UNINFORMED SEARCH (FINALE) + INFORMED SEARCH

Lara J. Martin (she/they)
TA: Aydin Ayanzadeh (he)

9/14/2023

CMSC 671

By the end of class today, you will be able to:

1. Assess the limitation of uninformed search
2. Select the right uninformed search algorithm for a given task.

Modified from slides by Dr. Cassandra Kent

RECAP

BFS VS DFS

Breadth-First Search

- **Complete**
- **Optimal**
- $\boldsymbol{O(b^d)}$ **time complexity**
- $O(b^d)$ space complexity

Depth-First Search

- Not complete
- Not optimal
- $O(b^m)$ time complexity
- $\boldsymbol{O(bm)}$ **space complexity**

RECAP

BFS VS UCS

Breadth-First Search

- Complete
- Optimal
- $O(b^d)$ time complexity
- $O(b^d)$ space complexity

Uniform-Cost Search

- Complete
- Optimal
- $O(b^{1+C^*/\epsilon})$ time complexity
- $O(b^{1+C^*/\epsilon})$ space complexity

DEPTH-LIMITED SEARCH

PROPERTIES OF DEPTH-LIMITED SEARCH

- Is depth-limited search complete?
- Is depth-limited search optimal?

Take a couple minutes and try to figure out under what conditions depth-limited search is complete or optimal.

Some terms that may help:

b: branching factor
d: depth of the goal
l: depth limit
m: maximum depth of tree

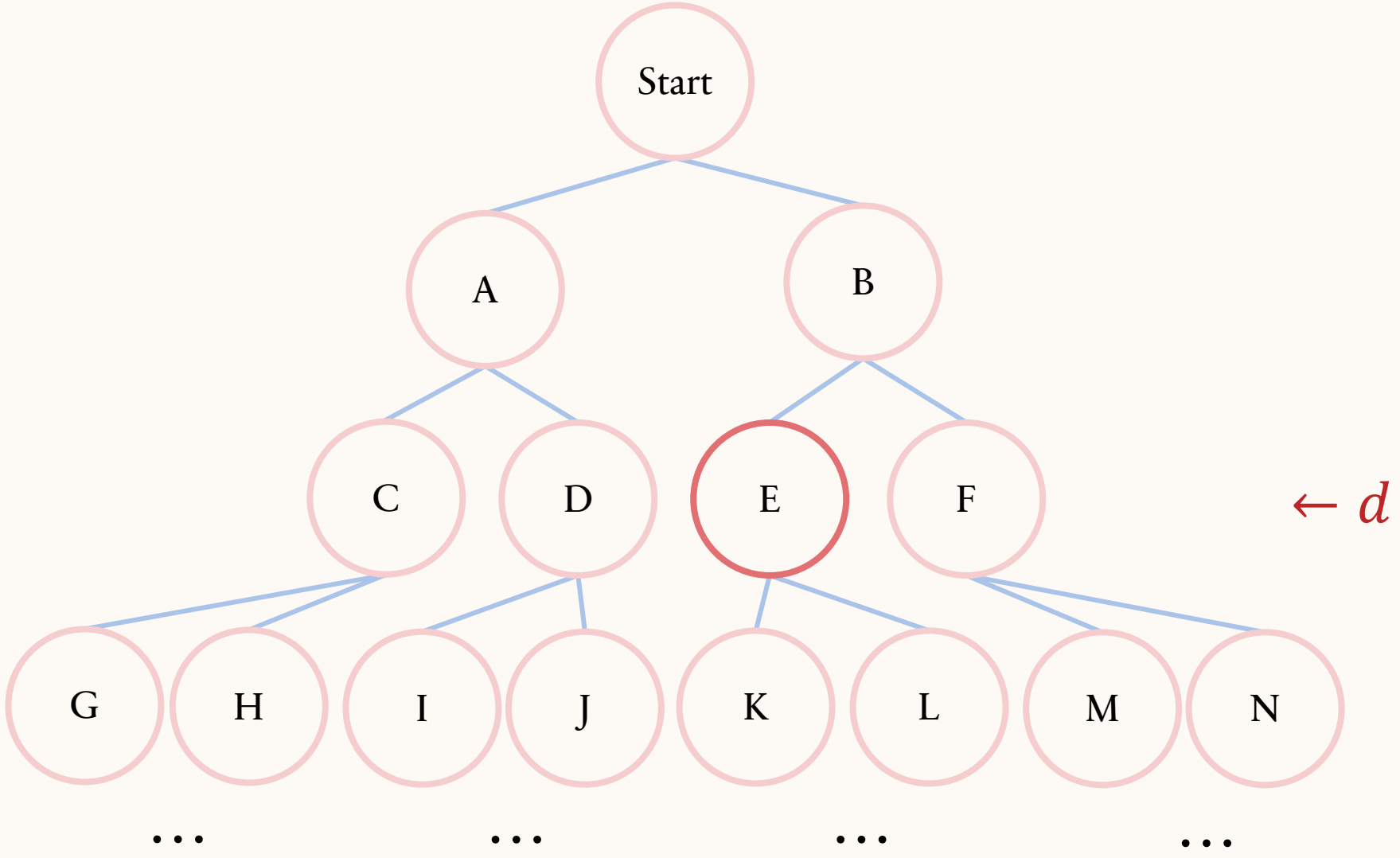
PROPERTIES OF DEPTH-LIMITED SEARCH

When is depth-limited search complete?

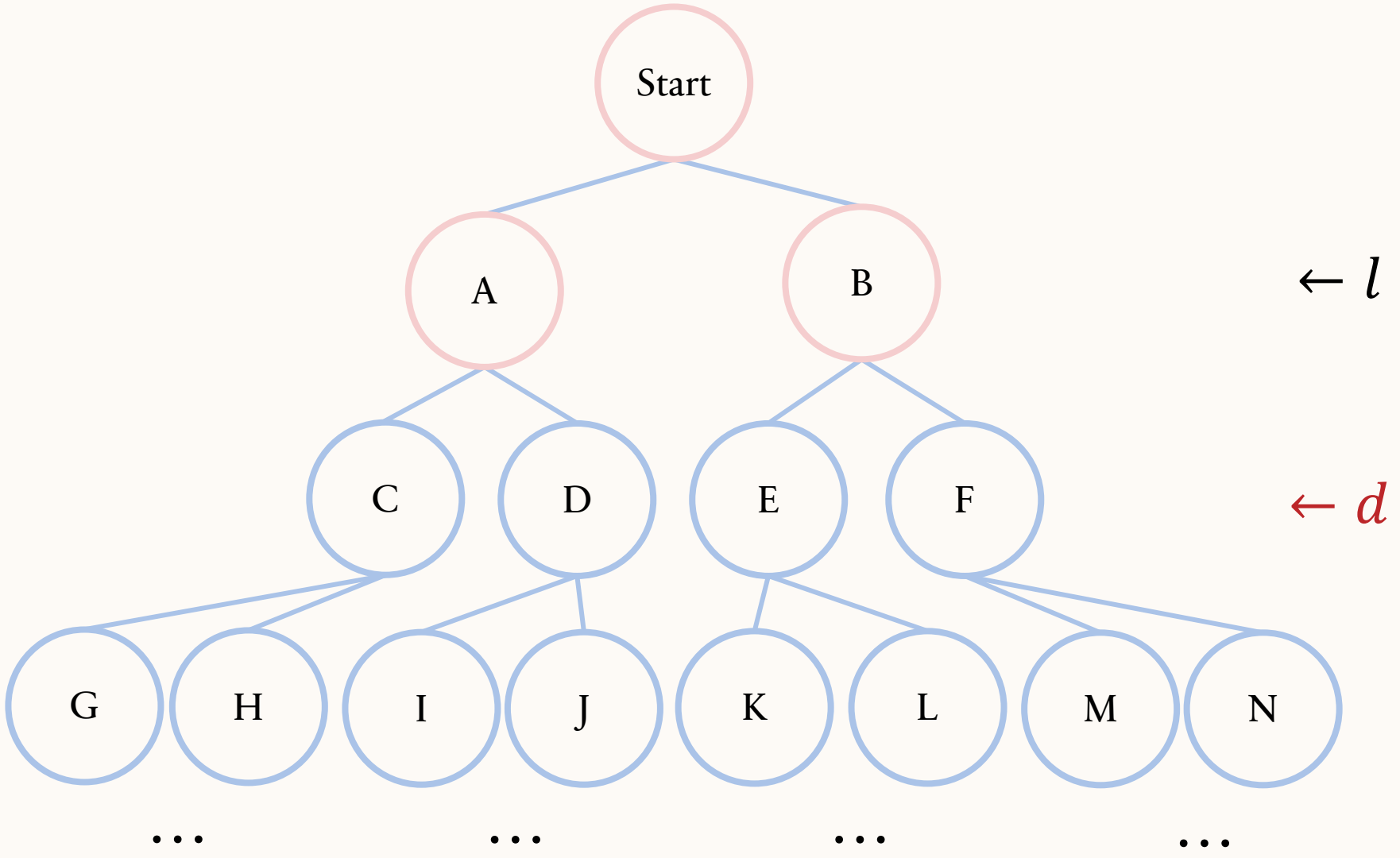
- If the depth limit is **deeper than** the solution depth ($l \geq d$)

