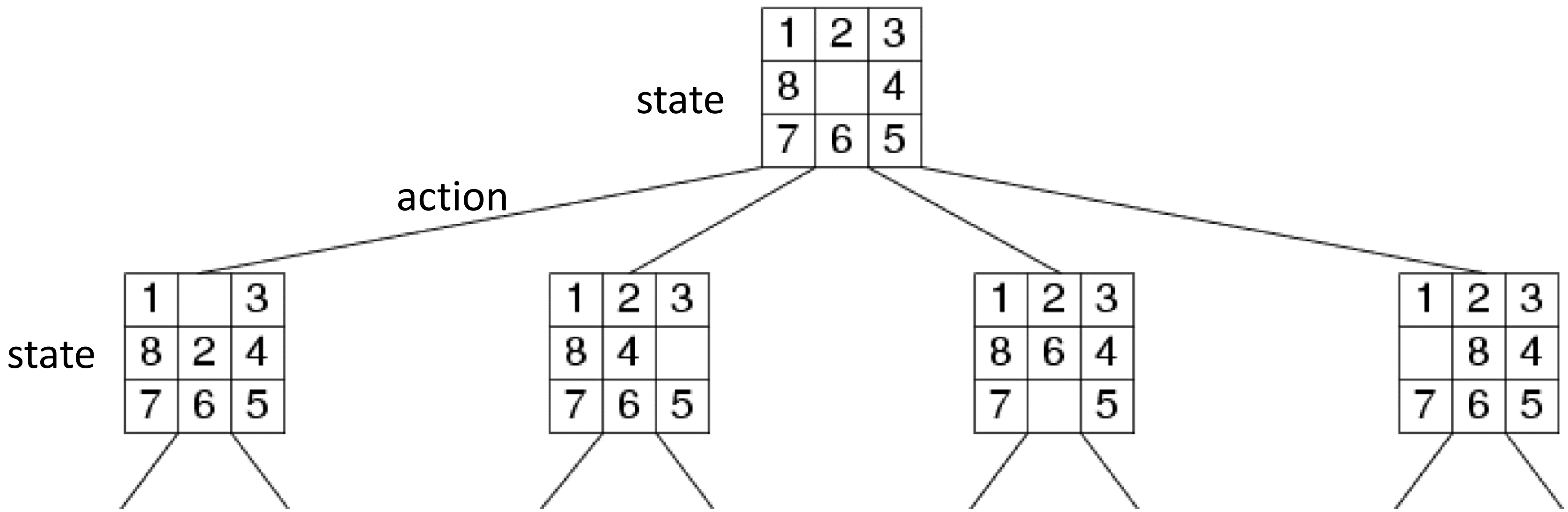
If move is right and Xb==0

state[0][Yb]=state[1][Yb], state[1][Yb]=-1

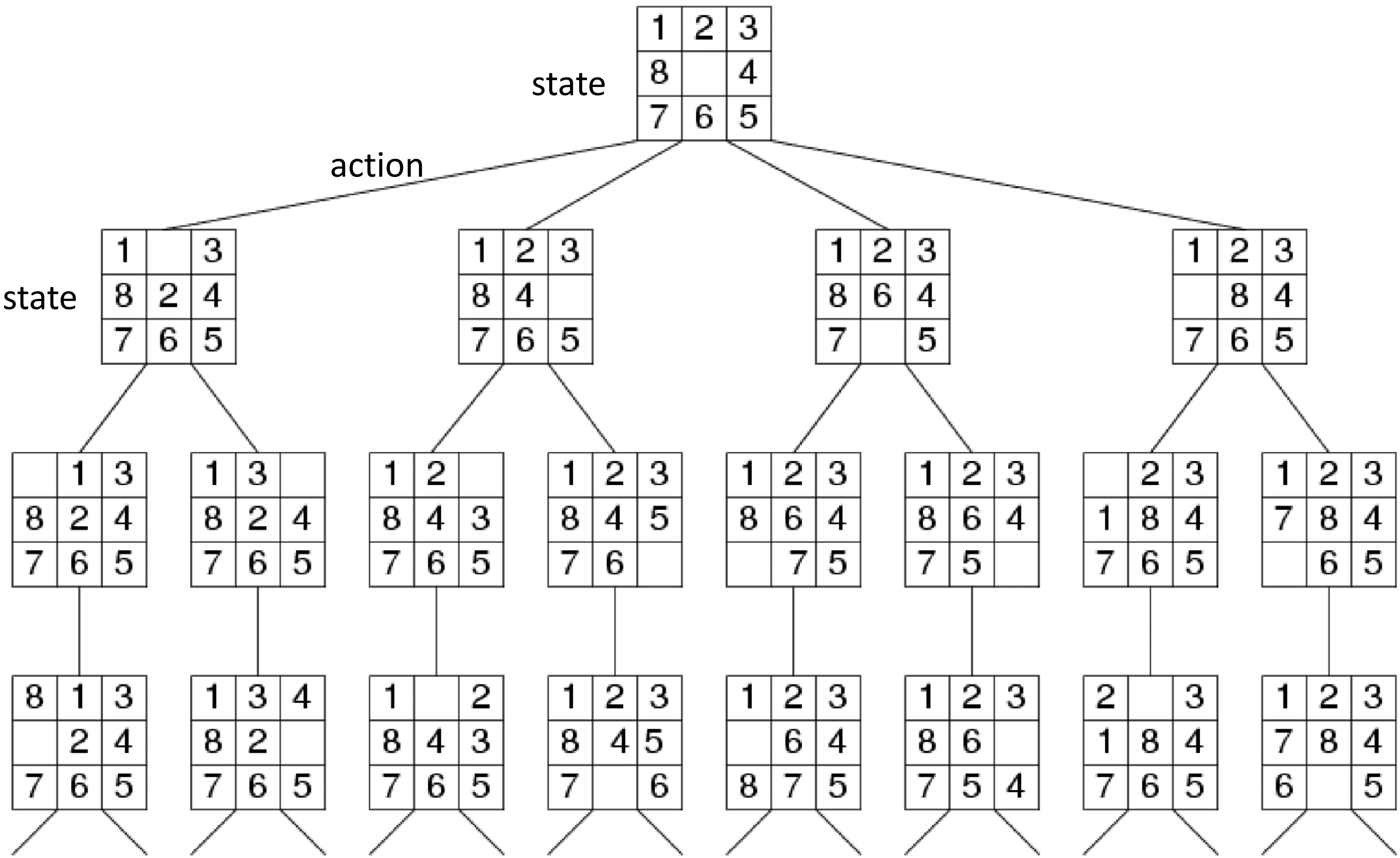
Representation as a tree



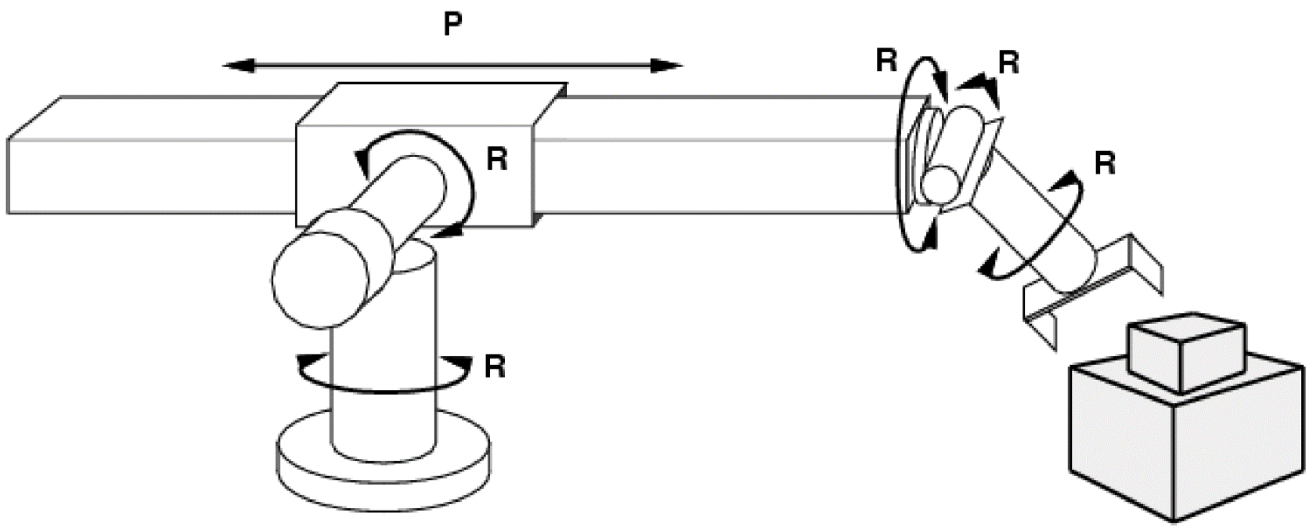
Representation as a tree



Representation as a tree



Example: robotic reaching



- States?:
  - robot joint angles
- Actions?
  - Motion of joints
- Goal test?
  - Distance from End effector to goal = 0

Finding a goal: Tree search

Basic idea: offline, simulate exploration of state space by generating successors of already-explored states (a.k.a. expanding states)

Frontier = root node
Repeat while Frontier not-empty
    Take s from Frontier
    For all s’ in Expand(s)
        if s’ is goal, return s’
        else add s’ to Frontier
Return failure

Finding a goal: Tree search

Basic idea: offline, simulate exploration of state space by generating successors of already-explored states (a.k.a. expanding states)

Frontier = root node
Repeat while Frontier not-empty
    Take s from Frontier
    For all s' in Expand(s)
        if s' is goal, return s'
        else add s' to Frontier
Return failure

How to avoid repeated states and loops?

