

COMP219: Artificial Intelligence

Lecture 7: Search Strategies

1

Problem Solving as Search

- In the state space view of the world, finding a solution is finding a **path** through the **state space**.
- When we (as humans) solve a problem like the 8-puzzle we have some **idea** of what constitutes the next best move.
- It is hard to program this kind of approach.
- Instead we start by programming the kind of **repetitive task** that computers are good at.
- A *brute force* approach to problem solving involves **exhaustively** searching through the space of **all** possible action sequences to find one that achieves the goal.

3

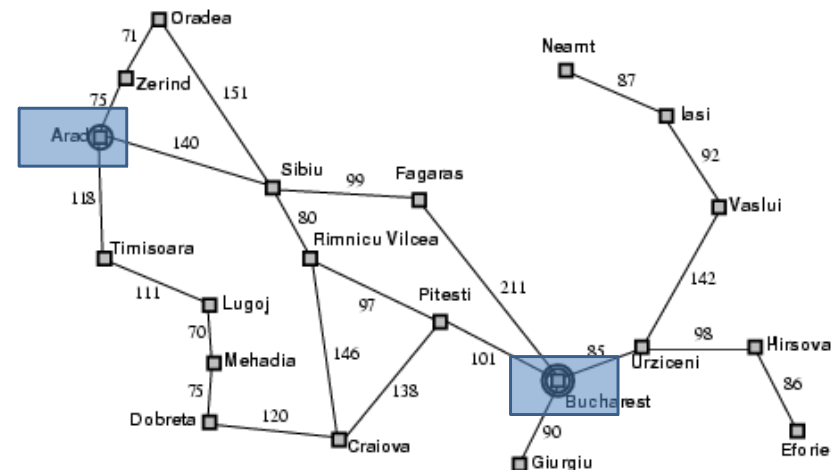
Overview

- **Last time**
 - basic ideas about problem solving;
 - state space;
 - solutions as paths;
 - the notion of solution cost;
 - the importance of using the correct level of abstraction.
- **Today**
 - Automating search
 - Blind (uninformed, brute force) strategies.
- Learning outcome covered today:
Identify, contrast and apply to simple examples the major search techniques that have been developed for problem-solving in AI;

2

Example: Romania Problem

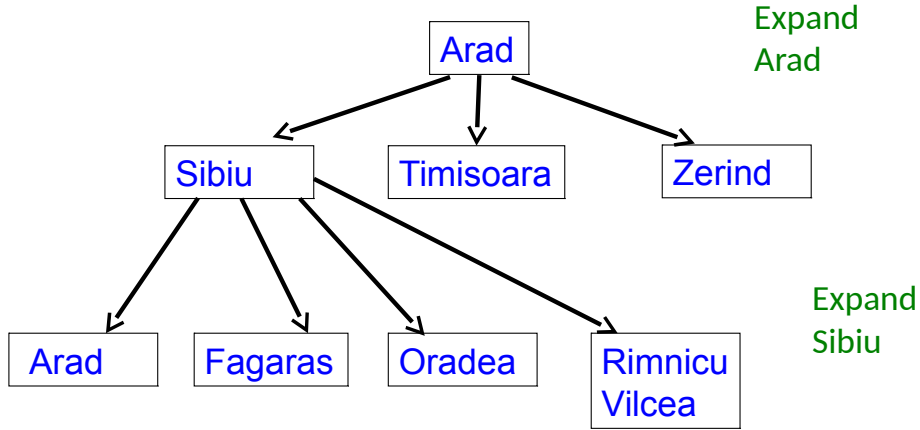
Travel from Arad to Bucharest



4



The Search Tree



Search strategy: how do we choose which node to expand?

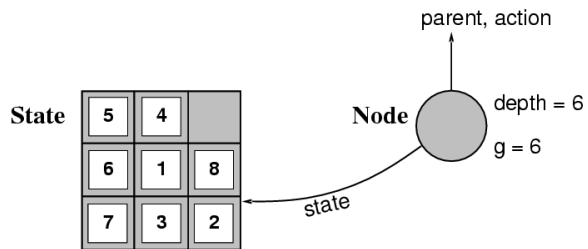
Search Tree Exploration



- The tree is built by taking the **initial** state and identifying the states that can be obtained by a single application of the **operators/actions** available.
- These new states become the **children** of the initial state in the tree.
- These new states are then examined to see if they are the **goal** state.
- If not, the process is **repeated** on the new states.
- We can formalise this description by giving an algorithm for it.
- We have different algorithms for different **choices** of nodes to expand.

Implementation: States vs. Nodes

- A **state** is a (representation of) a physical configuration.
- A **node** is a data structure constituting part of a search tree that includes **state**, **parent node**, **action**, **path cost $g(x)$** , **depth**.



Expanding the tree creates new nodes, filling in the various fields and creating the corresponding states.

General Algorithm for Search

```

agenda = [initial state];
while agenda not empty do
  pick node from agenda;
  new nodes = apply operations to state;
  if goal state in new nodes then
    return solution;
  else add new nodes to agenda;

```

- Question: How to pick states for expansion?
- Two obvious strategies:
 - **depth** first search;
 - **breadth** first search.

