香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

COMP7015 Artificial Intelligence (S1, 2024-25)

Lecture 1: Introduction to AI and Uninformed Search

Instructor: Dr. Kejing Yin (cskjyin@hkbu.edu.hk)

Department of Computer Science
Hong Kong Baptist University

September 6, 2024

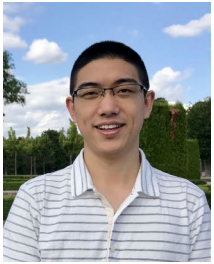
Outline for Today

1. Course Logistics
2. Introduction to AI
3. Solving Problems by Searching I

Course Logistics

- Instruction Team
- Resources
- Course Content and Assessment
- Course Policies

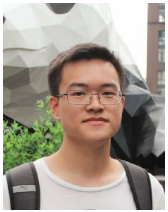
Instructor



Dr. YIN Kejing

- *Ph.D. in Computer Science*, Hong Kong Baptist University
- *B.Eng. in Thermal Energy and Power Engineering*, South China University of Technology
- Research Interest:
  - AI in healthcare
  - Time series analysis
  - Clinical multimodal learning
- Office Hour for Q&A:
  - Wednesday 2pm – 4pm, at RRS732
  - 12 sessions, from Sep. 11 to Dec. 4, excluding public holiday
  - Other time slots: by appointment
- Email: `cskjyin@comp.hkbu.edu.hk`

Teaching Assistants



Mr. LI Ruiqi
csrqli@comp.hkbu.edu.hk

Computer vision,
Neural rendering

Mr. CHI Pinzhen
cspzchi@comp.hkbu.edu.hk

Molecule generation,
Generative model for drug discovery

Communications and Learning Support

- Course Materials and Assignment Submission: Moodle (`https://buelearning.hkbu.edu.hk/course/view.php?id=111847`)
- Q&A platform: Piazza (`https://piazza.com/hkbu.edu.hk/fall2024/comp7015`)
  - Almost all questions should be asked in Piazza via public posts (post to entire class).
  - Try to answer questions from others.
  - Email or private posts should be used when sensitive information is involved.
  - Send an email to the instructor if you are not enrolled in Piazza.
- Office Hours for Q&A:
  - Instructor office hour: Wednesday 2 – 4pm, from Sep. 11 to Nov. 27, except public holidays.
  - TA office hours (for programming questions): Thursday 2 – 4pm, from Sep. 19 to Nov. 21.

Our Goal: Combining Theory with Practice

- **Theoretical Concepts**
  - What problems can be solved using AI? What AI algorithms are there?
  - How to choose suitable AI methods/algorithms for your task?
  - Why do the algorithms work?
  - When do the algorithms work, and when not?
  - How to *correctly* apply the algorithms?

- **Practical Experience**
  - Gaining hands-on experience of relevant algorithms.

⚠ Course Scope and Depth

This course is an "intro" level course to AI. We cover a broad range of topics in AI, but the depth could be limited. Read more and code more if you wish to dig deeper.

Theoretical Concepts

- Lectures
  - Every Friday 6:30 – 9:20 pm
  - Venue: LT3 (SCT503, 5/F, Cha Chi Ming Science Tower, Ho Sin Hang Campus)
  - Attend lectures: there are many live demos to help you understand the content
- In-/after-class exercises
  - They will not be assessed.
  - You are encouraged to solve them.
- Written assignments
  - You can discuss with others, but write your own answers independently.
- In-class quiz
  - Mark the date: Oct. 18, 2024 (Week 7)

Practical Experience

- Four lab sessions:
  - Arranged in Saturdays and divided into two sections due to lab availability.
  - Lab 1 (Oct. 5): Solving Problems Using Search
  - Lab 2 (Oct. 26): Machine Learning with Scikit-learn
  - Lab 3 (Nov. 2): Getting Started with PyTorch
  - Lab 4 (Nov. 16): Using Pre-trained Models in PyTorch
- Programming assignments
  - Solve them independently.
  - Do not directly copy from lab materials.
- Course project
  - Work in group of up to four people.
  - Choose from three topics.
  - Details will be released in Week 5.