Search strategies

- Search strategy is defined by the order of node expansion
- Strategies are evaluated along the following dimensions:
  - Completeness: Can it find a solution (if one exists)?
  - Time complexity: number of nodes generated
  - Space complexity: maximum number of nodes in memory
  - Optimality: does it always find the shortest solution?
- Space and Time complexity is measured in terms of
  - b: maximum branching factor of the tree
  - d: depth of shallowest solution
  - m: maximum depth of state space (may be infinity)

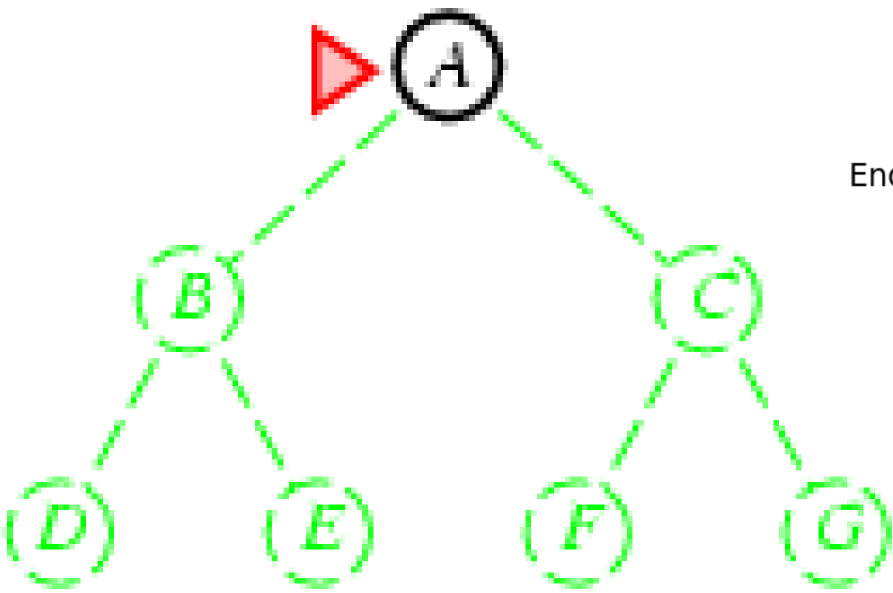
Uninformed search strategies

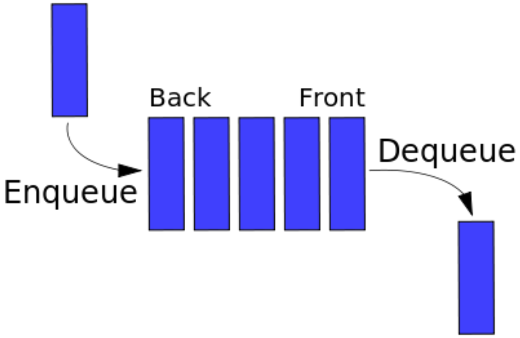
- Uninformed search strategies use only the information in the problem definition
  - Generate successor states
  - Detect goal state

1. Breadth-first
2. Depth-first
3. Depth-limited
4. Iterative deepening
5. Bi-directional

Breadth-first search

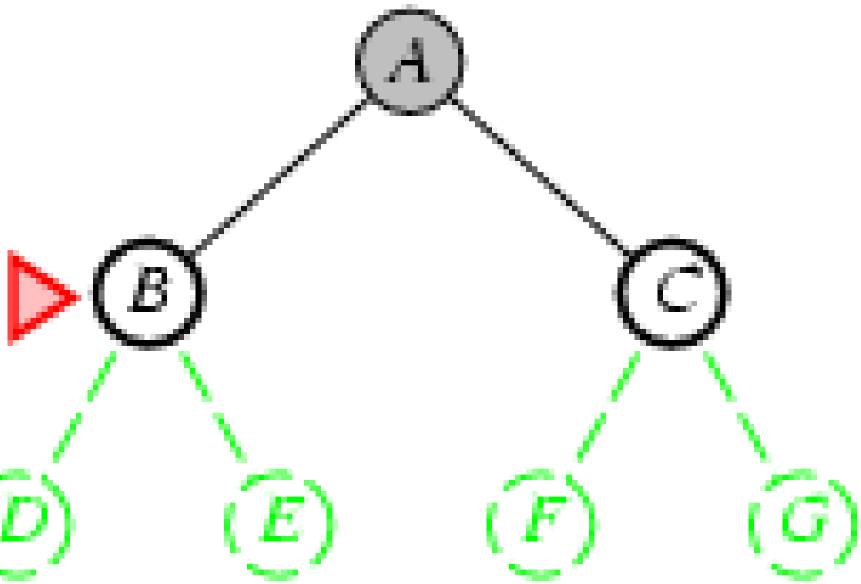
- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end





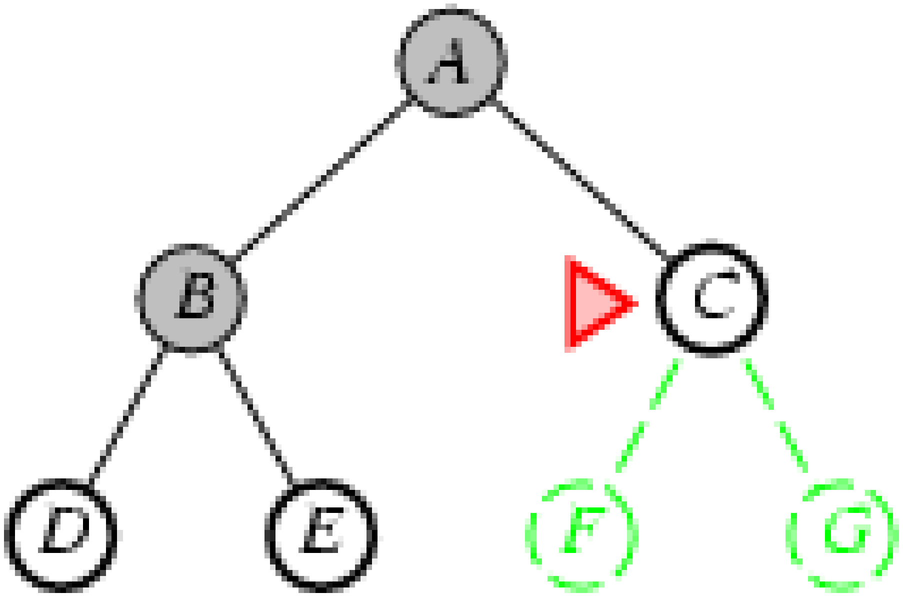
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



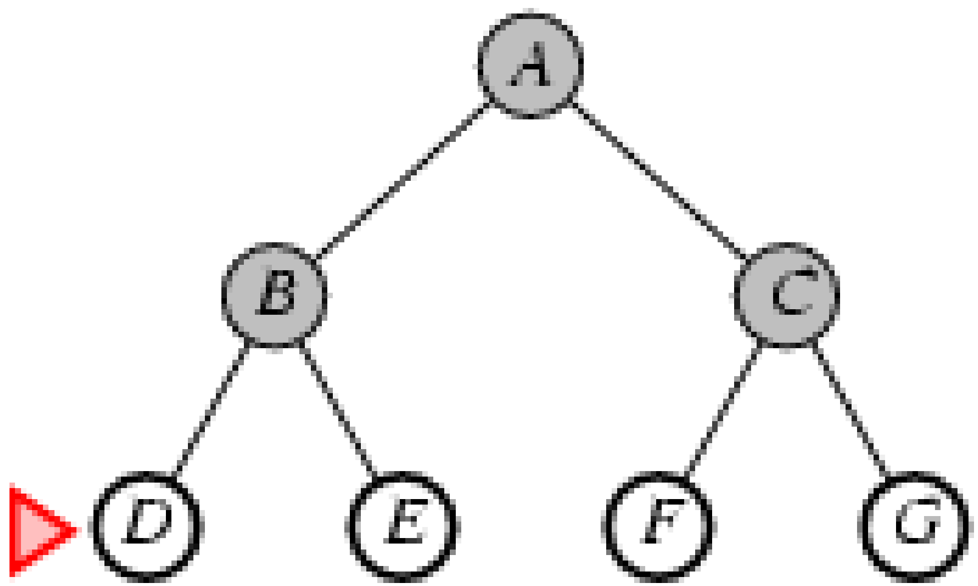
Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



Breadth-first search

- Expand shallowest unexpanded node
-
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end
- Evaluation
  - Completeness: yes if b is finite
  - Time complexity: O(b^d)
  - Space complexity: O(b^d)
  - Optimality: yes

Time and Memory requirements for BFS

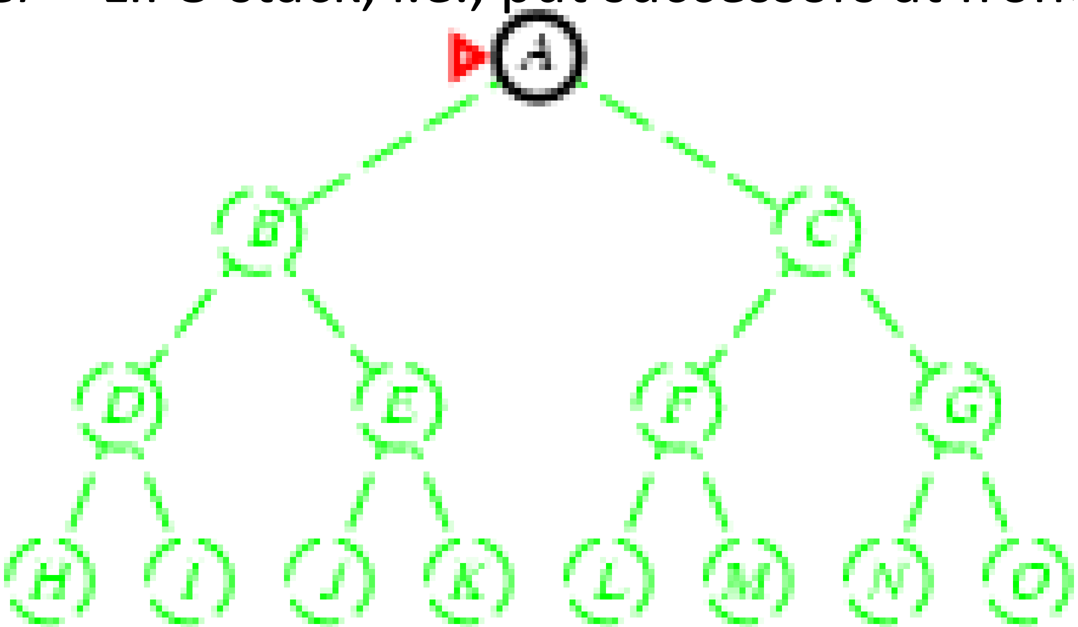
Depth	Nodes	Time	Memory
2	1100	.11 sec	1 MB
4	111,100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

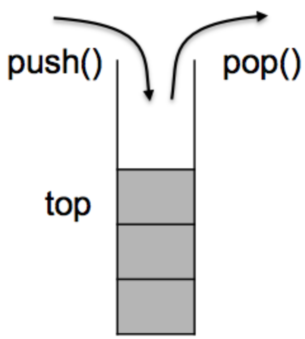
BFS with b=10, 10,000 nodes/sec; 10 bytes/node

