

CPSC 322: Introduction to Artificial Intelligence

Search: Iterative Deepening,
Search with Costs,
Heuristic Search

Textbook reference: [[3.5.3](#), [3.5.4](#), [3.6](#)]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Assignment 0 grades have been released.
- If you have any grading-related questions, contact Kyle on Piazza or by email (clarkson@cs.ubc.ca)
- Assignment 1 has been released.
- When you write your assignment, please make it clear what question you are answering.
- TAs might penalize you if they end up spending lots of time decoding your assignment.

Lecture outline

- **Recap from last lecture** (~10 mins) 📌
- Iterative deepening (~20 mins)
- Least Cost First Search (~15 mins)
- Break (~5 mins)
- Heuristic search (~15 mins)
- Summary and wrap-up (~5 mins)

Search strategies



Search strategies are different with respect to how they

- A. Check what node on a path is the goal
- B. Select and remove paths from the frontier
- C. Initialize the frontier
- D. Check if a state is a goal
- E. Choose which tunes to listen to while studying

Generic search algorithm

In what aspects do DFS and BFS differ when we look at the generic graph search algorithm?

Inputs: a graph,
a start node n_0
a boolean procedure $goal(n)$ that tests if n is a goal node

frontier := [$\langle n_0 \rangle$: n_0 is a start node];

While frontier is not empty:

select and remove path $\langle n_0, n_1, \dots, n_k \rangle$ from frontier;

If $goal(n_k)$

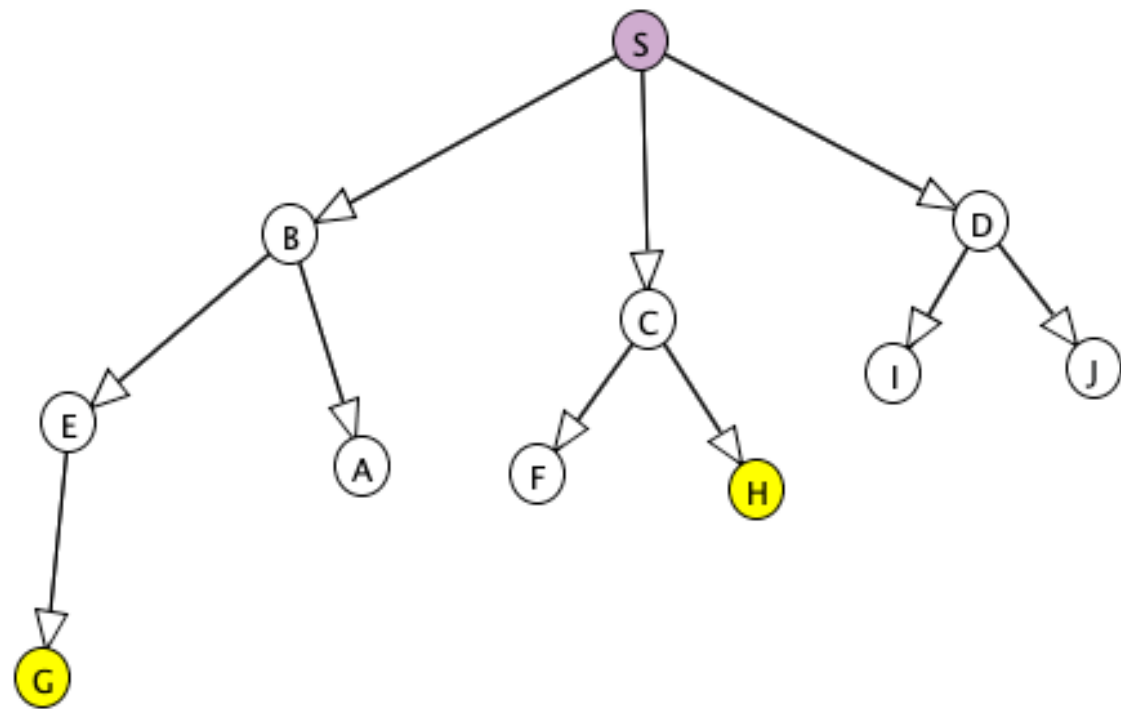
return $\langle n_0, n_1, \dots, n_k \rangle$;

For every neighbour n of n_k

add $\langle n_0, n_1, \dots, n_k, n \rangle$ to frontier;

