Uniformed Search (cont.)

Computer Science cpsc322, Lecture 6

(Textbook finish 3.5)

May, 18, 2017

Lecture Overview

- **Recap DFS vs BFS**
- Uninformed Iterative Deepening (IDS)
- Search with Costs

Search Strategies

Search Strategies are different with respect to how they:

A. Check what node on a path is the goal
B. Initialize the frontier
C. Add/remove paths from the frontier
D. Check if a state is a goal

Recap: Graph Search Algorithm

Input: a graph, a start node, Boolean procedure *goal(n)* that tests if *n* is a goal node

frontier:= [⟨*s*⟩: *s* is a start node];

While *frontier* is not empty:

  select and **remove** path ⟨$n_o, \cdots, n_k$⟩ from *frontier;*

  If *goal*(n_k)

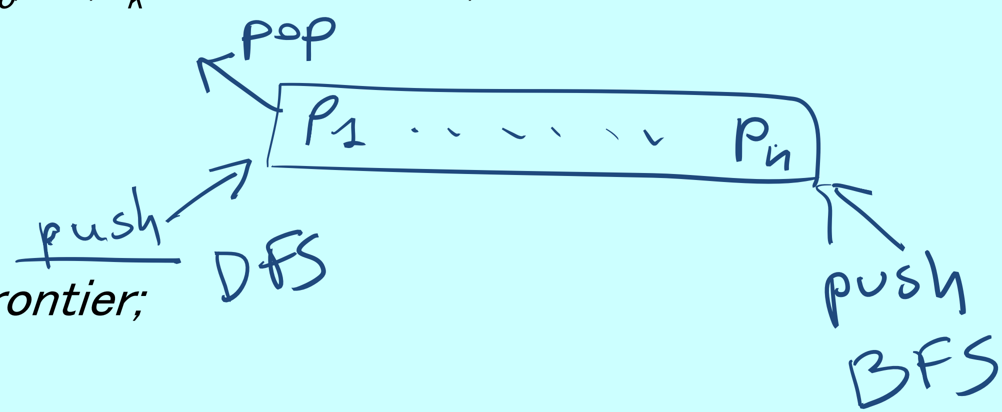
    return ⟨$n_o, \cdots, n_k$⟩;

  For every neighbor *n* of n_k

    add ⟨$n_o, \cdots, n_k$, *n*⟩ to *frontier;*

end

No solution found



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

When to use BFS vs. DFS?

- The search graph has cycles or is infinite

  BFS DFS

- We need the shortest path to a solution

  BFS DFS

- There are only solutions at great depth

  BFS DFS

- There are some solutions at shallow depth

  BFS DFS

- Memory is limited

  BFS DFS

Lecture Overview

- Recap DFS vs BFS
- Uninformed Iterative Deepening (IDS)
- Search with Costs

Iterative Deepening (sec 3.6.3)

How can we achieve an acceptable (linear) space complexity maintaining completeness and optimality?

	Complete	Optimal	Time	Space
DFS	N	N	b^m	mb
BFS	Y	Y	b^m	b^m
IDS	Y	Y	b^m	mb

Key Idea: let's re-compute elements of the frontier rather than saving them.