Chess* b=35,d=20-40
Rubik's cube* b=13, d=20

Depth-first search

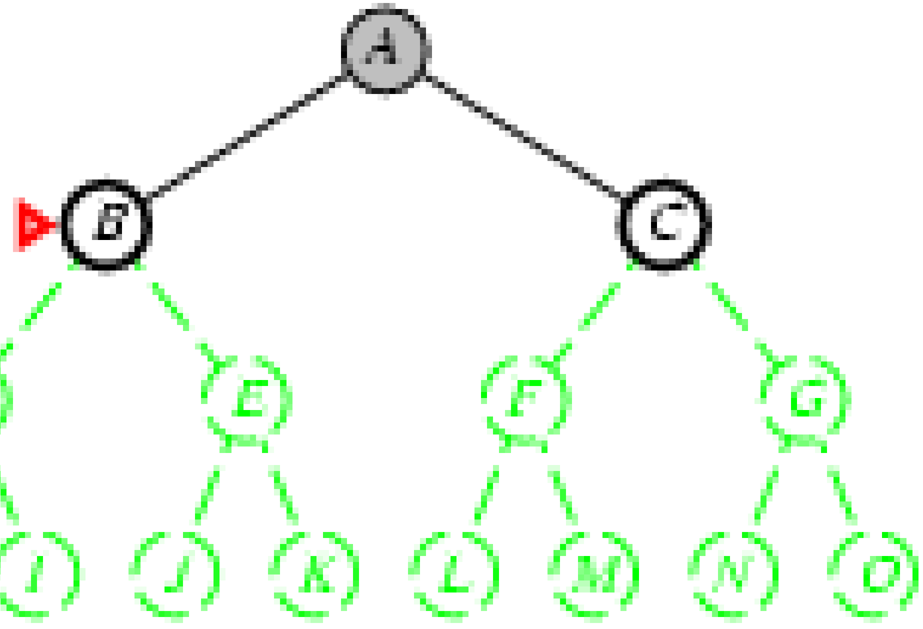
- Expand deepest unexpanded node
- Implementation:
  - *Frontier* = LIFO stack, i.e., put successors at front
  -