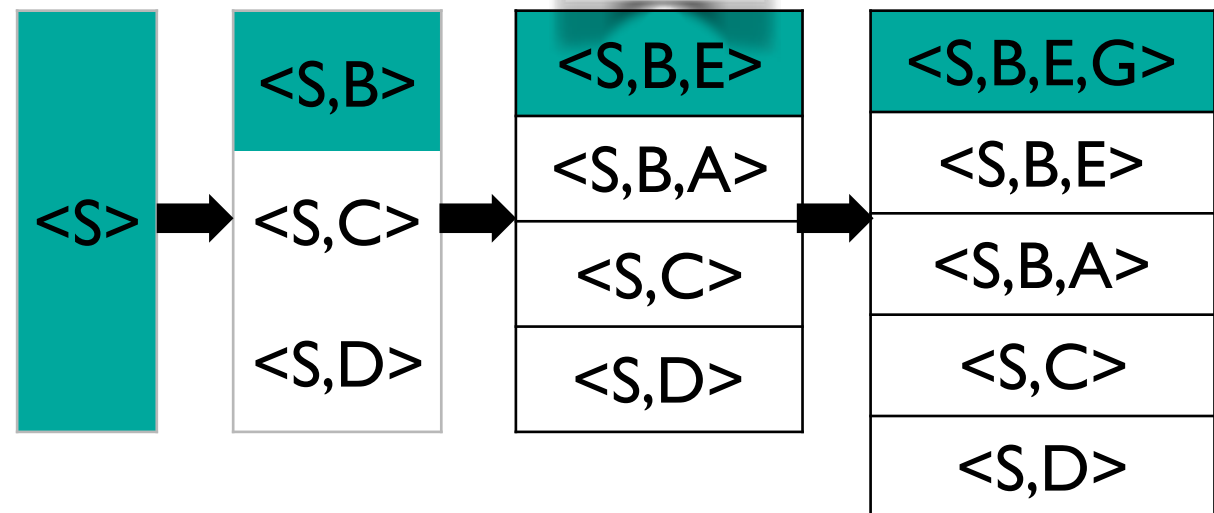
return NULL

DFS vs. BFS

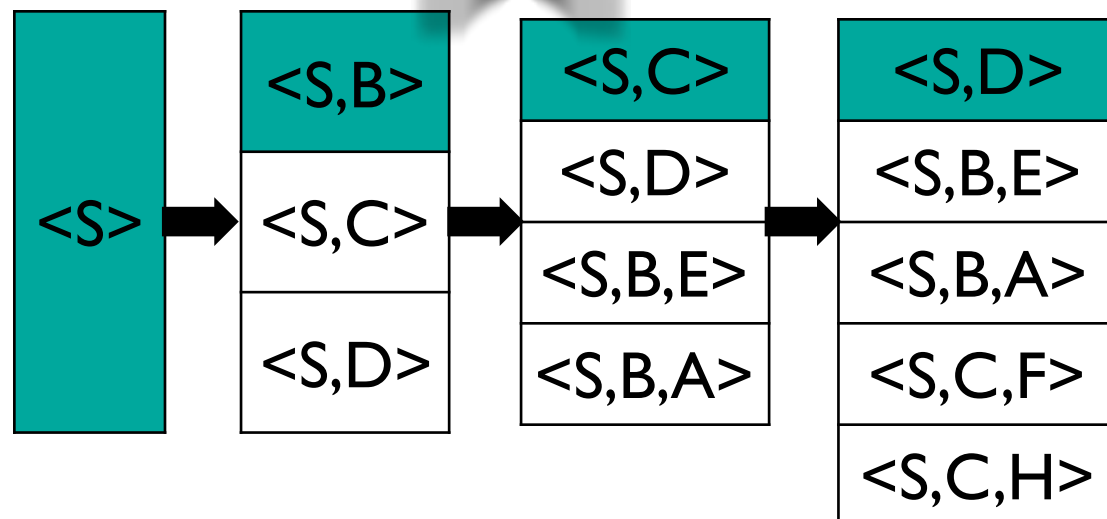


In DFS the **frontier** is a **last-in-first-out stack**.

DFS



BFS



In BFS the **frontier** is a **first-in-first-out queue**.

When to use BFS vs. DFS

A. BFS

B. DFS

- The search graph has cycles or is infinite.
- We need a shortest path to a solution.
- There are only solutions at great depth.
- There are some solutions at shallow depth.
- Memory is limited.

Midterm time



Which of the following times you absolutely **CANNOT** make for the midterm exam?


- A. Thursday Oct 24, **2 to 3pm**
- B. Thursday Oct 24, **7 to 8pm**
- C. Friday Oct 25, **6 to 7pm**

Today: Learning outcomes

At the end of the class you should be able to

- Select the most appropriate search algorithms for specific problems.
- BFS vs. DFS vs. IDS vs. LCFS
- Define/read/write/trace/debug the above search algorithms
- Explain the general idea of heuristic search

Lecture outline

- Recap from last lecture (~10 mins)
- **Iterative deepening** (~20 mins) 
- Least Cost First Search (~15 mins)
- Break (~5 mins) (if possible)
- Heuristic search (~15 mins)
- Summary and wrap-up (~5 mins)

Criteria to compare search strategies

So far we have looked at completeness, optimality, time and space complexity of DFS and BFS.

	complete?	optimal?	time $O()$	space $O()$
DFS	No	No	b^m	bm
BFS	Yes	Yes*	b^m	b^m
IDS				

*Assuming arcs all have the same cost. (We'll get to this later.)

Iterative Deepening Search (IDS)

Can we achieve an acceptable (linear) space complexity maintaining completeness and optimality? 🤔

	complete?	optimal?	time $O()$	space $O()$
DFS	No	No	b^m	bm
BFS	Yes	Yes*	b^m	b^m
IDS				

*Assuming arcs all have the same cost. (We'll get to this later.)

Iterative Deepening Search (IDS)

Can we achieve an acceptable (linear) space complexity maintaining completeness and optimality? 🤔

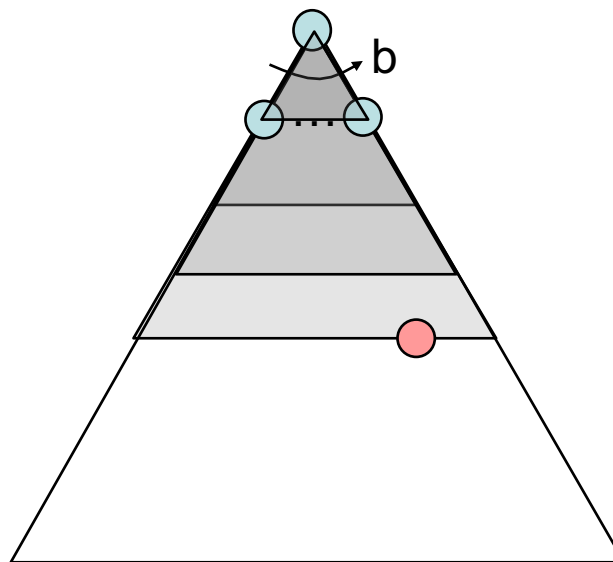
	complete?	optimal?	time $O()$	space $O()$
DFS	No	No	b^m	bm
BFS	Yes	Yes*	b^m	b^m
IDS				

*Assuming arcs all have the same cost. (We'll get to this later.)

Key idea: Recompute the elements of the frontier rather than saving them.