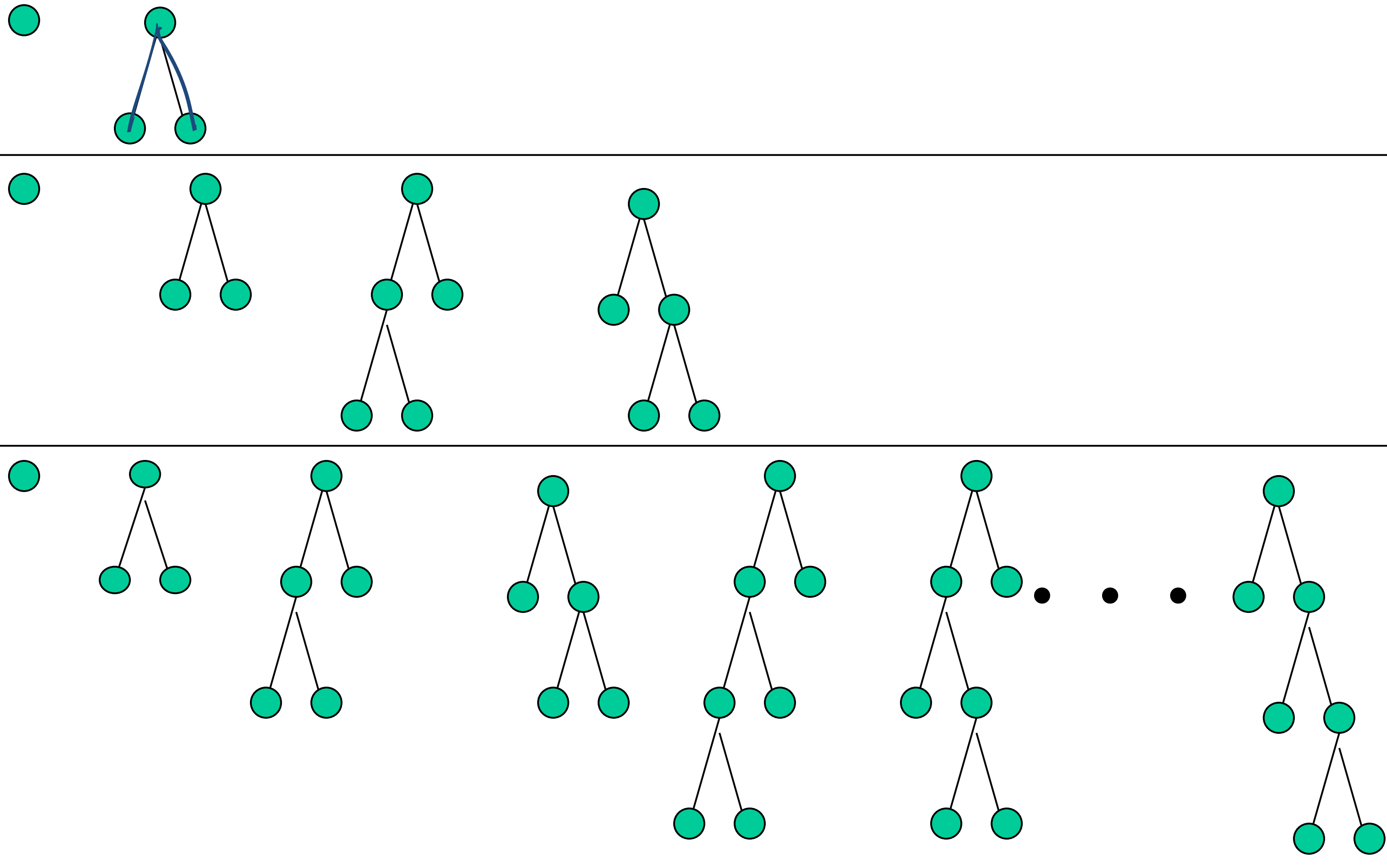
Iterative Deepening in Essence

- **Look with DFS** for solutions at depth 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.
- Given a bound B you simply assume that paths of length B cannot be expanded….

depth = 1

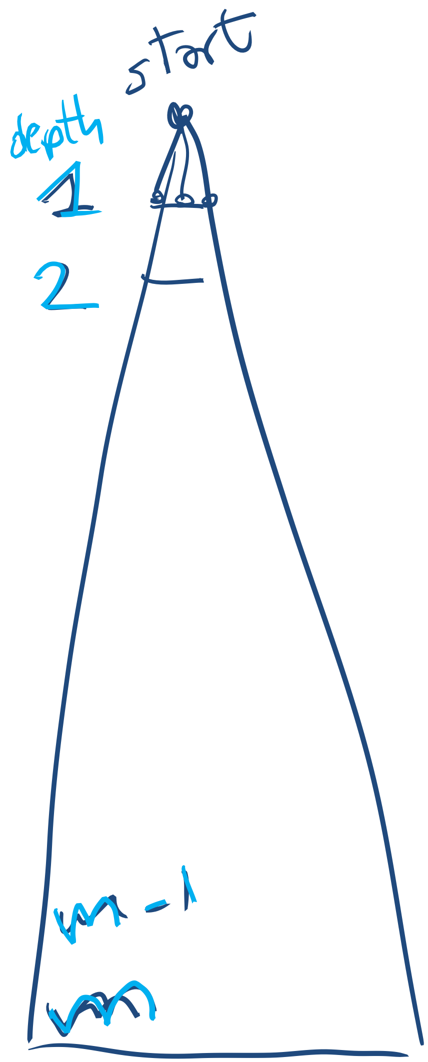
depth = 2

depth = 3



(Time) Complexity of Iterative Deepening

Complexity of solution at depth m with branching factor b



Total # of paths at that level	#times created by BFS (or DFS)	#times created by IDS	total # of paths created by IDS
b	1	m	$m b$
b^2	1	$m-1$	$(m-1) b^2$
			$2 b^{m-1}$
b^m	1	1	b^m