b: branching factor
l: depth limit
d: depth of the goal
m: maximum depth of tree

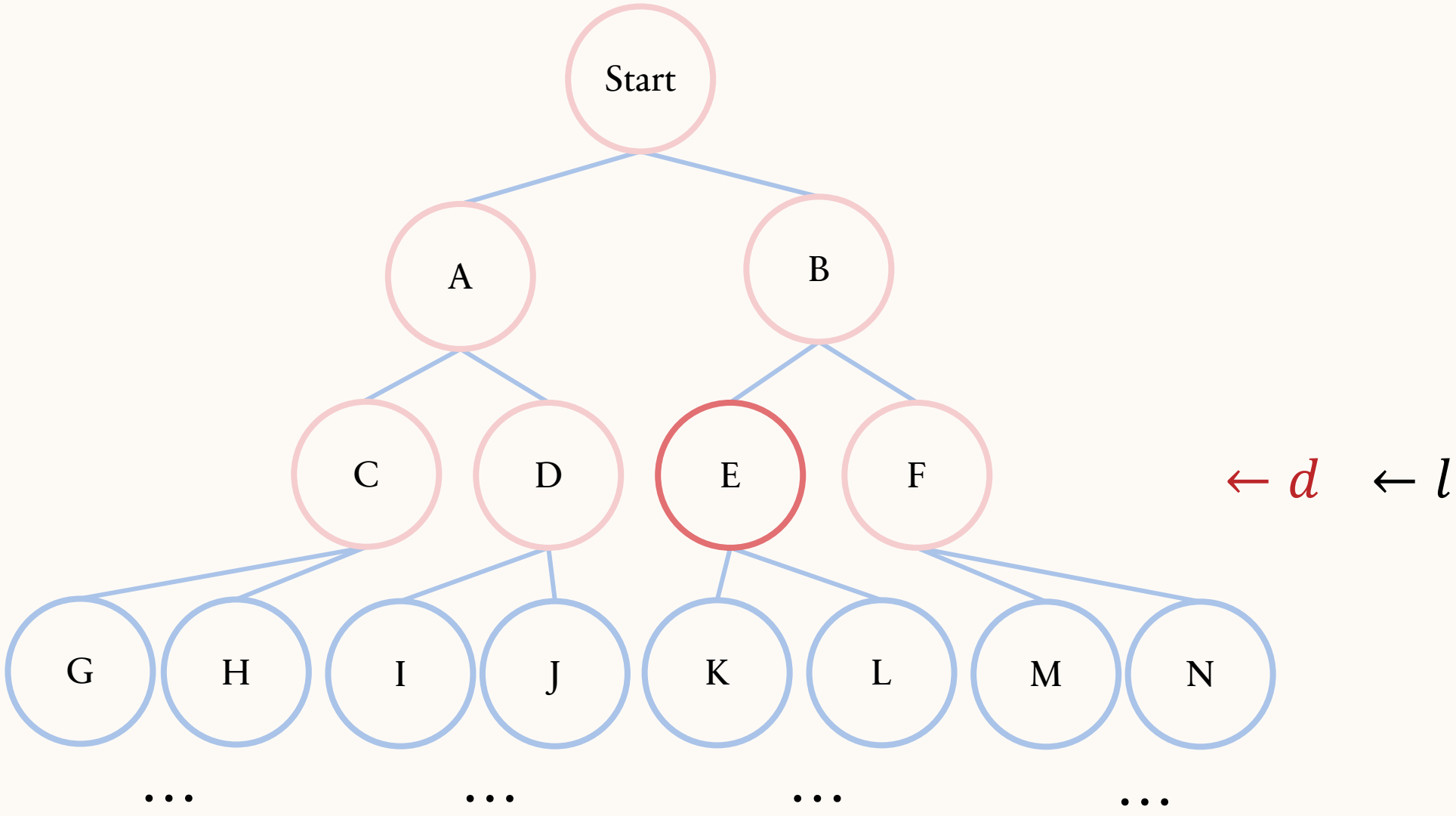
PROPERTIES OF DEPTH-LIMITED SEARCH



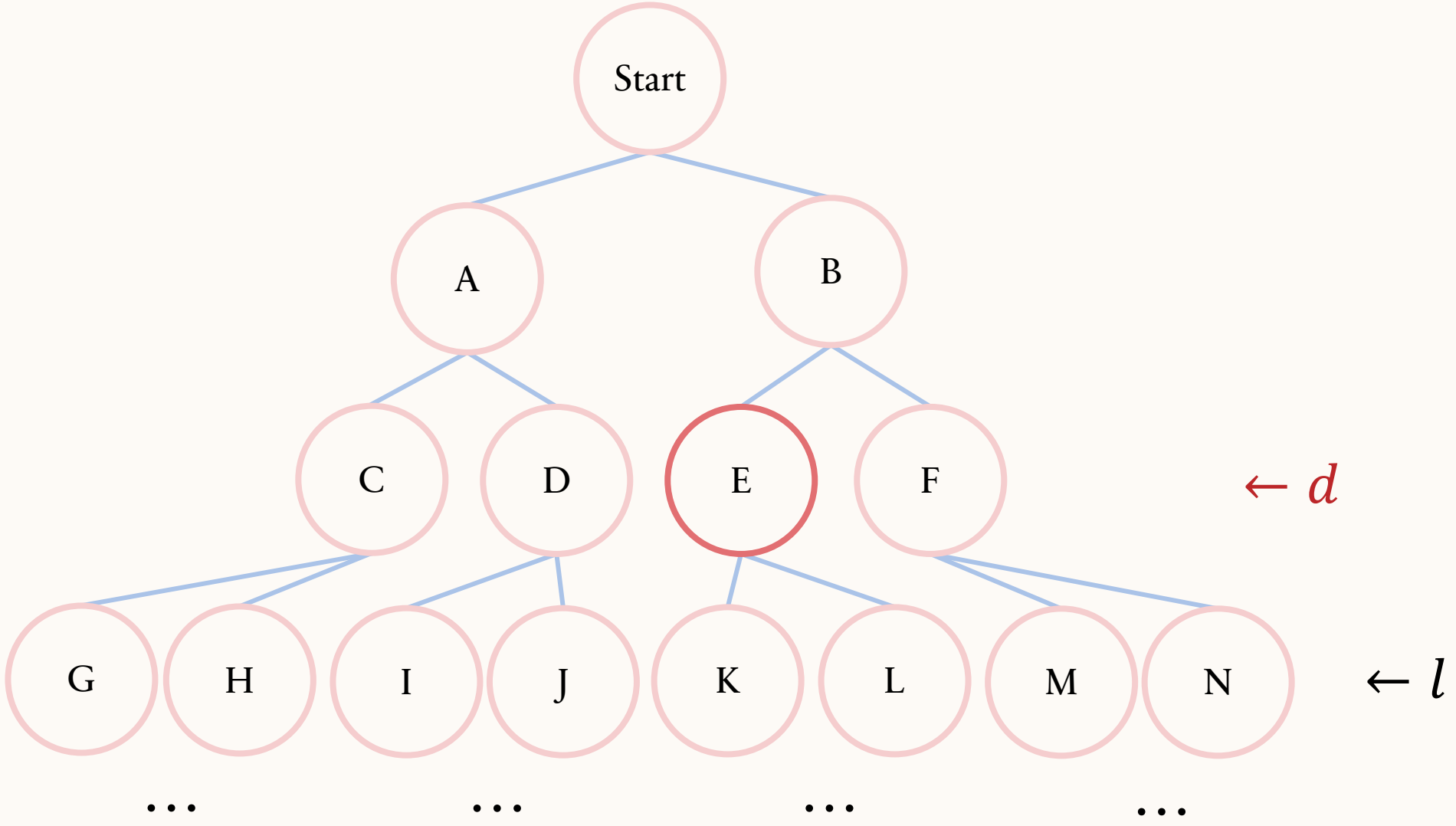
PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH

When is depth-limited search complete?