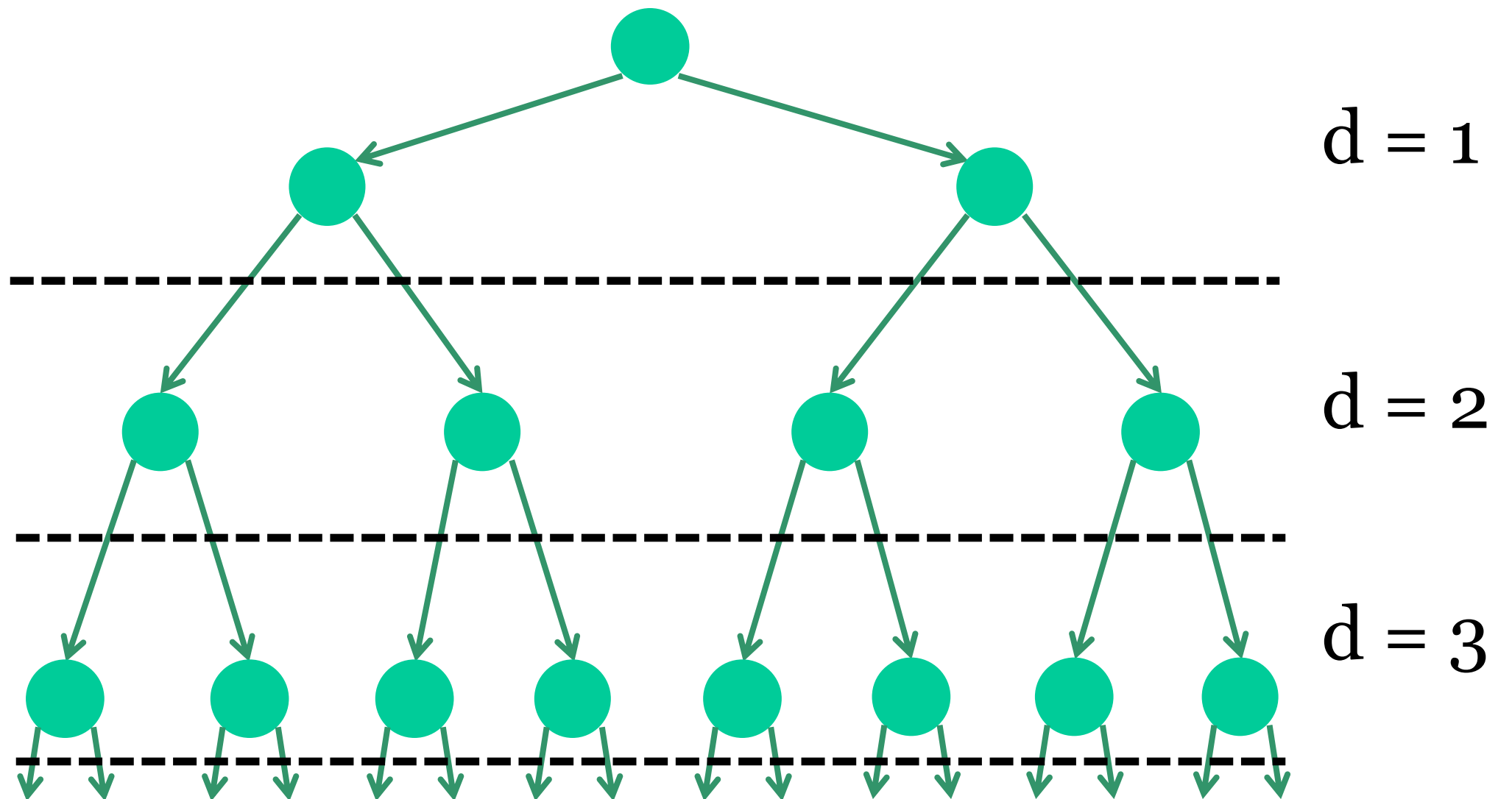
IDS

- Define a depth bound b
- Run a DFS from scratch with depth limit $d = 1$. If no solution...
- Run a DFS from scratch with depth limit $d = 2$. If no solution...
- Run a DFS from scratch with depth limit $d = 3$

