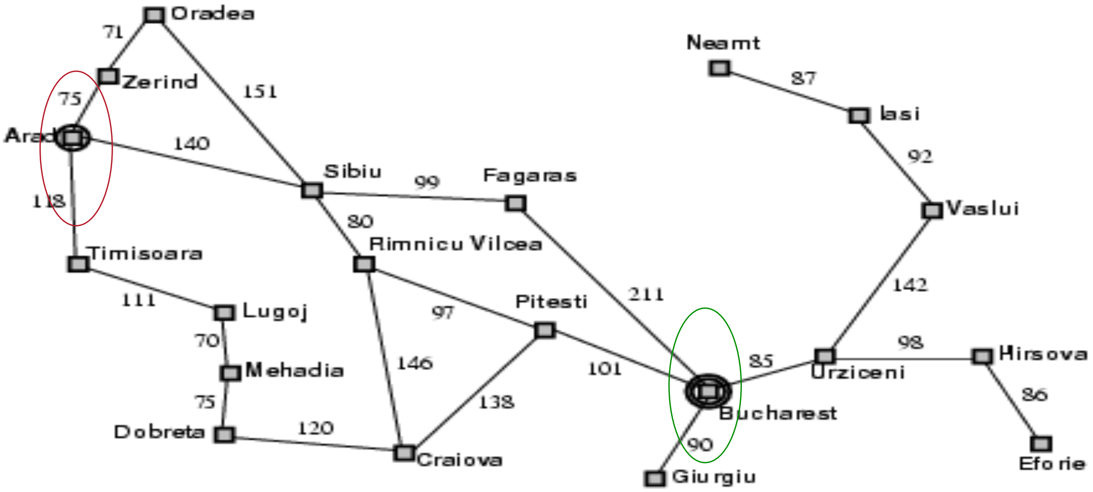
15381: Artificial Intelligence

Behrang Mohit

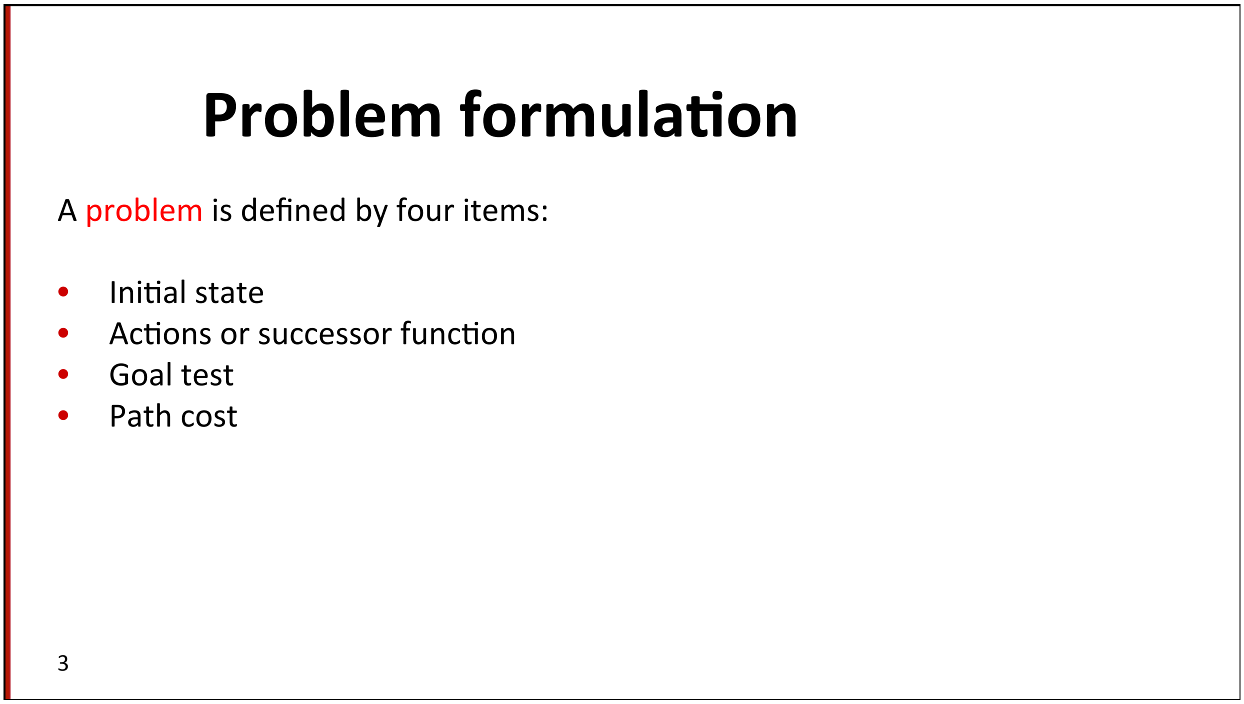
Solving Problems With Search

Some slides, graphics and ideas are borrowed or adapted from courses offered by Stuart Russell, Hwee Tou Ng, Rebecca Hwa and Milos Hauskrecht.

Example Problem: Romania Trip



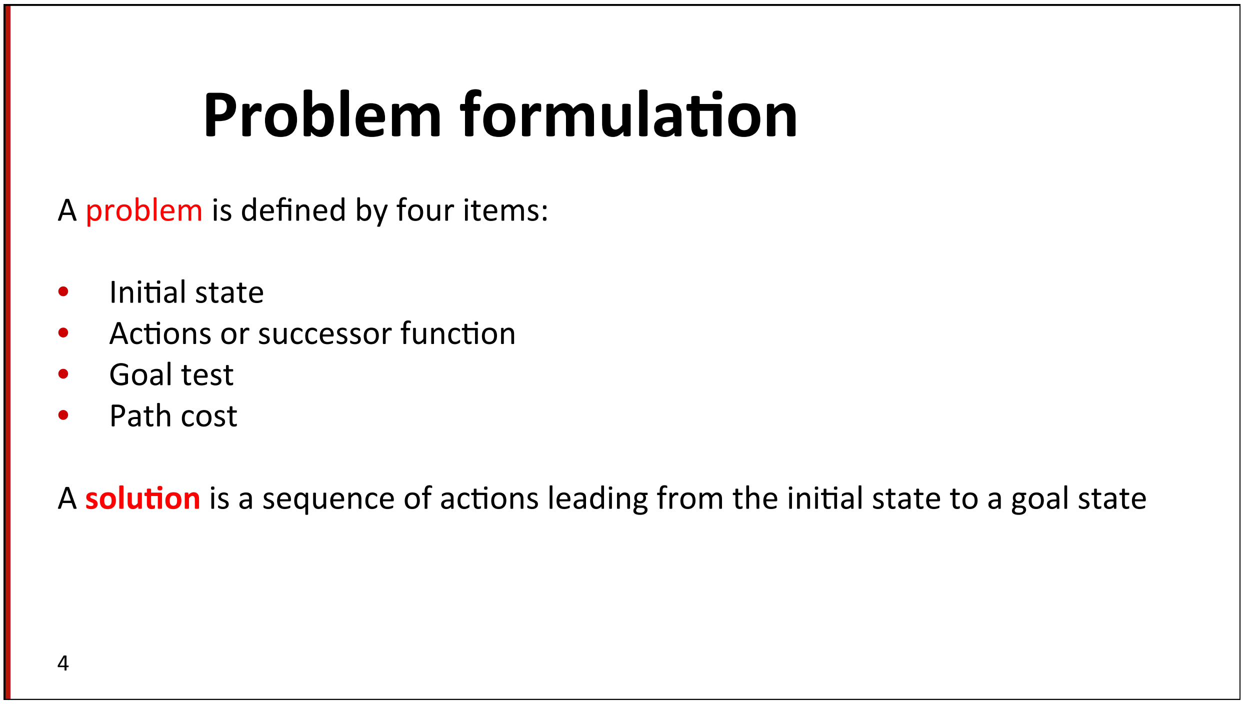
2



Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test
- Path cost



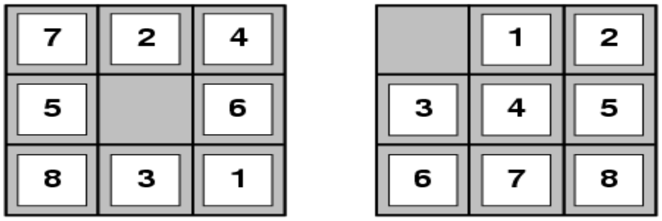
Problem formulation

A problem is defined by four items:

- Initial state
- Actions or successor function
- Goal test
- Path cost

A **solution** is a sequence of actions leading from the initial state to a goal state

Example: The 8-puzzle



7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

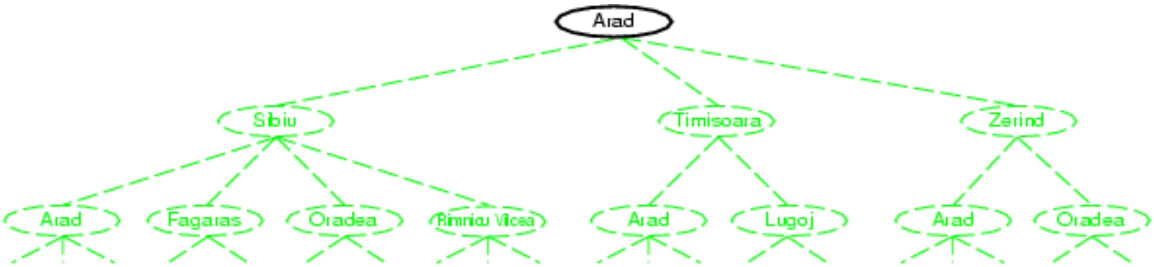
Goal State

- States?
- Actions?
- Goal test?
- Path cost?