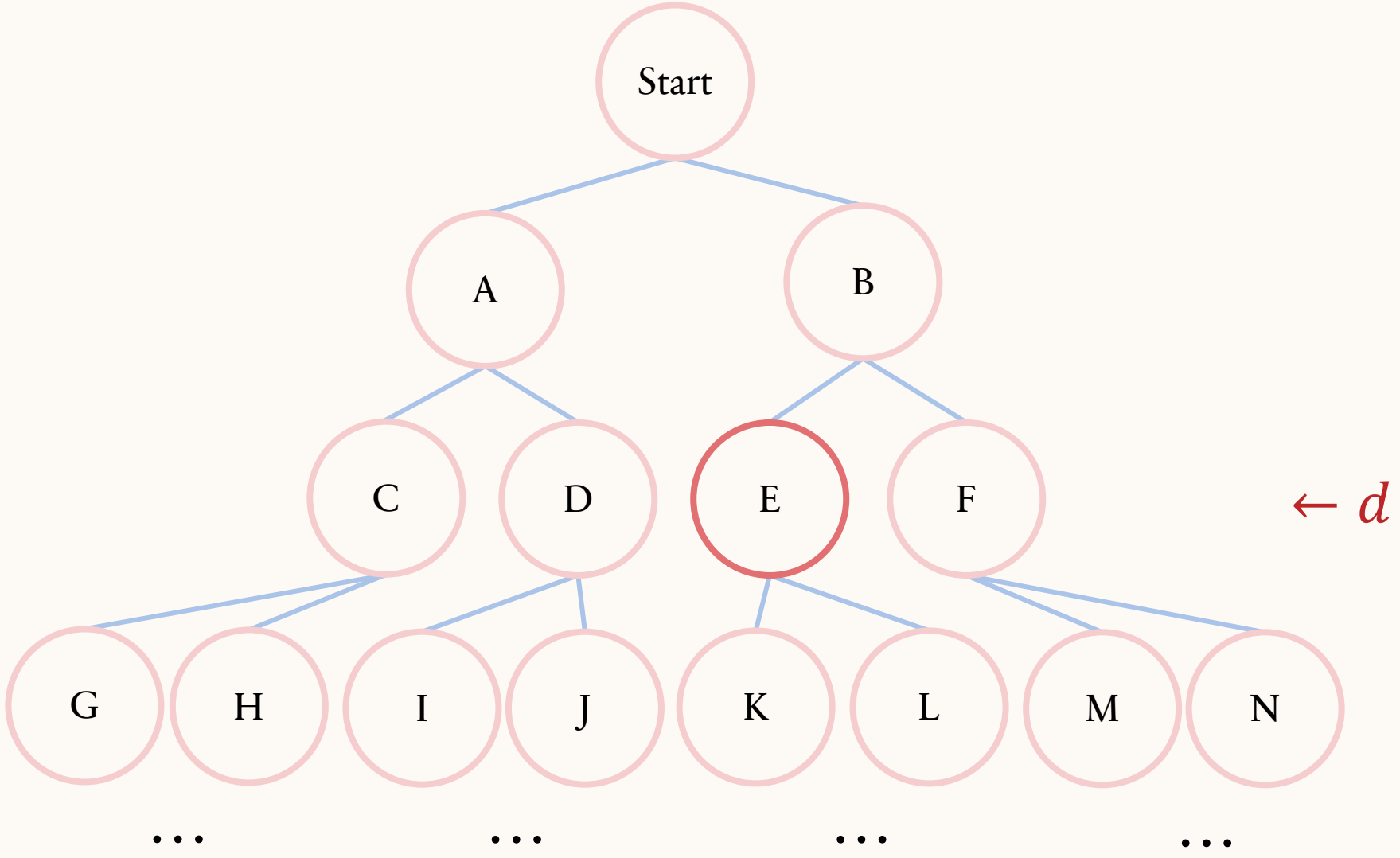
- If the depth limit is **deeper than** the solution depth ($l \geq d$)

When is depth-limited search optimal?

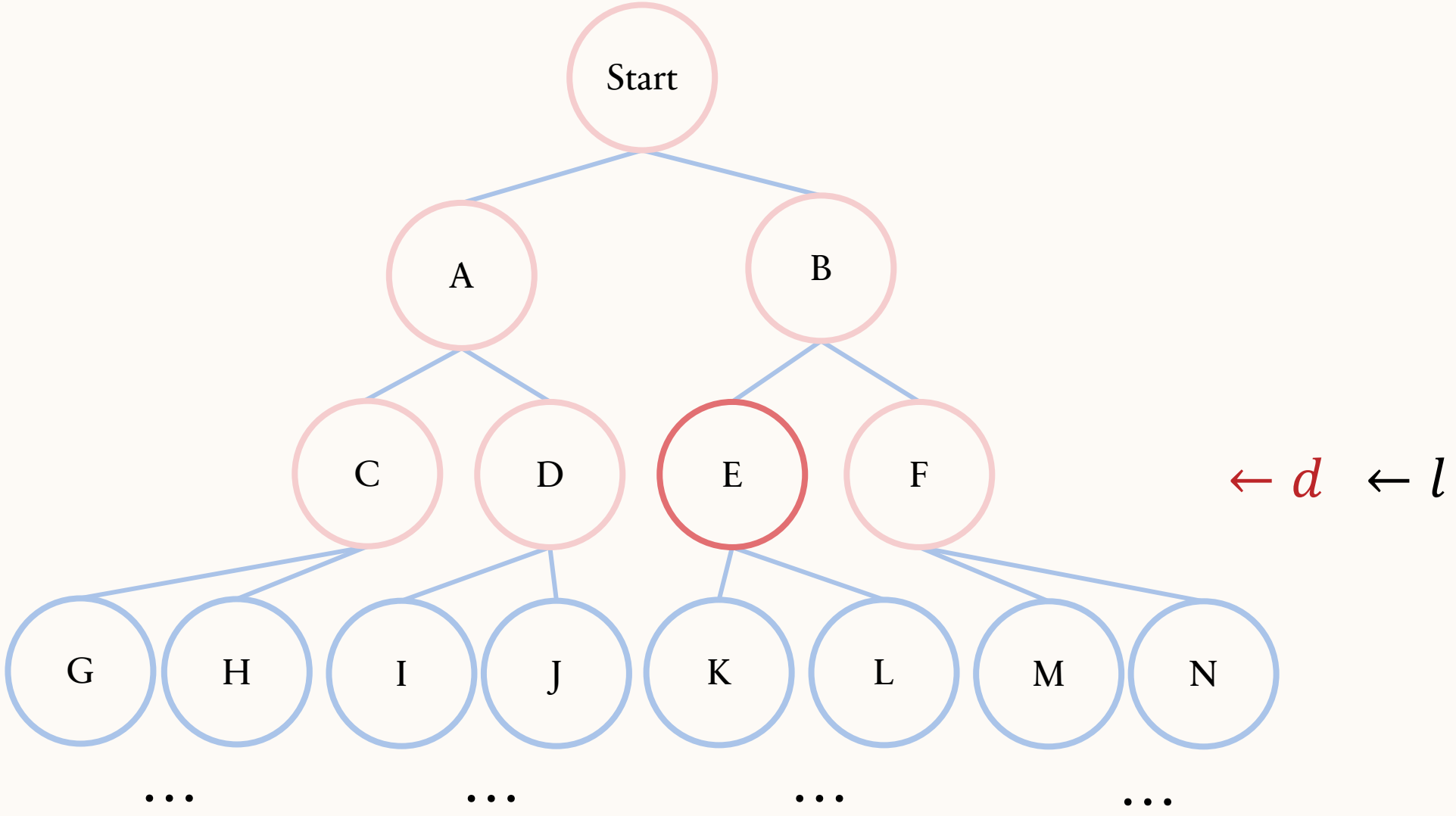
- If the depth limit is **at** the optimal solution depth ($l = d$)

