浙江大学伊利诺伊大学厄巴纳香槟校区联合学院
Zhejiang University-University of Illinois at Urbana Champaign Institute

ECE 448: Artificial Intelligence

Lecture 4: Search Intro

Prof. Hongwei Wang hongweiwang@intl.zju.edu.cn

Prof. Mark Hasegawa-Johnson jhasegaw@illinois.edu

Spring 2023

Review: Types of agents

- Reflex agent

- Consider how the world IS
- Choose action based on current percept
- Do not consider the future consequences of actions

- Goal-directed agent

- Consider how the world WOULD BE
- Decisions based on (hypothesized) consequences of actions
- Must have a model of how the world evolves in response to actions
- Must formulate a goal

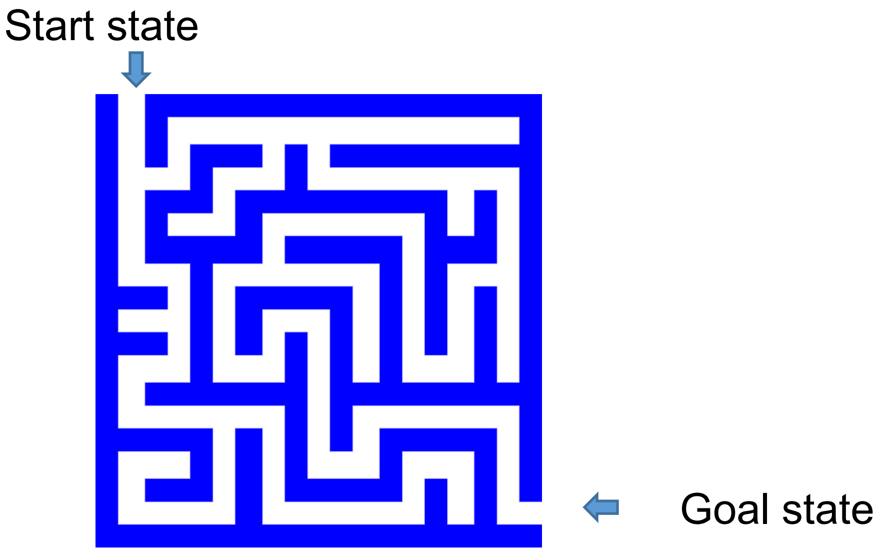
Source: D. Klein, P. Abbeel

Outline

Outline

1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: a frontier list
    2. Second data structure: a search tree
    3. Third data structure: a “visited states” list
3. Depth-first search: very fast, but not guaranteed
4. Breadth-first search: guaranteed optimal
5. Uniform cost search = Dijkstra’s algorithm = BFS with variable costs

We will consider the problem of designing **goal-based agents** in **fully observable**, **deterministic**, **discrete**, **static**, **known** environments



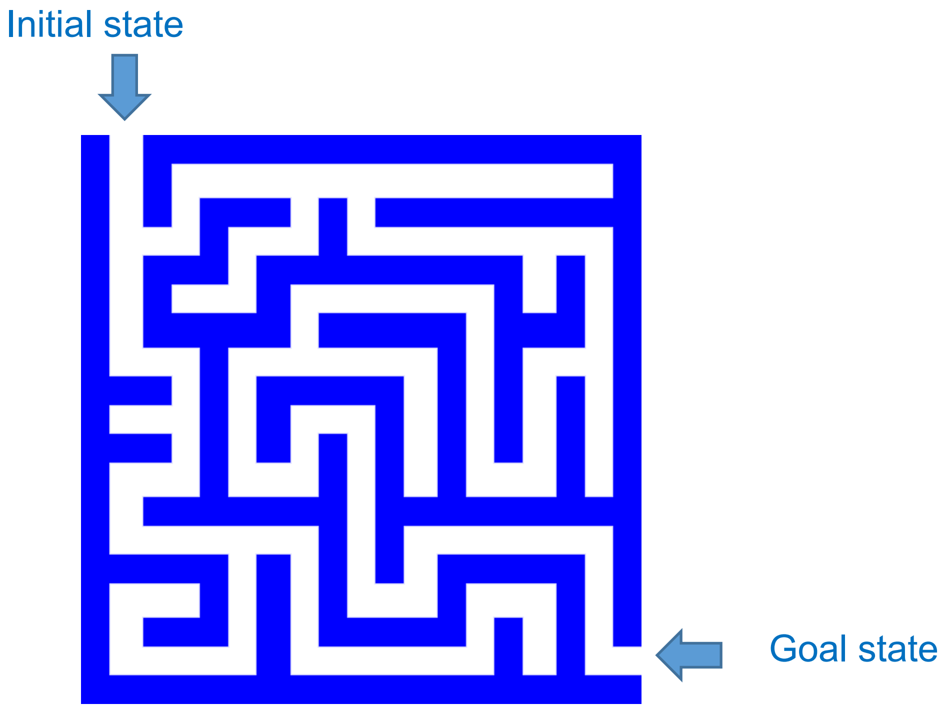
We will consider the problem of designing **goal-based agents** in **fully observable, deterministic, discrete, known** environments

- The agent must find a *sequence of actions* that reaches the goal
- The **performance measure** is defined by (a) reaching the goal and (b) how "expensive" the path to the goal is
    - The **agent doesn't know** the performance measure. This is a goal-directed agent, not a utility-directed agent
    - The **programmer (you) DOES know** the performance measure. So you design a goal-seeking strategy that minimizes cost.
- We are focused on the process of finding the solution; while executing the solution, we assume that the agent can safely ignore its percepts (**static environment, open-loop system**)

Search problem components

ZJUI

- **Initial state**
- **Actions**
- **Transition model**
  - What state results from performing a given action in a given state?
- **Goal state**
- **Path cost**
  - Assume that it is a sum of nonnegative *step costs*



- The **optimal solution** is the sequence of actions that gives the *lowest* path cost for reaching the goal

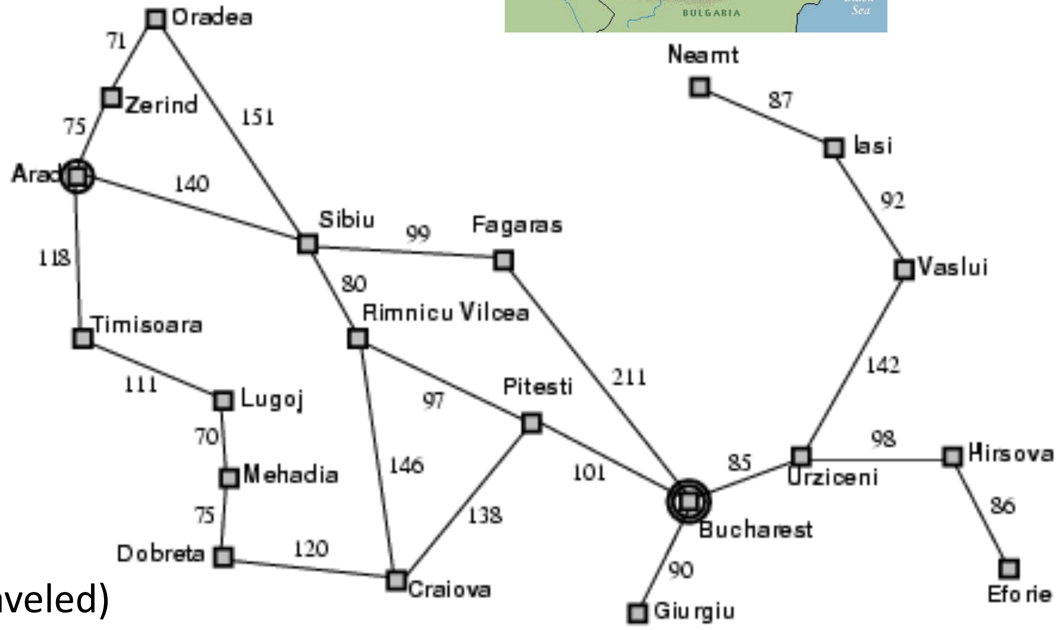
Knowledge Representation: State

- State = description of the world
  - Must have enough detail to decide whether or not you're currently in the **initial state**
  - Must have enough detail to decide whether or not you've reached the **goal state**
  - Often but not always: "defining the state" and "defining the transition model" are the same thing

Example: Romania

- On vacation in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

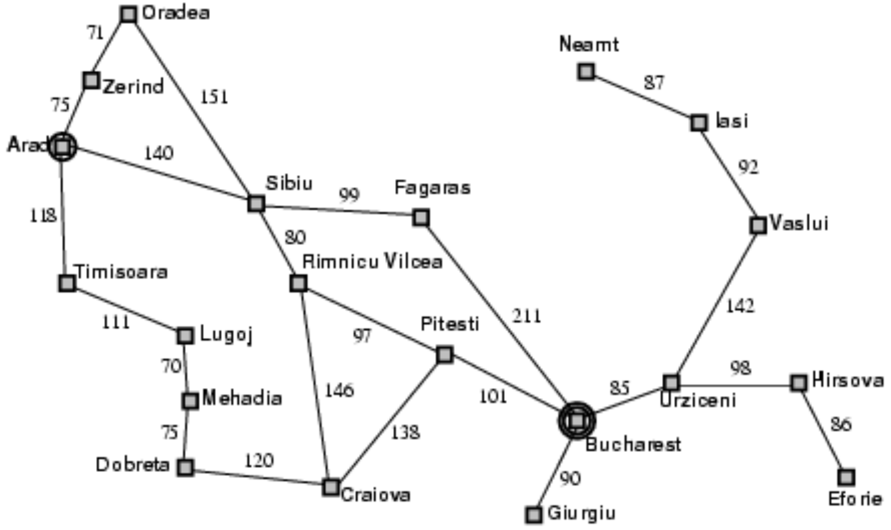
- **Initial state**
  - Arad
- **Actions**
  - Go from one city to another
- **Transition model**
  - If you go from city A to city B, you end up in city B
- **Goal state**
  - Bucharest
- **Path cost**
  - Sum of edge costs (total distance traveled)