Tree search algorithms

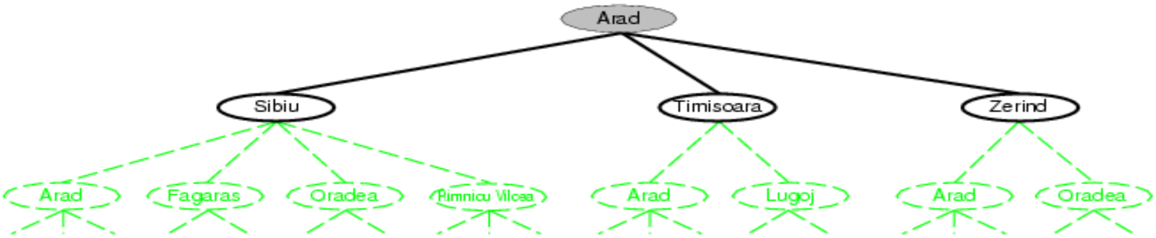
- Use a tree analogy for the movement from the initial state to the goal.
- Basic idea:
  - offline, simulated exploration of state space by generating successors of already-explored states (a.k.a.~expanding states)

Tree search example



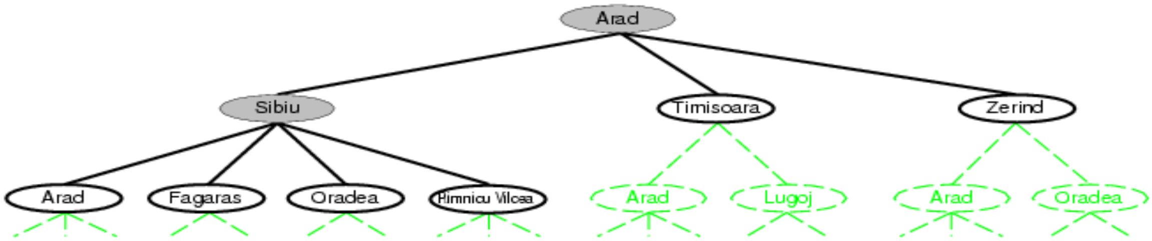
7

Tree search example



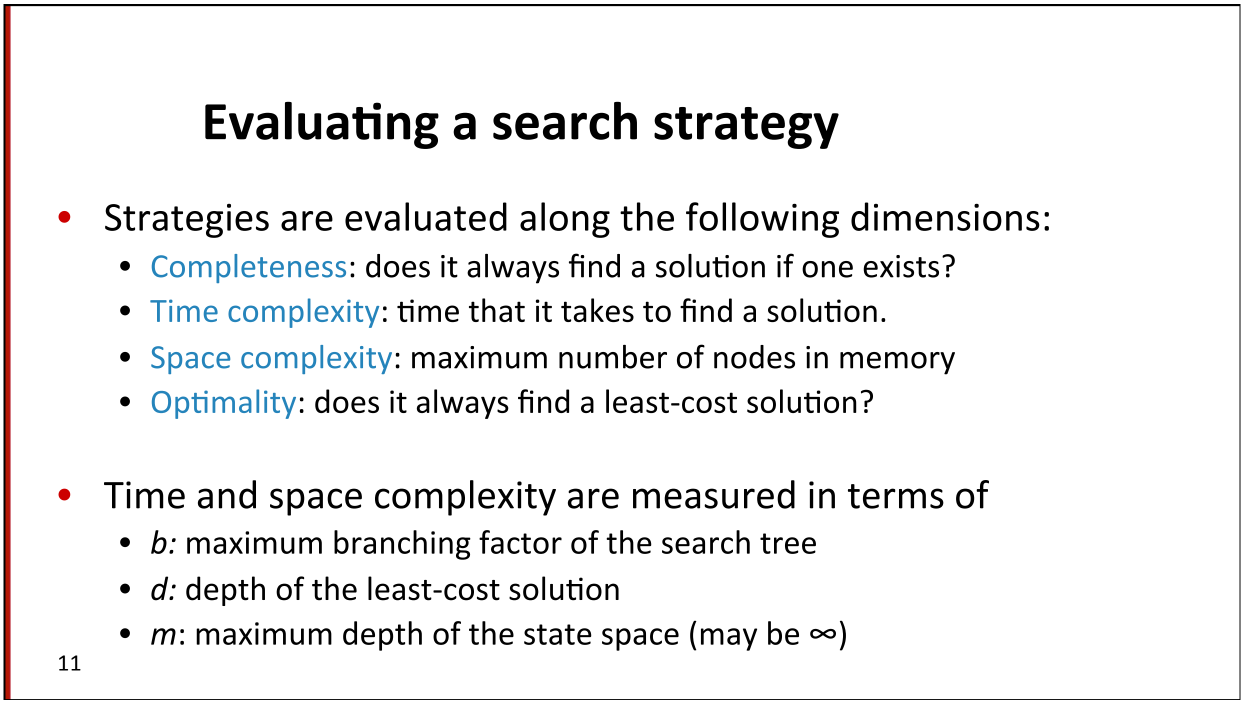
8

Tree search example



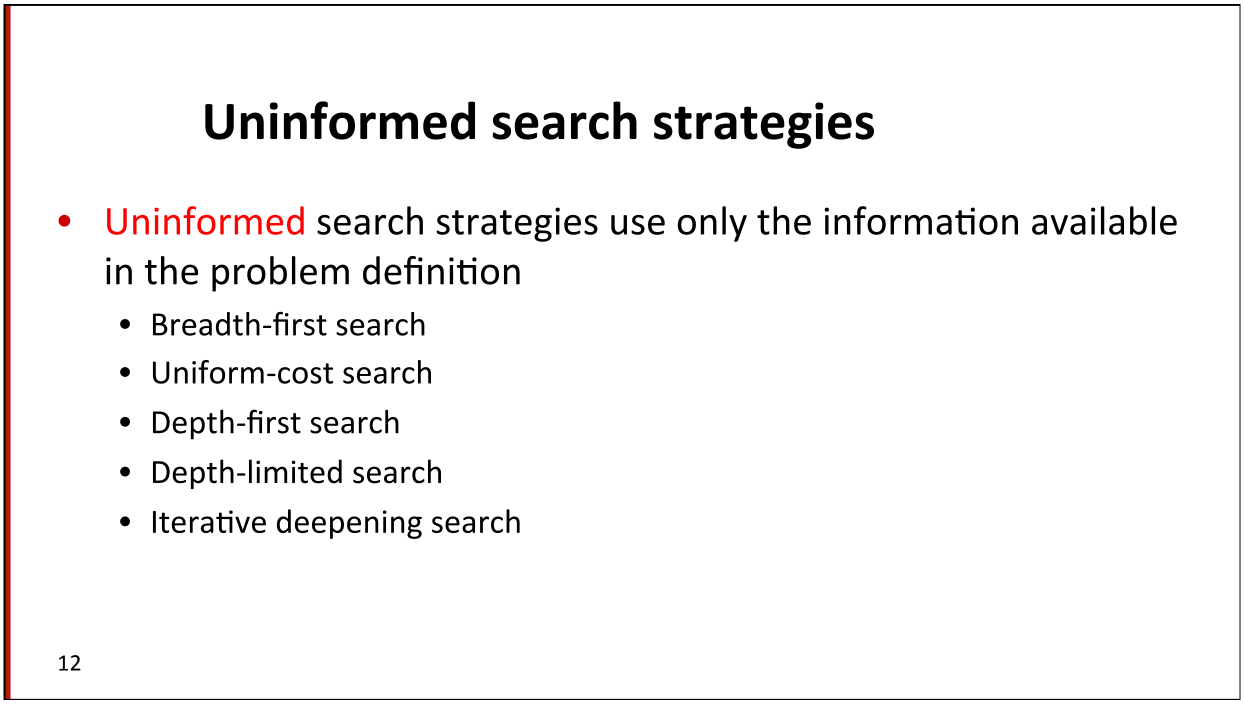
Search strategies

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - Completeness:
  - Time complexity:
  - Space complexity:
  - Optimality:



Evaluating a search strategy

- Strategies are evaluated along the following dimensions:
  - Completeness: does it always find a solution if one exists?
  - Time complexity: time that it takes to find a solution.
  - Space complexity: maximum number of nodes in memory
  - Optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

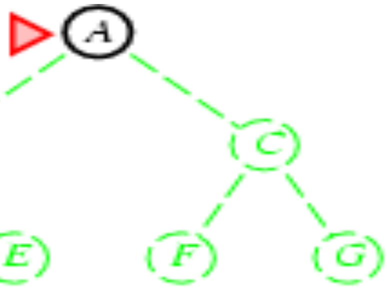


Uninformed search strategies

- Uninformed search strategies use only the information available in the problem definition
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