b: branching factor
l: depth limit
d: depth of the goal
m: maximum depth of tree

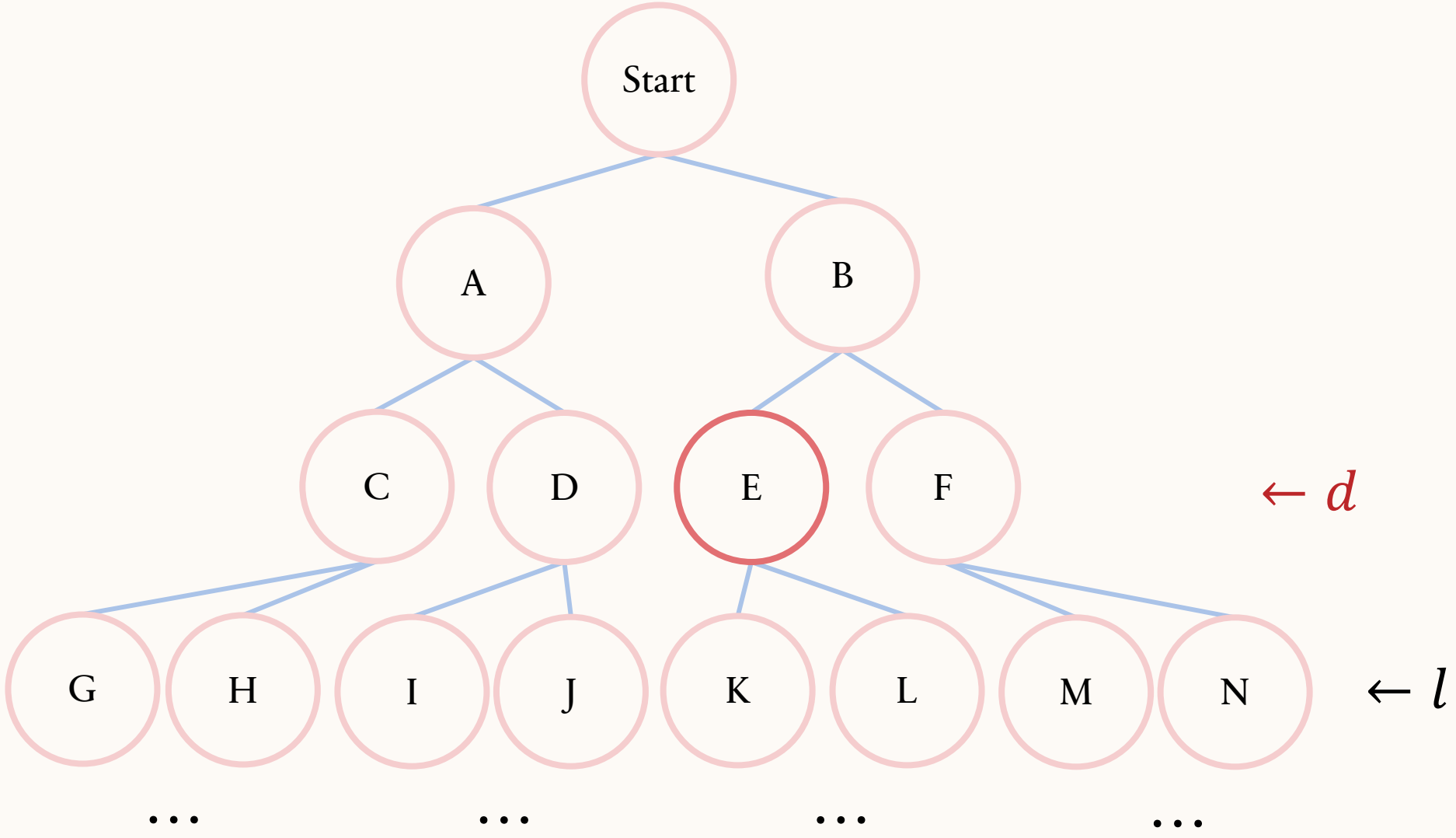
PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH



PROPERTIES OF DEPTH-LIMITED SEARCH

We now have a **complete**, **optimal** version of DFS…

…as long as we know the optimal solution depth.

For most problems, we can not determine the optimal solution depth without (optimally) solving the problem first!

Using Depth-Limited Search, can we ensure that when we *first* reach a goal state, it's at the *lowest possible depth*?

ITERATIVE DEEPENING DEPTH-FIRST SEARCH

Brute-force approach to depth limit selection: run depth-limited search with steadily increasing depth limits until a solution is found

```
function Iterative-Deepening-Search(problem)
  for depth = 0 to ∞ do
    result ← Depth-Limited-Search(problem, depth)
    if result ≠ failure then return result
```

BFS VS DFS: COMPLETENESS

Problem: Brute-force simple password guessing agent

Depth-Limited Search

- States expanded (iteration 1):
  a, b, c
- States expanded (iteration 2):
  a, aa, ab, ac, …, b, ba, bb, bc, …
- States expanded (iteration 2):
  a, aa, aaa, aab, aac, …, b, ba, baa, bab, bac, …

Breadth-First Search

- States expanded:
  a, b, c, …, aa, ab, ac, …, ba, bb, bc, …, ca, cb, …, aaa, aab, aac, … baa, bab, bac, …

Similar behavior, expands all nodes at lowest level before continuing

ITERATIVE DEEPENING DFS: TIME COMPLEXITY

Compared to BFS, Iterative Deepening **re-generates** $l - 1$ nodes at every iteration. Does this affect the time complexity?

We can answer this question using knowledge from Algorithms, specifically Big-O Notation!

ITERATIVE DEEPENING DFS: TIME COMPLEXITY

How many nodes does iterative deepening generate in the worst case?

$$(d)b^1 + (d-1)b^2 + (d-2)b^3 + \cdots + (1)b^d$$

Generate first layer of nodes once for every iteration

Generate second layer of nodes once for every iteration, after the first

Generate first layer of nodes once for every iteration, after the second

Generate final layer of nodes once

ITERATIVE DEEPENING DFS: TIME COMPLEXITY

How many nodes does iterative deepening generate in the worst case?

$$\underbrace{(d)b^1 + (d-1)b^2 + (d-2)b^3}_{\text{Asymptotically negligible}} + \cdots + (1)b^d$$

$$= O(b^d)$$

Equivalent to BFS!

ITERATIVE DEEPENING PROPERTIES

Breadth-First Search

- **Complete**
- **Optimal**
- $O(b^d)$ **time complexity**
- $O(b^d)$ space complexity

Depth-First Search

- Not complete
- Not optimal
- $O(b^m)$ time complexity
- $O(bm)$ **space complexity**

Iterative Deepening

- **Complete**
- **Optimal**
- $O(b^d)$ **time complexity**
- $O(bd)$ **space complexity**

Finally, a complete, optimal, and memory efficient algorithm!

Iterative Deepening DFS is a standard uninformed search method for **large search spaces** with **unknown solution depth.**

(i.e. many problems where we would use search)