State space

ZJUI

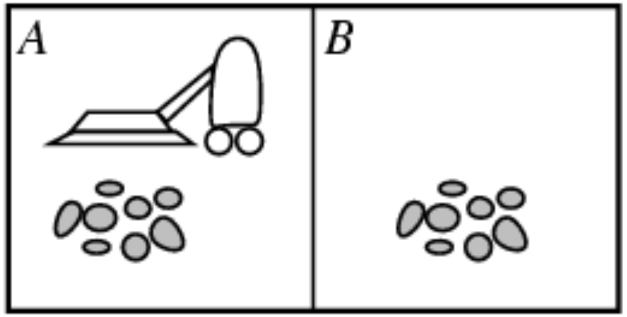
- The initial state, actions, and transition model define the **state space** of the problem
  - The set of all states reachable from initial state by any sequence of actions
  - Can be represented as a **directed graph** where the nodes are states and links between nodes are actions
- What is the state space for the Romania problem?
  - State Space = O{# cities}



Traveling Salesman Problem

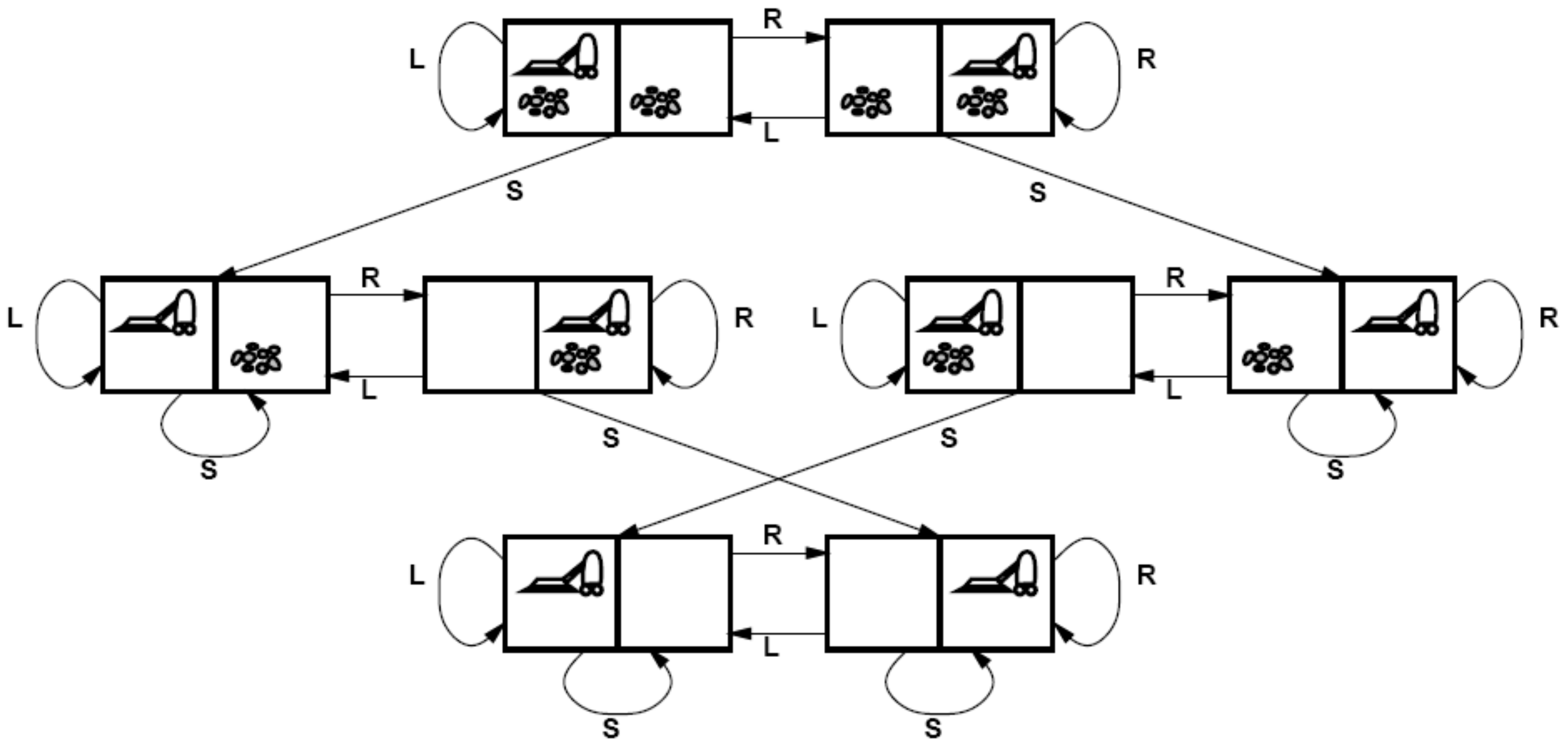
- Goal: visit every city in the United States
- Path cost: total miles traveled
- Initial state: Champaign, IL
- Action: travel from one city to another
- Transition model: when you visit a city, mark it as "visited."
    - State Space = O{2^#cities}

Example: Vacuum world



- **States**
  - Agent location and dirt location
  - How many possible states?
  - What if there are *n* possible locations?
    - The size of the state space grows exponentially with the “size” of the world!
- **Actions**
  - Left, right, suck
- **Transition model**

Vacuum world state space graph



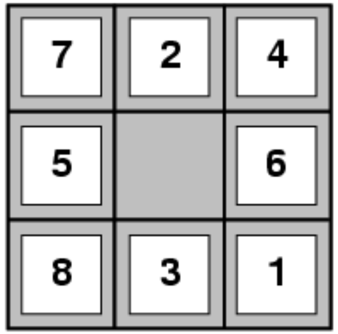
Complexity of the State Space

- Many "video game" style problems can be subdivided:
  - There are M different things your character needs to pick up: 2^M different world states
  - There are N locations you can be in while carrying any subset of those M objects: total number of world states = $O\{2^M N\}$
- Why a maze is nice: you don't need to pick anything up
  - Only N different world states to consider

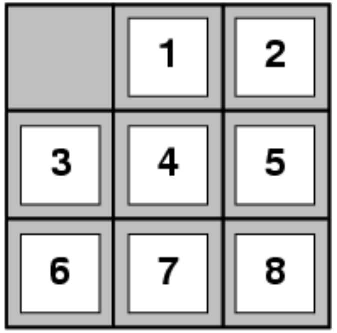
Example: The 8-puzzle

- **States**
  - Locations of tiles
    - 8-puzzle: 181,440 states (9!/2)
    - 15-puzzle: ~10 trillion states
    - 24-puzzle: ~10^{25} states
- **Actions**
  - Move blank left, right, up, down
- **Path cost**
  - 1 per move

- [Finding the optimal solution of n-Puzzle is NP-hard]

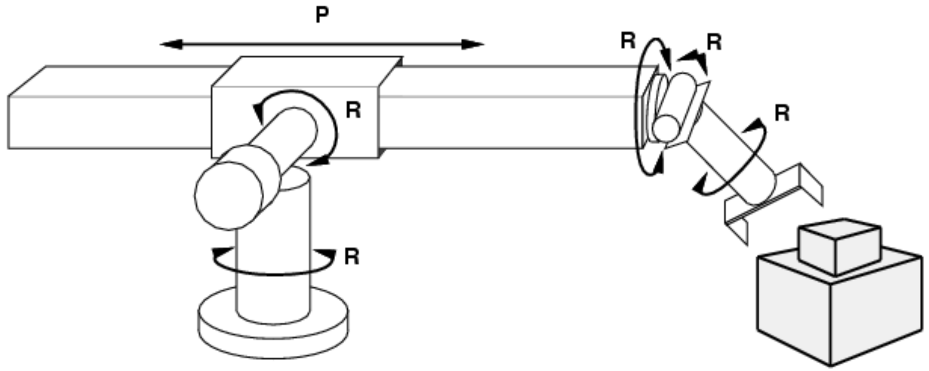


Start State



Goal State

Example: Robot motion planning



- **States**
  - Real-valued joint parameters (angles, displacements)
- **Actions**
  - Continuous motions of robot joints
- **Goal state**
  - Configuration in which object is grasped
- **Path cost**
  - Time to execute, smoothness of path, etc.

Outline

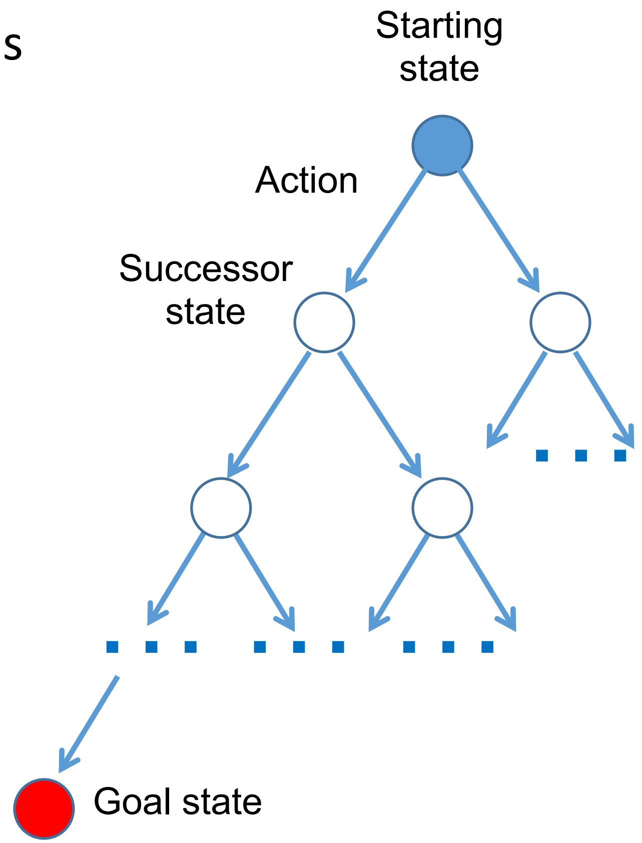
1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. <u>General algorithm for solving search problems</u>
    1. <u>First data structure: a frontier list</u>
    2. <u>Second data structure: a search tree</u>
    3. <u>Third data structure: a “visited states” list</u>
3. Depth-first search: very fast, but not guaranteed
4. Breadth-first search: guaranteed optimal
5. Uniform cost search = Dijkstra’s algorithm = BFS with variable costs