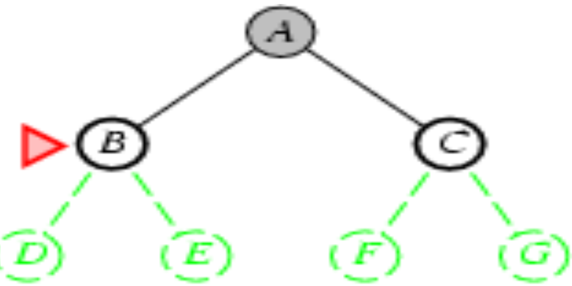
Breadth-first search (BFS)

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



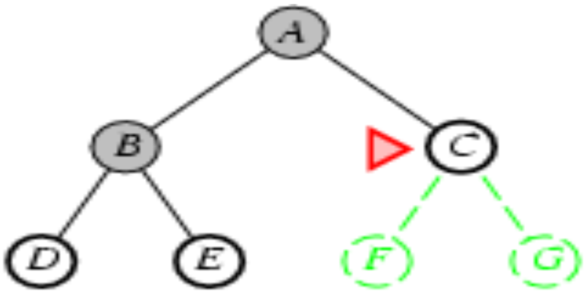
Breadth-first search (BFS)

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



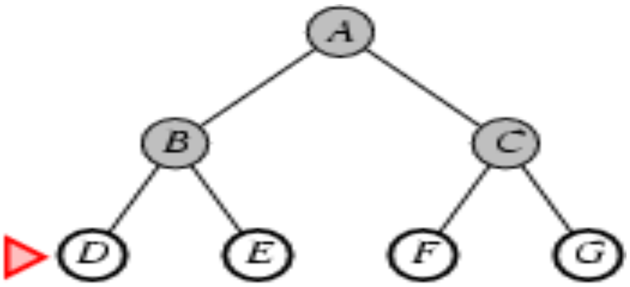
Breadth-first search (BFS)

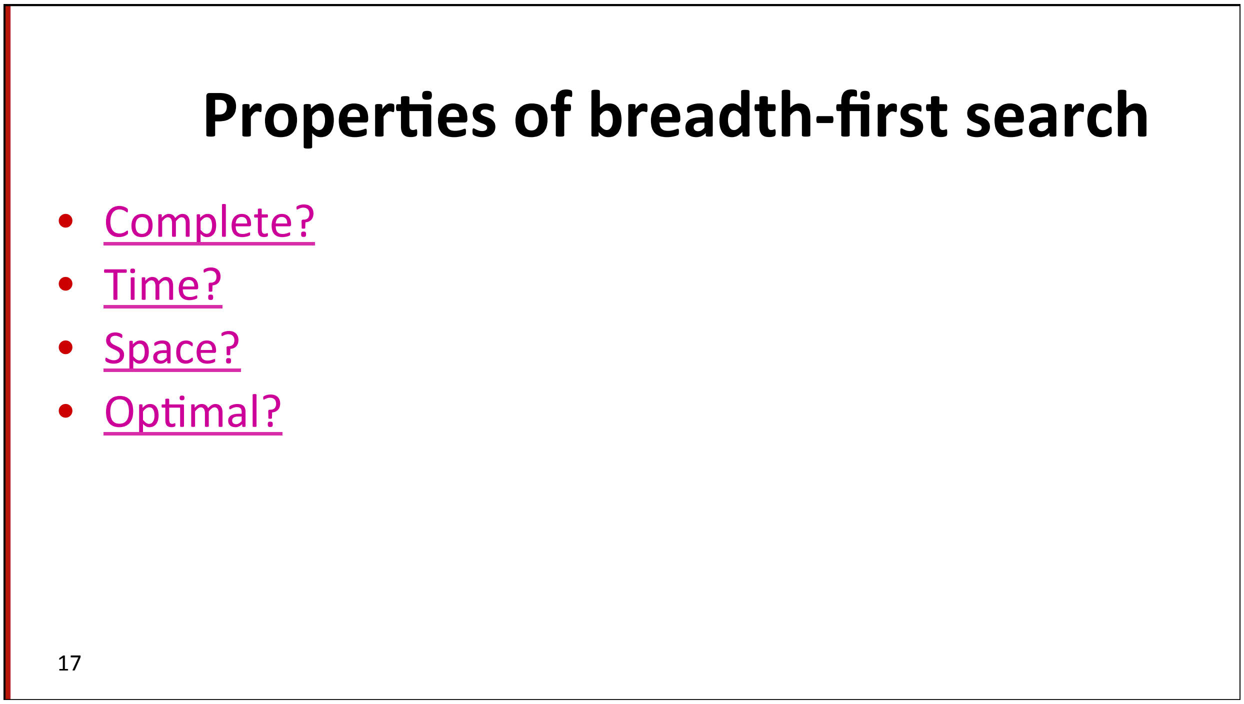
- Expand shallowest unexpanded node
- Implementation:
  - *Frontier* is a FIFO queue, i.e., new successors go at end



Breadth-first search (BFS)

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

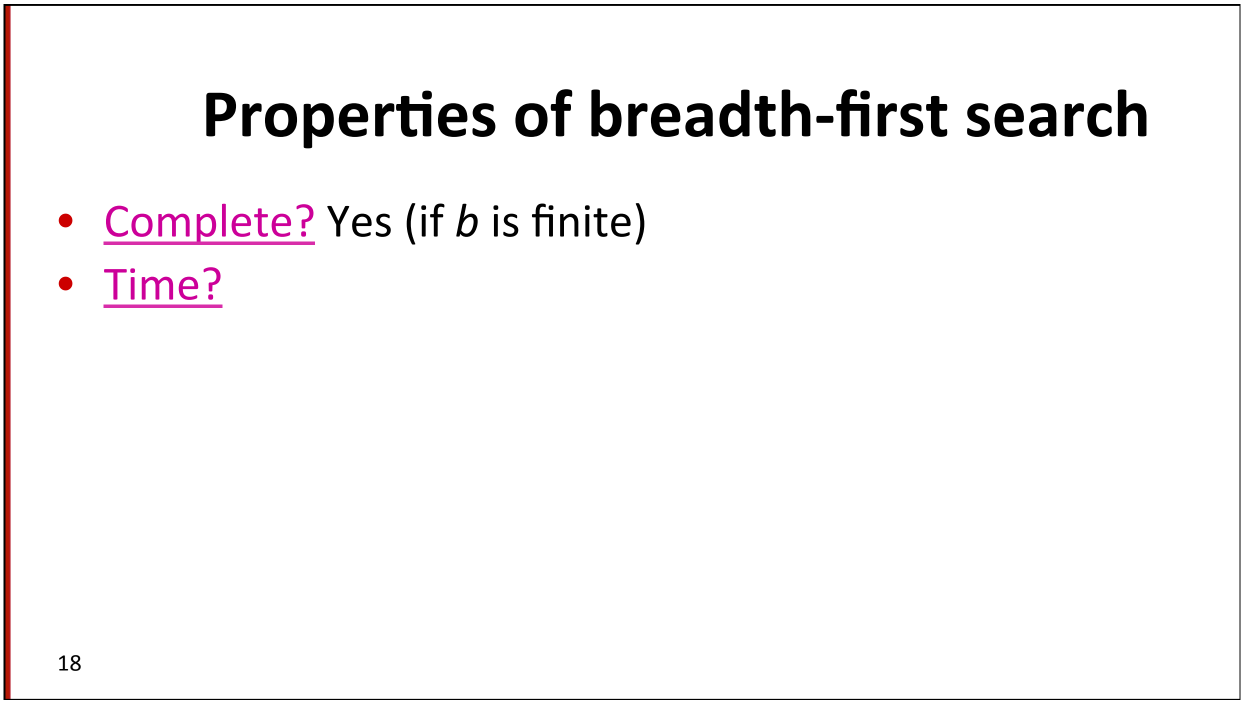




Properties of breadth-first search

- Complete?
- Time?
- Space?
- Optimal?

17

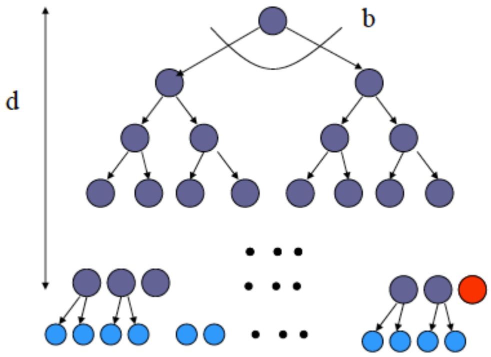


Properties of breadth-first search

- Complete? Yes (if *b* is finite)
- Time?

