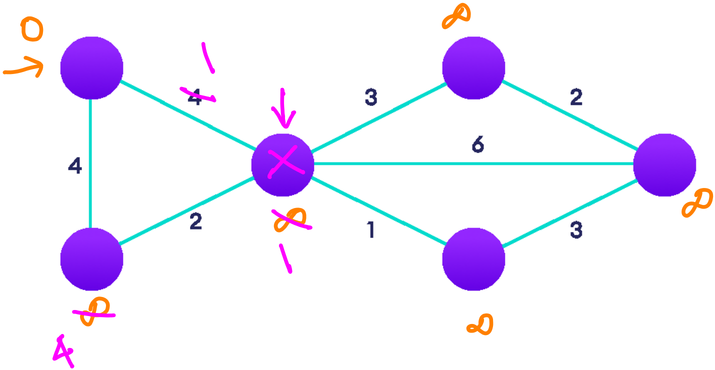
Complexity exponential in d

- Exponential complexity $O(b^d)$ leads to the following growth in time and space requirements:

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

Figure 3.13 Time and memory requirements for breadth-first search. The numbers shown assume branching factor $b = 10$; 1 million nodes/second; 1000 bytes/node.

- Dijkstra's algorithm



Breadth-first Search

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Uniform cost search

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

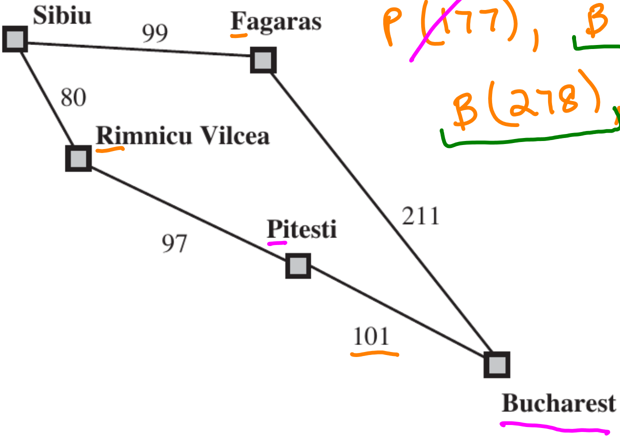
Uniform cost search

~~S(0)~~

~~R(80)~~, ~~F(99)~~

~~P(177)~~, B(~~310~~) 278

B(278)



Uniform cost search

Uniform cost search

- Complete and optimal?

Uniform cost search

- Complete and optimal?
- Time and Space complexity?

Uniform cost search

b^d

- Complete and optimal?
- Time and Space complexity?
  - $O(b^{1+\lfloor C^*/\epsilon \rfloor})$

$\underbrace{1 + \lfloor C^*/\epsilon \rfloor}$

C^*

ϵ

d

$O(b^d)$

Infinite State Space: Knuth's Problem

0 3! 6

The problem definition :

- **States**: Positive numbers.
- **Initial state**: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- **Transition model**: As given by the mathematical definitions of the operations.
- **Goal test**: State is the desired positive integer.

Infinite State Space: Knuth's Problem

The problem definition :

- **States**: Positive numbers.
- **Initial state**: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- **Transition model**: As given by the mathematical definitions of the operations.
- **Goal test**: State is the desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{\underbrace{(4!)!}}}}}} \right\rfloor = 5$$

2

Infinite State Space: Knuth's Problem

The problem definition:

- **States**: Positive numbers.
- **Initial state**: 4.
- **Actions**: Apply factorial, square root, or floor operation (factorial for integers only).
- **Transition model**: As given by the mathematical definitions of the operations.
- **Goal test**: State is the desired positive integer.

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

$$(4!)! = 24! = 620,448,401,733,239,439,360,000$$

Depth First Search

b * b

b * d

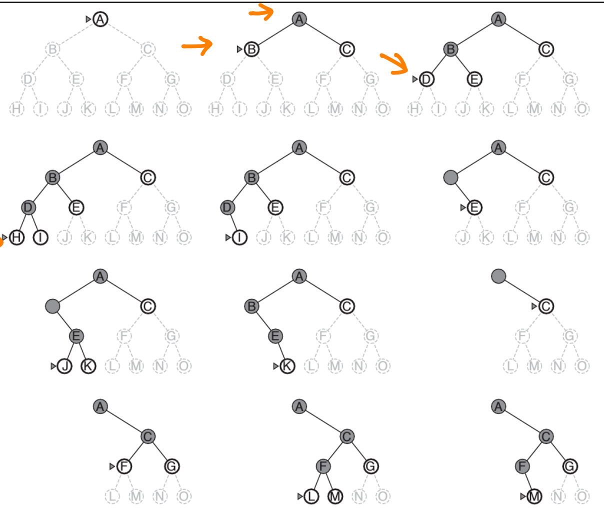


Figure 3.16 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Depth First Search

- Complete?

Depth First Search

- Complete?
- Optimal?

Depth First Search

- Complete?
- Optimal?
- Time complexity?

$m \leftarrow$

Depth First Search

- Complete?
- Optimal?
- Time complexity? $O(b^m)$

Depth First Search

- Complete?
- Optimal?
- Time complexity? $O(b^m)$
- Space compexity?

Depth First Search

- Complete?
- Optimal?
- Time complexity? $O(b^m)$
- Space compexity? $O(bm)$

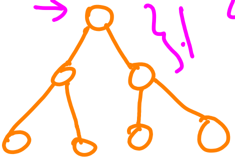
Iterative Deepening Search

function ITERATIVE-DEEPENING-SEARCH(*problem*) **returns** a solution, or failure
  for *depth* = 0 **to** ∞ **do**
    result ← DEPTH-LIMITED-SEARCH(*problem*, *depth*)
    if *result* $\neq$ cutoff **then return** *result*

Figure 3.18 The iterative deepening search algorithm, which repeatedly applies depth-limited search with increasing limits. It terminates when a solution is found or if the depth-limited search returns *failure*, meaning that no solution exists.

$\underbrace{O(bd)}$

d



$\underbrace{O(b^d)}$