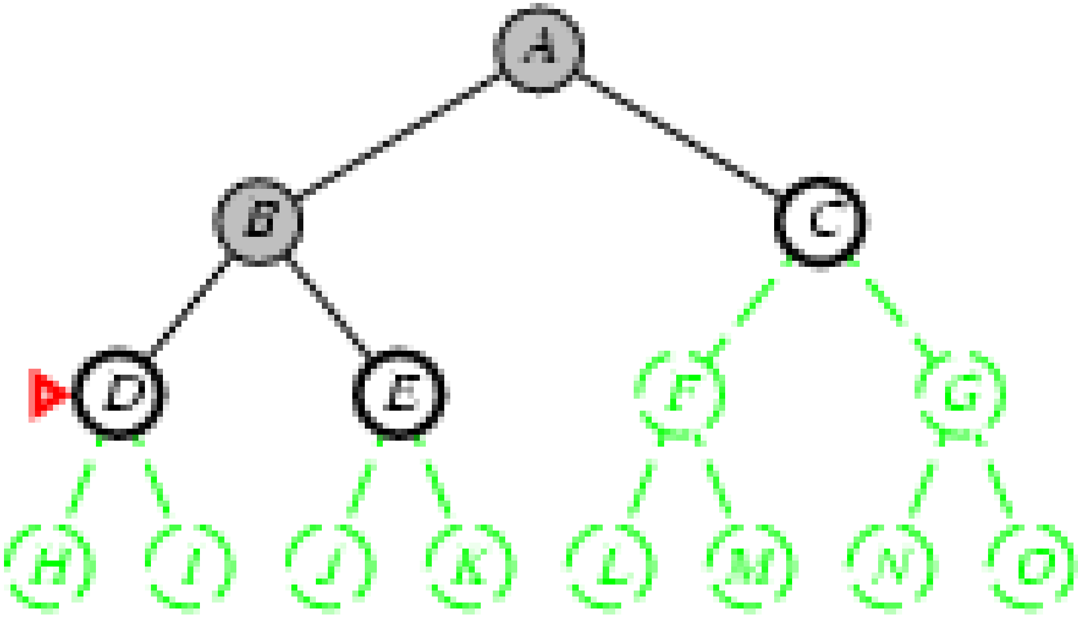
Depth-first search

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



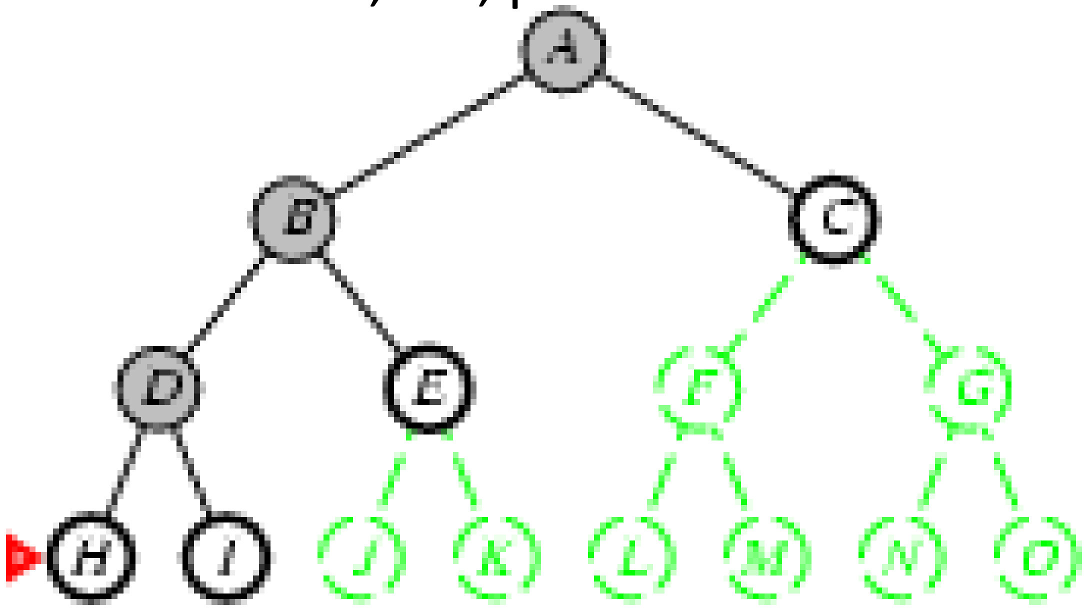
Depth-first search

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



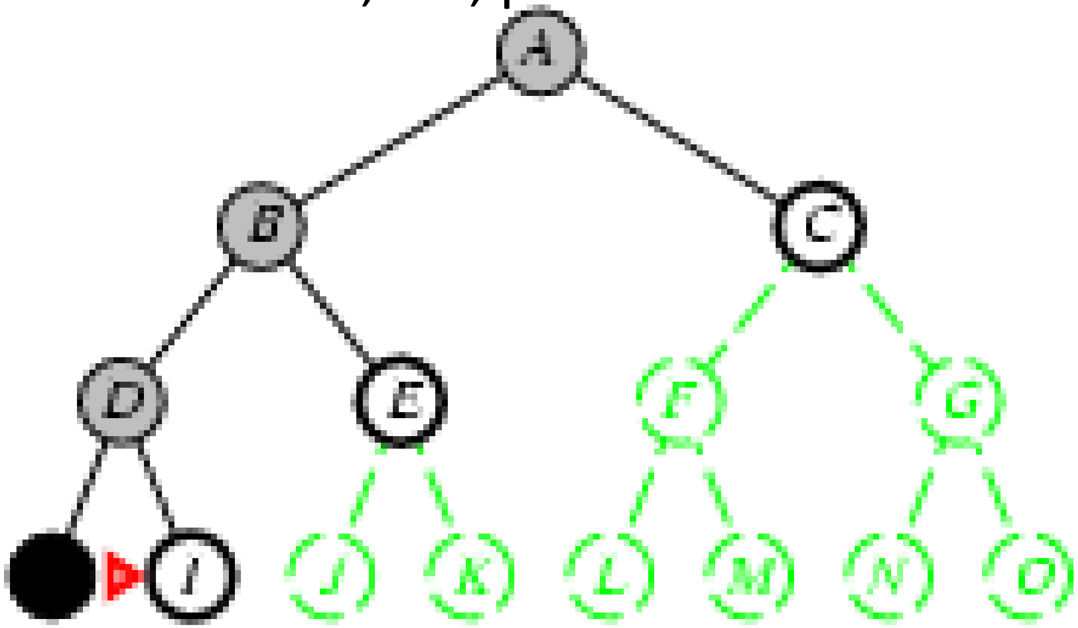
Depth-first search

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



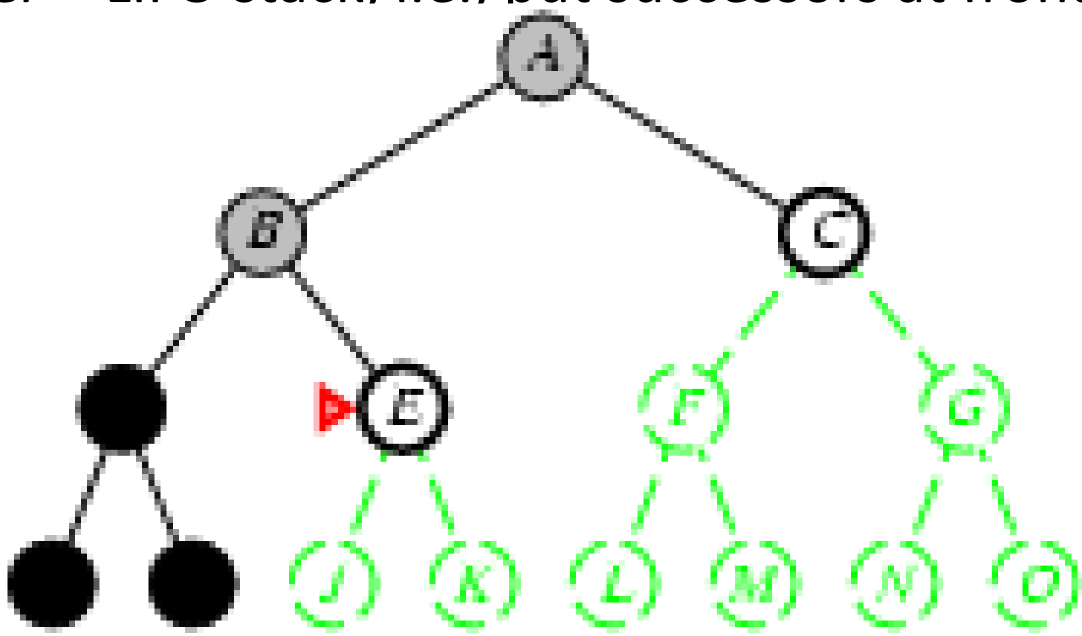
Depth-first search

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



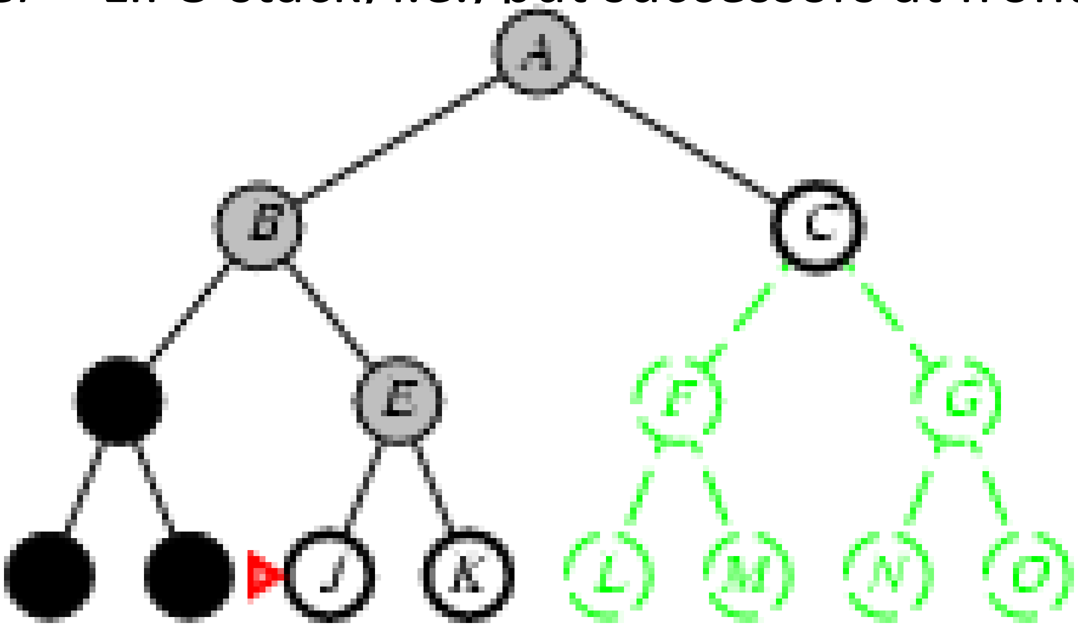
Depth-first search

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



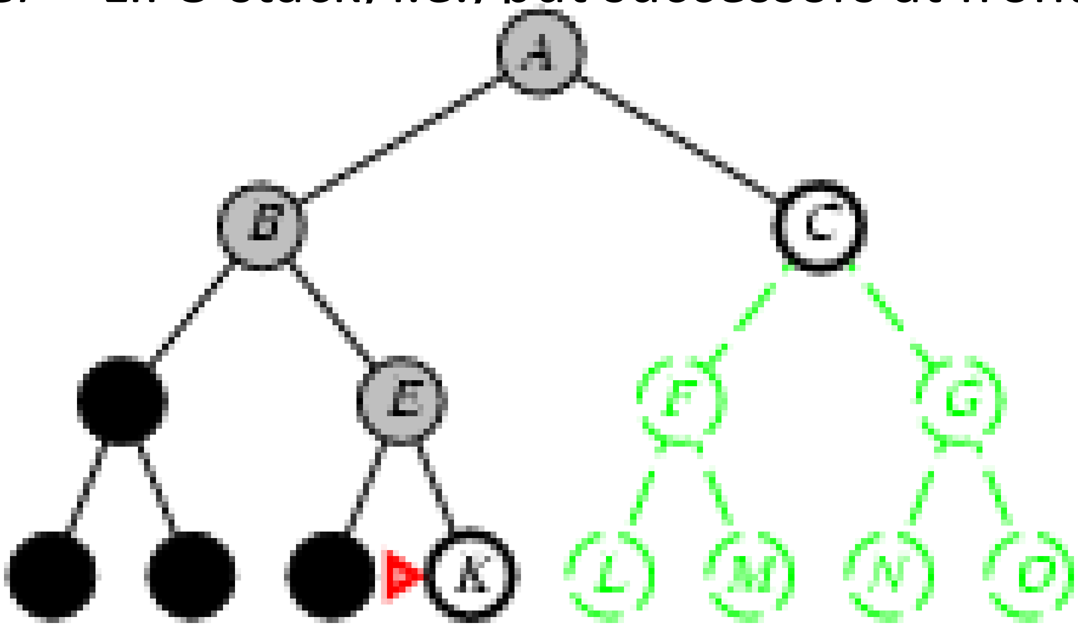
Depth-first search

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



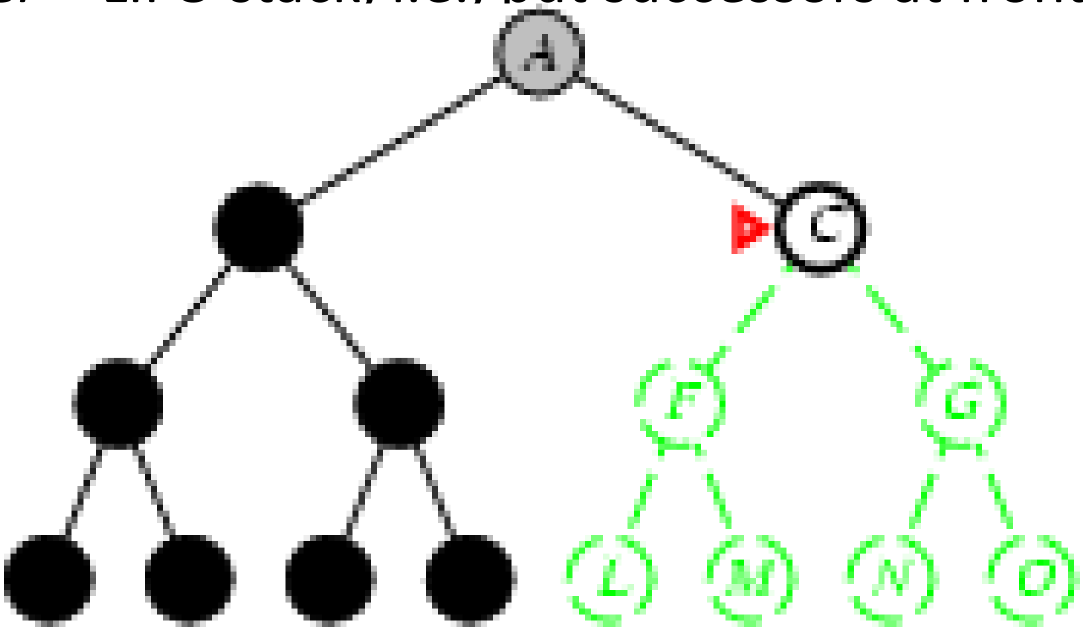
Depth-first search

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



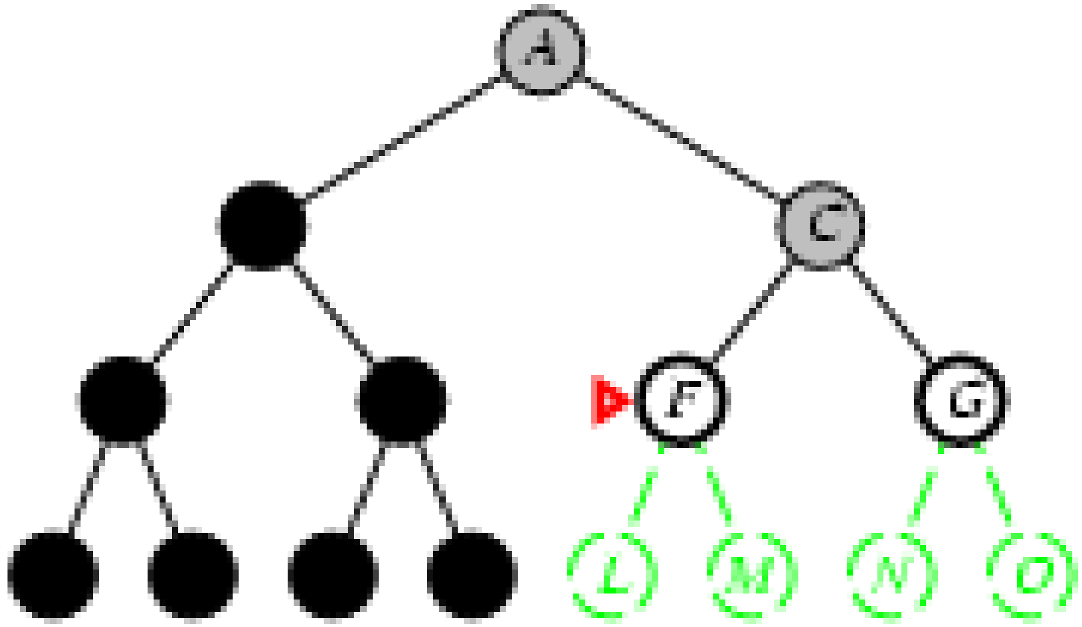
Depth-first search

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



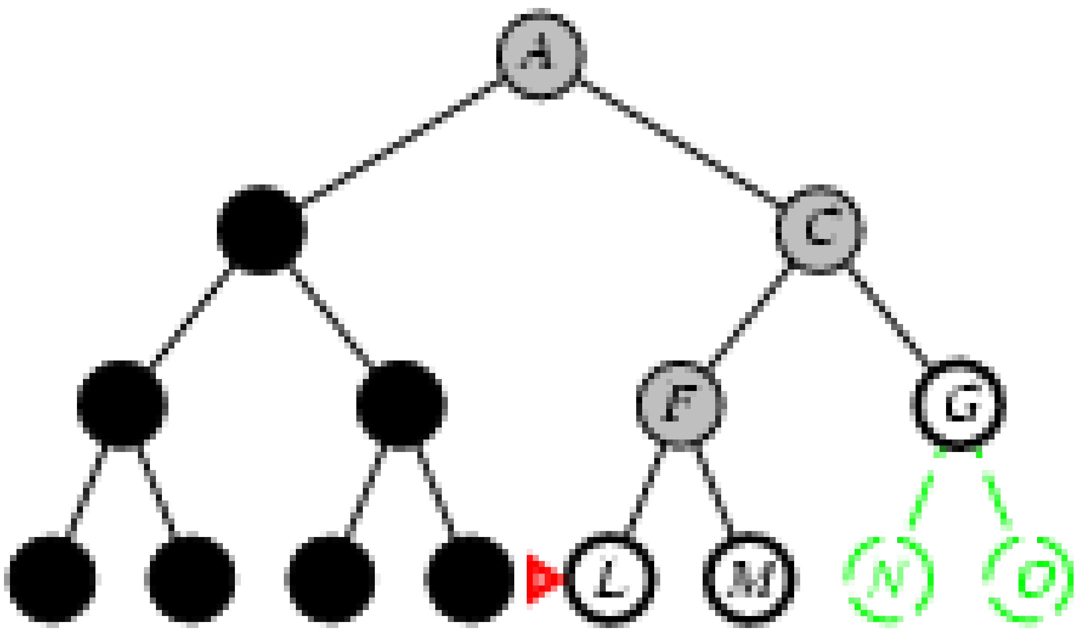
Depth-first search

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



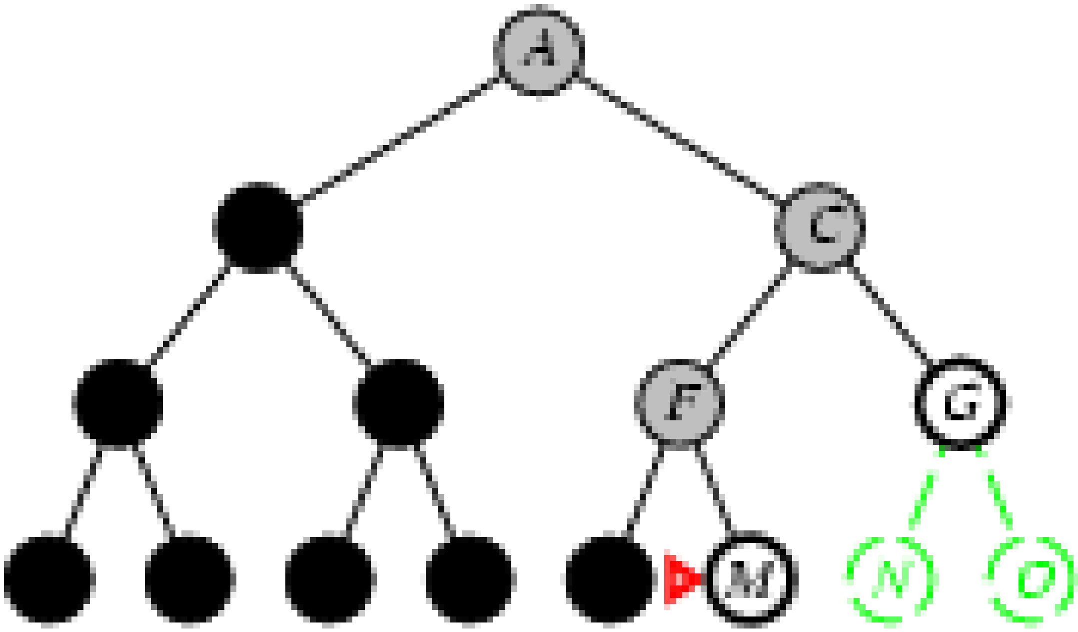
Depth-first search

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