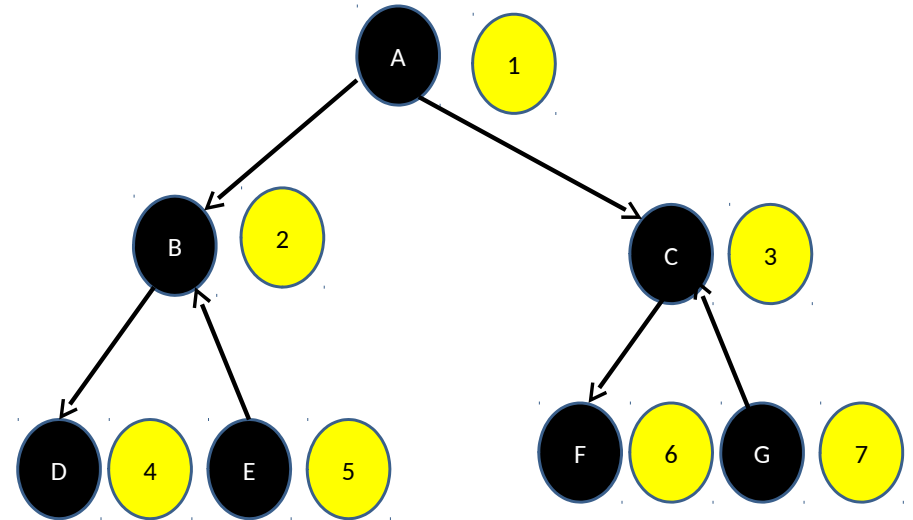
Breadth First Search



- Start by expanding initial state - gives tree of depth 1.
- Then expand **all** nodes that resulted from previous step – gives tree of depth 2.
- Then expand **all** nodes that resulted from previous step, and so on.
- Expand nodes all at depth n **before** going to level $n + 1$.

9

Breadth First Search



10

General Breadth First Search

```

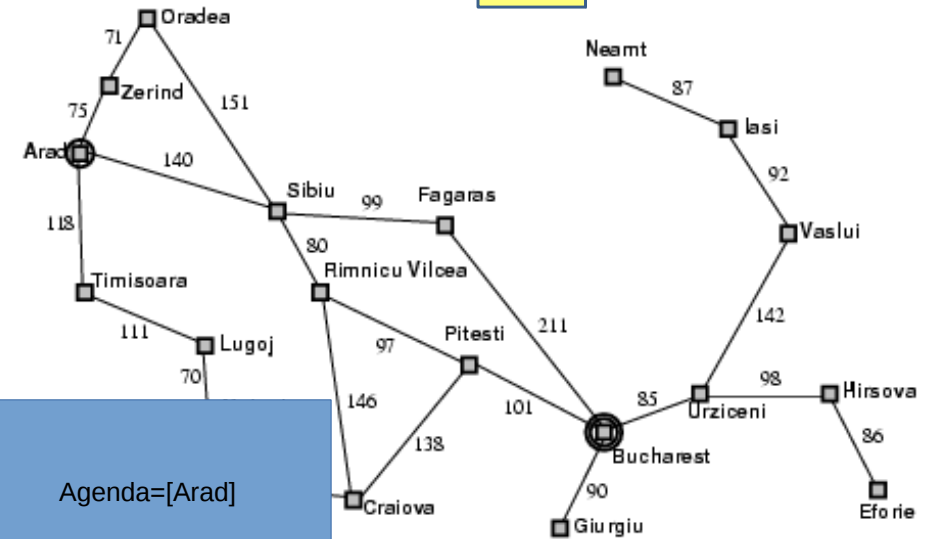
/* Breadth first search */
agenda = [initial state];
while agenda not empty do
  pick node from front of agenda;
  new nodes = apply operations to state;
  if goal state in new nodes then
    return solution;
  else APPEND new nodes to END of agenda
  