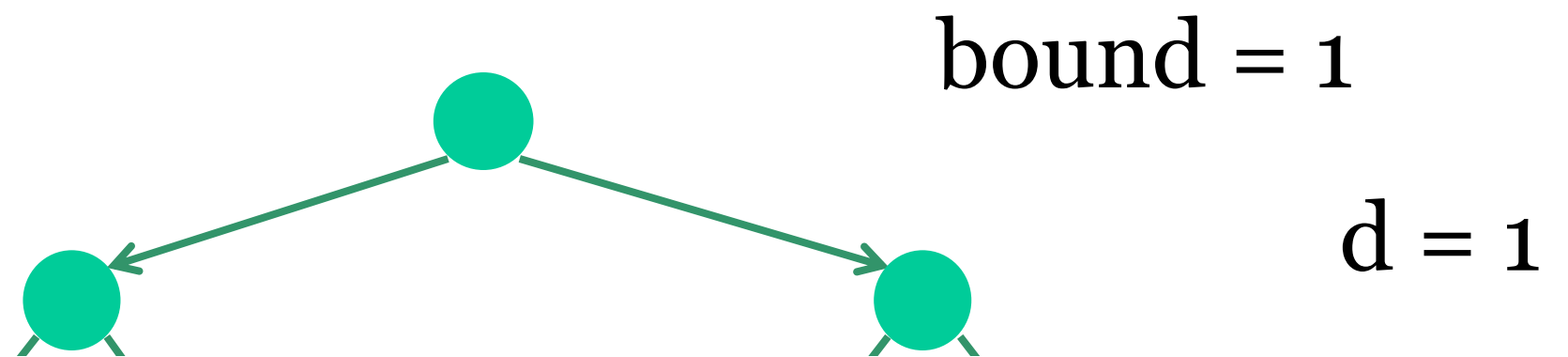


Credit: Berkeley AI course material

IDS

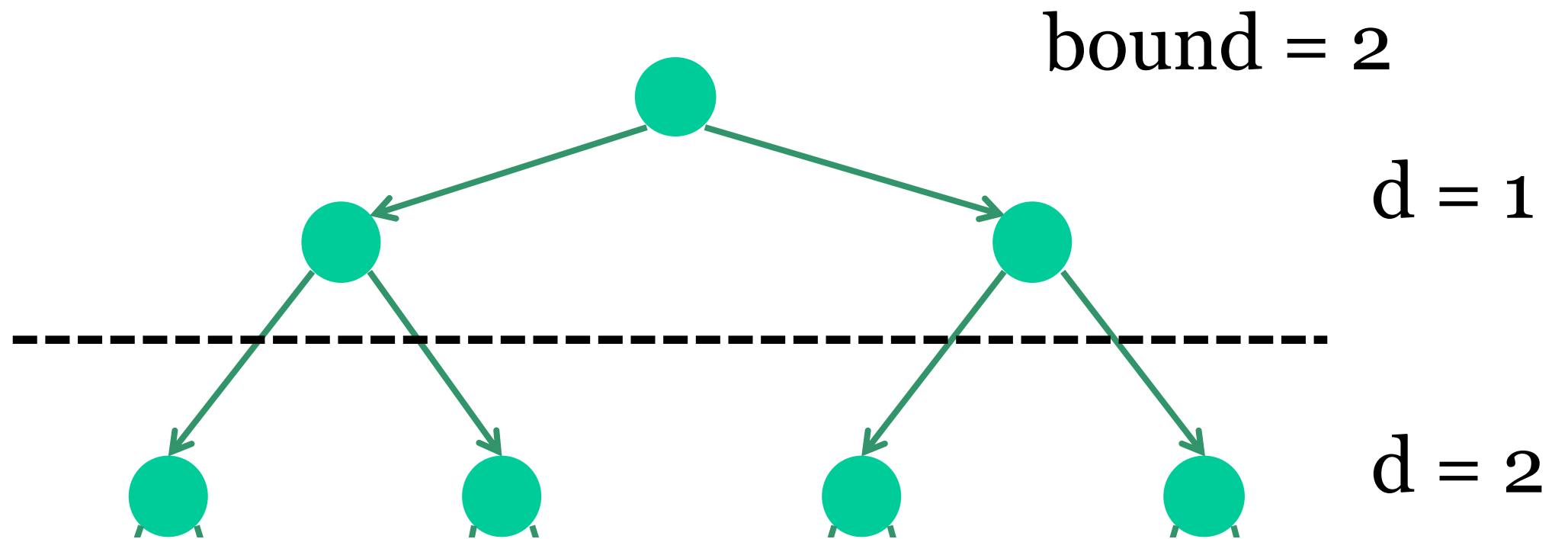


IDS



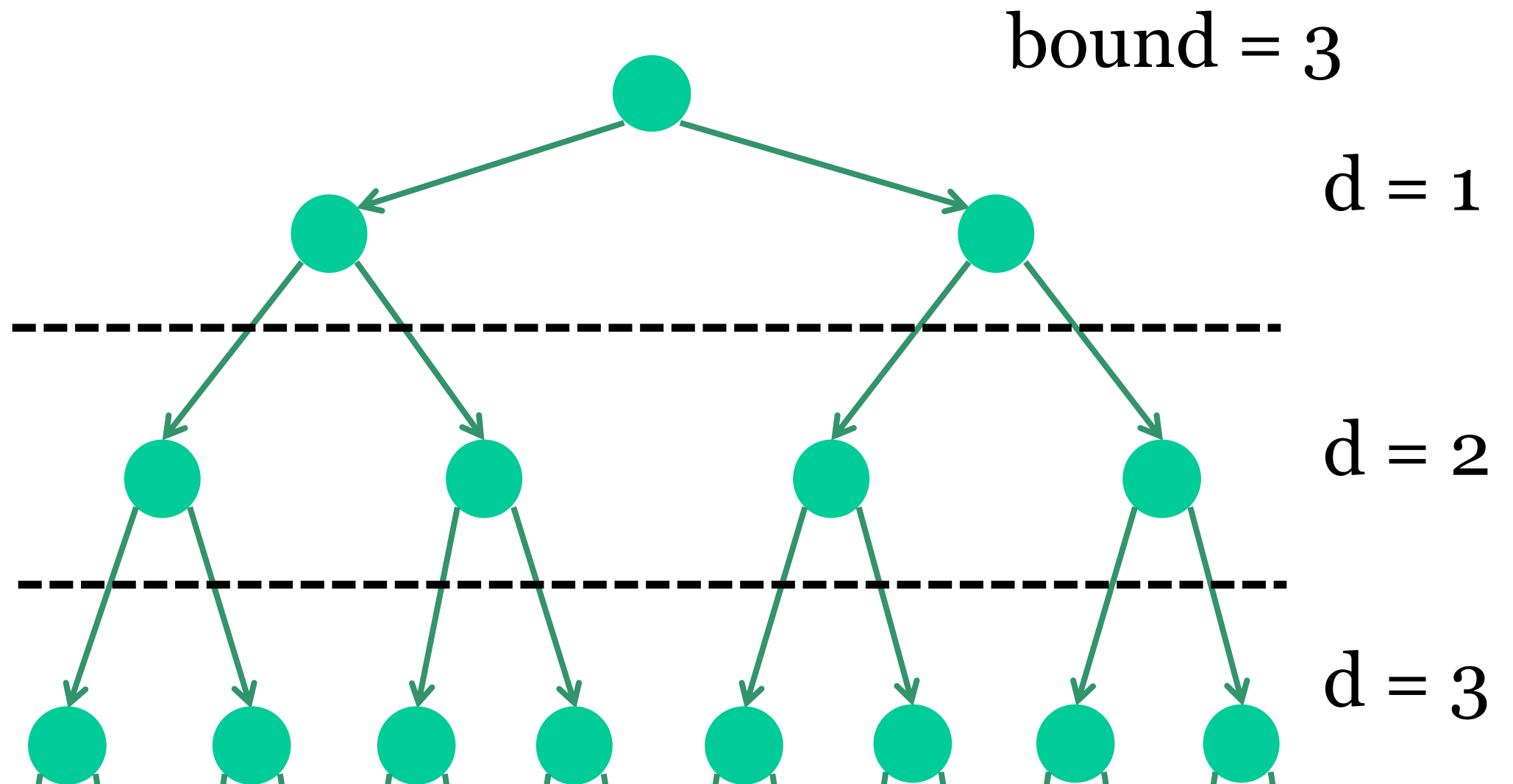
Run a DFS with depth size 1

IDS



If no solution with $d = 1$, run a DFS with depth size 2.