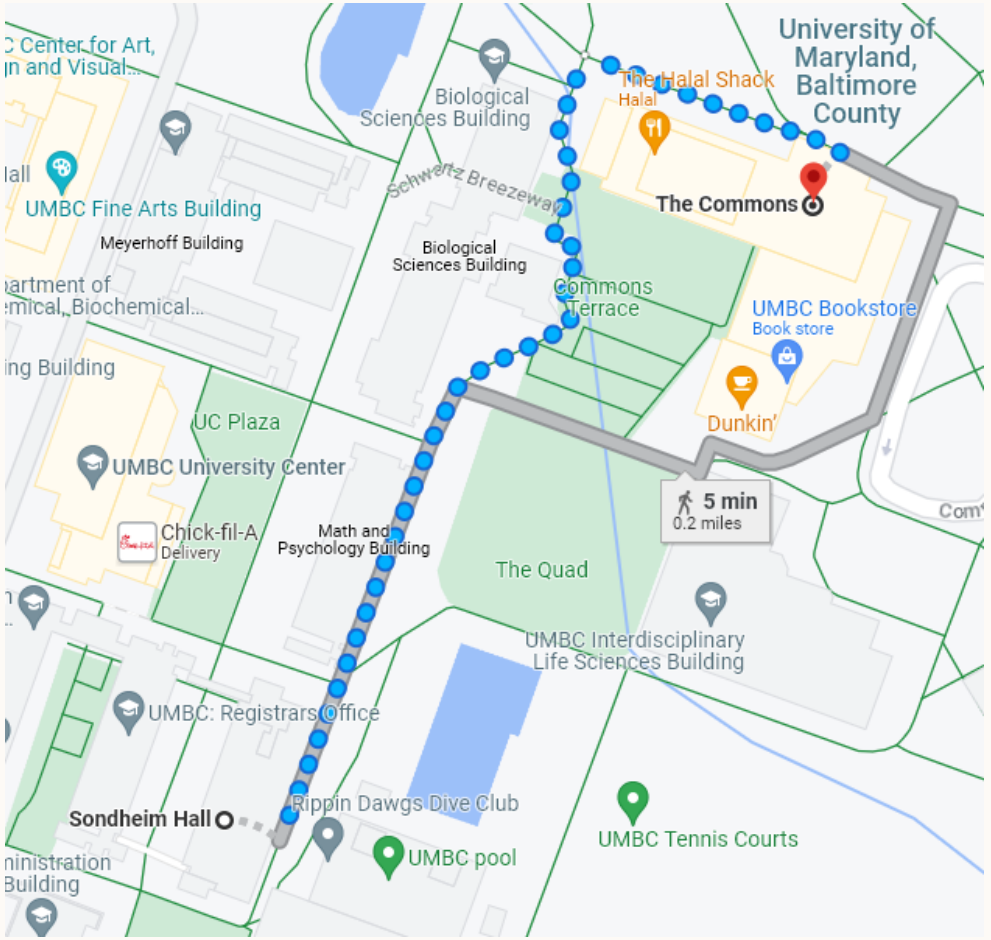
ROUTE FINDING

State: agent location (which path intersection are you at)

Actions: follow path *x* to the next intersection

Goal: find the shortest path to the end location

Algorithm : Uniform-Cost Search, because we're optimizing for route distance instead of minimum number of actions



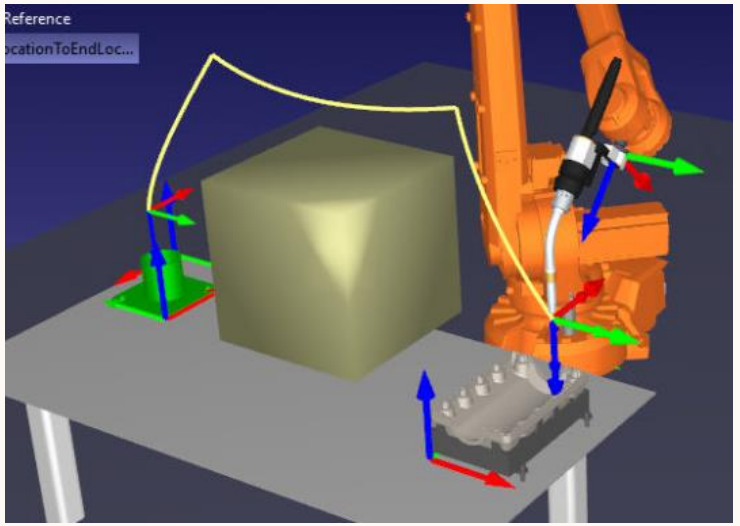
MOTION PLANNING

State: location of obstacles in environment, joint angles of robot

Actions: rotate joints *0*, *1*, *2*, … to angles *i*, *j*, *k*, …

Goal: move to the goal configuration over the shortest distance

Algorithm : Uniform-Cost Search, because we're optimizing for the distance the robot moves instead of minimum number of actions



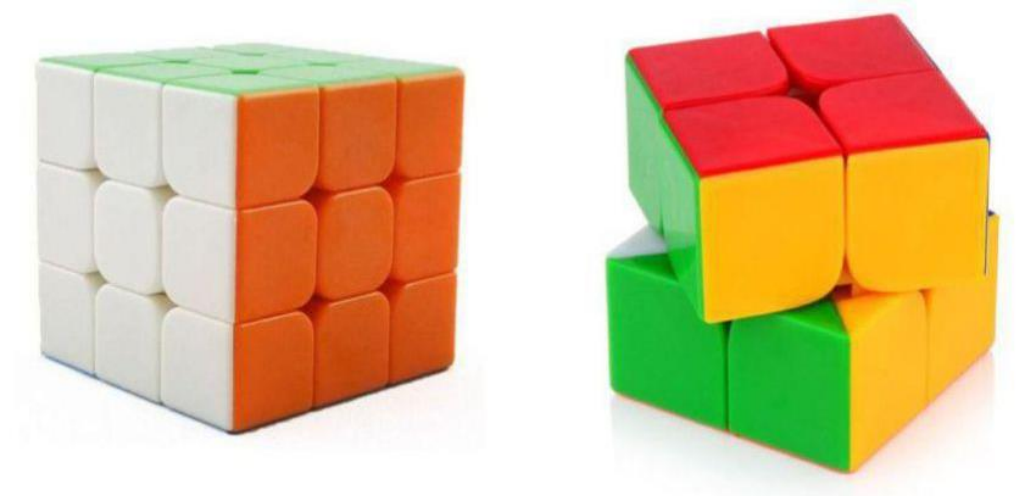
RUBIK'S CUBE

State: configuration of blocks on the cube

Actions: rotate each face clockwise/counterclockwise 90 degrees

Goal: reach solved state in the fewest moves

Algorithm : Iterative Deepening DFS, because we're optimizing for fewest moves in a large search space (i.e. memory efficiency matters), and we don't know the optimal solution depth



CHESS

State: configuration of pieces on the board

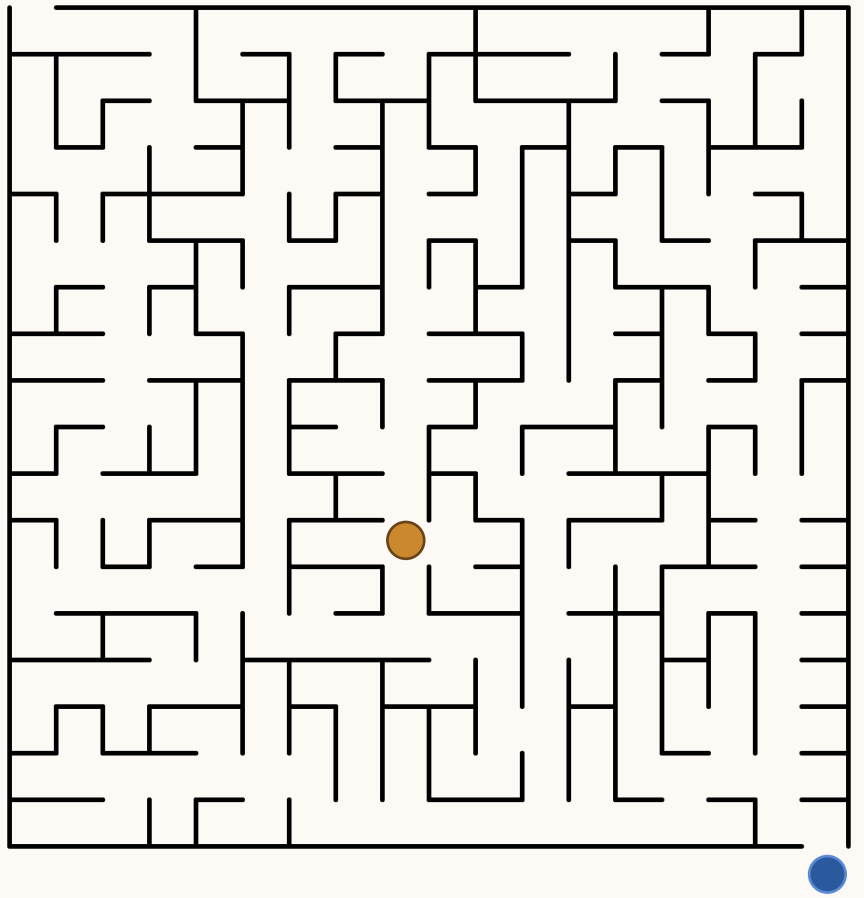
Actions: move 1 piece according to the rules of chess

Goal: checkmate the opponent before they checkmate you

Algorithm: Iterative Deepening DFS, because we're optimizing for fewest moves in a large search space (i.e. memory efficiency matters), and we don't know the optimal solution depth. Note that we will have to make some changes to account for the opponent, which we'll talk about when we get to Adversarial Search!