$\sum$

(Time) Complexity of Iterative Deepening

Complexity of solution at depth m with branching factor *b*

Total # of paths generated

$$b^m + 2\,b^{m-1} + 3\,b^{m-2} + .. + mb = \quad A$$

$$b^m (1 + 2\,b^{-1} + 3\,b^{-2} + .. + m\,b^{1-m}) \le$$

$$b^m \left(\sum_{i=1}^{\infty} i b^{1-i}\right) = b^m \left(\frac{b}{b-1}\right)^2 = O(b^m)$$

B

$A < B$

$b = 2 \quad 4$

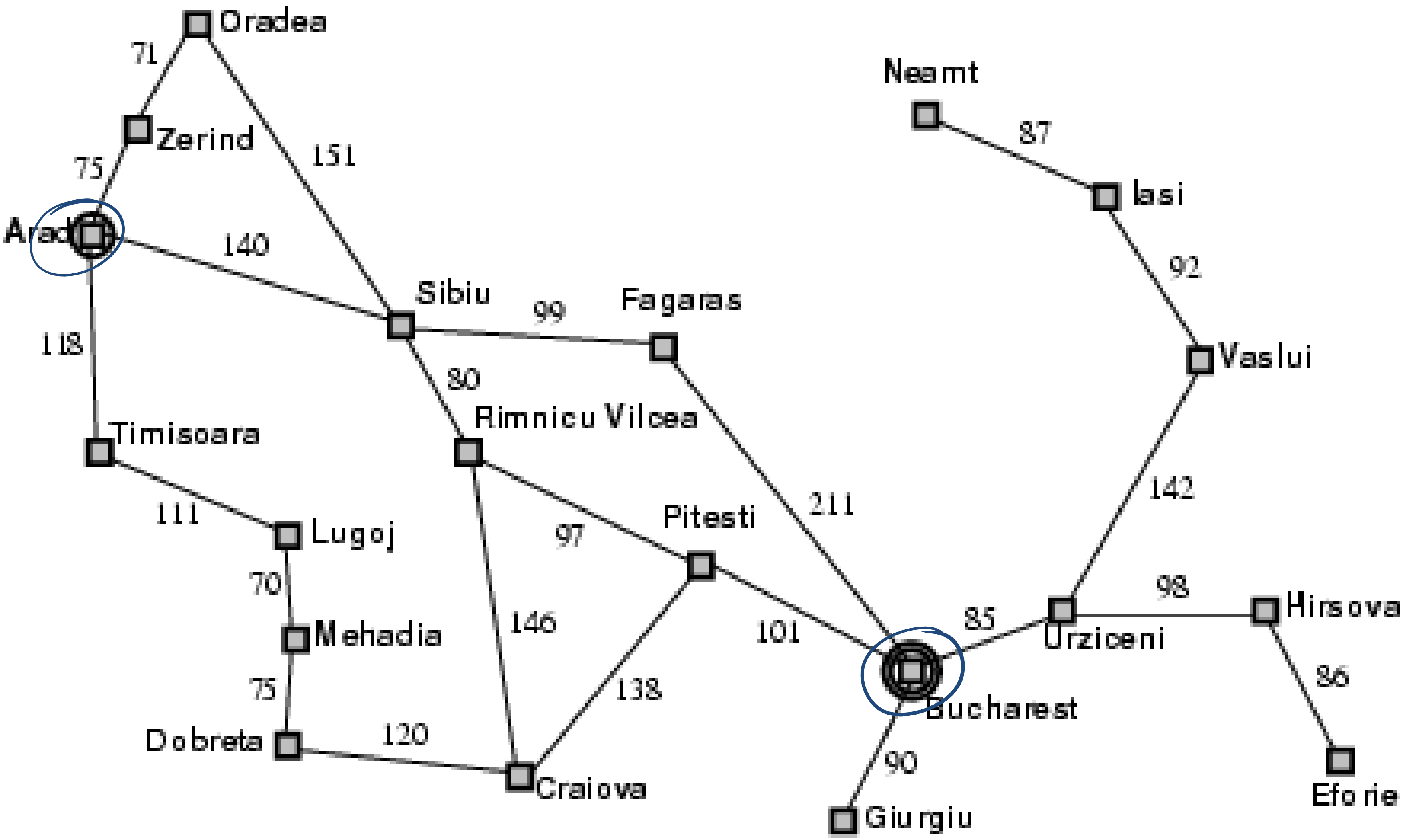
$b = 3 \quad \frac{9}{4} = 2.25$

$b = 4 \quad \frac{16}{9} < 2$

Lecture Overview

- Recap DFS vs BFS
- Uninformed Iterative Deepening (IDS)
- Search with Costs

Example: Romania



Search with Costs

Sometimes there are costs associated with arcs.