Depth-first search

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



Depth-first search

- Expand deepest unexpanded node
- Implementation:
  - *Frontier* = LIFO queue, i.e., put successors at front
  -
- Evaluation:
  - Completeness: only for finite spaces
  - Time complexity: O(b^m)
  - Space complexity: O(mb)
  - Optimality: no!

  - (m is max depth of nodes)

DFS as a recursive function

```
Function DFS(node)
        if goal(node) == TRUE
                Return node
        For all n in Expand(node)
                if DFS(n)!=FAIL
                        Return DFS(n)
        Return FAIL
```

Program is then DFS(start_node)

DFS as a recursive function+path

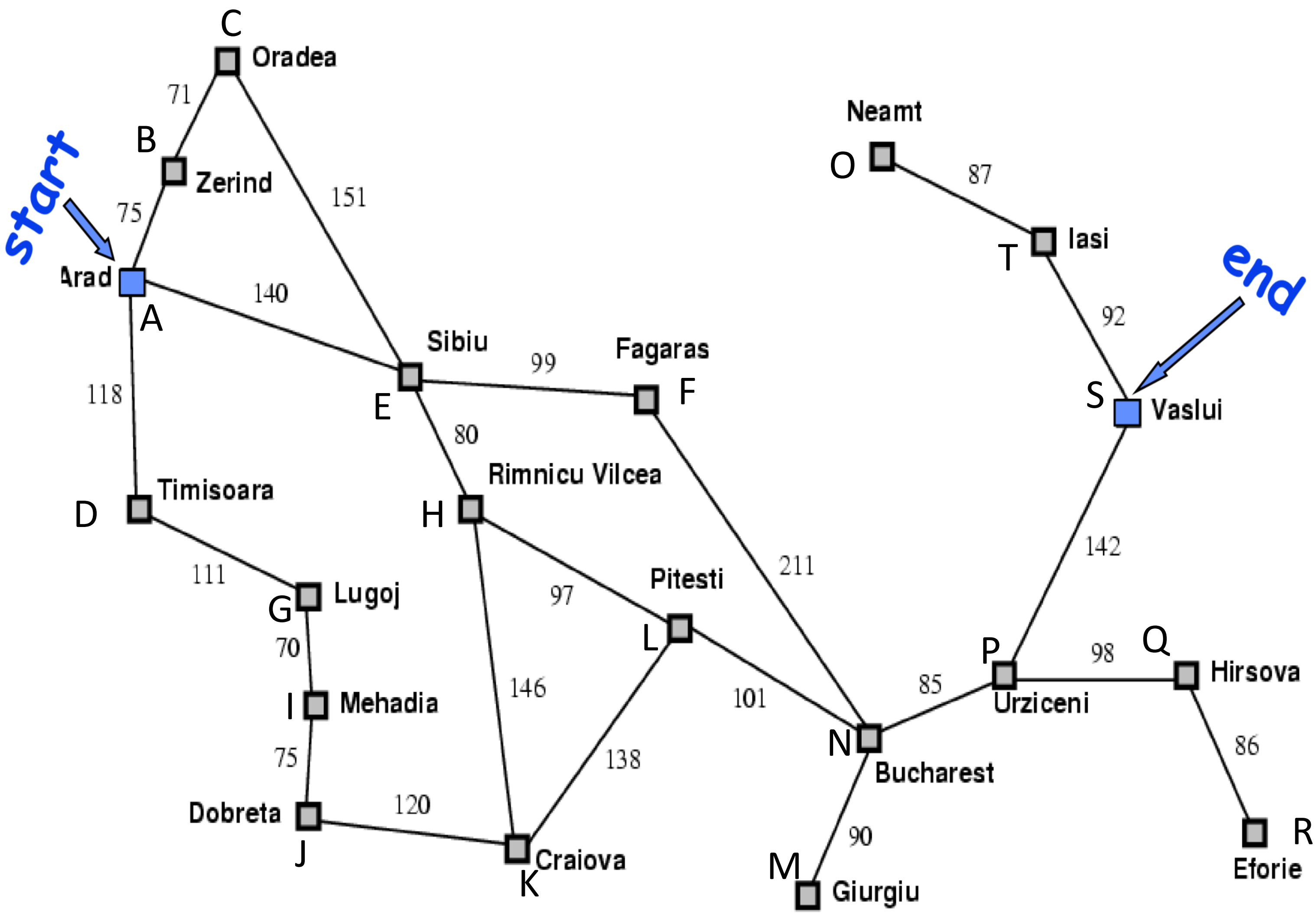
Path=empty
DFS(root_node)

```
Function DFS(node)
        if goal(node) == TRUE
                path<-node
                Return node
        For all n in Expand(node)
                if DFS(n)!=FAIL
                        path<-node
                        Return DFS(n)
        Return FAIL
```

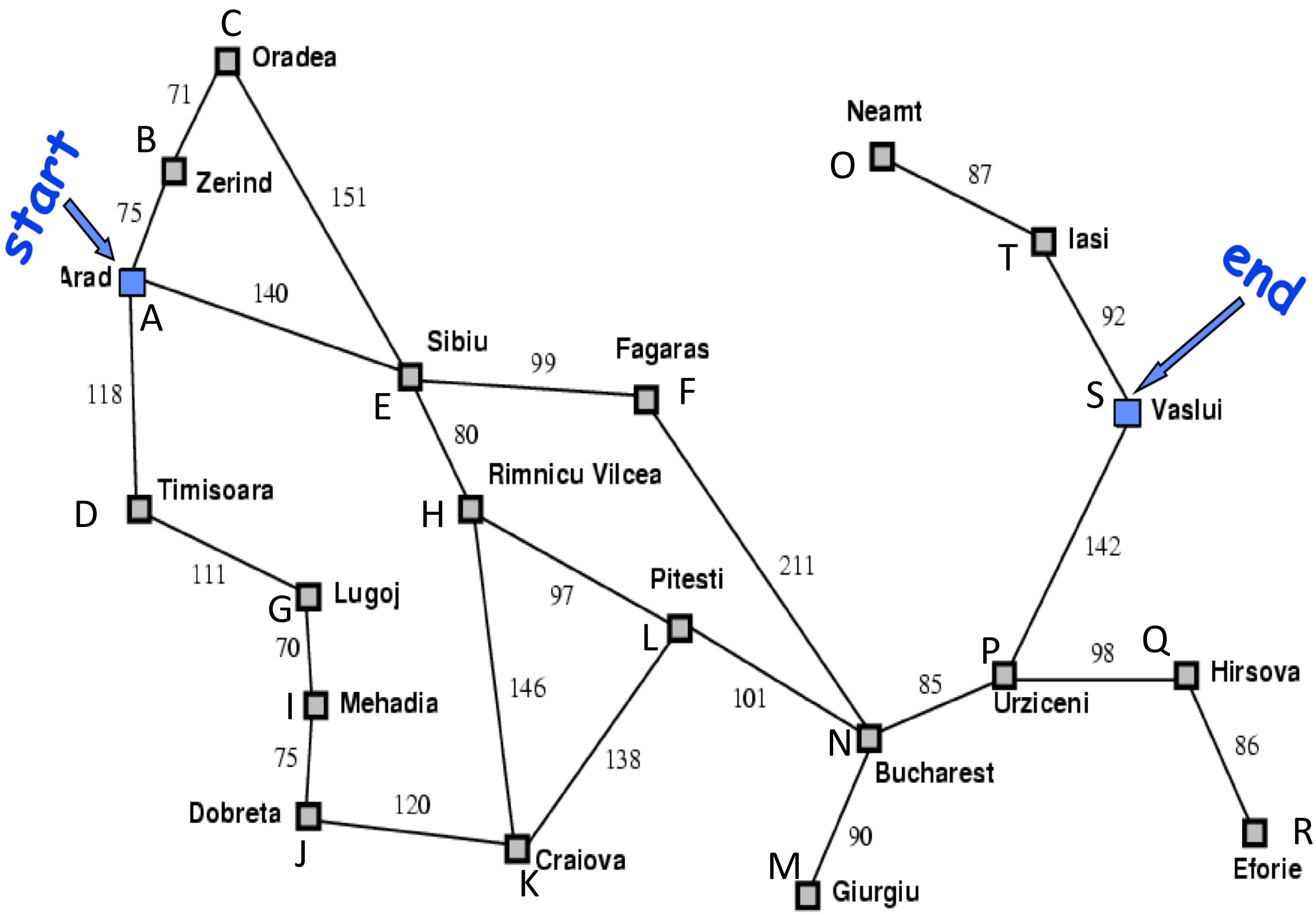
Problems with BFS and DFS

- BFS
  - memory! ☹
- DFS
  - Not optimal
  - And not even necessarily complete!