18

Time and space complexity of BFS



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
d	2^d (b^d)
d+1	2^{d+1} (b^{d+1})

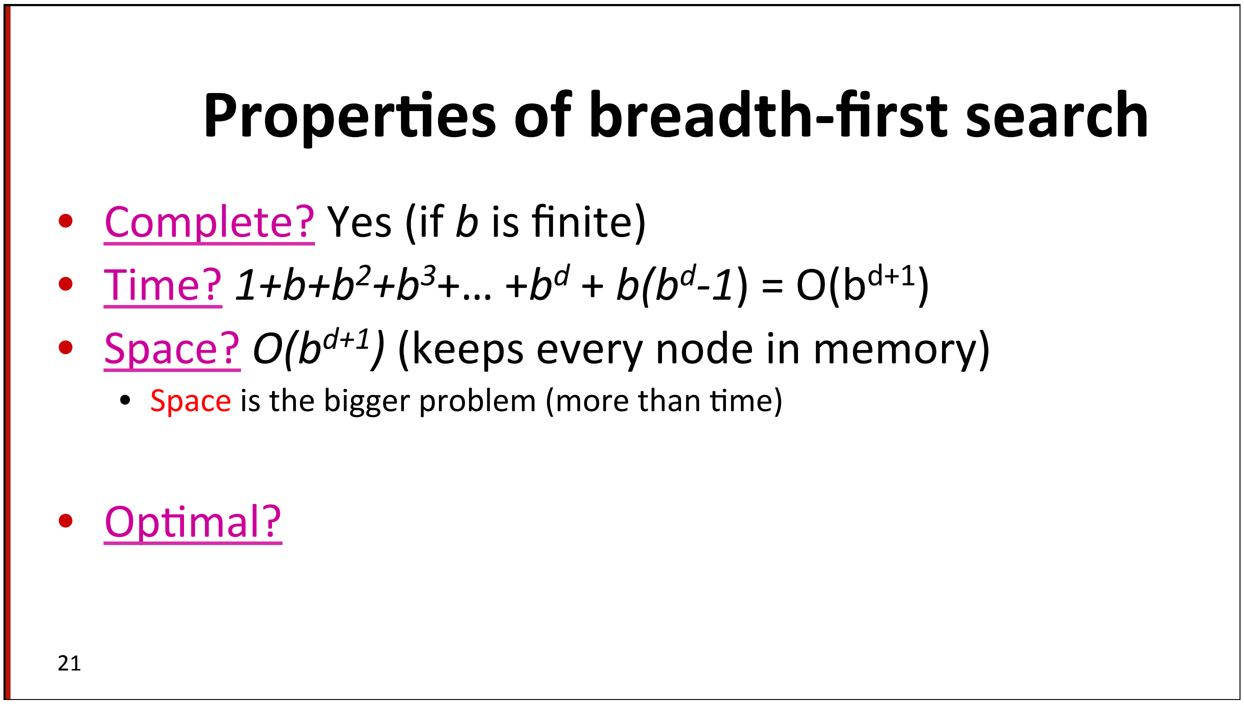
Expanded nodes: $O(b^d)$

Total nodes: $O(b^{d+1})$

Properties of breadth-first search

- Complete? Yes (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d+b(b^d-1)$ = $O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)

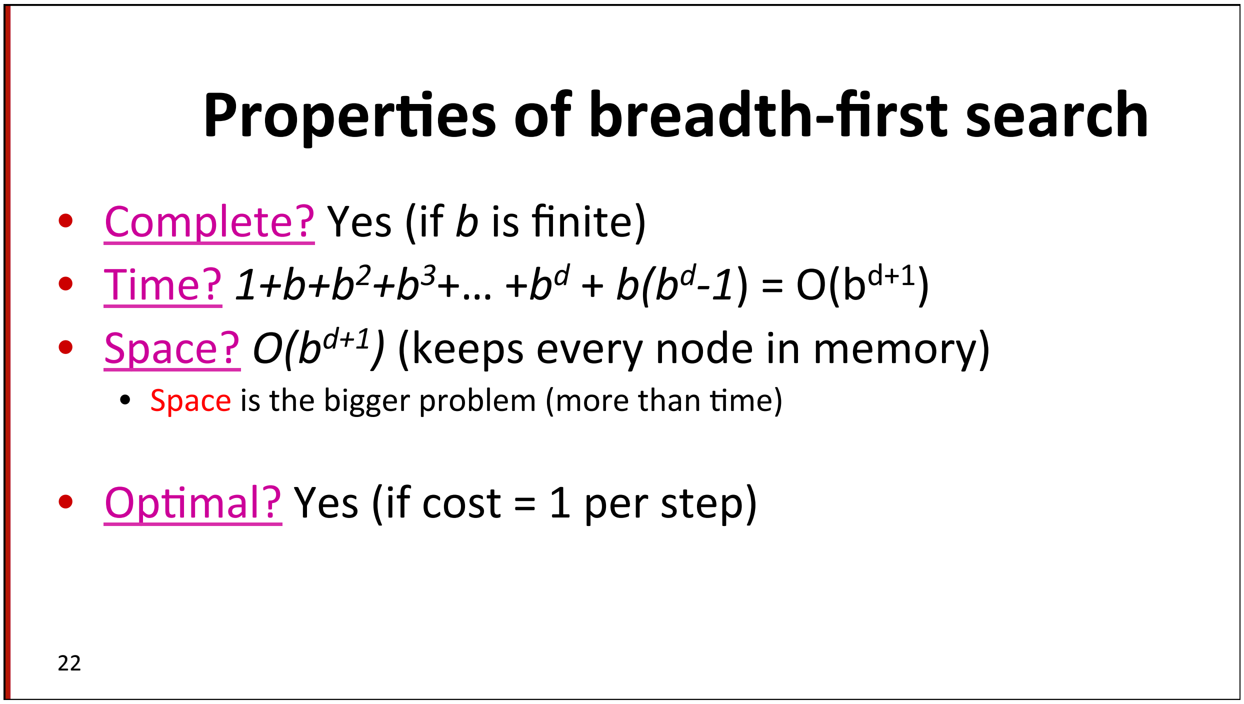
Depth	Nodes	Time	Memory
2	110	.11 m sec.	107 kB
12	10 ^ 12	13 days	1 peta bytes
16	10 ^ 16	350 years	10 exta bytes



Properties of breadth-first search

- Complete? Yes (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
  - Space is the bigger problem (more than time)

- Optimal?



Properties of breadth-first search

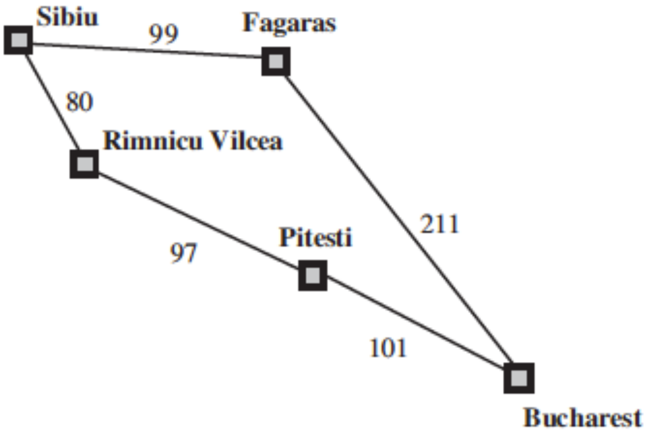
- Complete? Yes (if *b* is finite)
- Time? $1+b+b^2+b^3+\ldots+b^d + b(b^d-1) = O(b^{d+1})$
- Space? $O(b^{d+1})$ (keeps every node in memory)
  - Space is the bigger problem (more than time)

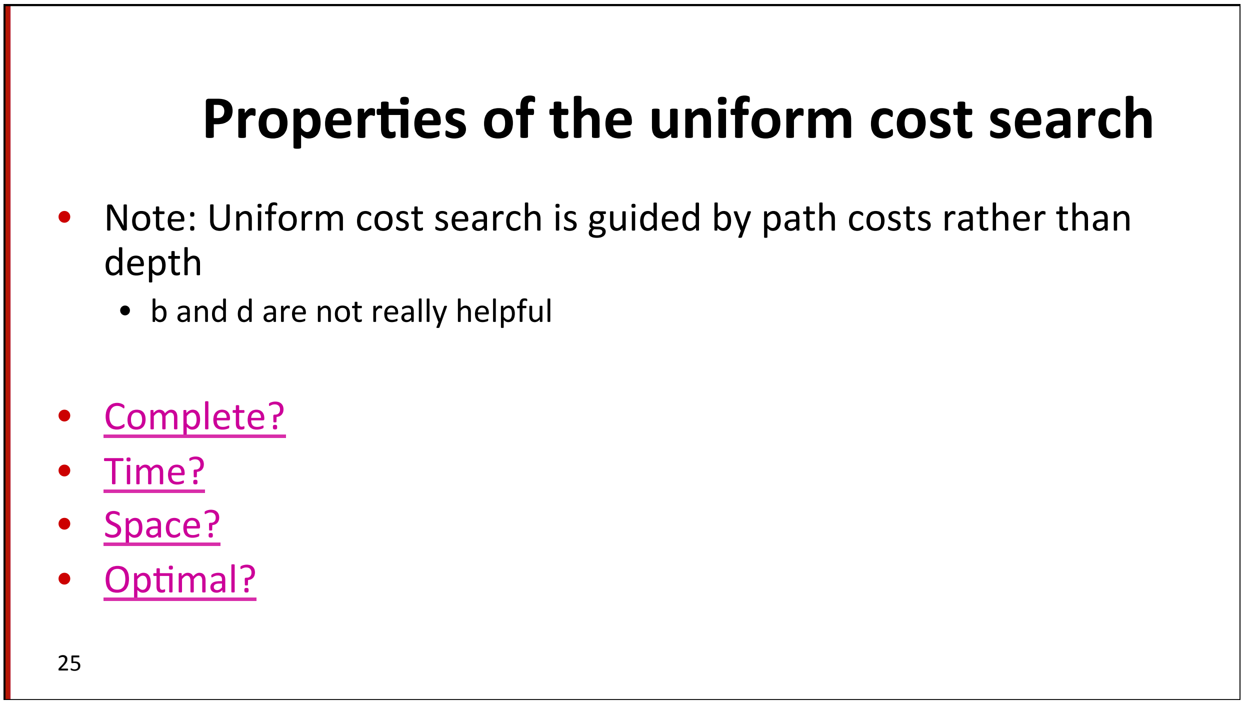
- Optimal? Yes (if cost = 1 per step)