IDS



If no solution with $d = 2$, run a DFS with depth size 3

IDS in essence

- Look with DFS for solutions at $d = 1$, then 2, then 3, etc.
- If a solution cannot be found at depth d , look for a solution at depth $d + 1$.
- You need a depth-bounded depth-first searcher.
- We continue this until the depth limit reaches the bound; we assume that paths of length bound cannot be expanded.

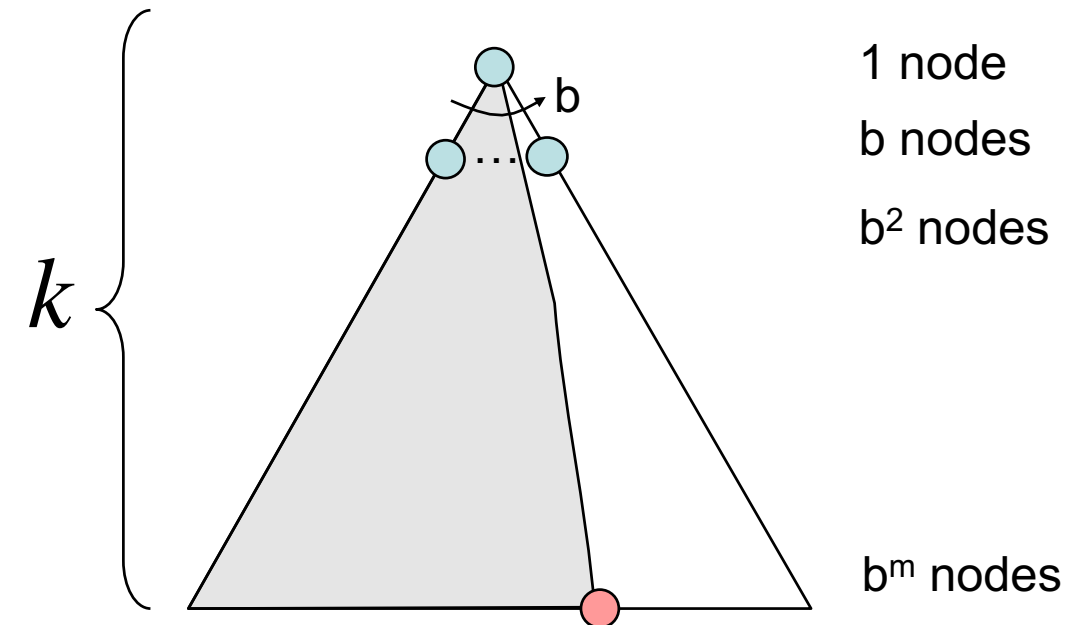
Isn't it wastefully redundant?

Time complexity of IDS

- Not so bad as one might think, especially when the branching factor b is large, as most work happens in the lowest level searched.

Time complexity of IDS

- At depth k there are b^k nodes, each has been generated once.
- The nodes at depth $k - 1$ have been generated twice and the nodes at depth 1 have been generated k times.



Branching factor $b > 1$

Depth bound = k

So the total number of paths expanded is

$$b^k + 2b^{k-1} + 3b^{k-2} + \dots + kb$$

$$= b^k(1 + 2b^{-1} + 3b^{-2} + \dots + kb^{1-k})$$

Time complexity of IDS

So the total number of paths expanded is

$$b^k + 2b^{k-1} + 3b^{k-2} + \dots + kb \\ = b^k(1 + 2b^{-1} + 3b^{-2} + \dots + kb^{1-k})$$

$$< b^k \left(\sum_{i=1}^{\infty} ib^{1-i} \right) \text{ which expands to } b^k \left(\frac{b}{b-1} \right)^2$$


Turns out that IDS has an asymptotic overhead of $\left(\frac{b}{b-1} \right)$ times the cost of expanding the nodes at depth k using breadth-first search. The time complexity is $O(b^m)$.