Definition (cost of a path)

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

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

In this setting we often don't just want to find just any solution

- **we usually want to find the solution that minimizes cost**

Definition (optimal algorithm)

A search algorithm is optimal if, when it returns a solution, it is the one with minimal cost.

Lowest-Cost-First Search

- 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**
  - We say ``a path'' because there may be ties

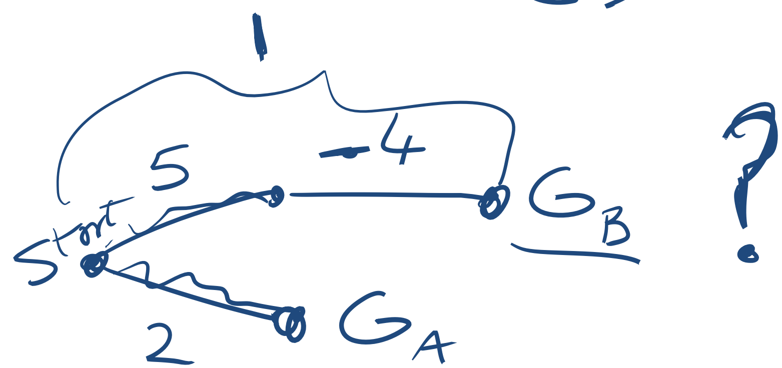
- Example of one step for LCFS:
  - the frontier is $[\langle p_2, 5\rangle, \langle p_3, 7\rangle, \langle p_1, 11\rangle,]$
  - p_2 is the lowest-cost node in the frontier
  - "neighbors" of p_2 are $\{\langle p_9, 10\rangle, \langle p_{10}, 15\rangle\}$
- What happens?
  - p_2 is selected, and tested for being a goal (its end).
  - (if not a goal) Neighbors of p_2 are inserted into the frontier
  - Thus, the frontier is now $[\langle p_3, 7\rangle, \langle p_9, 10\rangle, \langle p_1, 11\rangle, \langle p_{10}, 15\rangle]$.
  - ? ? is selected next.
  - Etc. etc.

- When arc costs are equal LCFS is equivalent to..

A. DFS
B. BFS
C. IDS
D. None of the Above

Analysis of Lowest-Cost Search (1)

- Is LCFS complete?
  - not in general: a cycle with zero or negative arc costs could be followed forever
  - yes, as long as arc costs are strictly positive



- Is LCFS optimal?
  - Not in general. Why not?
  - Arc costs could be negative: a path that initially looks high-cost could end up getting a ``refund''.
  - However, LCFS *is* optimal if arc costs are guaranteed to be non-negative.

Analysis of Lowest-Cost Search

- What is the time complexity, if the maximum path length is m and the maximum branching factor is b?
  - The time complexity is $O(b^m)$: must examine every node in the tree.
  - Knowing costs doesn't help here.

- What is the space complexity?
  - Space complexity is $O(b^m)$: we must store the whole frontier in memory.

Learning Goals for Search (up to today)

- Apply basic properties of search algorithms: completeness, optimality, time and space complexity of search algorithms.

	Complete	Optimal	Time	Space
DFS	N	N	b^m	bm
BFS	Y	Y	n	b^m
IDS	Y	Y	n	bm
LCFS	N Y if C>0	N Y if C≥0	b^m	b^m

Learning Goals for Search (cont')
(up to today)

- Select the most appropriate search algorithms for specific problems.
  - BFS vs DFS vs IDS vs BidirS
  - LCFS vs. BFS –
  - A* vs. B&B vs IDA* vs MBA*

uninformed

informed

- Define/read/write/trace/debug different search algorithms
  - With / Without cost
  - Informed / Uninformed

Beyond uninformed search….

What information we could use to better select paths from the frontier

A. an estimate of the distance from the last node on the path to the goal
B. an estimate of the distance from the start state to the goal
C. an estimate of the cost of the path
D. None of the above

Next Class on Tue

- **Heuristic Search**

(read textbook.: 3.6)

- Best-First Search
- Combining LCFS and BFS: A* (finish 3.6)
- A* Optimality

Finish Search (finish Chpt 3)

- Branch-and-Bound
- A* enhancements
- Non-heuristic Pruning
- Dynamic Programming