EFFECTIVENESS OF UNINFORMED SEARCH

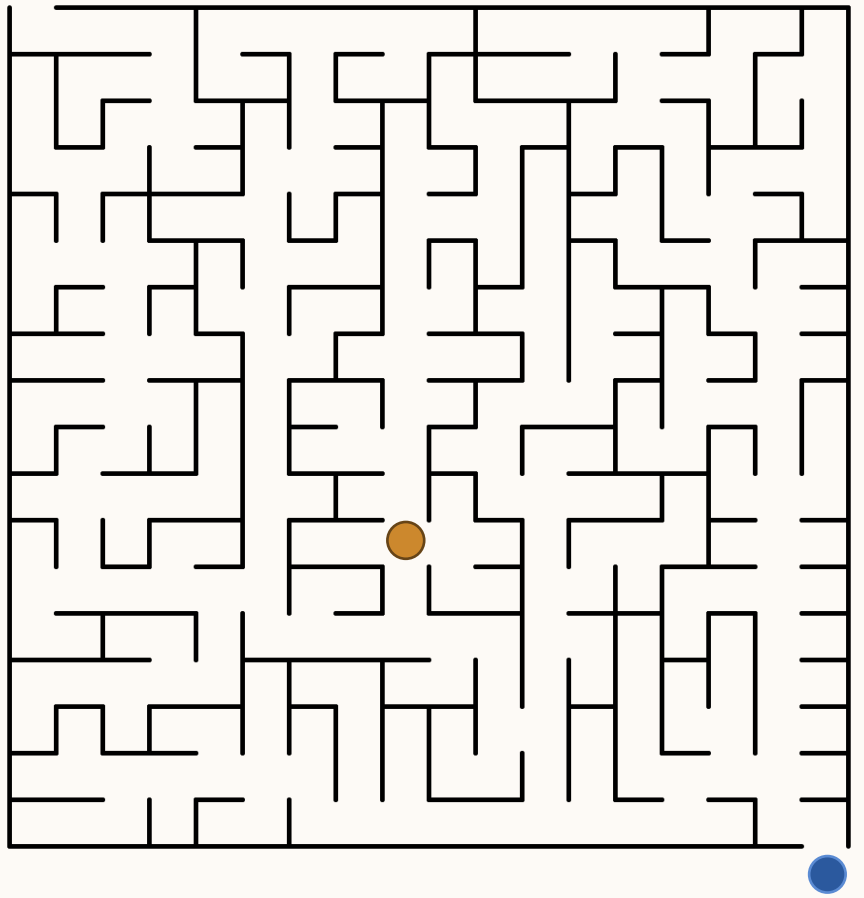
- Where do our search algorithms fit on the acting/thinking humanly/rationally spectrum?
- Consider the maze on the right
  - Find the shortest path from the orange start point to the blue goal point
  - You can move left, right, up, or down
  - Which directions would an uninformed search algorithm consider first?
  - Which directions would ***you***, a human, consider first?



EFFECTIVENESS OF UNINFORMED SEARCH

Levels of AI performance:

- Sub-human: can solve a problem, but not as well as a person
- Human: comparable performance to a human (goal of the Turing test?)
- Super-human: outperforms people on the same problems



EFFECTIVENESS OF UNINFORMED SEARCH

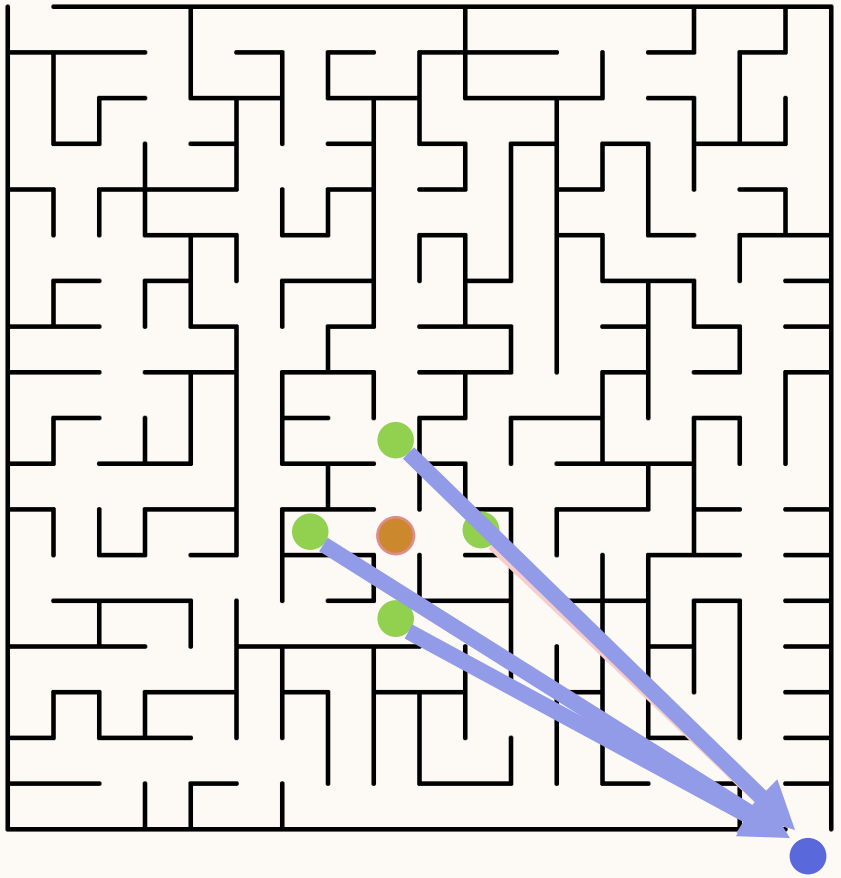
Humans can search some of these toy problems very efficiently, so can we also develop algorithms to solve the same search problems more efficiently?

Informed Search: consider additional information to guide our search to be more efficient; move beyond brute-force techniques

INFORMED SEARCH

INFORMED SEARCH

- down/right states are *closer* to the goal than the up/left states
- Define a function to represent approximate distance to the goal state: a **heuristic function** $h(n)$
  - Example: Euclidean (straight-line) distance
- Use $h(n)$ to select nodes for expansion



BEST-FIRST SEARCH

```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier
```

Store frontier as a priority queue, prioritized by heuristic $h(n)$

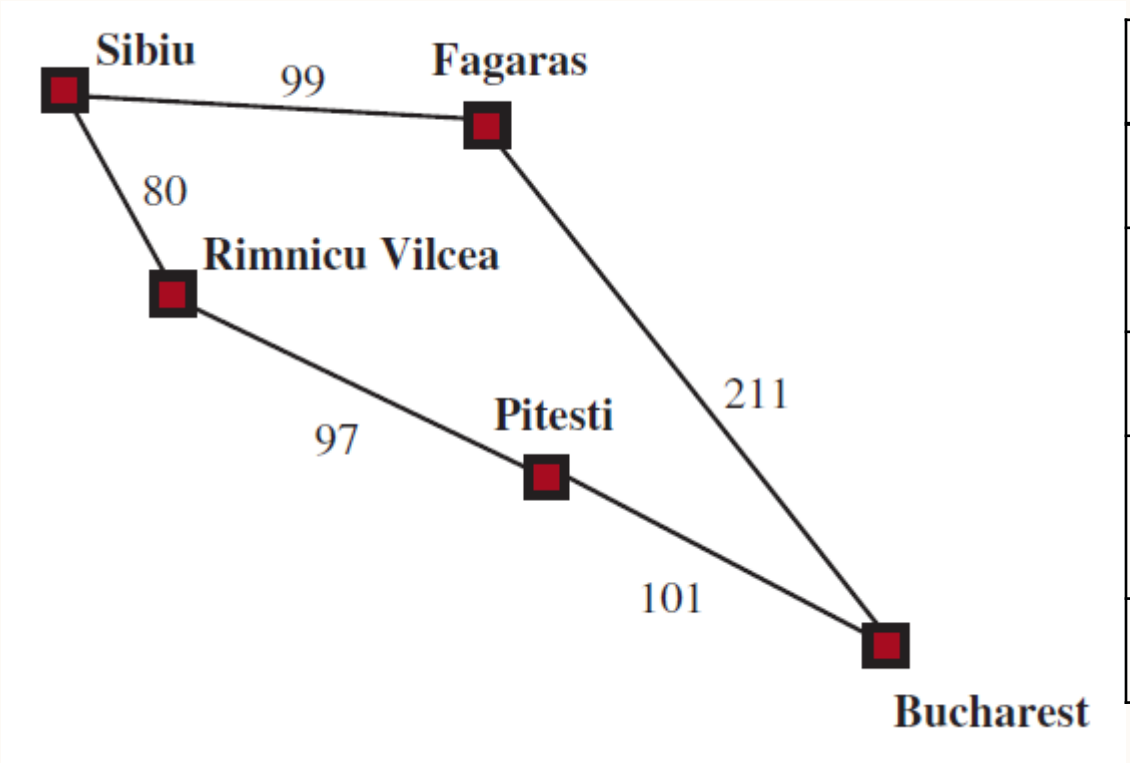
- Heuristic approximates the cost-to-go to reach the goal

BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[Sibiu (253)]

Current node:
none



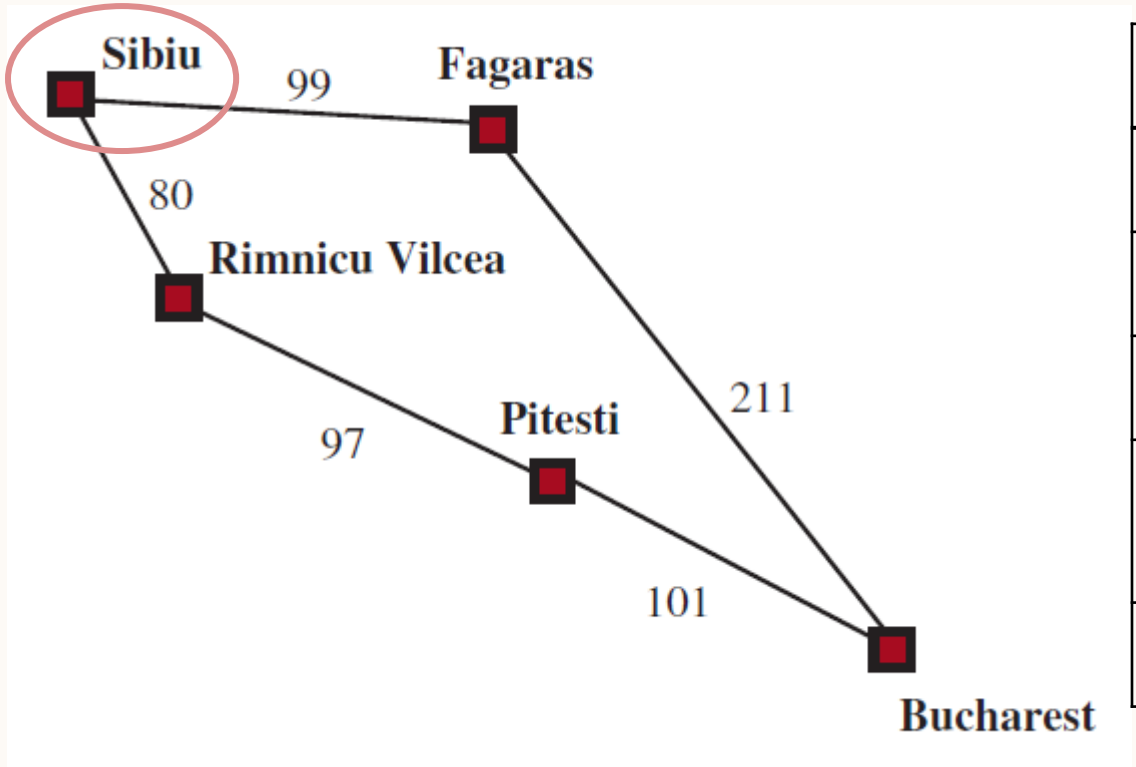
State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[]

Current node:
Sibiu (253)



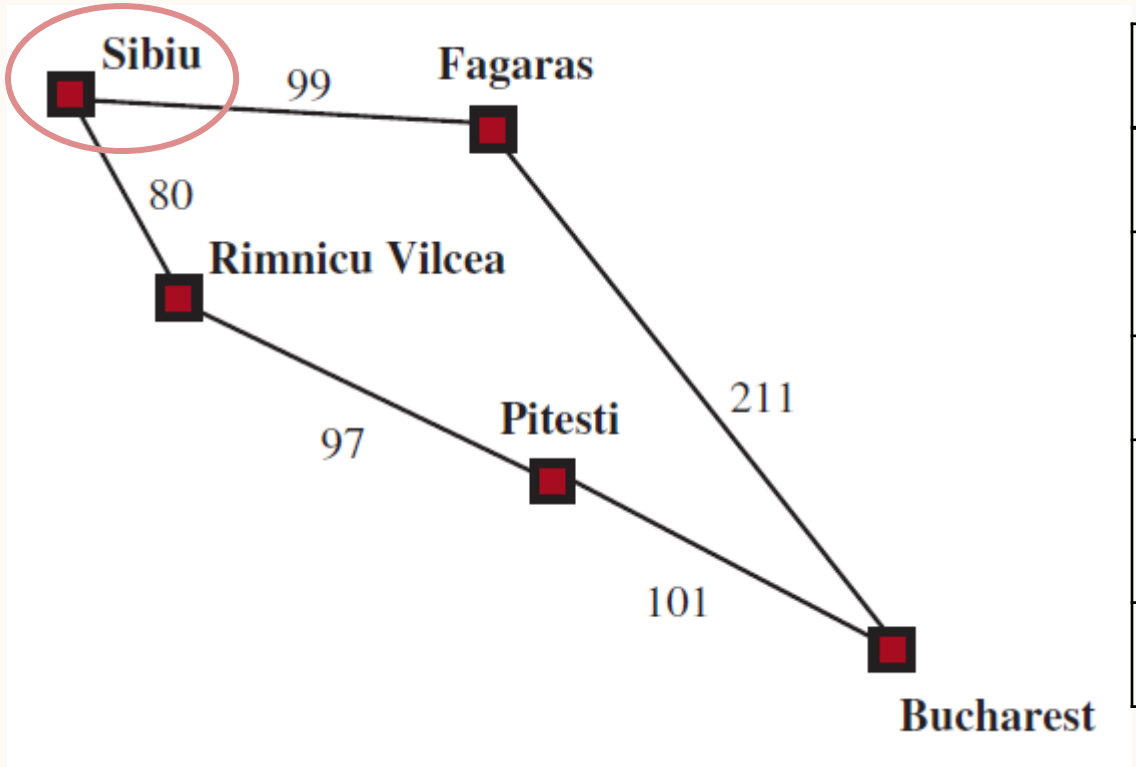
State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[F. (176), R.V. (193)]

Current node:
Sibiu (253)