Interim summary

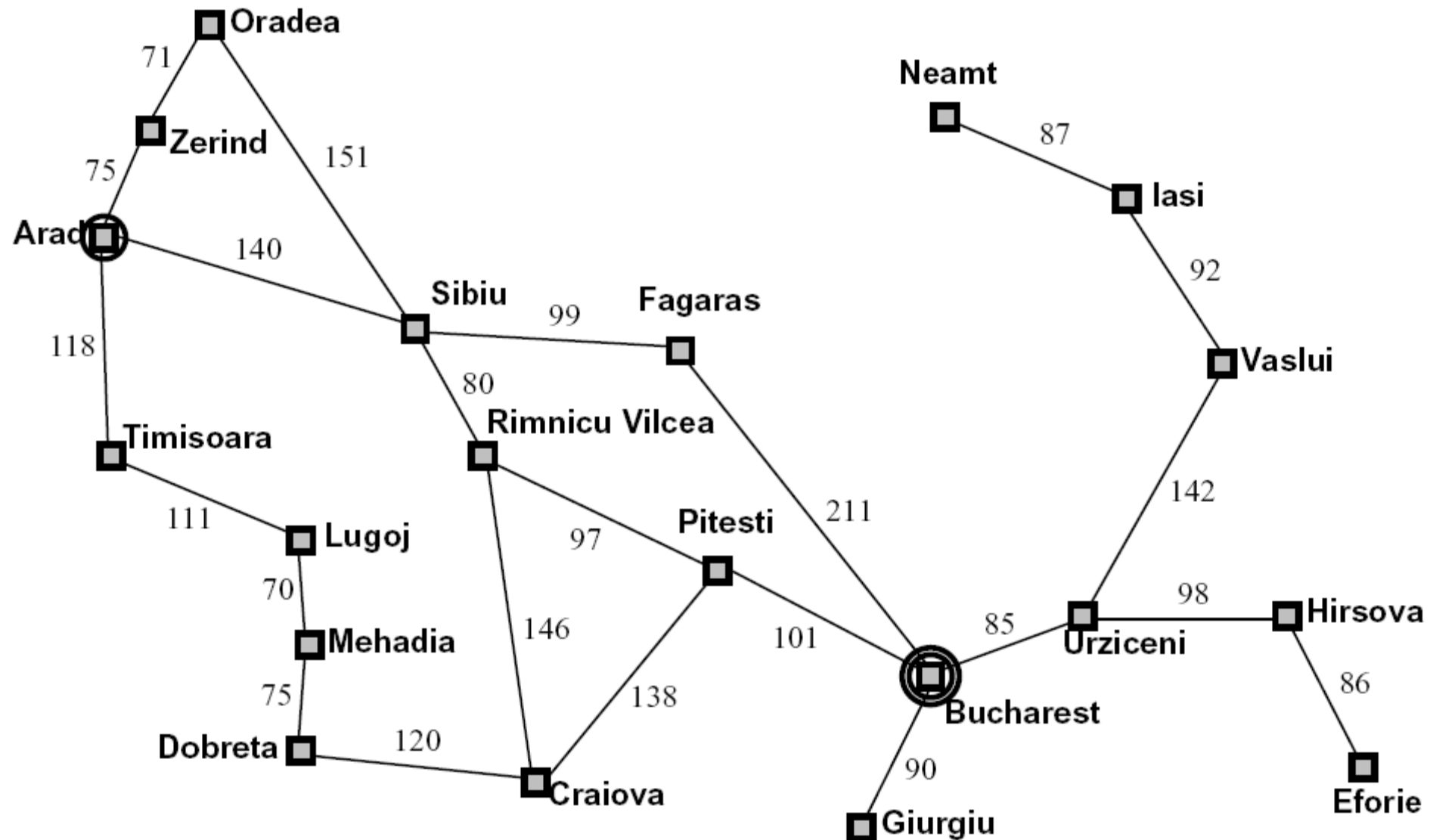
	complete?	optimal?	time $O()$	space $O()$
DFS	No	No	b^m	bm
BFS	Yes	Yes*	b^m	b^m
IDS	Yes	Yes*	b^m	bm

* Assuming arcs cost are equal. (We'll get to this later.)

Lecture outline

- Recap from last lecture (~10 mins)
- Iterative deepening (~20 mins)
- **Least Cost First Search** (~15 mins) 
- Break (~5 mins) (if possible)
- Heuristic search (~15 mins)
- Summary and wrap-up (~5 mins)

Search with costs



Sometimes there are costs associated with arcs.

Search with costs

The cost of a path is the sum of the costs of its arcs.

$$\text{cost}(\langle n_0, n_1, \dots, n_k \rangle) = \sum_{i=1}^k \text{cost}(\langle n_{i-1}, n_i \rangle)$$

In this setting we often do not just want to find just any solution; we usually want to find the solution that **minimizes cost**.

A search algorithm is **optimal** if, when it returns a solution, it is the best solution: **it has the lowest path cost**.

Lowest Cost First Search (LCFS)

- At each stage, lowest-cost-first search selects a path on the frontier with **lowest cost**.
- The frontier is a priority queue ordered by path cost.

In LCFS the **frontier** is a **priority queue**.

Alspace: LCFS example



- In the Search Applet toolbar select the “Vancouver Neighbourhood Graph” problem.
- Set “Search Options → Search Algorithms” to “Lowest-Cost-First”.
- Select “Show Edge Costs” under “View”.
- Run “Auto Search”.
- Now create a new arc from UBC to SP with cost 9 and trace the algorithm using “Fine Step”.

LCFC



When arc costs are equal LCFS is equivalent to

- A. Depth-first search (DFS)
- B. Breadth-first search (BFS)
- C. Iterative-depth search (IDS)
- D. None of the above
- E. I don't care.

Completeness of LCFS

A search algorithm is **complete** if, whenever at least one solution exists, the algorithm is guaranteed to find a solution within a finite amount of time.

Is LCFS complete?

A. Yes

B. No

Completeness of LCFS

Is LCFS complete?

A. Yes

B. No

- Not in general: a cycle with **zero or negative** arc costs could be followed forever.
- Yes, as long as arc costs are strictly positive and the branching factor is finite.

Optimality of LCFS

A search algorithm is **optimal** if, when it returns a solution, it is the best solution: **it has the lowest path cost.**

Is LCFS optimal?

A. Yes

B. No

Optimality of LCFS

Is LCFS optimal?

- Not in general. Why not?
- Arc costs could be negative: A path that looks high-cost could end up getting a refund.
- However, LCFC is optimal if all costs are guaranteed to be strictly positive.

Time and space complexity

- What is the **time complexity**, if the maximum path length is m and the maximum branching factor is b ?
- The time complexity is $O(b^m)$: may need to examine every node in the tree. Knowing costs does not help here.
- What is the **space complexity**?
- Space complexity is $O(b^m)$: in the case where arc costs are equal (and > 0), LCFS behaves like BFS.

Summary: uninformed search strategies


	complete?	optimal?	time $O()$	space $O()$
DFS	No	No	b^m	bm
BFS	Yes	Yes*	b^m	b^m
IDS	Yes	Yes*	b^m	bm
LCFS	Yes**	Yes**	b^m	b^m

*Assuming arc costs are equal

** Assuming arc costs are positive

Why are they called “uninformed” or “blind” search strategies?