```

11

Example: Romania BFS

Travel from Arad to Bucharest

D=0

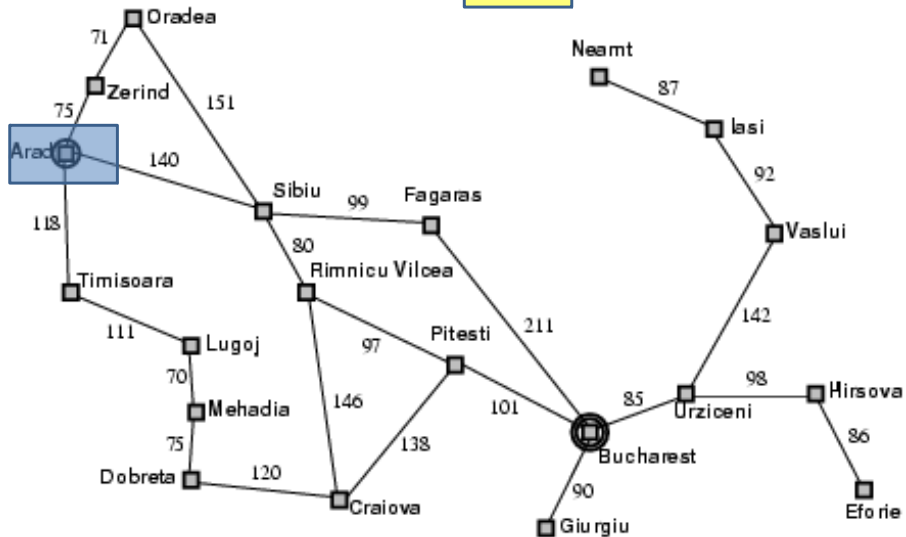


12

Example: Romania BFS

Travel from Arad to Bucharest

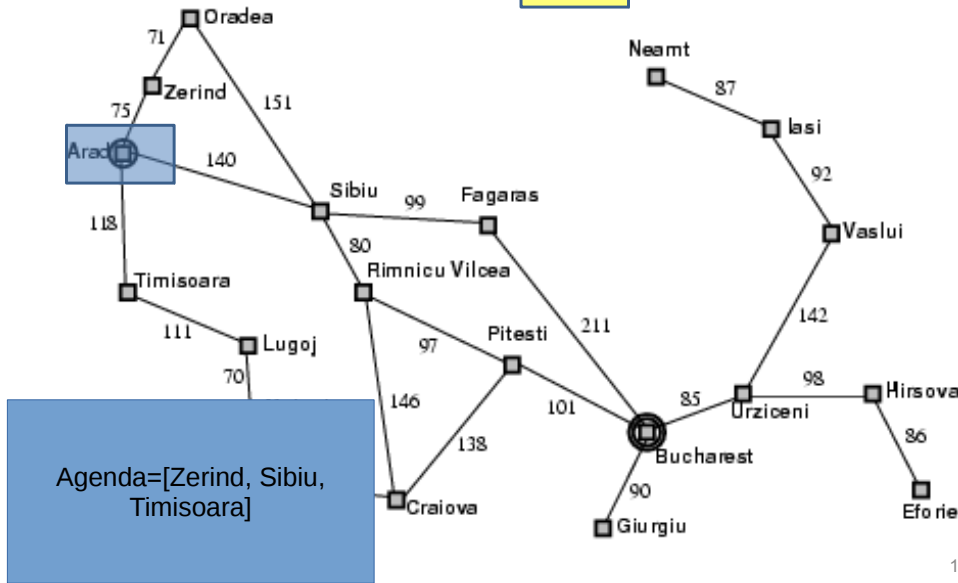
D=0



Example: Romania BFS

Travel from Arad to Bucharest

D=0

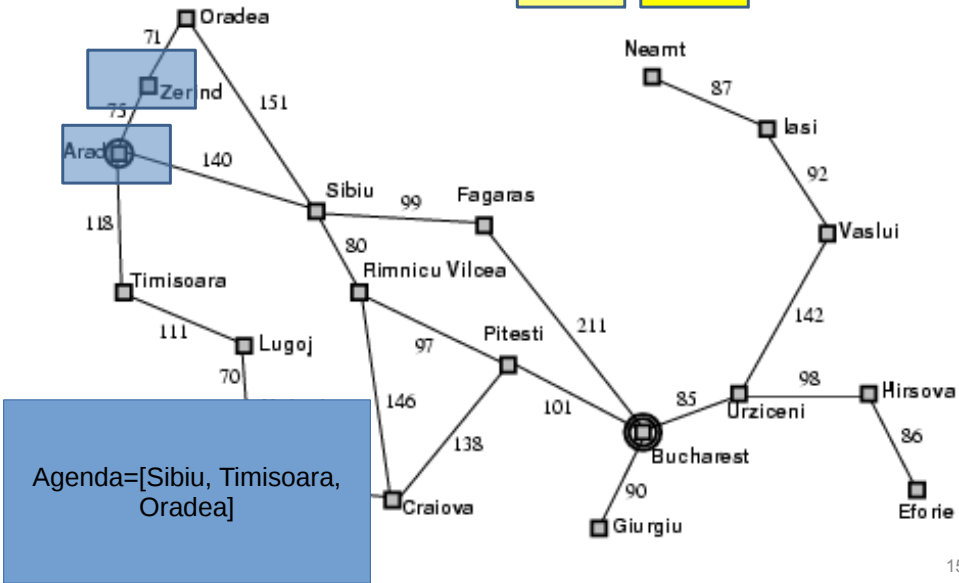


Example: Romania BFS

Travel from Arad to Bucharest

D=0

D=1

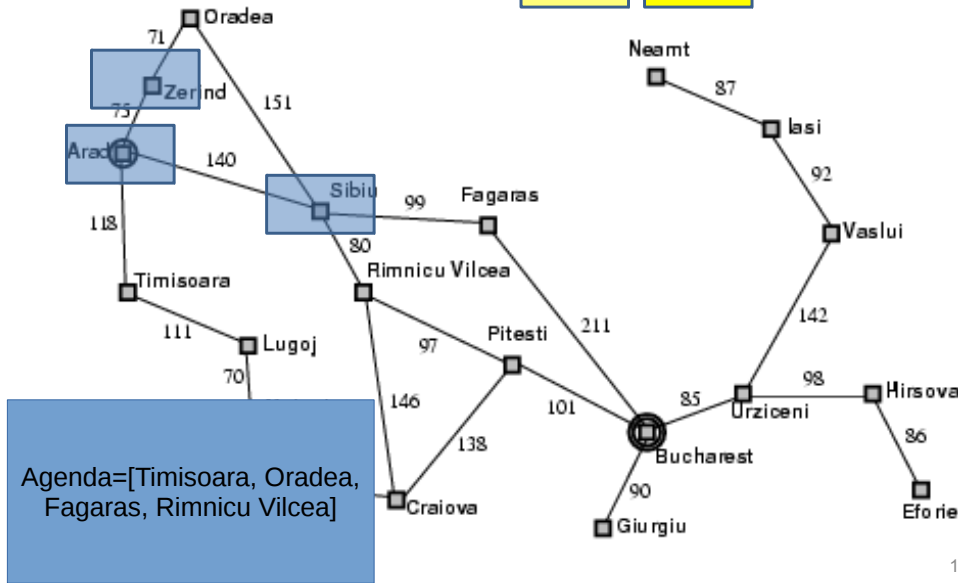


Example: Romania BFS

Travel from Arad to Bucharest

D=0

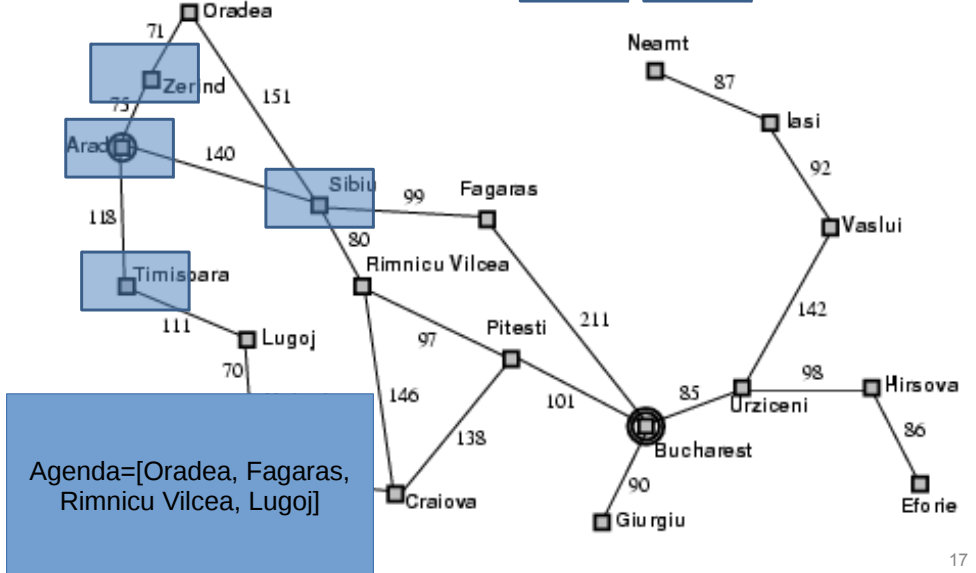
D=1



Example: Romania BFS

Travel from Arad to Bucharest