Exercise DFS (stack)

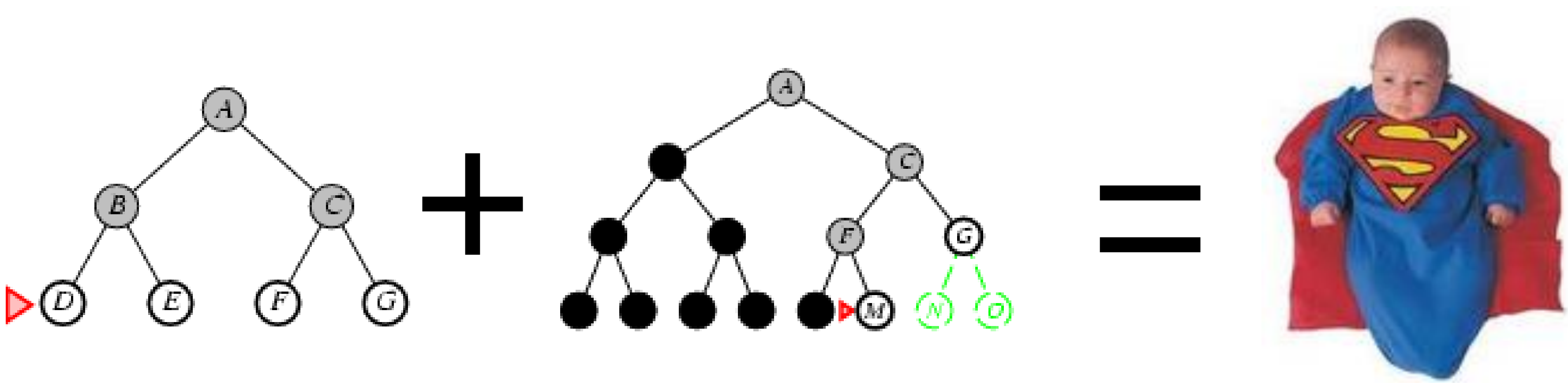


Exercise BFS (queue)



Ideas?

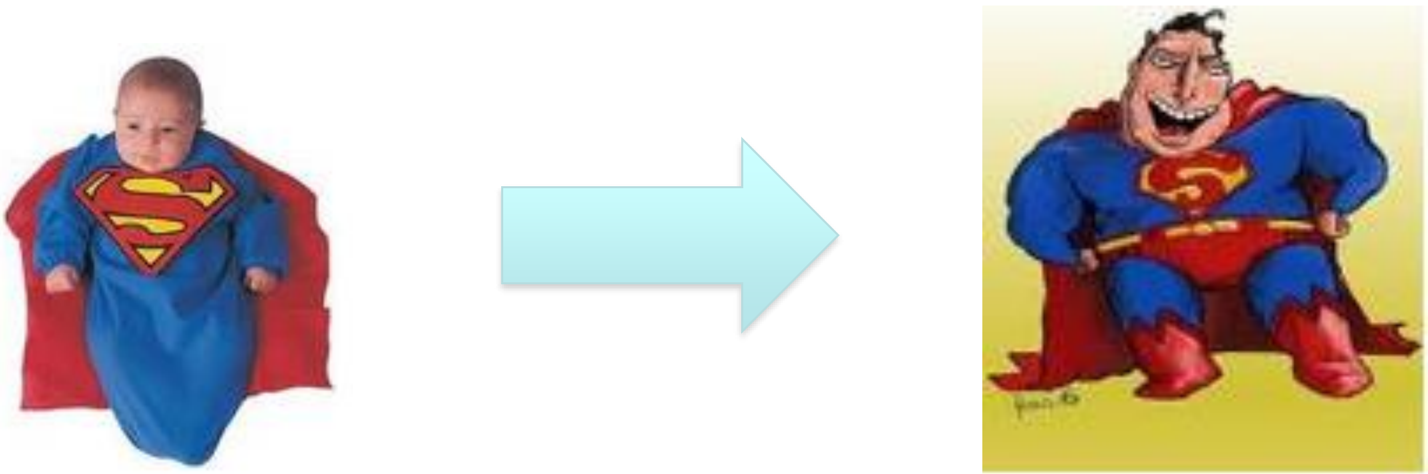
Can we combined the optimality and completeness of BFS with the memory of DFS?



Depth limited DFS

- DFS, but with a depth limit **L** specified
  - nodes at depth **L** are treated as if they have no successors
  - we only search down to depth **L**
- Time?
  - O(b^L) , it was O(b^m)
- Space?
  - O(bL), it was O(bm)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn’t

Ideas?



Iterative deepening search

For depth 0, 1, …., ∞

    run depth limited DFS

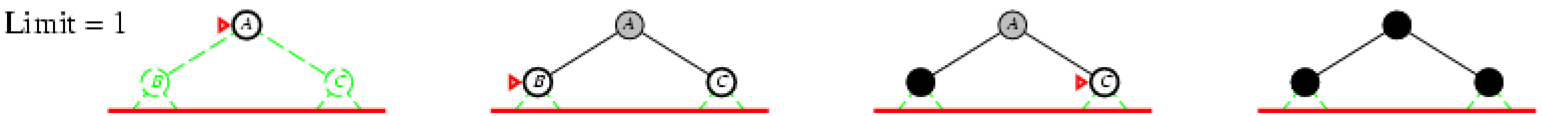
    if solution found, return result

- Blends the benefits of BFS and DFS
  - searches in a similar order to BFS
  - but has the memory requirements of DFS
- Will find the solution when **L** is the depth of the shallowest goal