Lecture outline

- Recap from last lecture (~10 mins)
- Iterative deepening (~20 mins)
- Least Cost First Search (~15 mins)
- Break (~5 mins) (if possible)
- **Heuristic search** (~15 mins) 
- Summary and wrap-up (~5 mins)

How are these strategies “uninformed”?

- Because they do not consider any information about the states (end nodes) to decide which path to expand first on the frontier.
- All they know is generating successors and distinguishing goal node from a non-goal node.
- In other words, they are general; they do not take into account the specific nature of the problem.

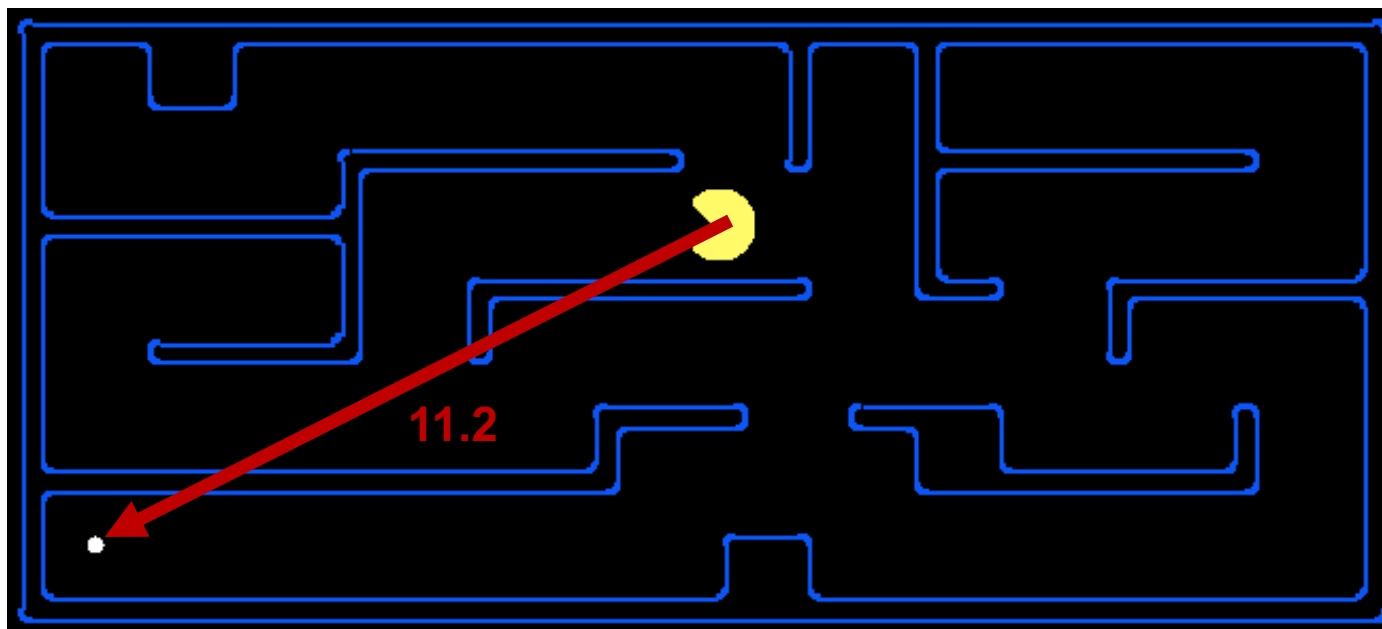
Informed search strategies

- Blind search algorithms do not take into account the goal until they are at a goal node.
- Often there is extra knowledge that can be used to guide the search:
 - an estimate of the distance/cost from node n to a goal node.
- This estimate is called a **search heuristic**.

Search heuristic

A heuristic is

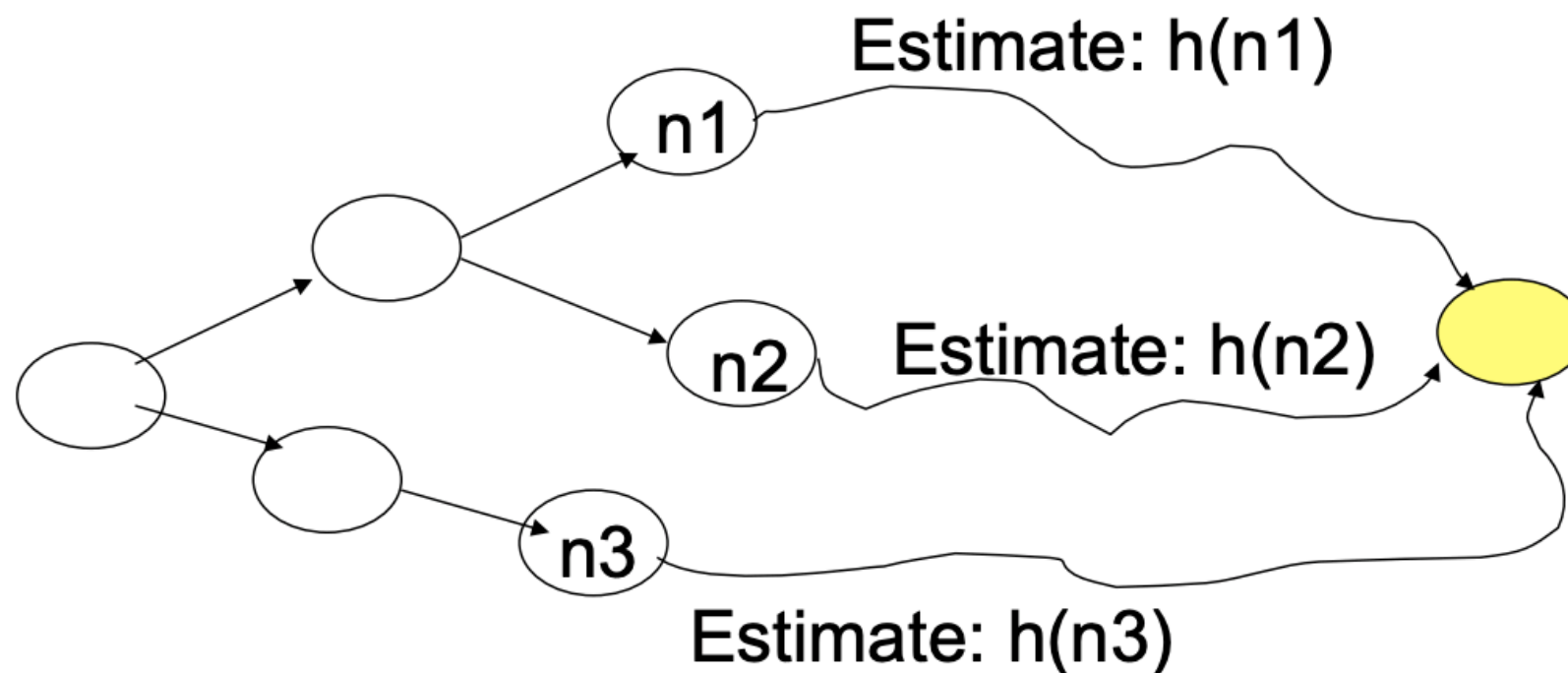
- A function that estimates how close a state is to the goal
- Designed for a particular search problem
- Example: Euclidean distance for pathing