First data structure: a frontier list

- Let’s begin at the start state and **expand** it by making a list of all possible successor states
- Maintain a **frontier** or a list of unexpanded states
- At each step, pick a state from the frontier to expand:
  - Check to see if it’s a goal state
  - If not, find the other states that can be reached from this state, and add them to the frontier, if they’re not already there
- Keep going until you reach a goal state

Second data structure: a search tree

- "What if" tree of sequences of actions and outcomes
- The root node corresponds to the starting state
- The children of a node correspond to the **successor states** of that node's state
- A path through the tree corresponds to a sequence of actions
    - A solution is a path ending in the goal state



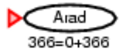
Knowledge Representation: States and Nodes

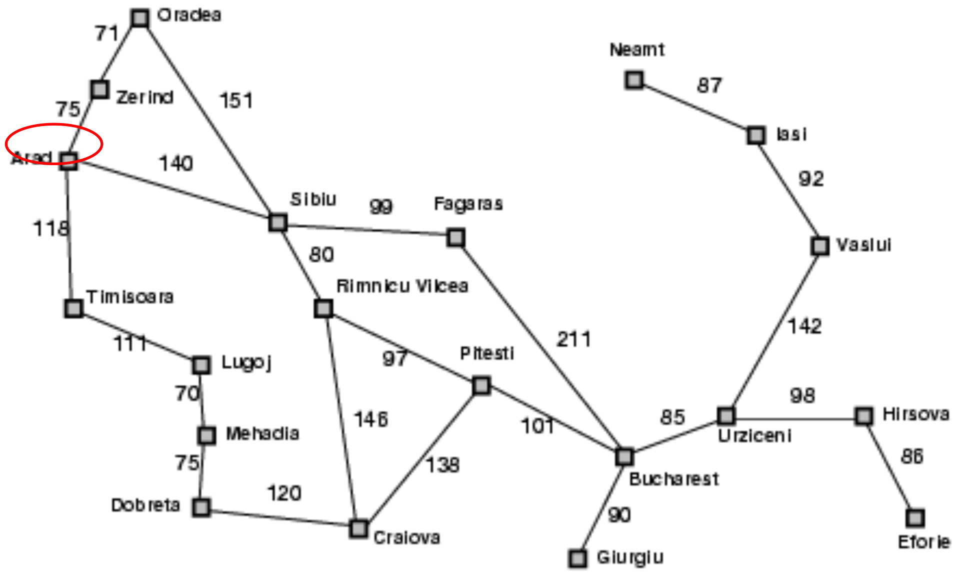
- State = description of the world
  - Must have enough detail to decide whether or not you’re currently in the **initial state**
  - Must have enough detail to decide whether or not you’ve reached the **goal state**
  - Often but not always: “defining the state” and “defining the transition model” are the same thing
- Node = a point in the search tree
  - Private data: ID of the state reached by this node
  - Private data: the ID of the parent node

Tree Search Algorithm Outline

- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the frontier
- **Search strategy** determines
  - Is this process guaranteed to return an optimal solution?
  - Is this process guaranteed to return ANY solution?
  - Time complexity: how much time does it take?
  - Space complexity: how much RAM is consumed by the frontier?
- For now: assume that search strategy = random

Tree search example

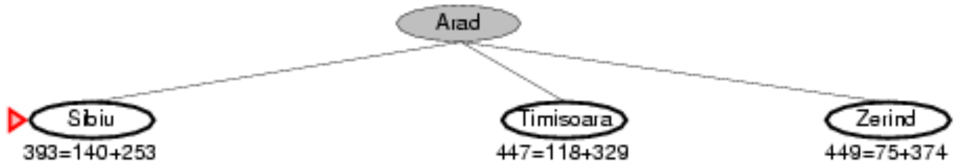


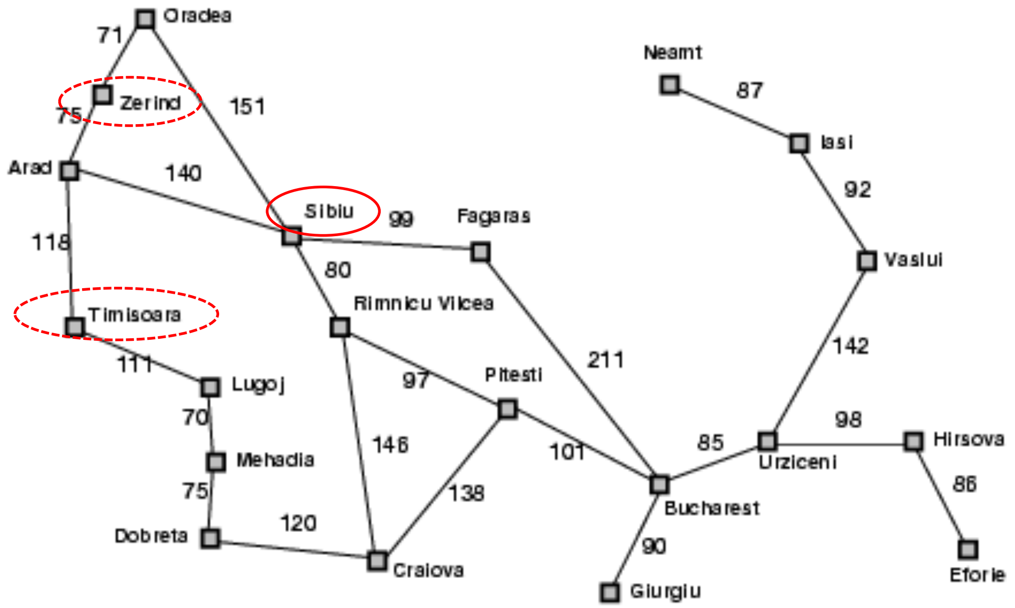


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search example



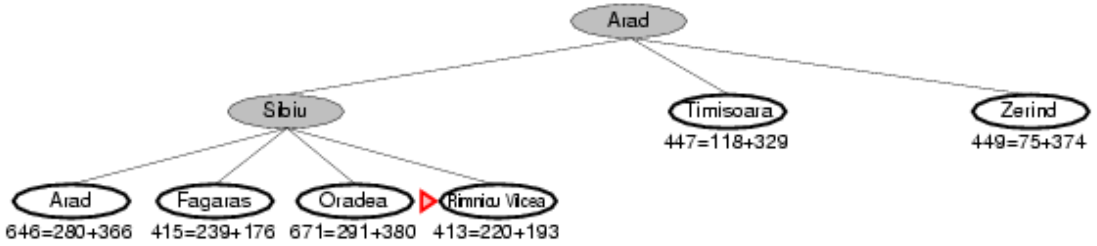


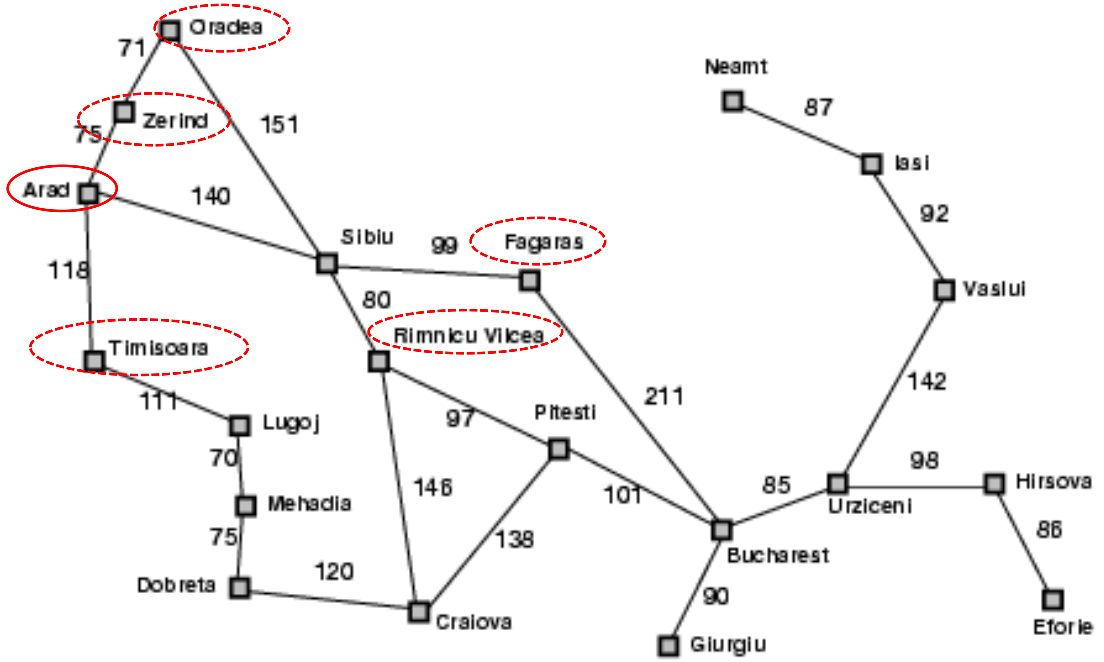
Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search example