Uniform-cost search

- Expand least-cost unexpanded node
  - Lowest path cost: g(n)
- Implementation:
  - *frontier* = queue ordered by path cost (priority queue)
- Equivalent to breadth-first if step costs are all equal
  - Does not care about the number of steps

Applying Uniform Cost Search

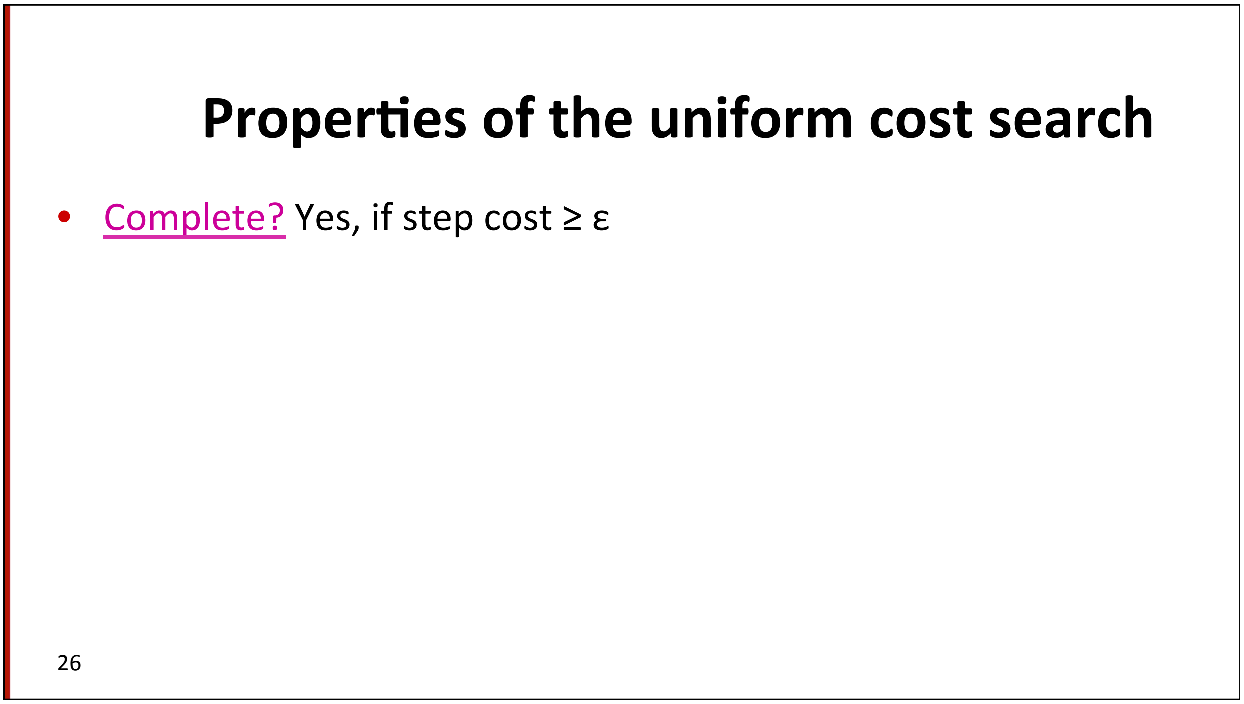




Properties of the uniform cost search

- Note: Uniform cost search is guided by path costs rather than depth
  - b and d are not really helpful

- Complete?
- Time?
- Space?
- Optimal?



Properties of the uniform cost search

- Complete? Yes, if step cost ≥ ε

Properties of the uniform cost search

- Complete? Yes, if step cost ≥ ε
- Time?
- Space?

Properties of the uniform cost search

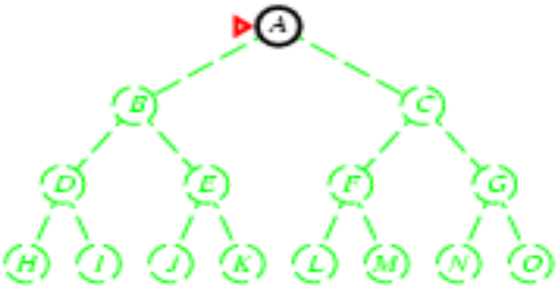
- Complete? Yes, if step cost ≥ ε
- Time? $O(b^{ceiling(C^{*}/\varepsilon)})$ where C^{*} is the cost of the optimal solution
  - Can be much larger than b^{d}
- Space? $O(b^{ceiling(C^{*}/\varepsilon)})$
- Optimal?

Properties of the uniform cost search

- Complete? Yes, if step cost ≥ ε
- Time? $O(b^{ceiling(C^*/\varepsilon)})$ where C^* is the cost of the optimal solution
- Space? $O(b^{ceiling(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of *g(n)*

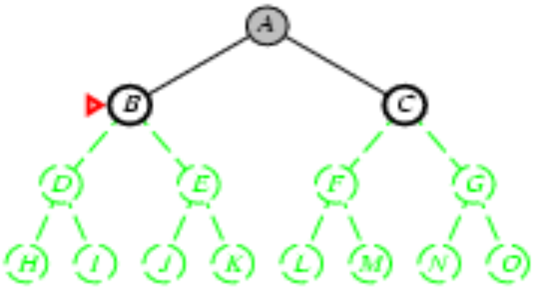
Depth-first search (DFS)

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



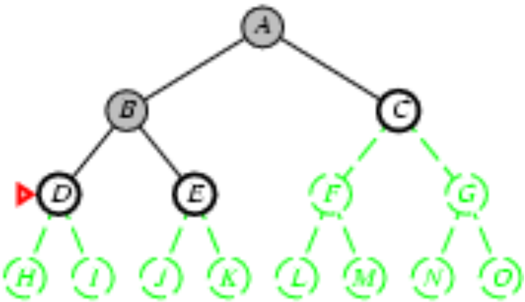
Depth-first search (DFS)

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



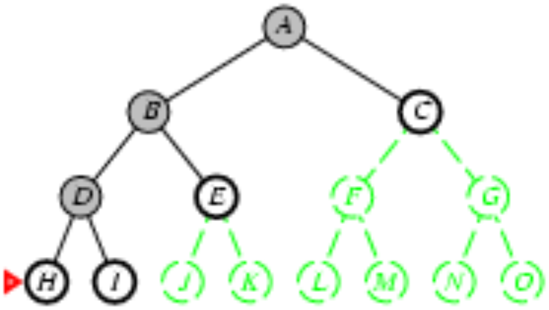
Depth-first search (DFS)

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



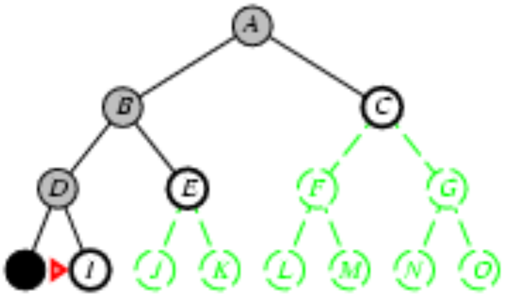
Depth-first search (DFS)

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



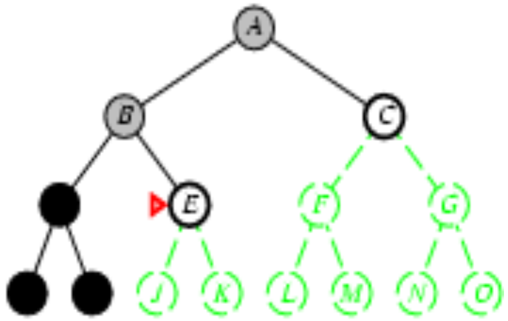
Depth-first search (DFS)

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



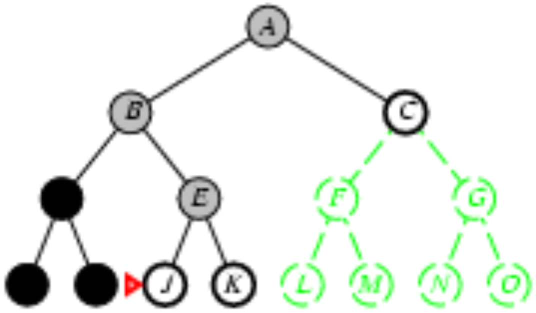
Depth-first search (DFS)