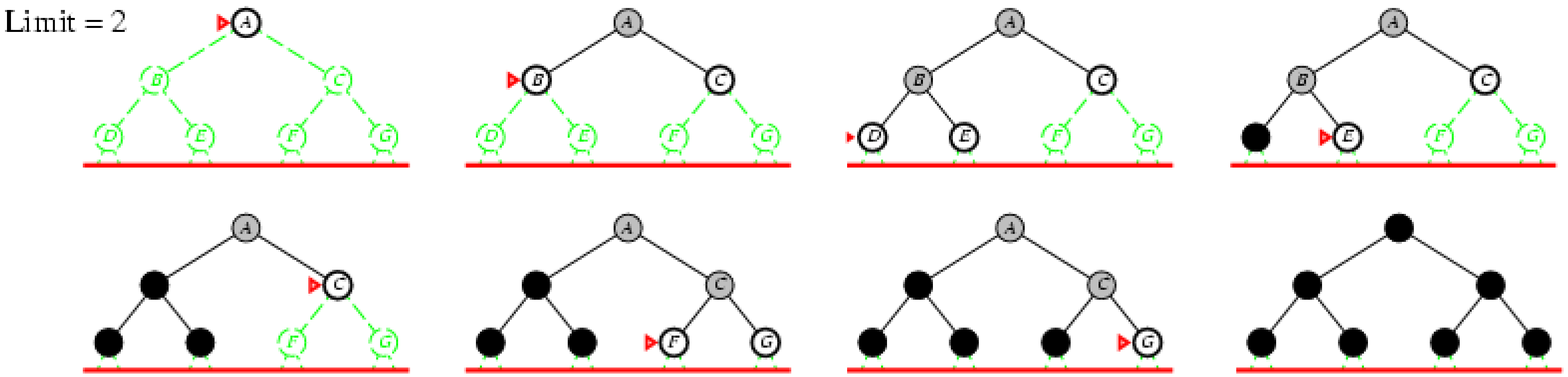
Iterative deepening search L =0

Limit = 0

Iterative deepening search L =1

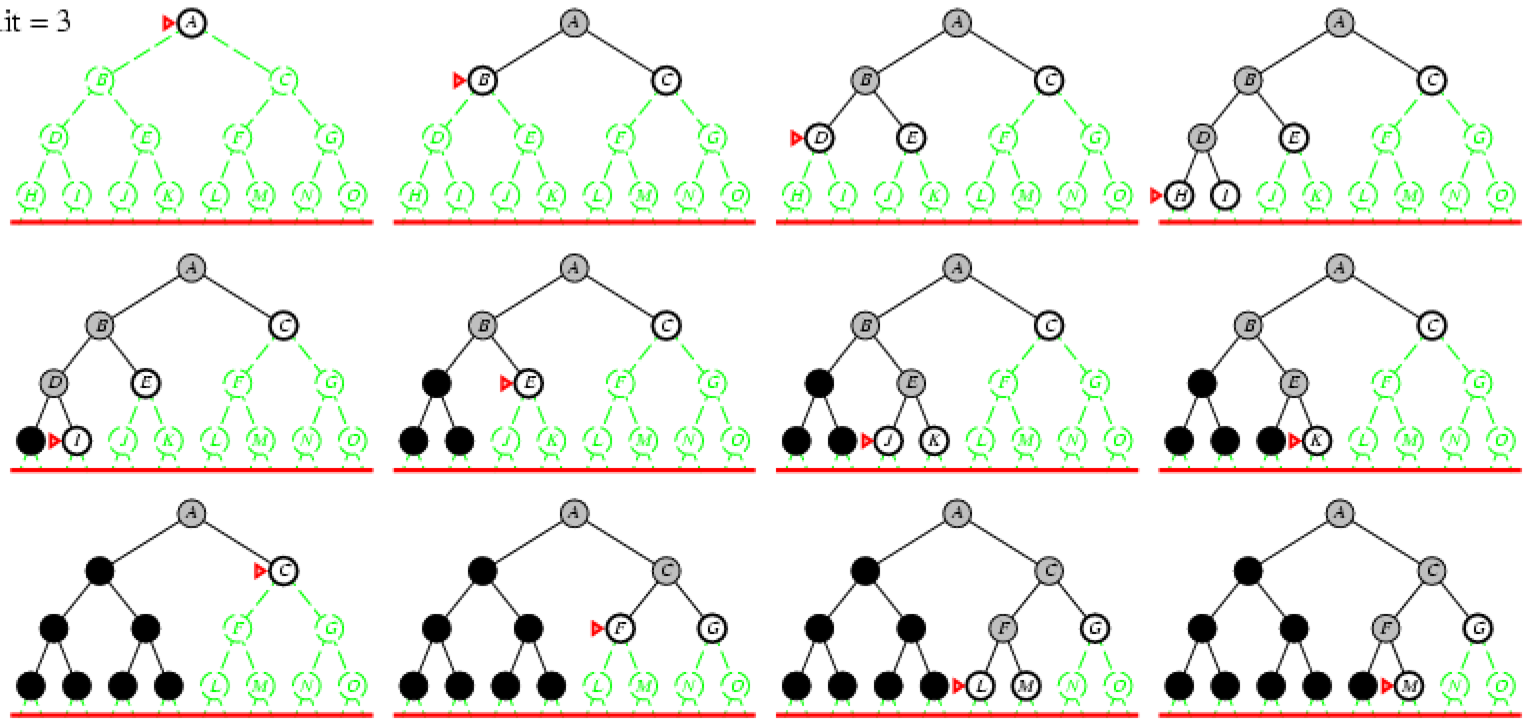


Iterative deepening search L =2



Iterative deepening search L =3

Limit = 3



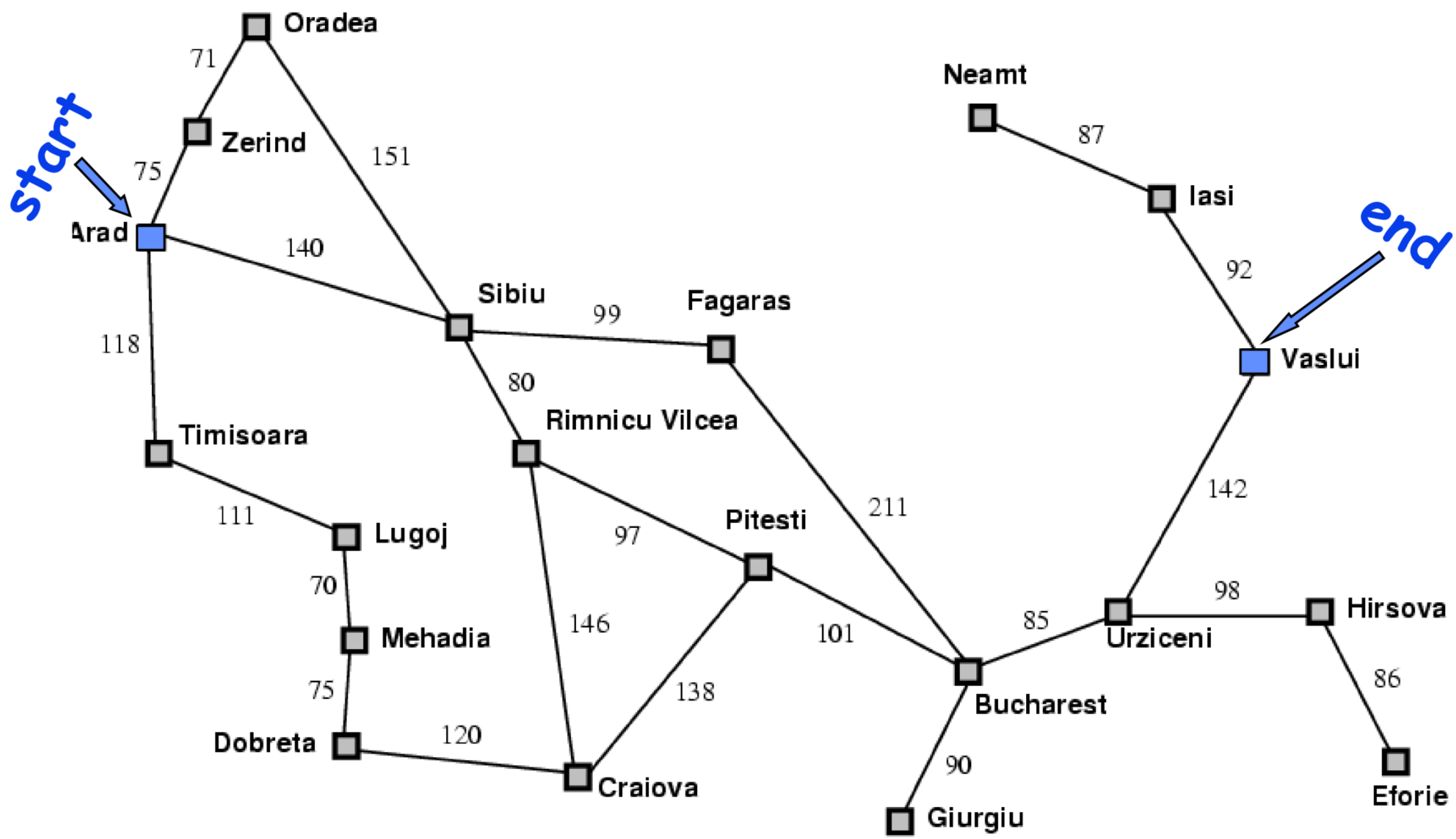
Time?

- $L = 0$: 1
- $L = 1$: $1 + b$
- $L = 2$: $1 + b + b^2$
- $L = 3$: $1 + b + b^2 + b^3$
- …
- $L = d$: $1 + b + b^2 + b^3 + \ldots + b^d$
- Overall:
  - $d(1) + (d-1)b + (d-2)b^2 + (d-3)b^3 + \ldots + b^d$
  - $O(b^d)$ (same as BFS)
  - the cost of the repeat of the lower levels is subsumed by the cost at the highest level

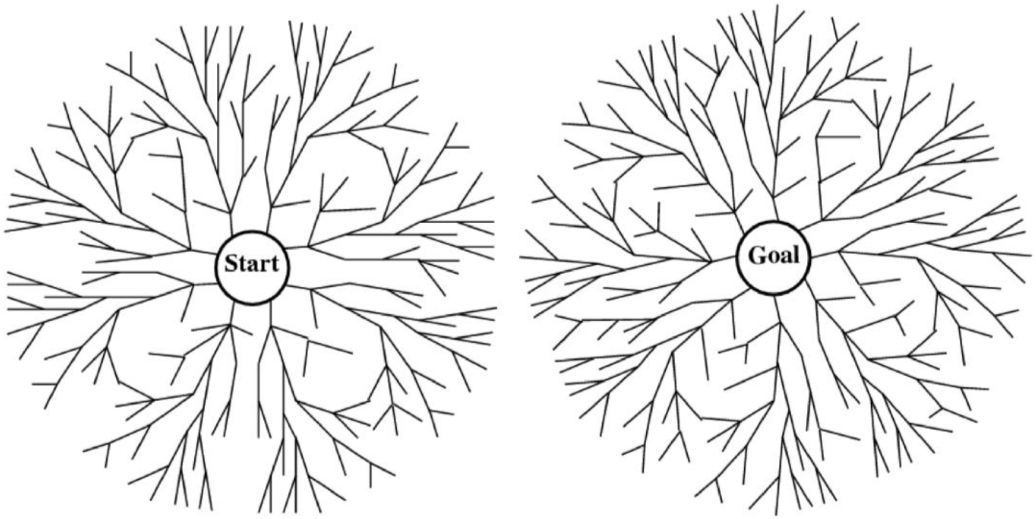
Properties of iterative deepening search

- <u>Space?</u>
  - *O(bd) recall BFS is* O(b^d)
- <u>Complete?</u>
  - Yes
- <u>Optimal?</u>
  - Yes
- Efficient?
    - no