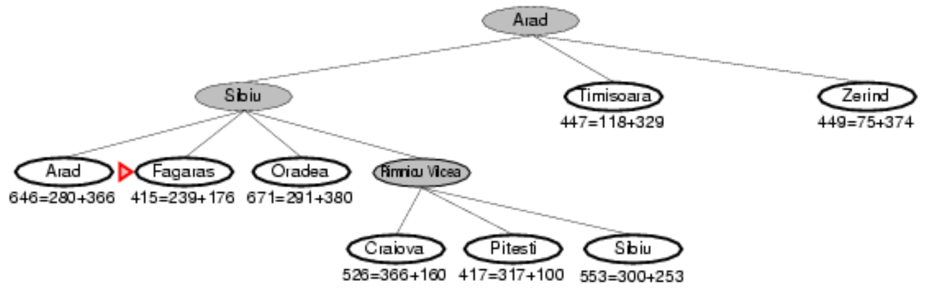


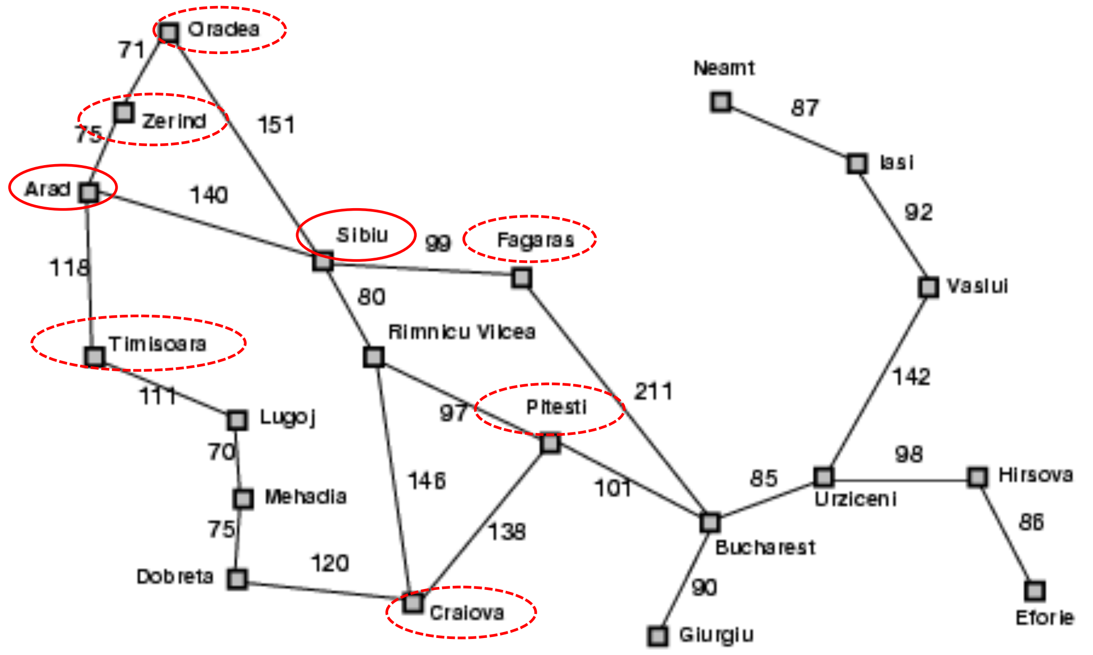
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search example



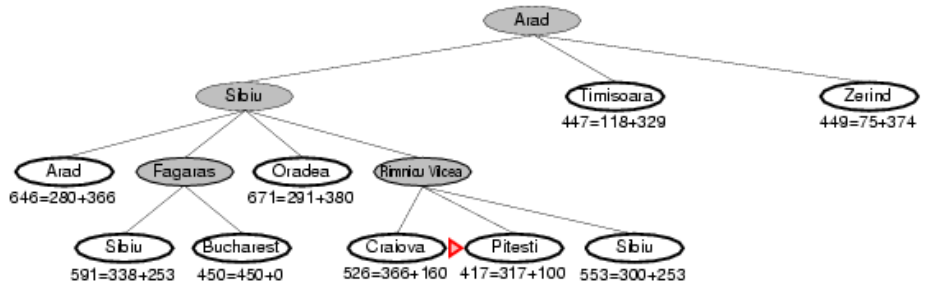


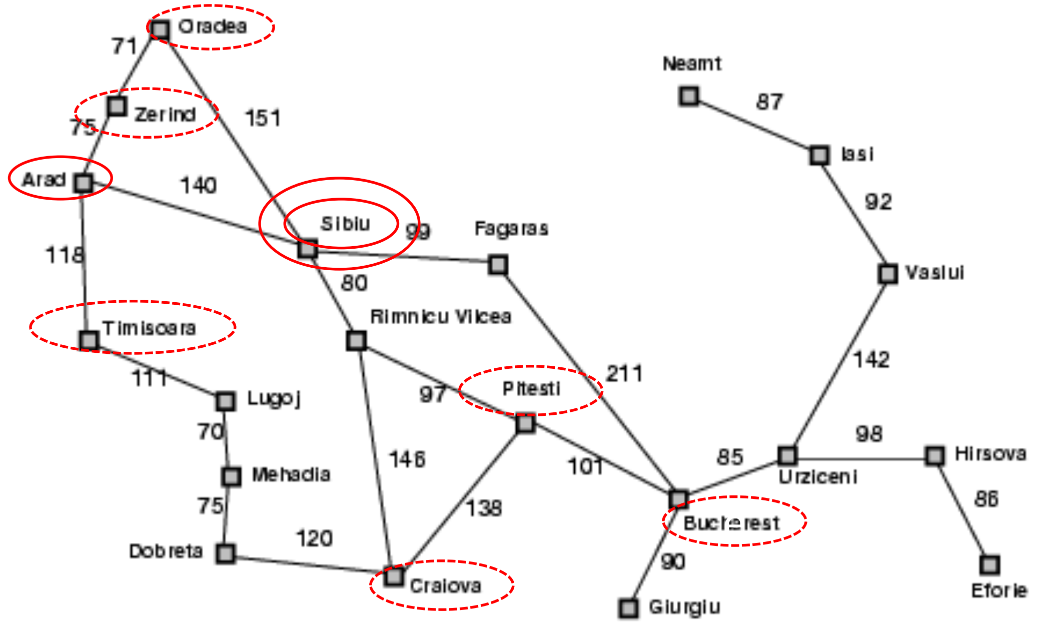
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search example



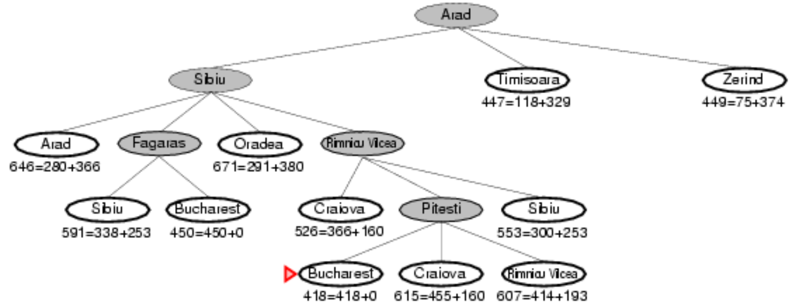


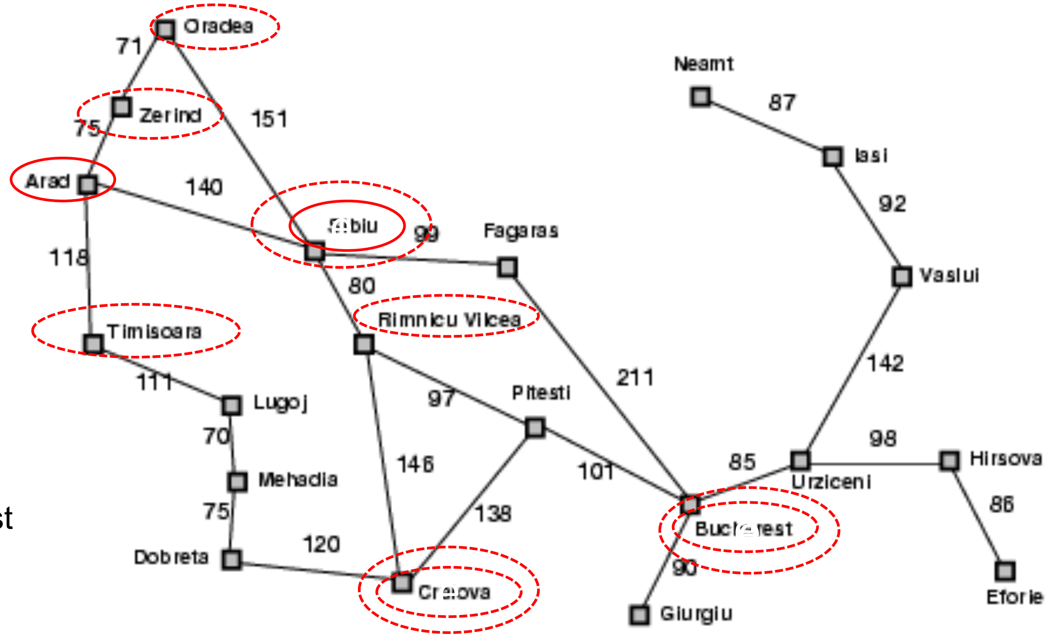
Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search example





Straight–line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Handling repeated states

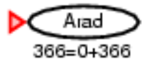
- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
  - Choose a frontier node according to **search strategy** and take it off the frontier
  - If the node contains the **goal state**, return solution
  - Else **expand** the node and add its children to the frontier
- To handle repeated states:
  - Every time you expand a node, add that state to the **explored set**
  - When adding nodes to the frontier, CHECK FIRST to see if they've already been explored