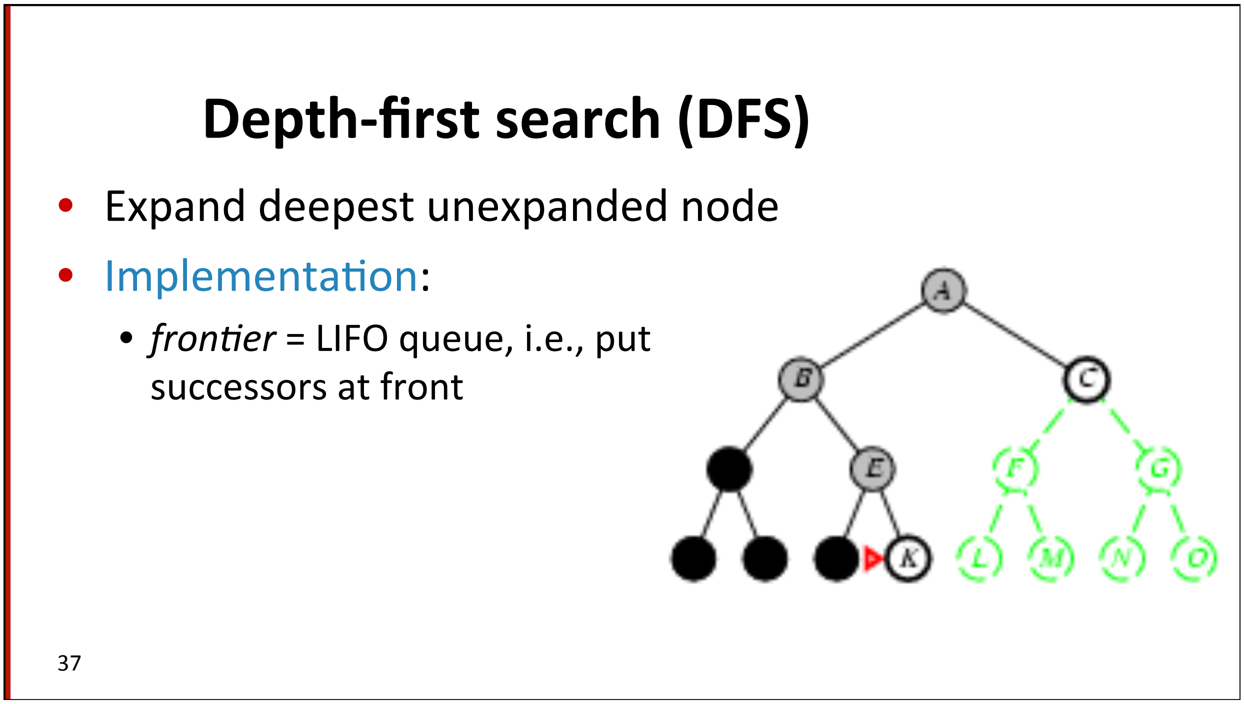
- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



Depth-first search (DFS)

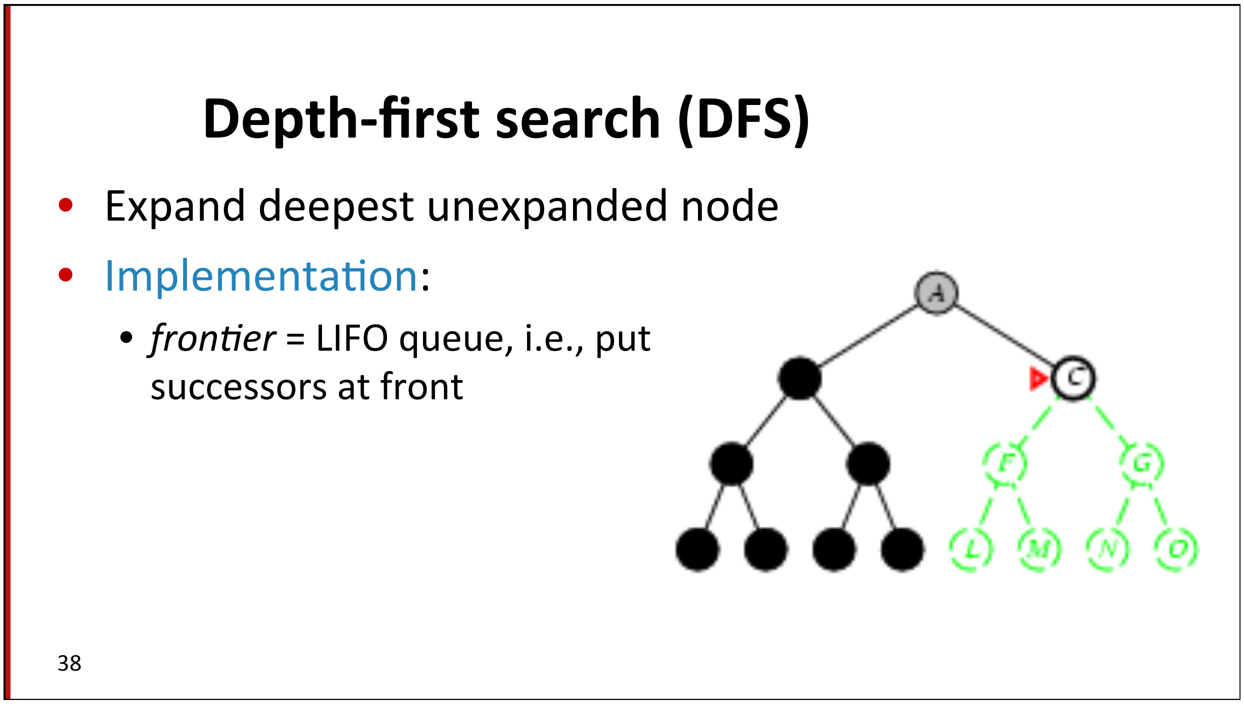
- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front





Depth-first search (DFS)

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

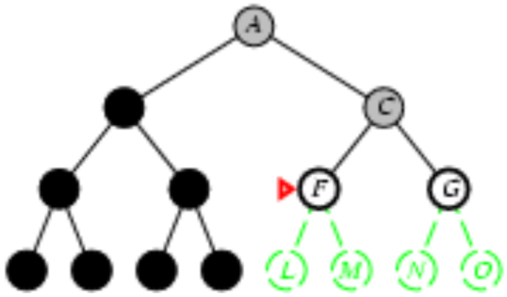


Depth-first search (DFS)

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

Depth-first search (DFS)

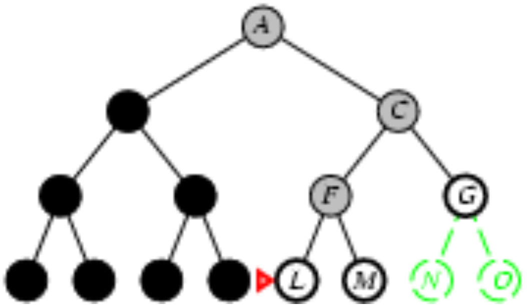
- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



39

Depth-first search (DFS)

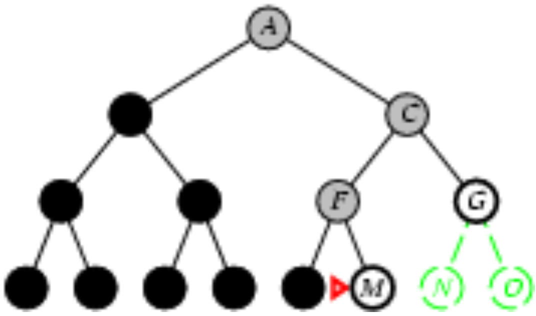
- Expand deepest unexpanded node
- Implementation:
  - *frontier* = LIFO queue, i.e., put successors at front



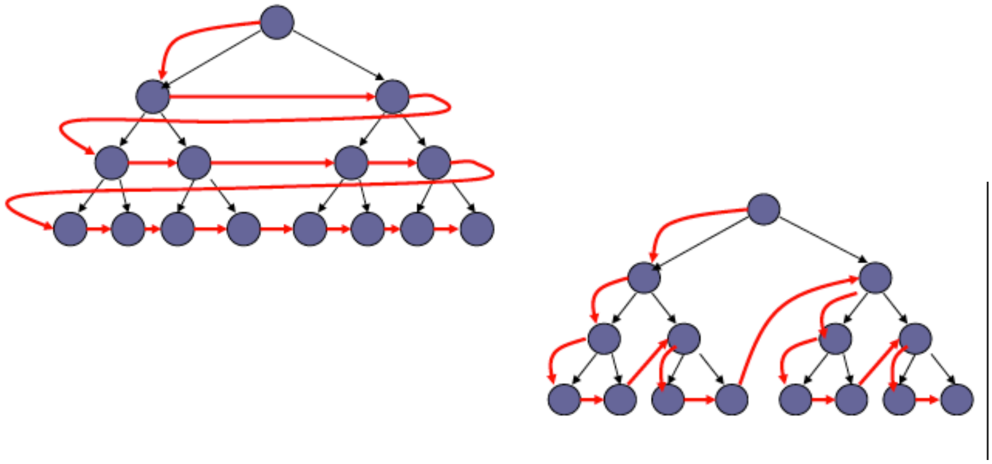
40

Depth-first search (DFS)