Not perfect but
a good guess

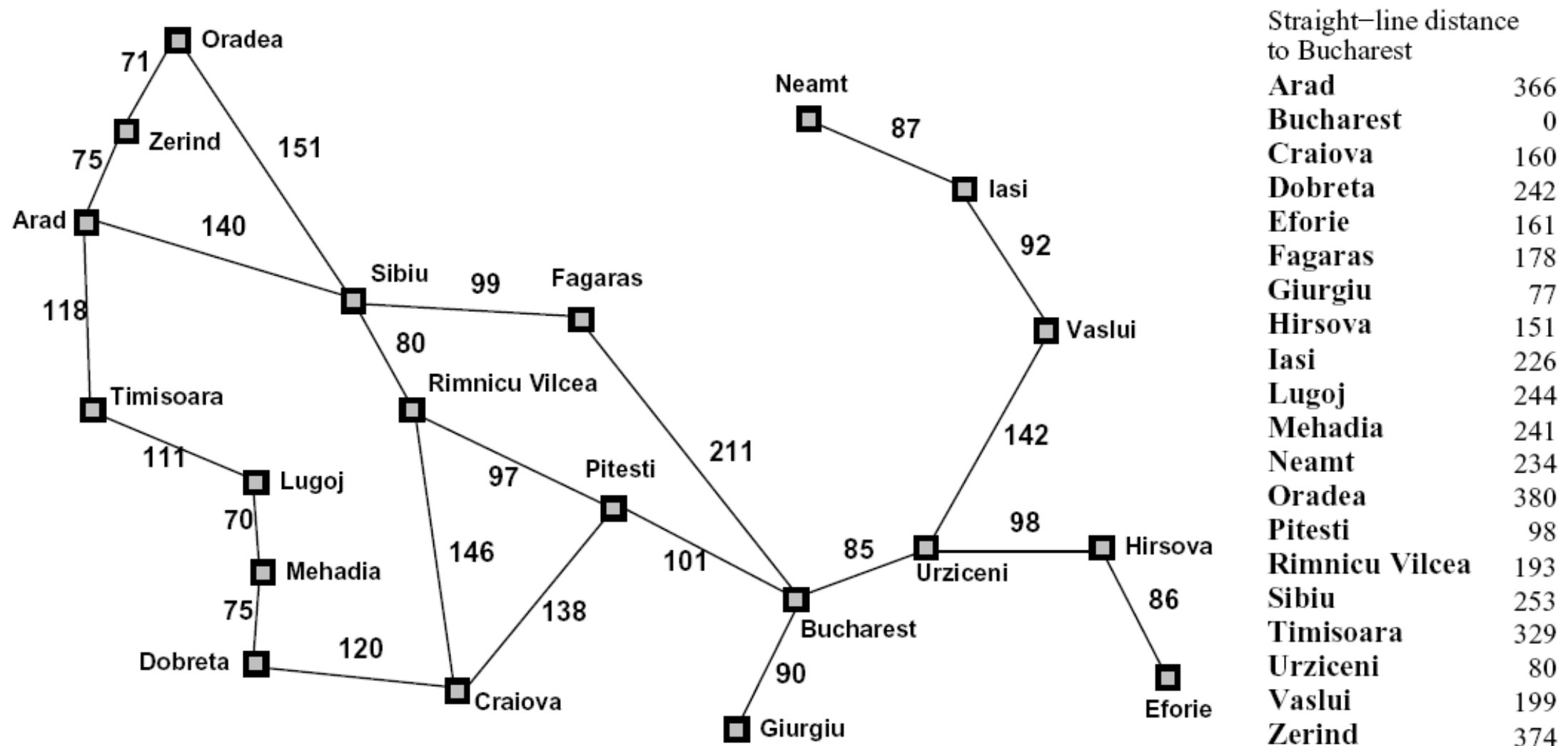
More formally

A **search heuristic** $h(n)$ is an estimate of the cost of the lowest-cost path from node n to a goal node.



h can be extended to paths $h(\langle n_0, \dots, n_k \rangle) = h(n_k)$.
But for now think of $h(n)$ as only using readily obtainable information (that is easy to compute) about a node.

Example of $h(n)$



Euclidean distance: straight-line distance between source and goal node


Uninformed vs. informed search

Uninformed (blind) search strategies

- No additional information about the states beyond what's provided in the problem definition
- Generate successors and distinguish goal node from a non-goal node
- Usually less efficient

Informed (heuristic) search strategies

- Use problem-specific knowledge beyond the definition of a problem itself
- Know whether one non-goal state is more “promising” than another
- Usually more efficient

- 
- Best-first search [3.6]
 - A* search [3.6.1]