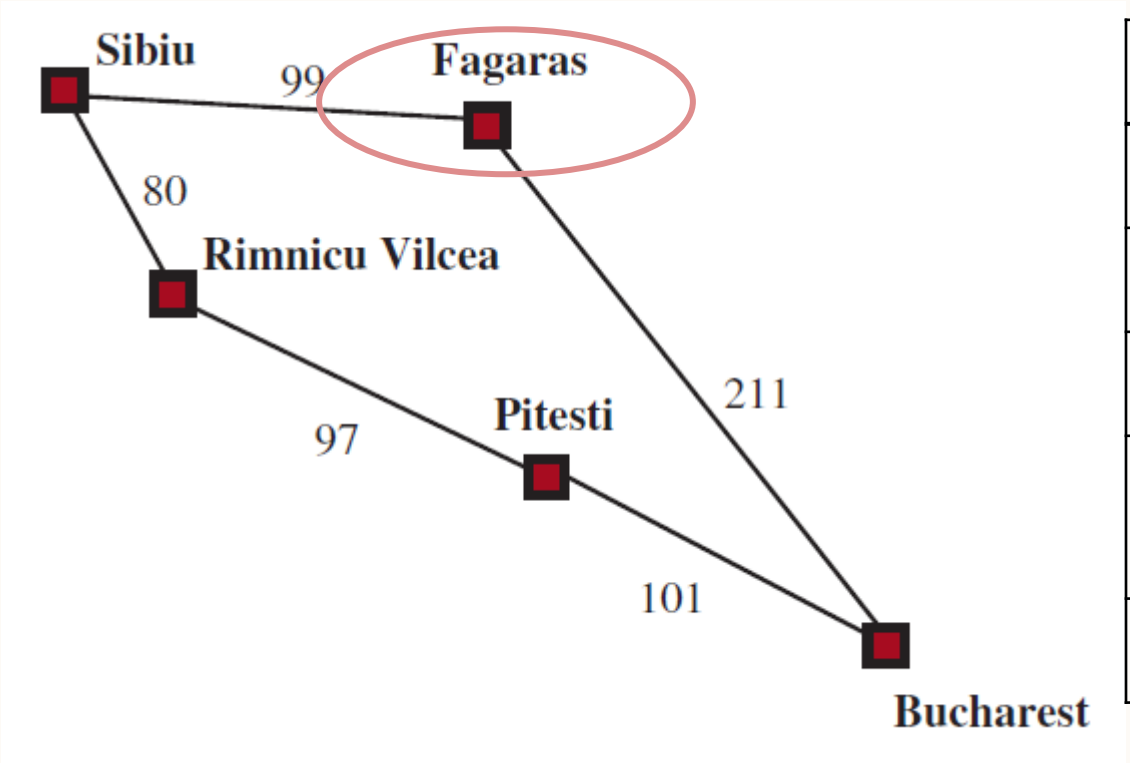
State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[R.V. (193)]

Current node:
F. (176)



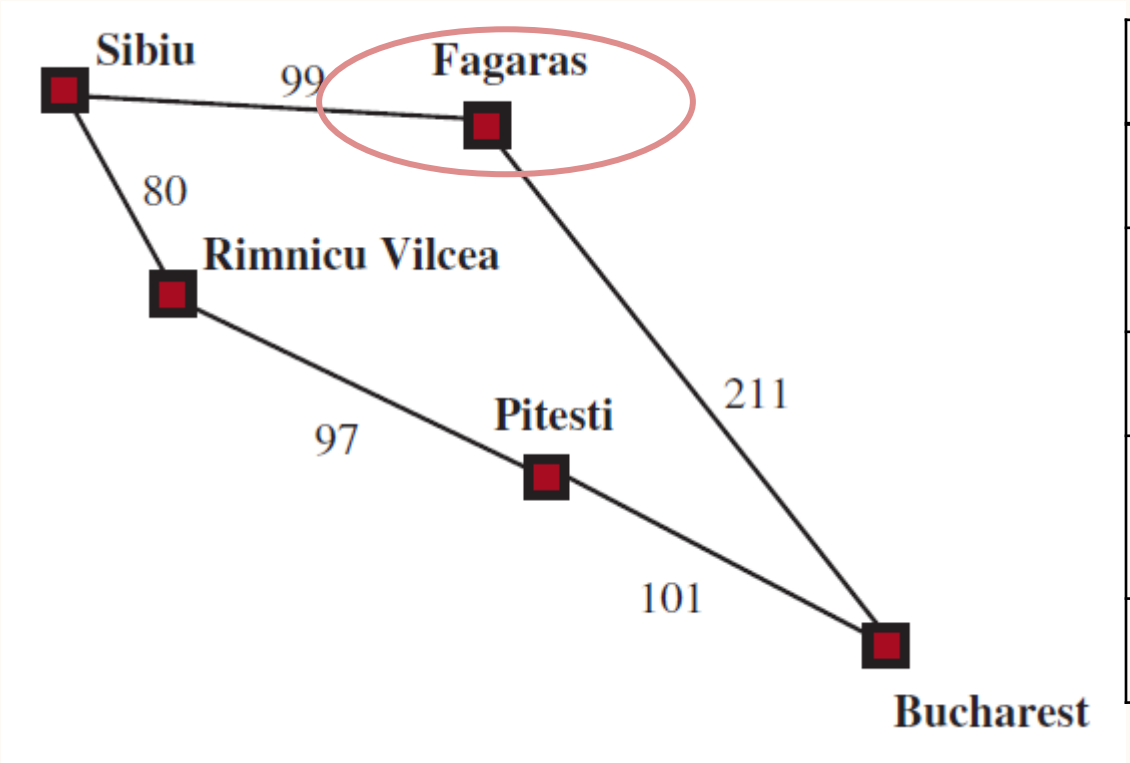
State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[B. (0), R.V. (193)]

Current node:
F. (176)



State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

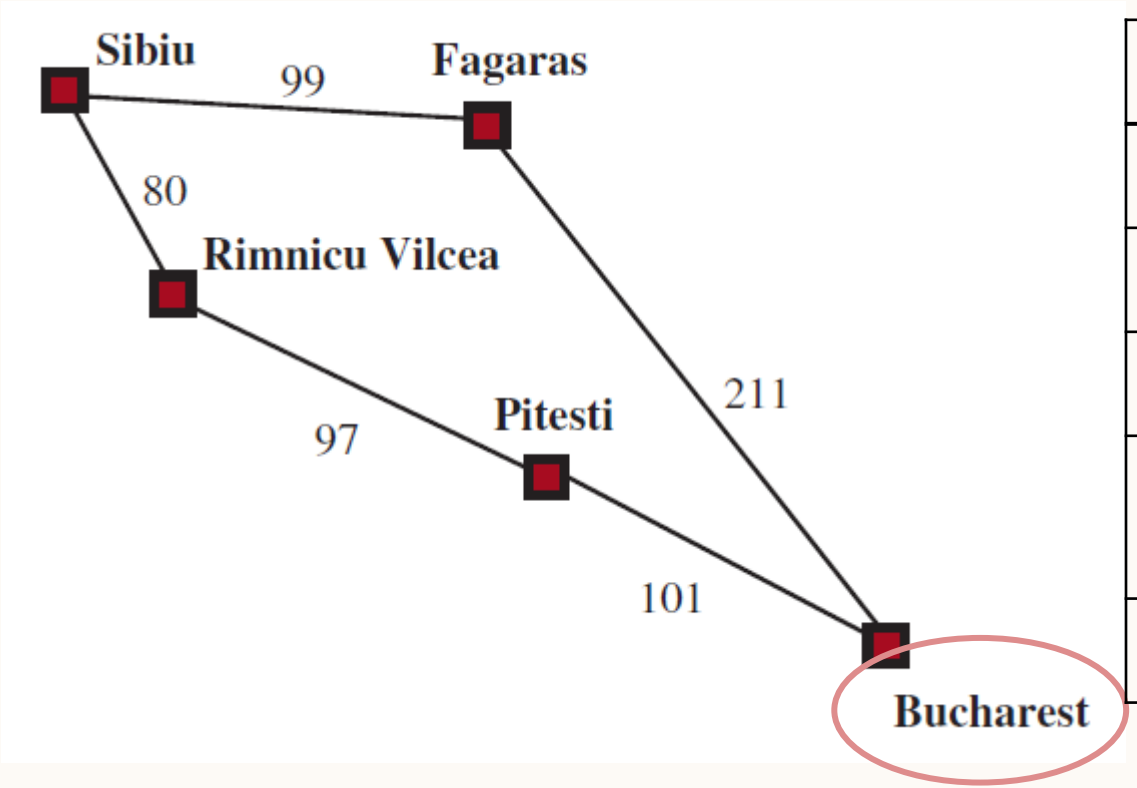
BEST-FIRST SEARCH EXAMPLE

Problem: find the shortest (in terms of distance) path from Sibiu to Bucharest

Frontier:
[R.V. (193)]

Current node:
B. (0)

Return:
[Sibiu, Fagaras, Bucharest]



State	h(State)
Bucharest	0
Fagaras	176
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253

BEST-FIRST SEARCH IN PRACTICE

Experiment URL (try it for yourself!): https://qiao.github.io/PathFinding.js/visual/

Experiment summary:

- Best-First Search is a **greedy** algorithm
  - Every step moves "closer" towards the goal
  - Can miss the bigger picture
- Can be *very* fast with good heuristic choice compared to uninformed search
- May not be optimal depending on the environment

FOR NEXT CLASS

- Read Chapter 3.5-3.7
- If you have Module 1/Search for your paper presentation, start looking for a paper and send it to me & Aydin
  - Presentation on Thursday Sept 21
- Work on the uninformed part of HW 1