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

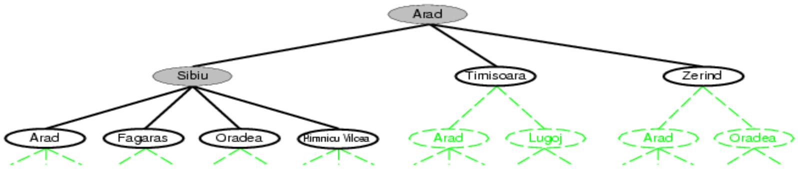


BFS and DFS in a shot



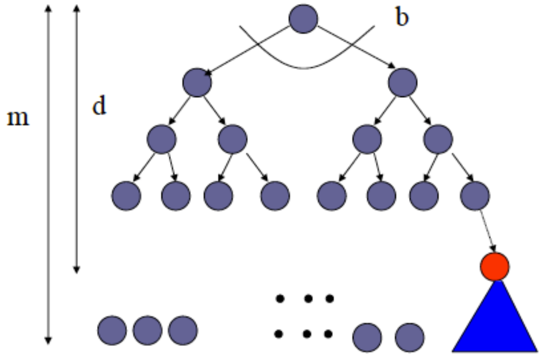
Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops



  - Modify to avoid repeated states along path
    → complete in finite spaces

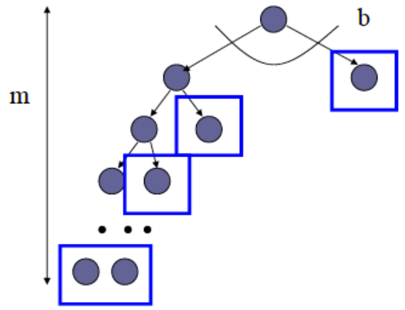
Time Complexity of DFS



depth	number of nodes
0	1
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
•	• •
d	2^d
•	• •
m	2^m-2^{m-d}

Complexity: $O(b^m)$

Space complexity of DFS

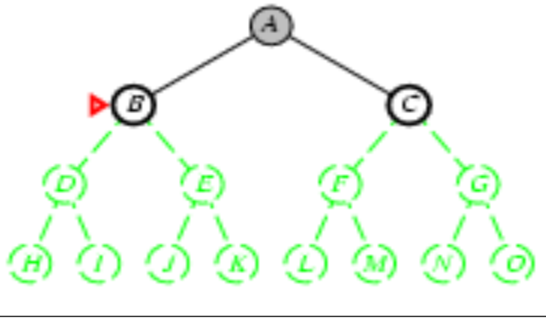


depth	number of nodes kept
0	0
1	1=(b-1)
2	1= (b-1)
3	1 =(b-1)
• • •	
m	2=b

Complexity: $O(bm)$

Properties of depth-first search

- Time? *O(b^m)*
- Space? *O(bm),* i.e., linear space!
- Optimal? No



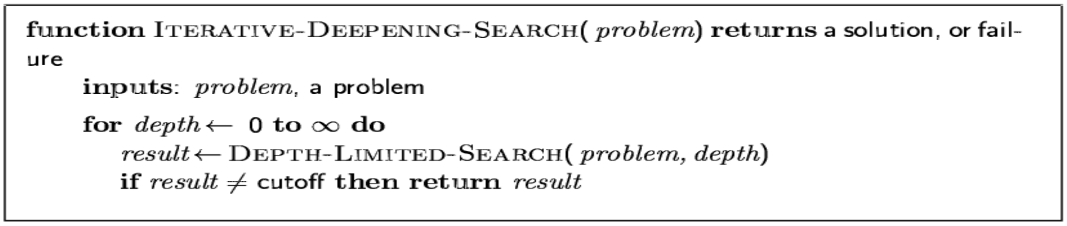
Depth-limited search

- Problem: DFS fails in infinite state spaces (unbounded tree)
- Depth limited search= depth-first search with depth limit
  - i.e., nodes at depth *l* have no successors
- Knowledge of the problem can provide leads about the maximum depth
  - *20 cities ➔ l = 19?*
    - *Map ➔ l = 9*

47

Iterative deepening search

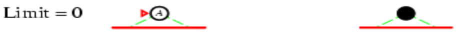
- Combines some benefits of BFS and DFS
  - Complete if branching factor is finite
  - Linear space: *O(bd)*
  - Optimal



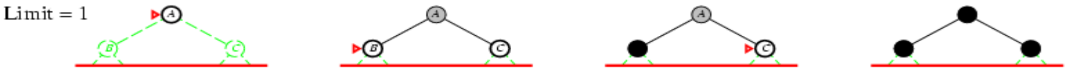
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or fail-
ure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