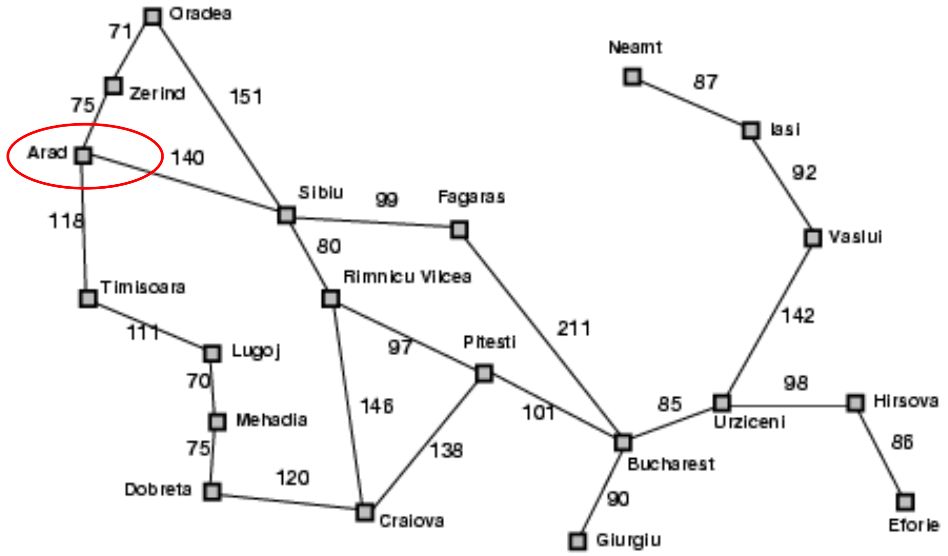
Time Complexity

ZJUI

- Without **explored set** :
  - $O\{1\}$/node
  - $O\{b^m\}$ = # nodes expanded
    - b = branching factor (number of children each node might have)
    - m = length of the longest possible path
- With **explored set** :
  - $O\{1\}$/node using a hash table to see if node is already in **explored set**
  - $O\{|S|\}$ = # nodes expanded
- Usually, $O\{|S|\} < O\{b^m\}$. I'll continue to talk about $O\{b^m\}$, but remember that it's upper-bounded by $O\{|S|\}$.

Tree search w/o repeats

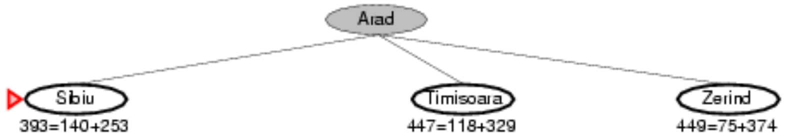




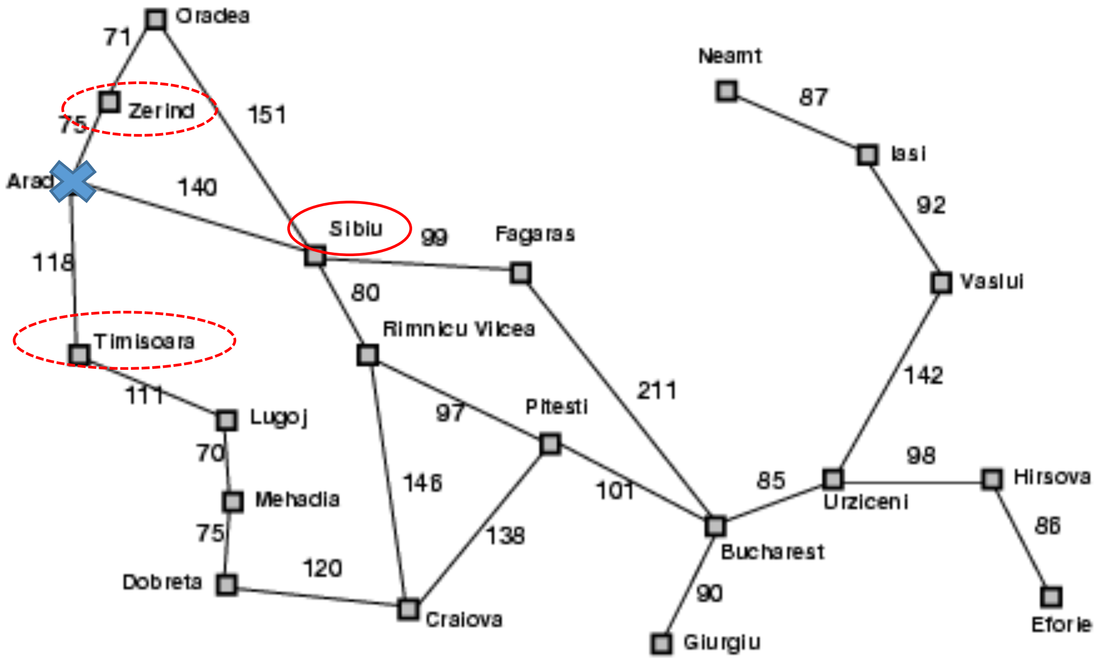
Start: Arad
Goal: Bucharest

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Tree search w/o repeats



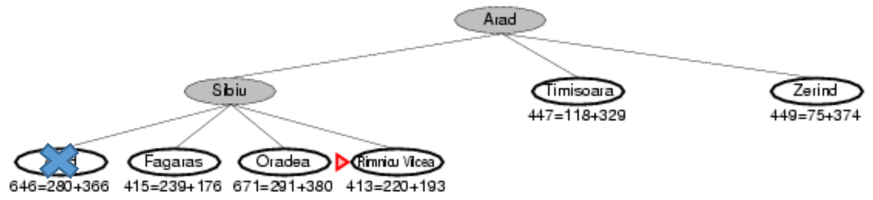
Explored:
Arad



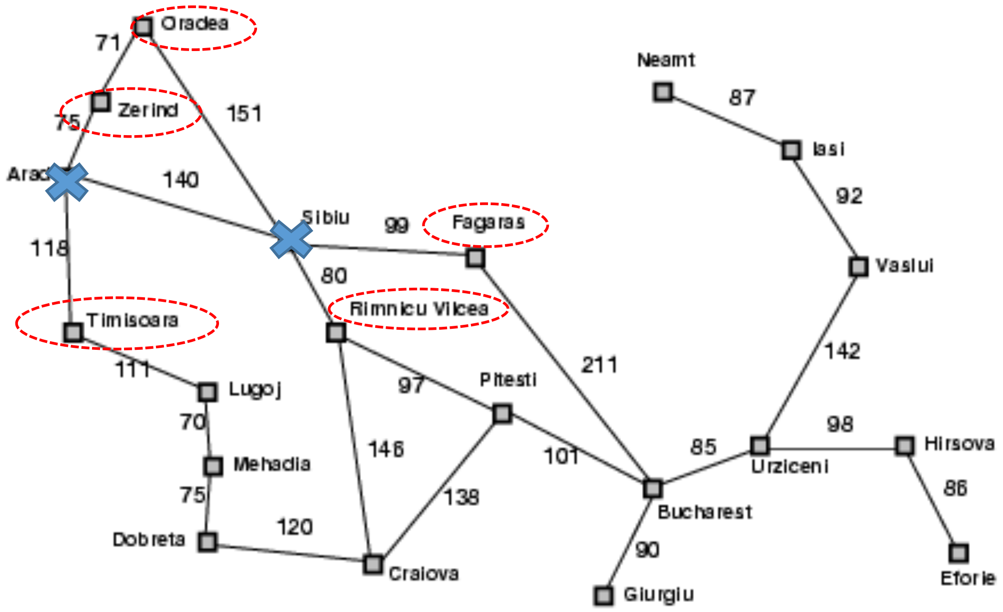
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search w/o repeats



Explored:
Arad
Sibiu

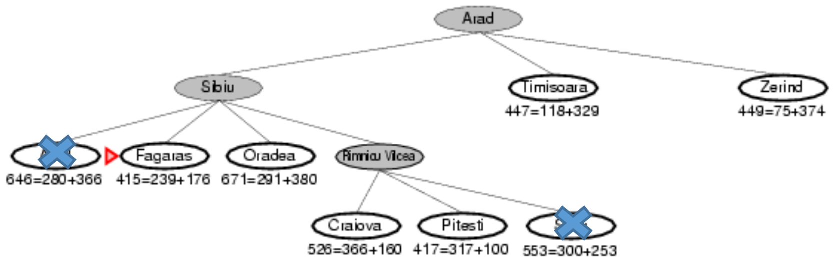


Straight–line distance to Bucharest

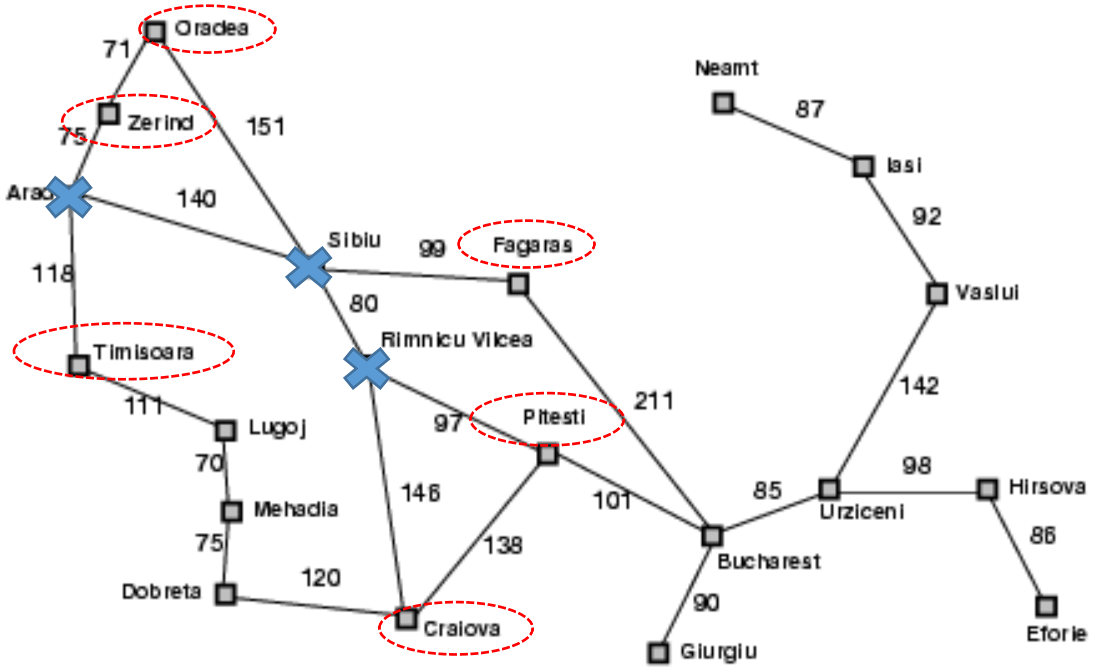
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search w/o repeats