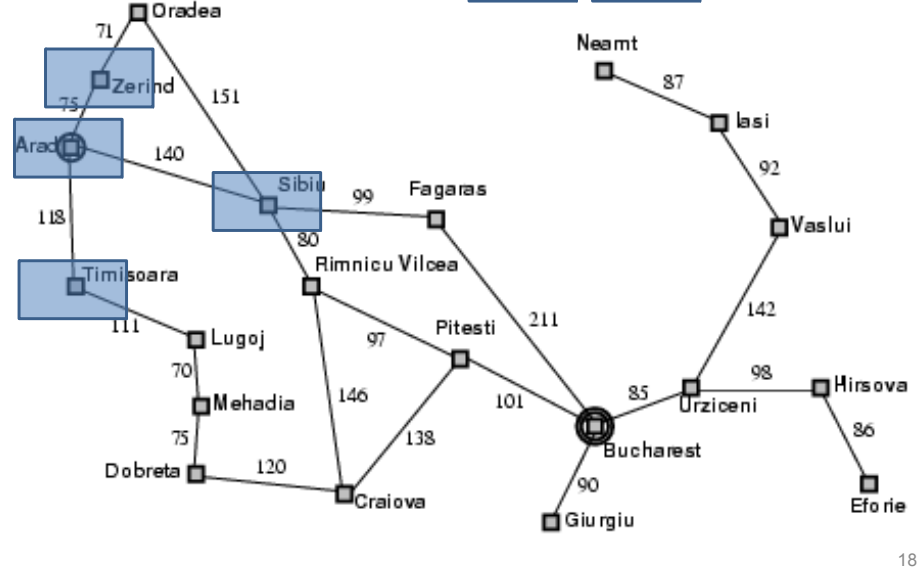
D=0 D=1



Example: Romania BFS

Travel from Arad to Bucharest

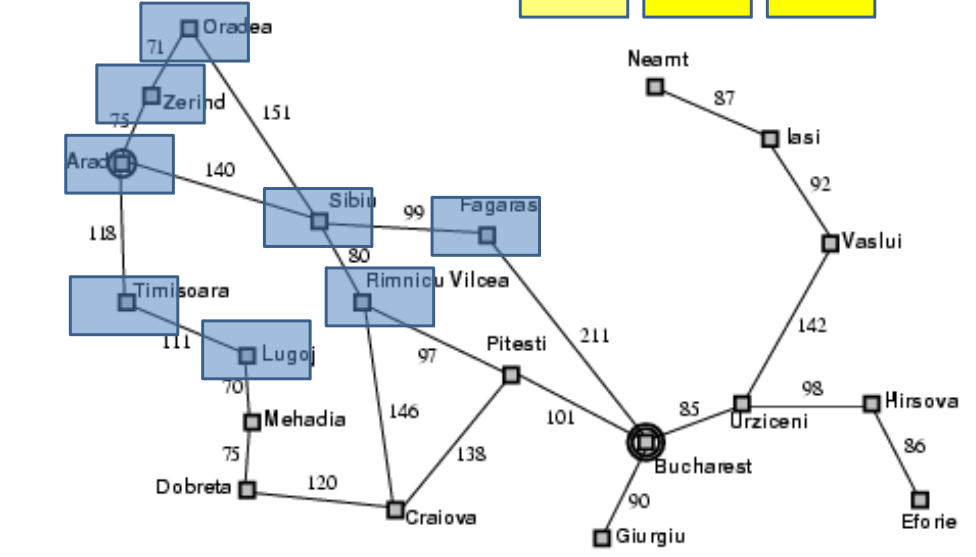
D=0 D=1



Example: Romania BFS

Travel from Arad to Bucharest

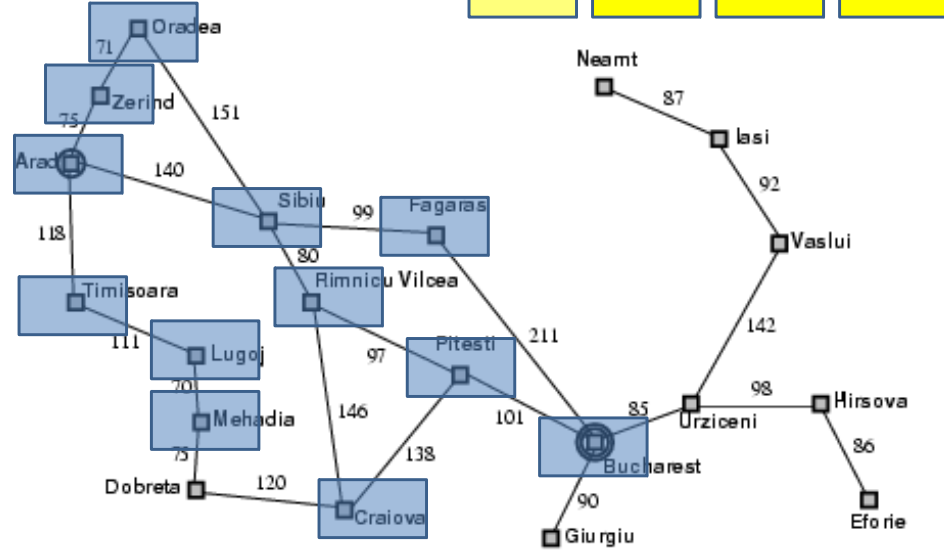
D=0 D=1 D=2



Example: Romania BFS

Travel from Arad to Bucharest

D=0 D=1 D=2 D=3



Properties of Breadth First Search

- **Advantage:** *guaranteed* to reach a solution if one exists.
- Finds the **shortest** (cheapest) solution in terms of the number of operations applied to reach a solution.
- **Disadvantage:** time taken to reach solution.
 - Let b be branching factor - average number of operations that may be performed from any level.
 - If solution occurs at depth d , then we will look at $b + b^2 + b^3 + \dots + b^d$ nodes before reaching solution - **exponential**.
 - The memory requirement is b^d

21



Depth First Search

- Start by expanding initial state.
- Pick **one** of nodes resulting from 1st step, and expand it.
- Pick **one** of nodes resulting from 2nd step, and expand it, and so on.
- Always expand *deepest node*.
- Follow one “branch” of search tree.