48

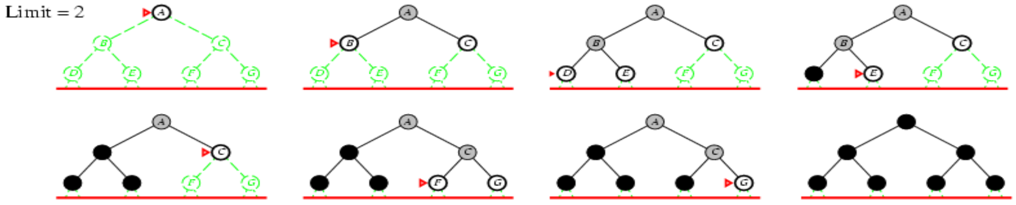
Iterative deepening search *l* =0



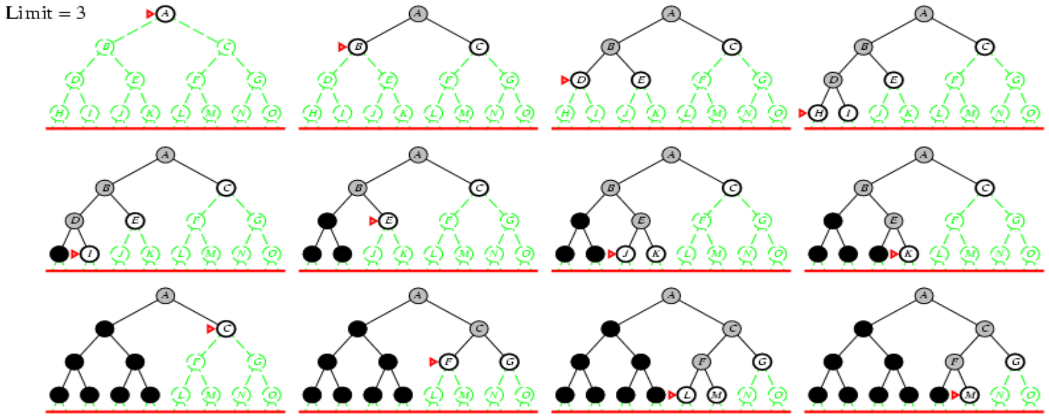
Iterative deepening search *l* =1

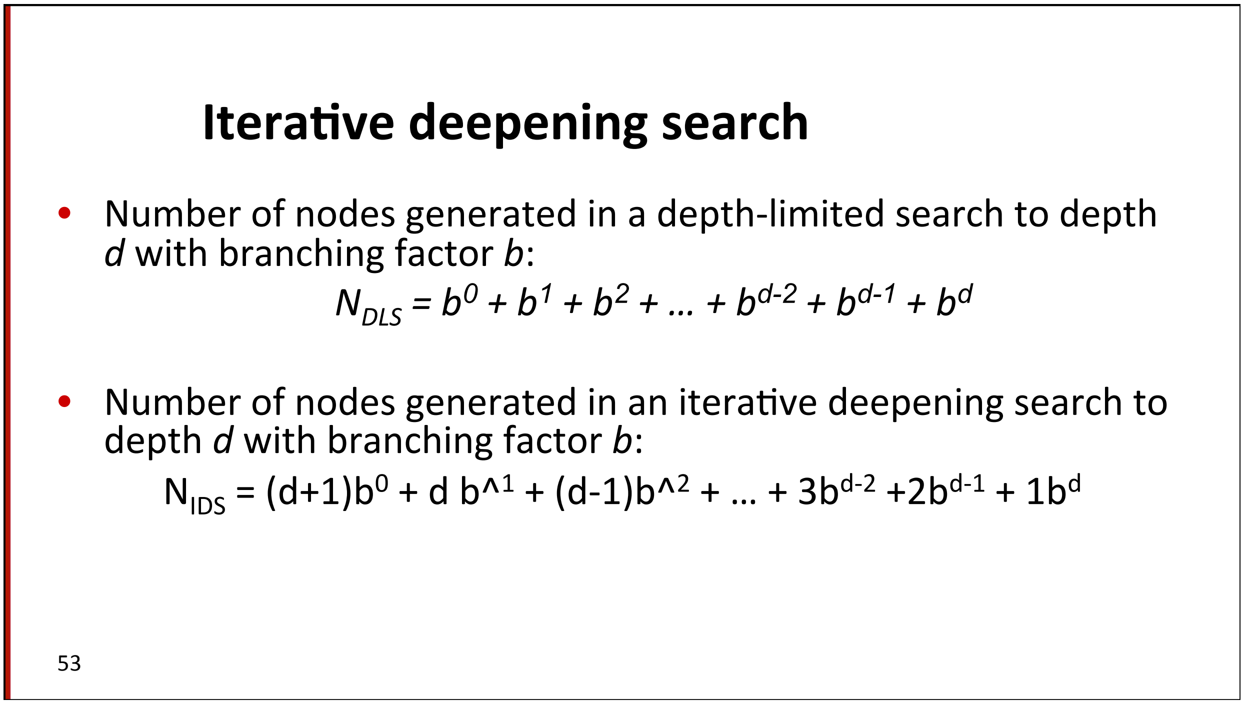


Iterative deepening search *l* =2



Iterative deepening search *l* =3





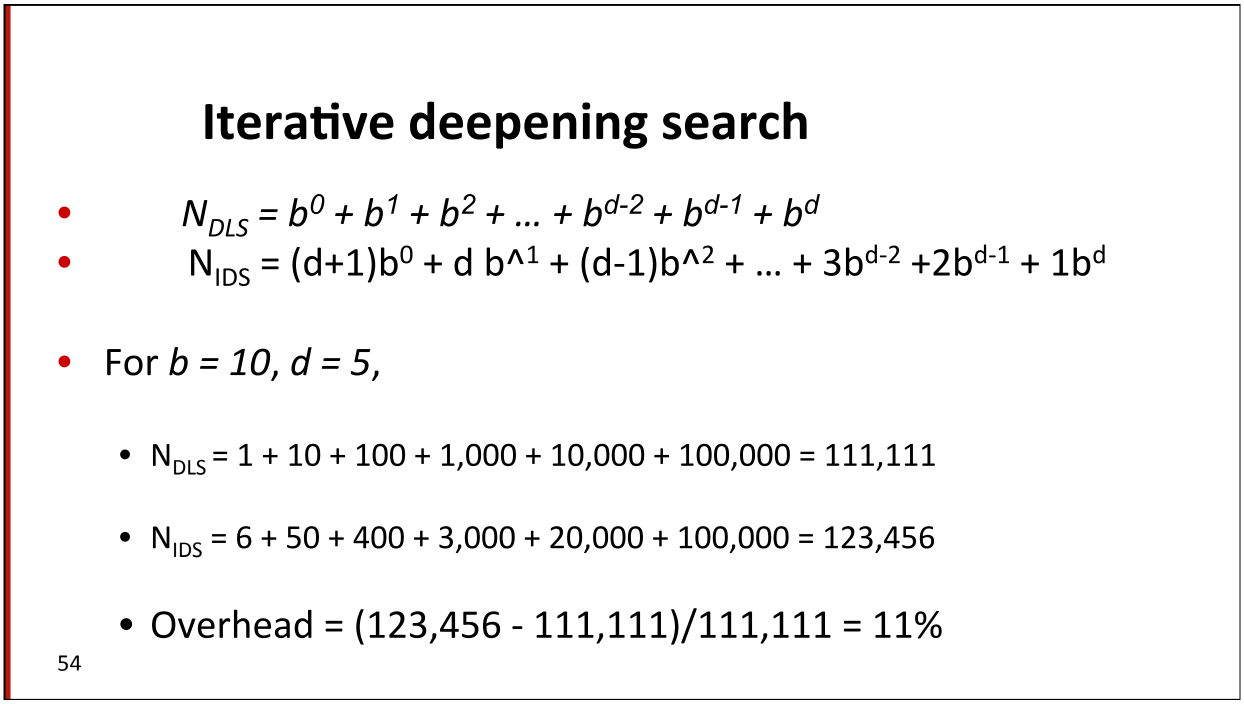
Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$\mathrm{N}_{\mathrm{IDS}} = (\mathrm{d}+1)\mathrm{b}^{0} + \mathrm{d}\ \mathrm{b}\text{^}^{1} + (\mathrm{d}-1)\mathrm{b}\text{^}^{2} + \ldots + 3\mathrm{b}^{\mathrm{d}-2} + 2\mathrm{b}^{\mathrm{d}-1} + 1\mathrm{b}^{\mathrm{d}}$$



Iterative deepening search

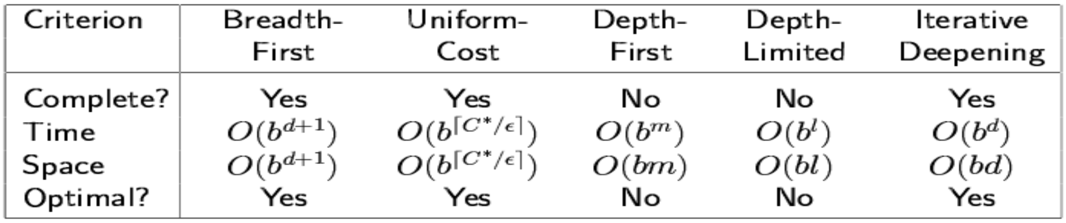
- $N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$
- $\mathrm{N}_{\mathrm{IDS}} = (\mathrm{d}+1)\mathrm{b}^{0} + \mathrm{d}\ \mathrm{b}\text{^}^{1} + (\mathrm{d}-1)\mathrm{b}\text{^}^{2} + \ldots + 3\mathrm{b}^{\mathrm{d}-2} + 2\mathrm{b}^{\mathrm{d}-1} + 1\mathrm{b}^{\mathrm{d}}$

- For *b = 10*, *d = 5*,
  - $\mathrm{N}_{\mathrm{DLS}}$ = 1 + 10 + 100 + 1,000 + 10,000 + 100,000 = 111,111
  - $\mathrm{N}_{\mathrm{IDS}}$ = 6 + 50 + 400 + 3,000 + 20,000 + 100,000 = 123,456
  - Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

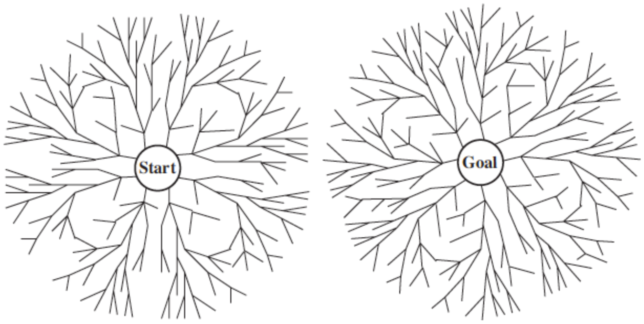
- Complete? Yes
- Time? $(d+1)b^0 + d\ b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1
- **Iterative deepening is the proffered search strategy when the search space is large and depth of the solution is unknown**

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

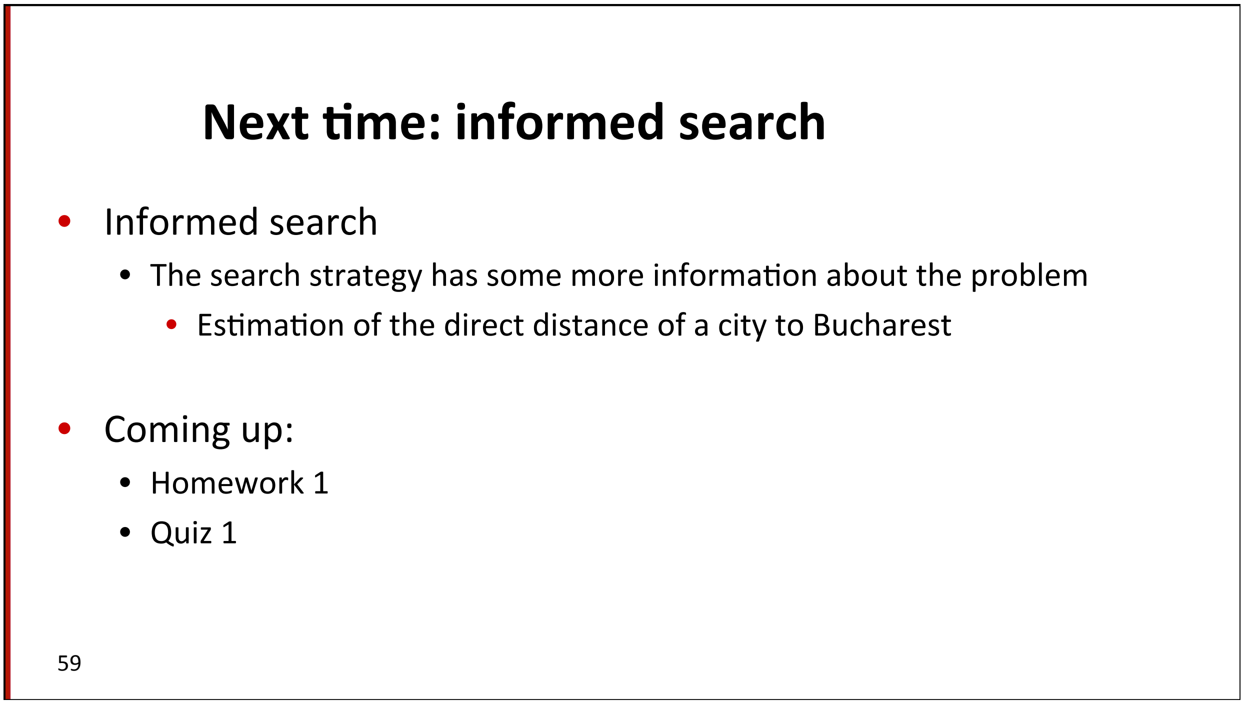
Bidirectional search



- Motivation: b ^ d/2 is much less than b ^ d
- Search from start and goal
  - Check if the frontiers of the two

Bidirectional search

- Can be useful when the goal state is clear
- Difficult for abstract problems like 8-queens



Next time: informed search

- Informed search
  - The search strategy has some more information about the problem
    - Estimation of the direct distance of a city to Bucharest
- Coming up:
  - Homework 1
  - Quiz 1