Bi-directional search

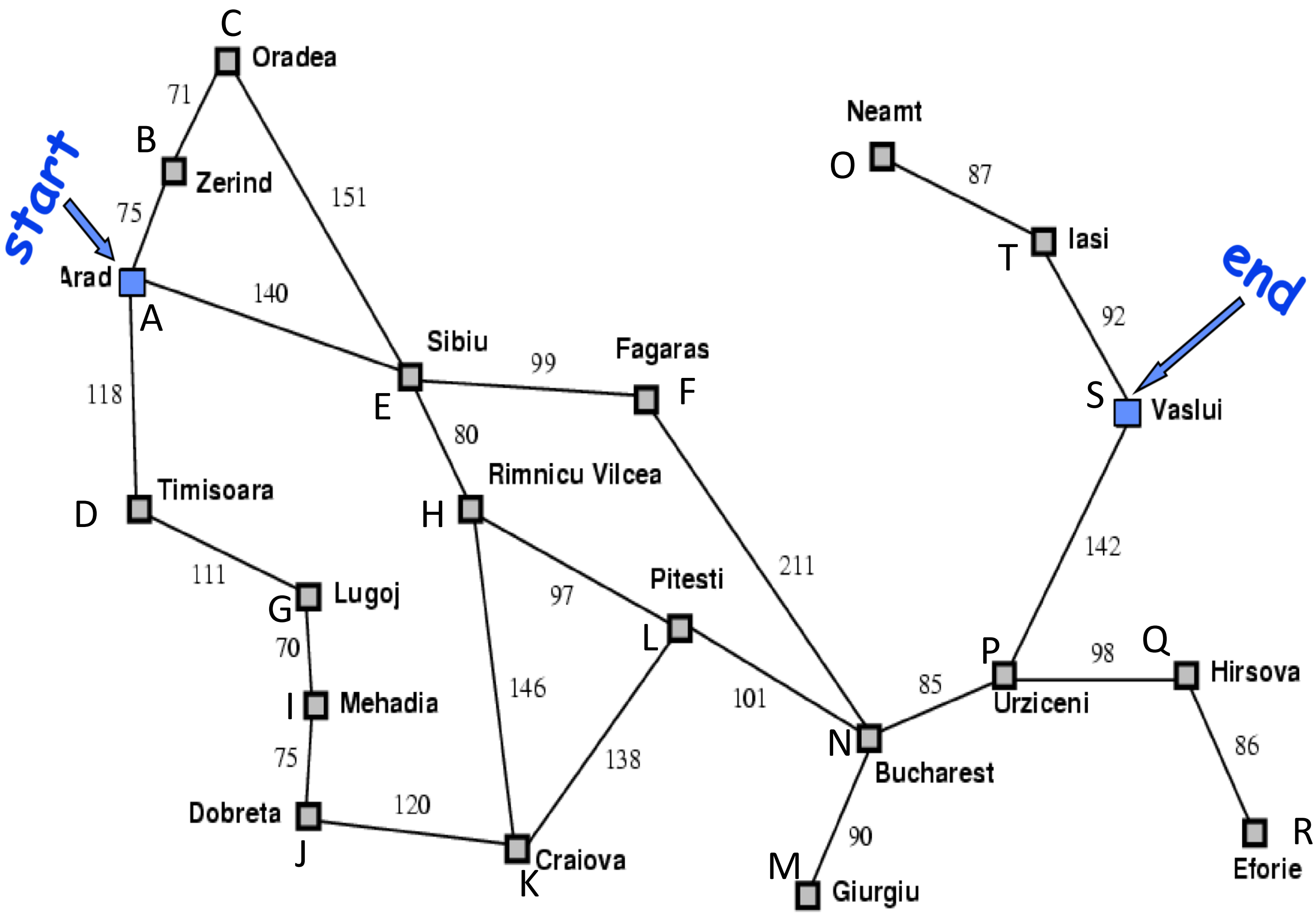


Bi-directional search



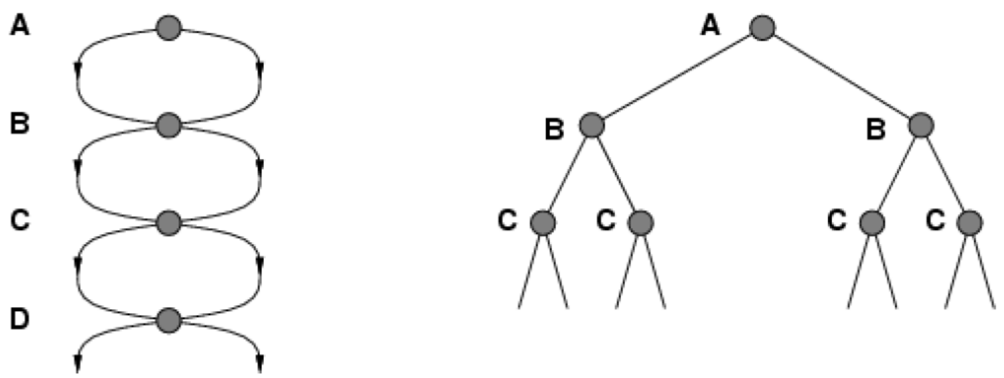
- Evaluation:
  - Completeness: yes (if BFS is used)
  - Time complexity: $O(b^{d/2})$
  - Space complexity: $O(b^{d/2})$
  - Optimality: yes
- Problems?
  - Goal known

Bi-directional search example



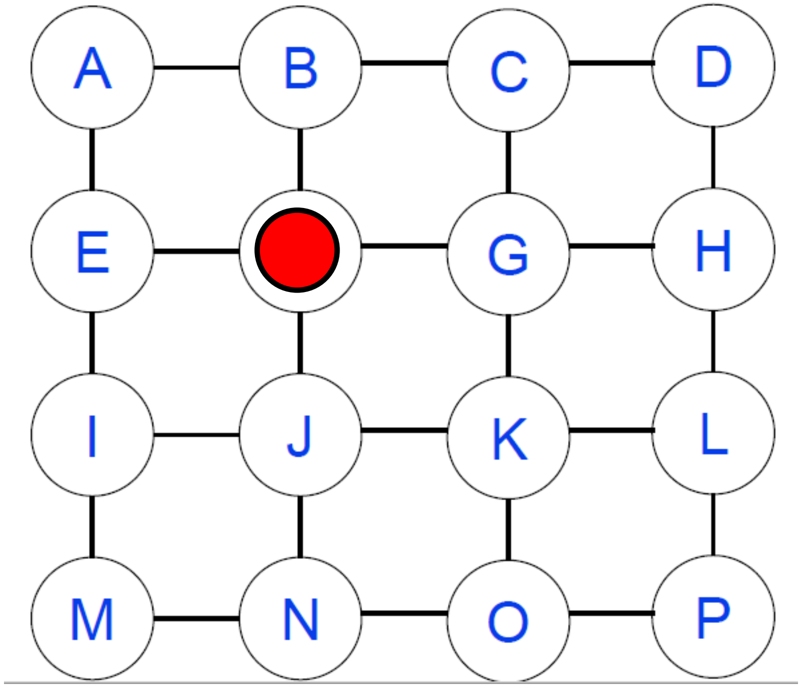
Repeated states

Failure to detect repeated states can turn linear problem to an exponential one!



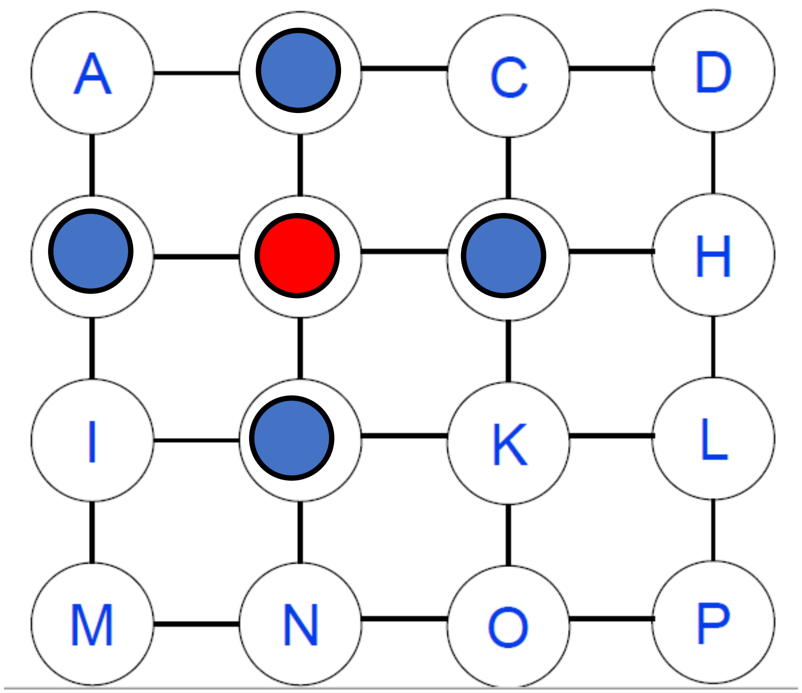
Repeated states

- More realistic case



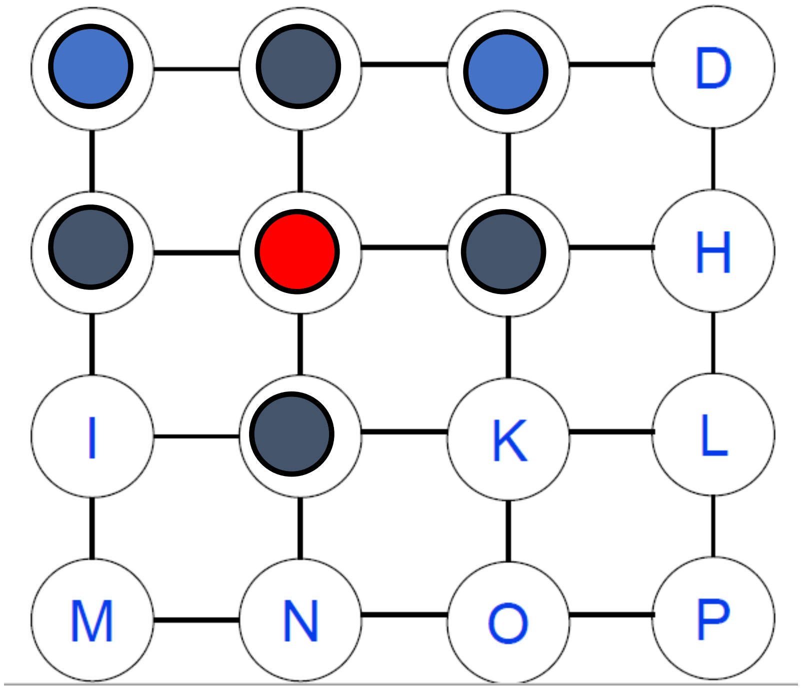
Repeated states

- More realistic case



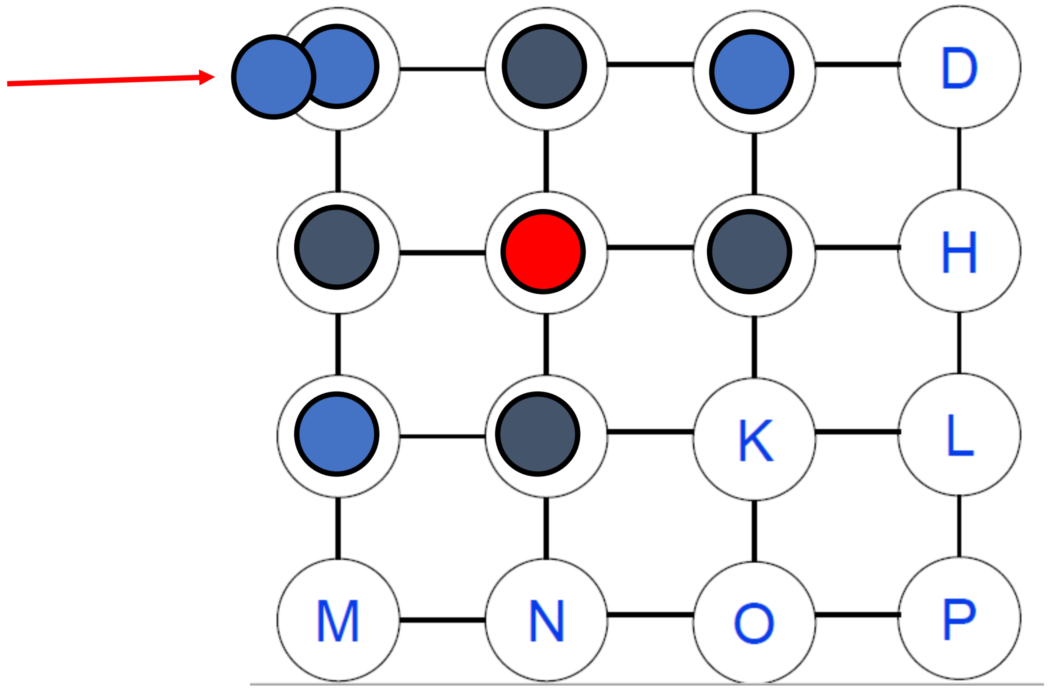
Repeated states

- More realistic case



Repeated states

- More realistic case



Summary

- Search problems
- Search strategies
  - DFS, BFS, Depth limited, ID, Bi-directional
  - Issue: all these methods are slow (blind)
  - Can we fix this by adding guidance…
  - Next time: Informed search

Homework 1 (due 4/14 7pm)

- Maze solver using uninformed search

1	1	1	1	0
0	2	0	0	0
0	1	1	1	1
0	1	1	3	1
0	0	0	0	0

- Write search functions
  - df_search() bf_search()
  - Modify map with 4's where searched, 5's on found path to goal
- Hint: size of map given to program so dynamic memory not necessary. (but don't assume the map size will always be the same as the test cases)
- Hint: tests given are a good start, but does not guarantee your homework will work on graded tests.
- Full homework description on canvas
- Review academic honesty statement