23

Complexity

Depth	Nodes	Time
2	110	0.11 msec
4	11,110	11 msec
6	10^6	1.1 sec
8	10^8	2 mins
10	10^{10}	3 hours
12	10^{12}	13 days
14	10^{14}	3.5 years
16	10^{16}	350 years

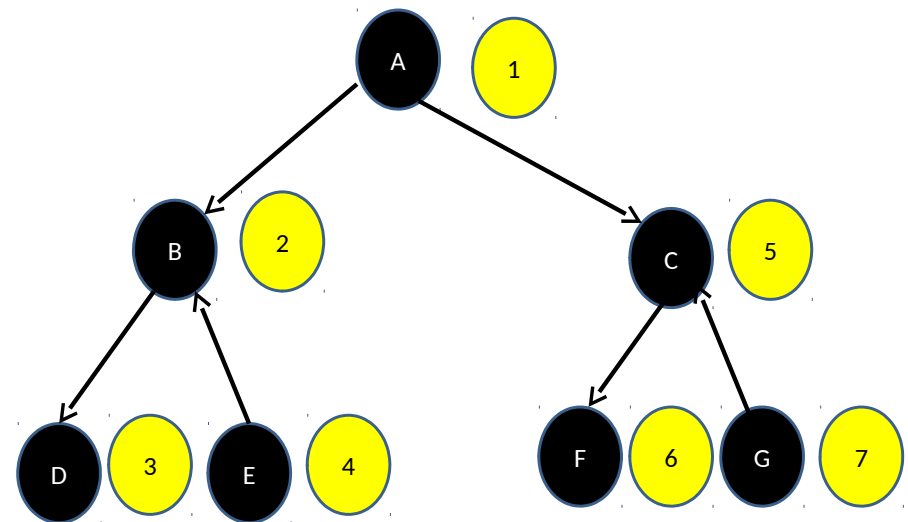
Time for BFS assuming a branching factor of 10 and 1 million nodes expanded per second.

Combinatorial Explosion !



22

Depth First Search



24

General Depth First Search

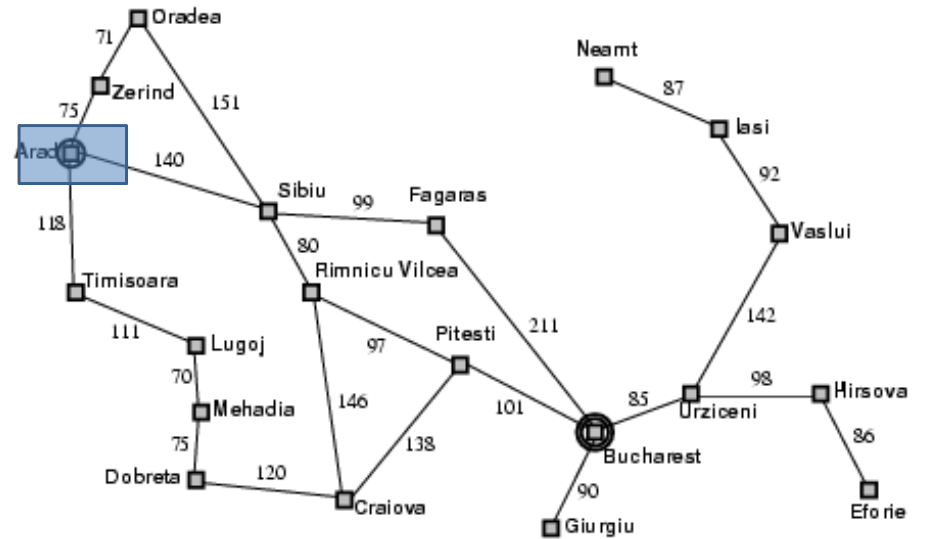
```

/* Depth first search */
agenda = [initial state];
while agenda not empty do
  pick node from front of agenda;
  new nodes = apply operations to state;
  if goal state in new nodes then
    return solution;
  else put new nodes on FRONT of agenda;