Explored:
Arad
Sibiu
Rimnicu Vilcea



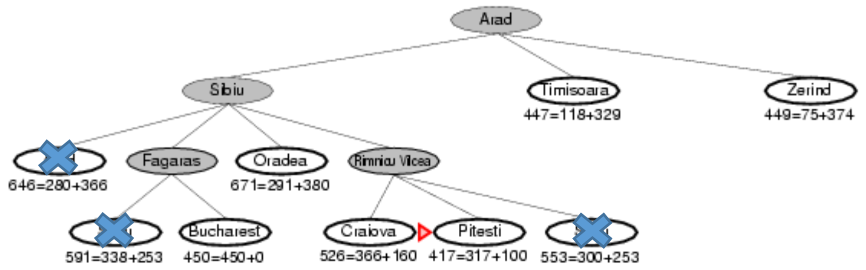
Straight–line distance to Bucharest

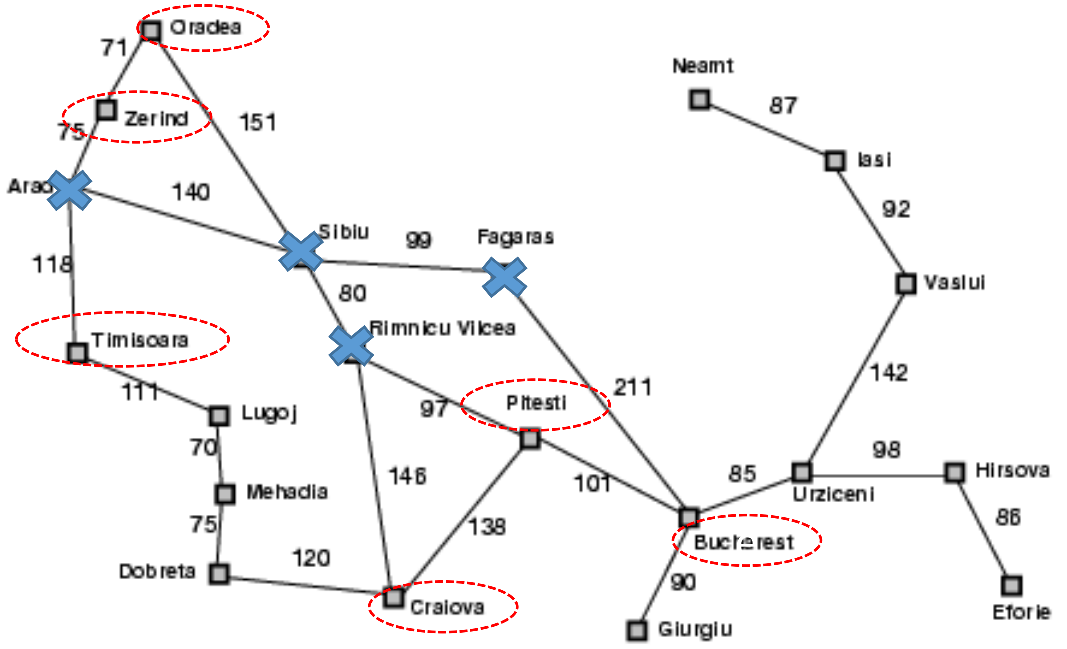
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search w/o repeats

Explored:
Arad
Sibiu
Rimnicu Vilces
Fagaras



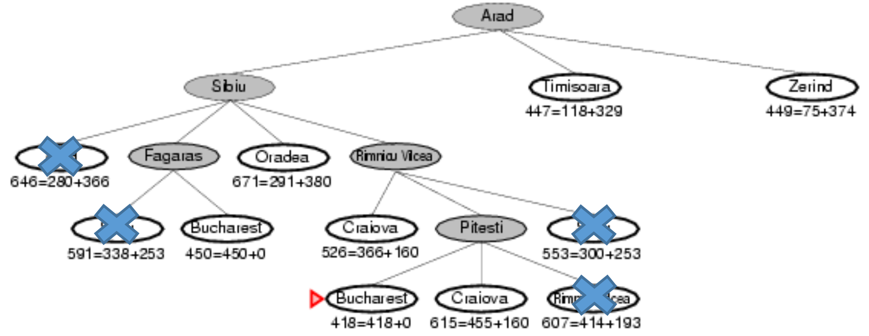


Straight–line distance to Bucharest

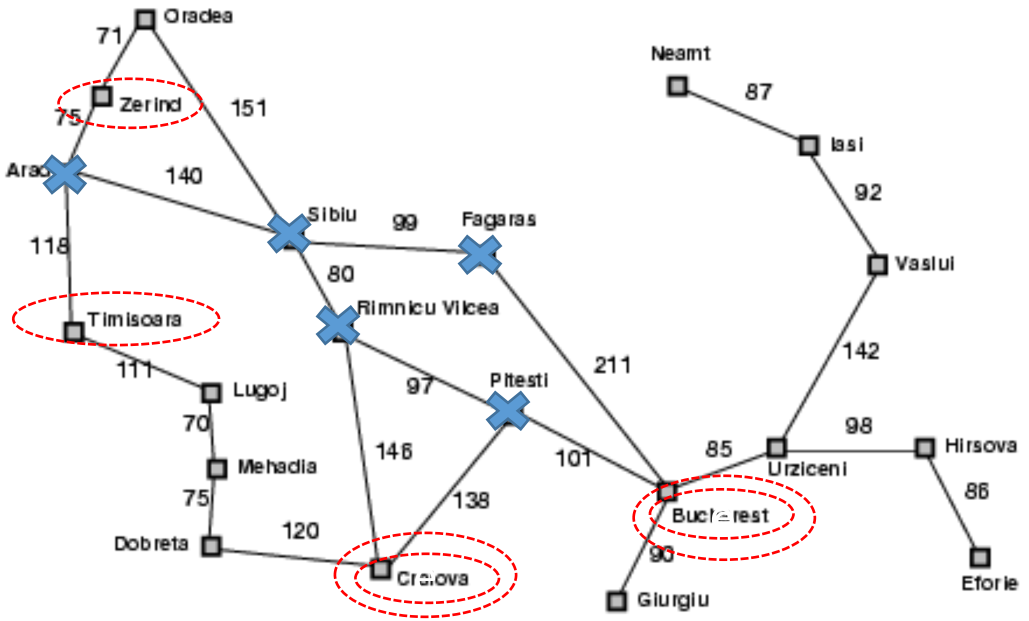
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Tree search w/o repeats



Explored:
Arad
Sibiu
Rinnicu Vilces
Fagaras
Pitesti



Straight–line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Start: Arad
Goal: Bucharest

Outline

ZJUI

1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: a frontier list
    2. Second data structure: a search tree
    3. Third data structure: a “visited states” list
3. **Depth-first search: very fast, but not guaranteed**
4. Breadth-first search: guaranteed optimal
5. Uniform cost search = Dijkstra’s algorithm = BFS with variable costs

Depth-First Search

- Basic idea

try to find a solution as fast as possible

- How:

From the frontier, always choose a node which is

AS FAR FROM THE STARTING POINT AS POSSIBLE

- How:

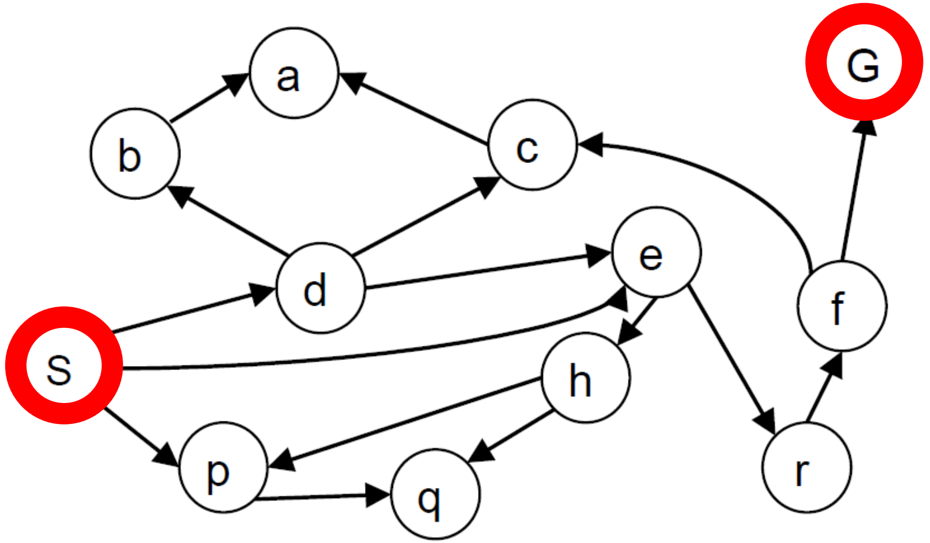
Frontier is a LIFO queue.

The node you expand = whichever node has been most recently placed on the queue.