```

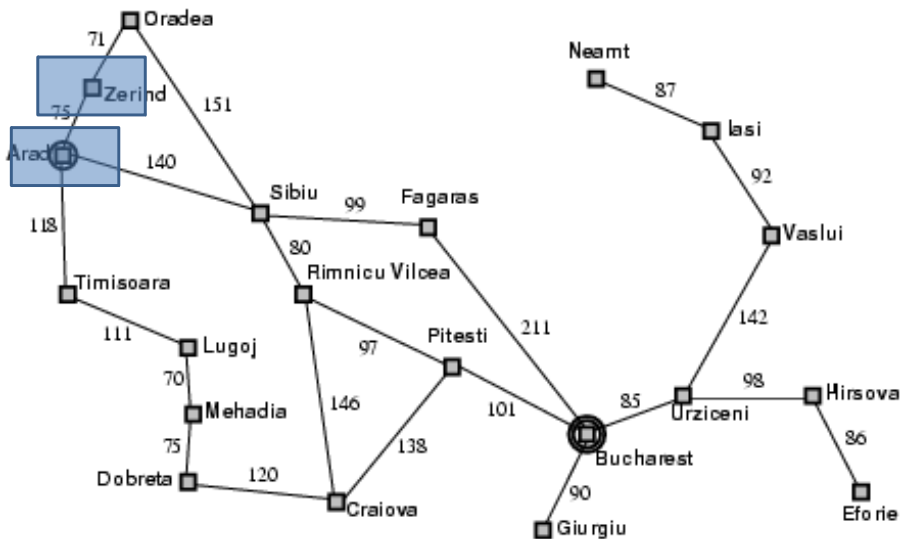
Example: Romania DFS

Travel from Arad to Bucharest



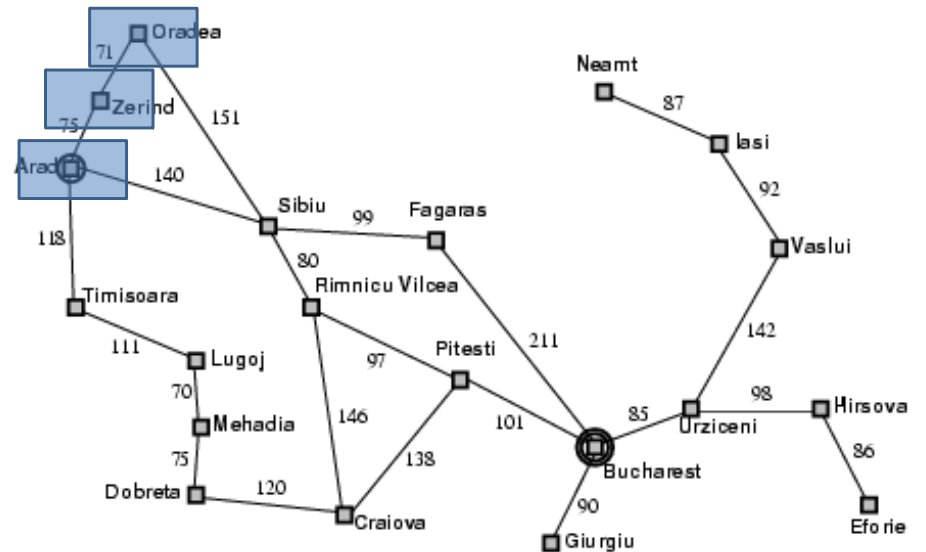
Example: Romania DFS

Travel from Arad to Bucharest



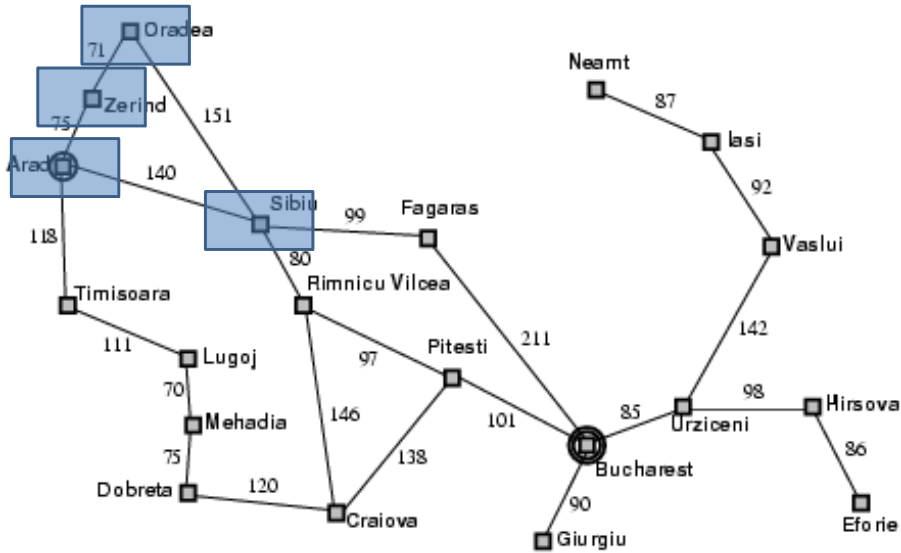
Example: Romania DFS

Travel from Arad to Bucharest



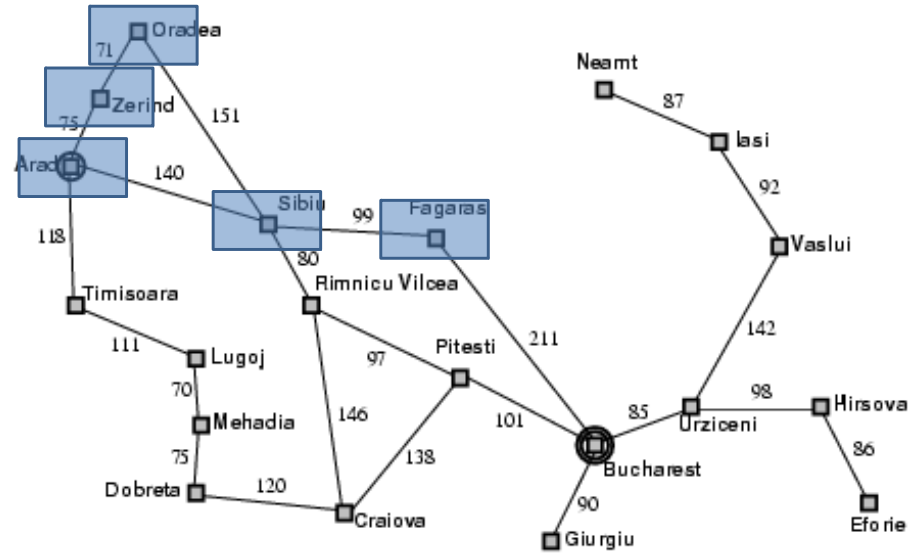
Example: Romania DFS

Travel from Arad to Bucharest



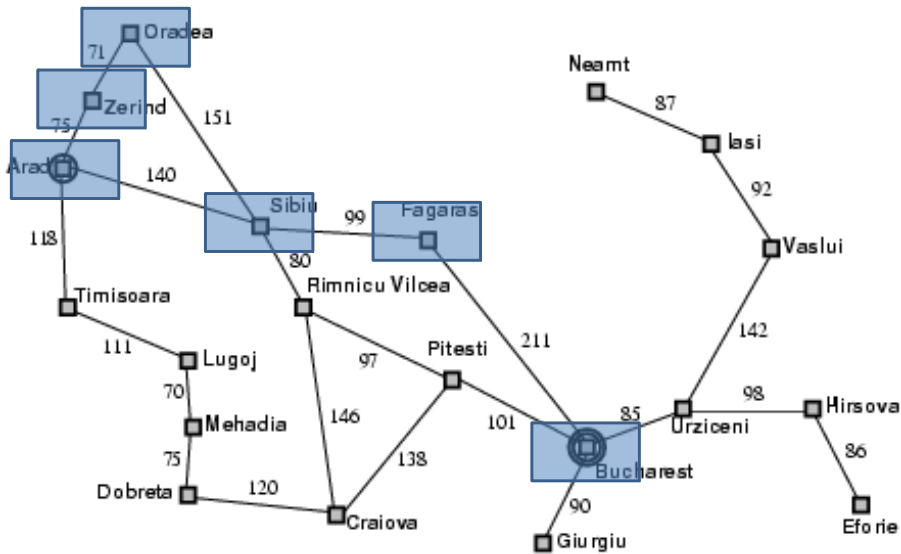
Example: Romania DFS

Travel from Arad to Bucharest



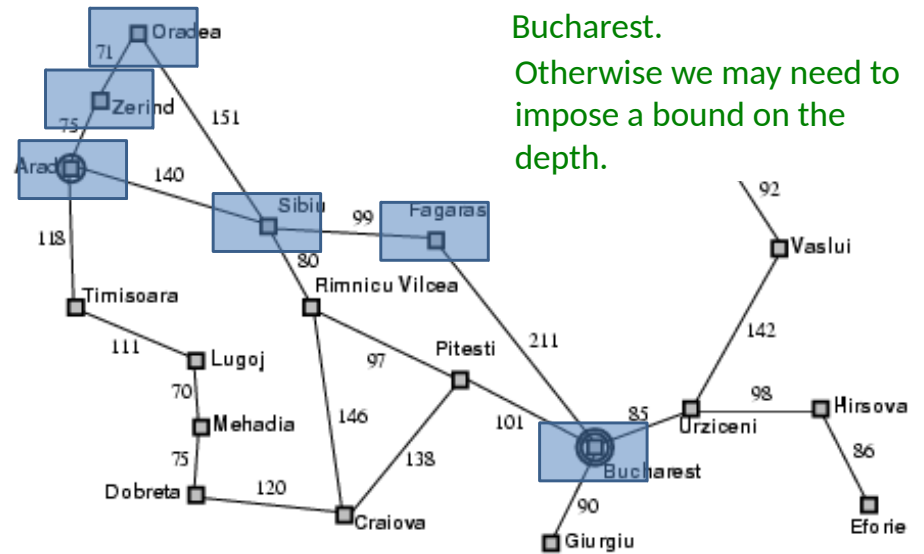
Example: Romania DFS

Travel from Arad to Bucharest



Example: Romania DFS

Travel from Arad to Bucharest



OK when all roads lead to Bucharest.
Otherwise we may need to impose a bound on the depth.

Properties of Depth First Search

- Depth first search is guaranteed to find a solution if one exists, unless there are **infinite** paths.
- Solution found is **not** guaranteed to be the **best**.
- The amount of time taken is **usually** much less than breadth first search.
- Memory requirement is **always** much less than breadth first search.
- For branching factor b and maximum depth of the search tree m , depth-first search requires the storage of only bm nodes.

33

Exercise

- Consider a state space where the start state is number 1 and the successor function for state n returns two states, numbers $2n$ and $2n+10$
 - 1) Draw the portion of the state space for the first 15 states.
 - 2) Suppose the goal state is 38. List the order in which the nodes will be visited for both breadth first search and depth first search.

Summary: Basic Search Strategies

- Introduced:
 - Breadth-first search: **complete** but **expensive**.
 - Depth-first search: **cheap** but **completeness not guaranteed**.
- **Next time**
 - More advanced search strategies

35