Depth-first search

- Expand deepest unexpanded node
- Implementation: *frontier* is LIFO (a stack)

Example state space graph for a tiny search problem



Depth-first search

Expansion order:

(s,d,b,a,

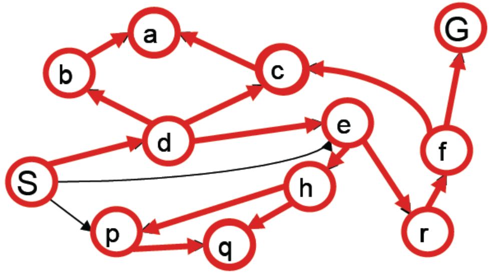
c,a,

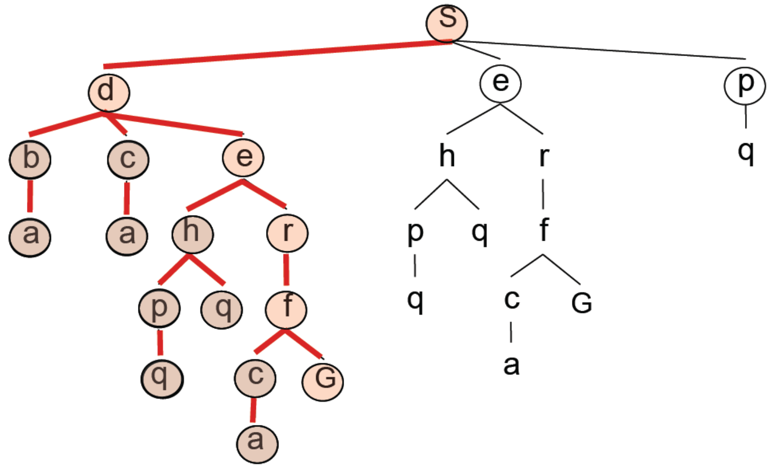
e,h,p,q,

q,

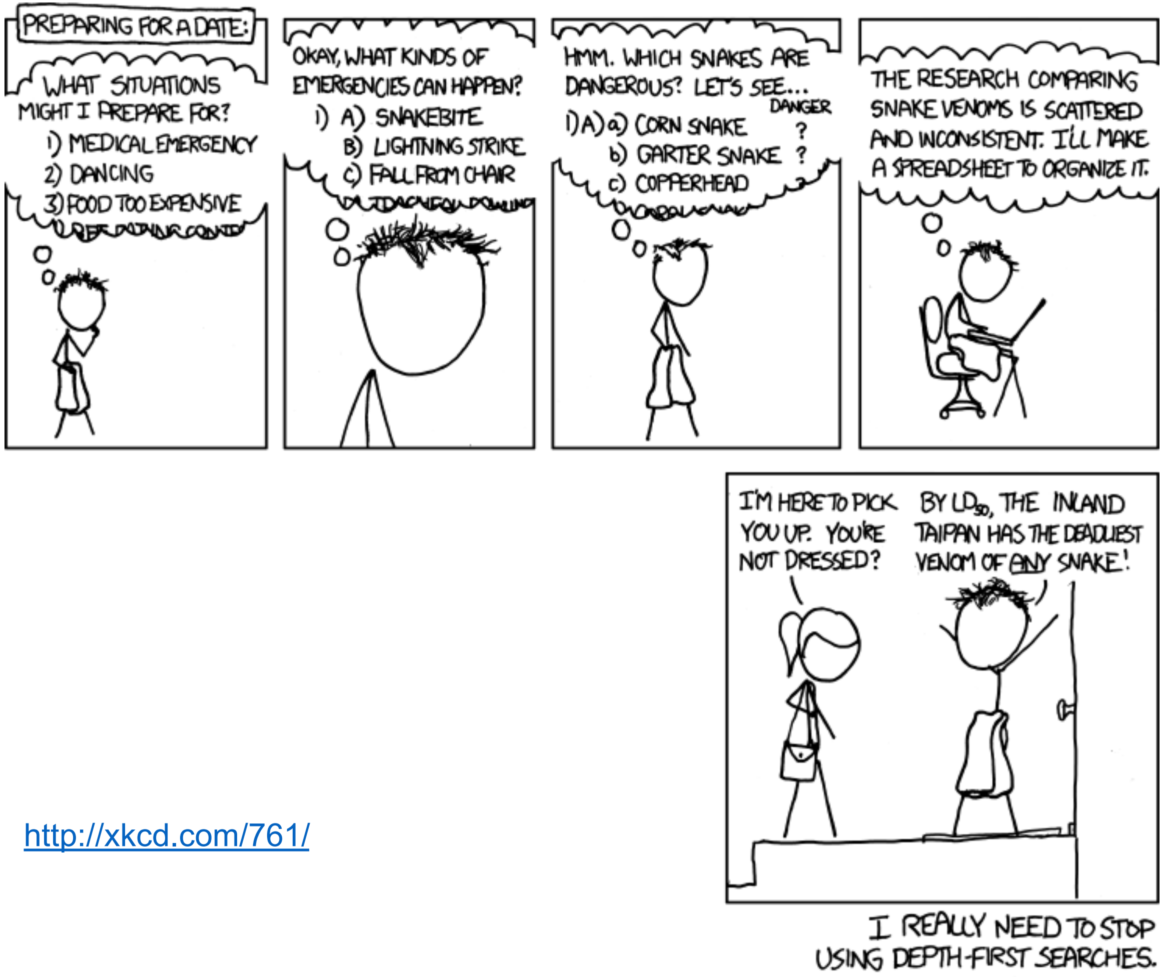
r,f,c,a,

G)





Depth-first search


PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR?
1) MEDICAL EMERGENCY
2) DANCING
3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN?
1) A) SNAKEBITE
B) LIGHTNING STRIKE
C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE...
DANGER
1)A) a) CORN SNAKE ?
b) GARTER SNAKE ?
c) COPPERHEAD
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD50, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

http://xkcd.com/761/

Analysis of search strategies

- Strategies are evaluated along the following criteria:
  - **Completeness:** does it always find a solution if one exists?
  - **Optimality:** does it always find a least-cost solution?
  - **Time complexity:** number of nodes generated
  - **Space complexity:** maximum number of nodes in memory
- Time and space complexity are measured in terms of
  - $\boldsymbol{b}$*:* maximum branching factor of the search tree
  - $\boldsymbol{d}$*:* depth of the optimal solution
  - $\boldsymbol{m}$: maximum length of any path in the state space (may be infinite)
  - $|\boldsymbol{S}|$: number of distinct states

Properties of depth-first search

ZJUI

- **Complete? (always finds a solution if one exists?)**

  Fails in infinite-depth spaces, spaces with loops

  Modify to avoid repeated states along path

  → complete in finite spaces

- **Optimal? (always finds an optimal solution?)**

  No – returns the first solution it finds

- **Time? (how long does it take, in terms of b, d, m?)**

  Could be the time to reach a solution at maximum depth *m*: $O\{b^m\}$

  Terrible if *m* is much larger than *d*

  But VERY FAST if there are LOTS of solutions

- **Space? (how much storage space, in terms of b, d, m?)**

  $O(bm)$, i.e., linear space!

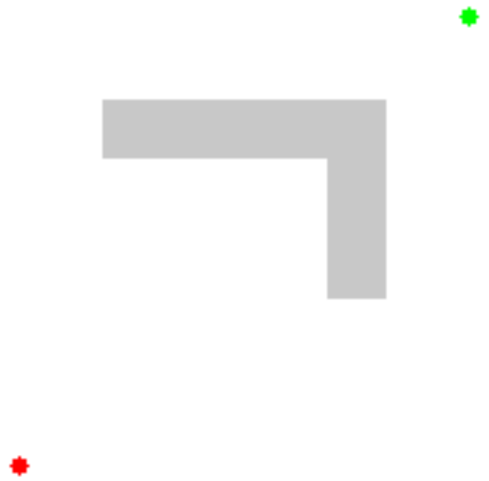
Outline

ZJUI

1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: a frontier list
    2. Second data structure: a search tree
    3. Third data structure: a “visited states” list
3. Depth-first search: very fast, but not guaranteed
4. <u>Breadth-first search: guaranteed optimal</u>
5. Uniform cost search = Dijkstra’s algorithm = BFS with variable costs

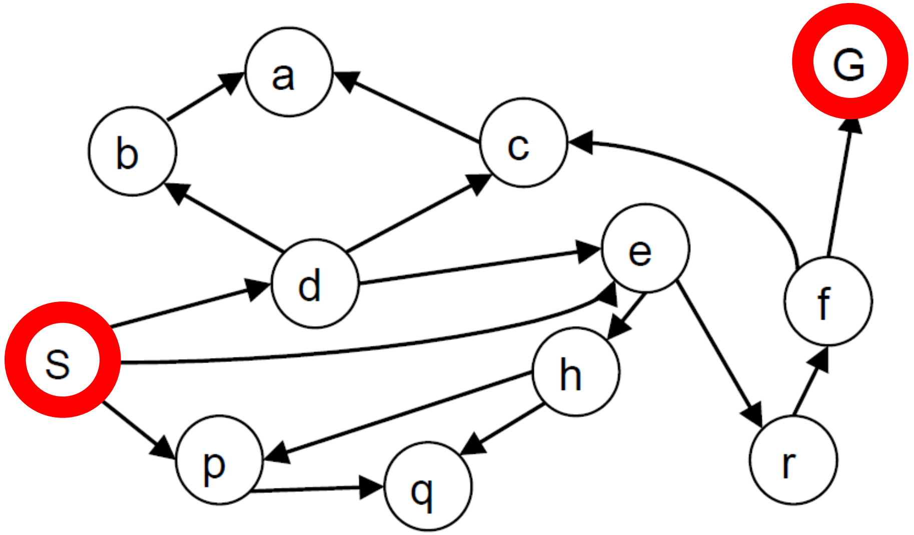
Breadth-first search

- Initialize the **frontier** using the **starting state**
- While the frontier is not empty
    - **Search strategy:** choose one of the hodes which is CLOSEST to the starting state
    - If the node contains the **goal state**, return solution
    - Else **expand** the node and add its children to the frontier



Breadth-first search

- Expand shallowest unexpanded node
- Implementation: *frontier* is FIFO (a queue)



Example from P. Abbeel and D. Klein

Breadth-first search

Expansion order:

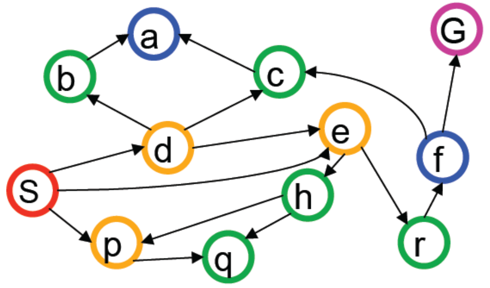
(s,

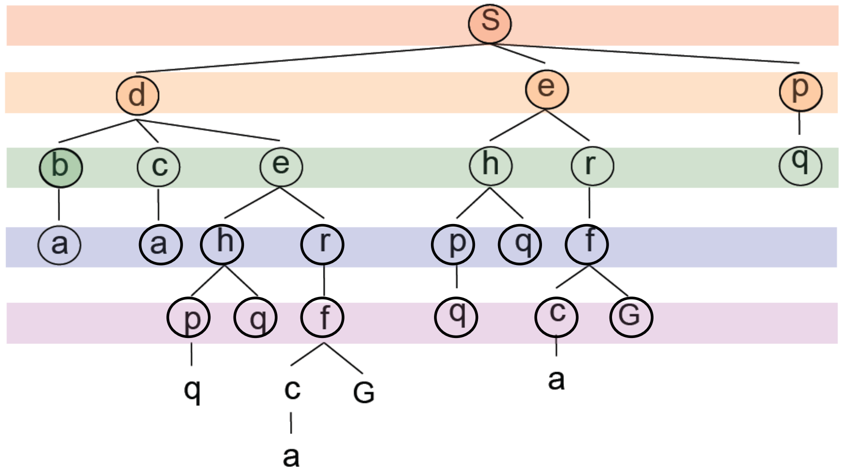
d,e,p,

b,c,e,h,r,q,

a,a,h,r,p,q,f,

p,q,f,q,c,G)





Properties of breadth-first search

ZJUI

- **Complete?**

  Yes (if branching factor *b* is finite).

  Even w/o repeated-state checking, it still works!!!

- **Optimal?**

  Yes – if cost = 1 per step (uniform cost search will fix this)

- **Time?**

  Number of nodes in a *b*-ary tree of depth *d*: $O\{b^d\}$

  (*d* is the depth of the optimal solution)

- **Space?**

  $O\{b^d\}$. --- much larger than DFS!

Outline

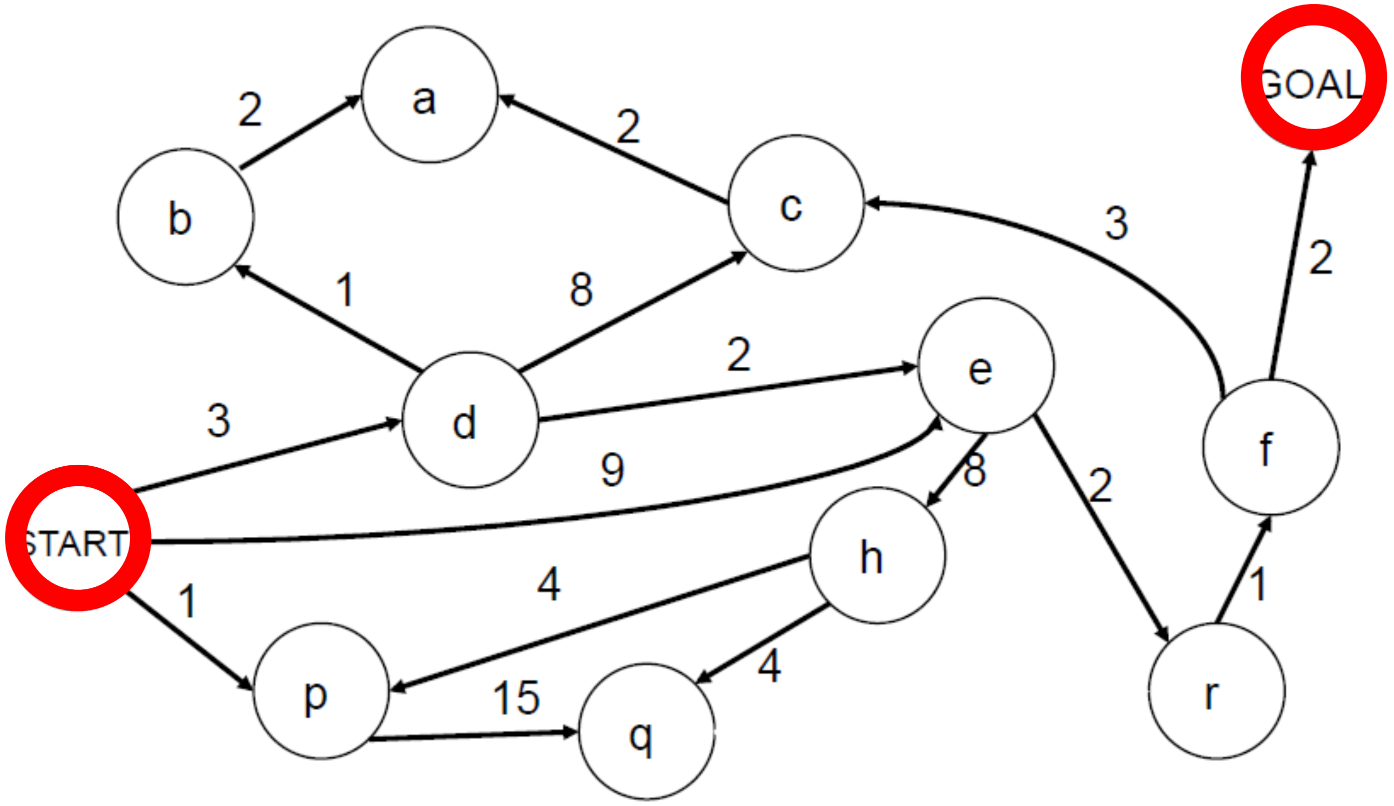
ZJUI

1. How to turn ANY problem into a SEARCH problem:
    1. Initial state, goal state, transition model
    2. Actions, path cost
2. General algorithm for solving search problems
    1. First data structure: a frontier list
    2. Second data structure: a search tree
    3. Third data structure: a “visited states” list
3. Depth-first search: very fast, but not guaranteed
4. Breadth-first search: guaranteed optimal
5. <u>Uniform cost search = Dijkstra’s algorithm = BFS with variable costs</u>

Uniform-cost search = Dijkstra’ s algorithm

- For each frontier node, save the total cost of the path from the initial state to that node
- Expand the frontier node with the lowest path cost
- Implementation: *frontier* is a priority queue ordered by path cost
- Equivalent to breadth-first if step costs all equal
- Equivalent to Dijkstra’s algorithm, if Dijkstra’s algorithm is modified so that a node’s value is computed only when it becomes nonzero

Uniform-cost search example



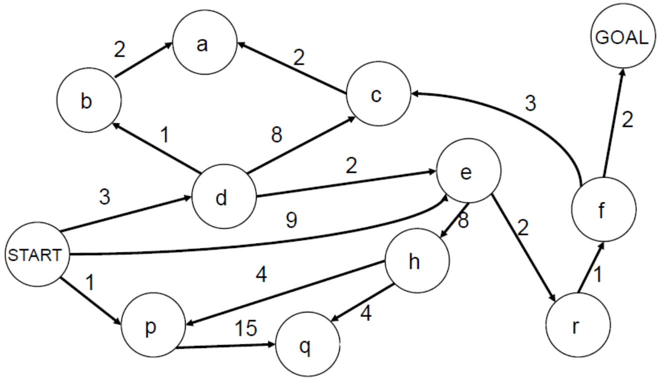
Uniform-cost search example

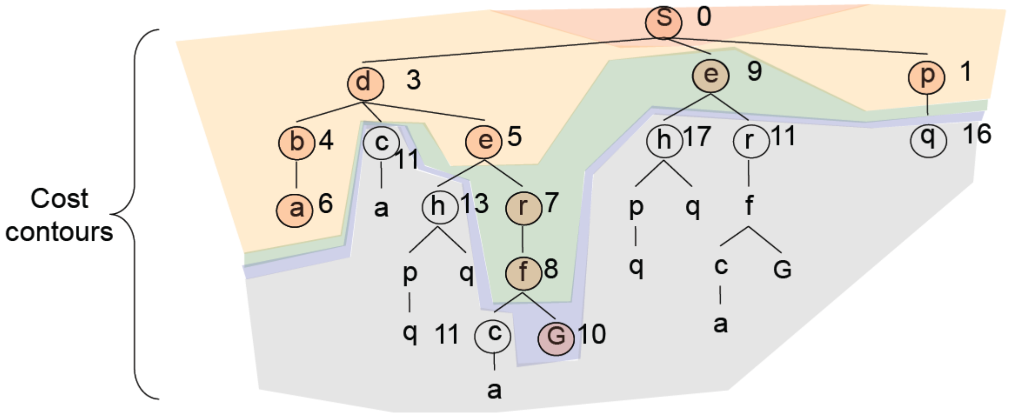
Expansion order:
(s,

p(1),d(3),b(4),e(5),a(6),

r(7),f(8),e(9),

G(10))





Properties of uniform-cost search

- **Complete?**

  Yes (if branching factor *b* is finite).

  Even w/o repeated-state checking, it still works!!!

- **Optimal?**

  Yes

- **Time?**

  Number of nodes in a *b*-ary tree of depth *d*: $O\{b^d\}$

  Priority queue is $O\{\log_2 d\}$/node

- **Space?**

  $O\{b^d\}$ --- much larger than DFS! This might be a reason to use DFS.

Search strategies so far

Algorithm	Complete?	Optimal?	Time complexity	Space complexity	Implement the Frontier as a…
BFS	Yes	If all step costs are equal	$O\{b^d\}$	$O\{b^d\}$	Queue
DFS	No	No	$O\{b^m\}$	$O\{bm\}$	Stack
UCS	Yes	Yes	$O\{b^d \log_2 d\}$	$O\{b^d\}$	Priority Queue

Next time

- Already we know how far it is, from the start point, to each node on the frontier.
- What if we also have an ESTIMATE of the distance from each node to the GOAL?