Practical Experience: Python

- Python is the mainstream programming language for AI.
- Python programming is covered in COMP7035.
- Seek help from RAs for any programming questions.
- Learning resources
  - `https://www.w3schools.com/PYTHON/`
  - `https://www.youtube.com/watch?v=rfscVS0vtbw`
- Practice makes perfect!

Assessment

- The overall assessment consists of:

Assignments	20%
In-class quiz	5%
Course project	25%
Final exam	50%

- You need to score at least 30 in the final exam to pass the course.

Academic Integrity

Guidelines for Students on Academic Integrity

HKBU adopts a zero tolerance policy for academic dishonesty. Any student who commits an act of academic dishonesty would have violated academic integrity and would therefore be subject to academic disciplinary actions.

- Make sure you read in full: `https://ar.hkbu.edu.hk/quality-assurance/university-policy-and-guidelines/academic-integrity`
- Discussions on course materials are encouraged.
- However, all assignments and project reports must be written in your own words.
- *Lending* and *borrowing* assignment or project solutions are both regarded as dishonest behaviors.

Avoiding Plagiarism

- Plagiarism means taking words or ideas from their original owners and presenting them as if they were your own (adapted from Hung, 1999).
- For example, copying and pasting from books, lecture notes, Web resources, and your work submitted to other courses constitutes plagiarism.
- Make sure to read the following resources on how to avoid plagiarism:
  `https://ar.hkbu.edu.hk/quality-assurance/university-policy-and-guidelines/academic-integrity/section-2-plagiarism`

On The Use of AI-Generated Content

- Follow HKBU policy:
  `https://securedoc.hkbu.edu.hk/hkbuOnly/ics/integration/GenAIPrinciples.pdf`
- Do NOT submit AI-generated content as your answers to assignments.
- If any generative AI tool is used, **declare the use** with sufficient detail.

Late Submission Policy

- **For assignments**: 12.5% penalties for each 6 hours late.
  - *Example: If one score 90, but the assignment is overdue for 5 minutes,* $90 \times 12.5\% = 11.25$ *scores will be deducted.*
  - Submit early to avoid last-minute network issues.
  - Always check the file you submitted.
  - Obtain prior approval under special circumstances (e.g., sick leave), no penalties will apply.

- **For course project:** No late submission is allowed!

Introduction to AI

- What is Artificial Intelligence?
- A Brief History of AI
- A Double Edged Sword: The Risks of AI
- The Course In a Nutshell

Real Examples of AI: Digital Assistants

Google Duplex: `https://www.youtube.com/embed/D5VN56jQMWM`

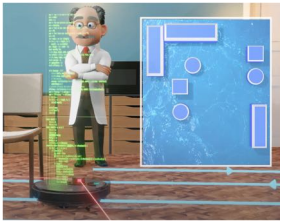
Behind the scenes:

- Voice to text
- Natural language understanding
- Natural language generation
- Text to speech (*sounding natural*)

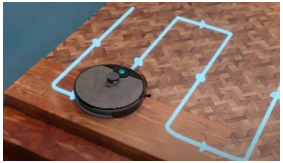
Real Examples of AI: Cleaning Robots

Cleaning robots: `https://www.youtube.com/embed/hoY2YxLGV98`

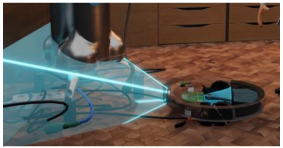
Behind the scenes:



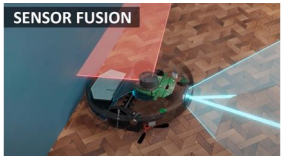
Signal Processing & Map Construction



Path Planning



Computer Vision

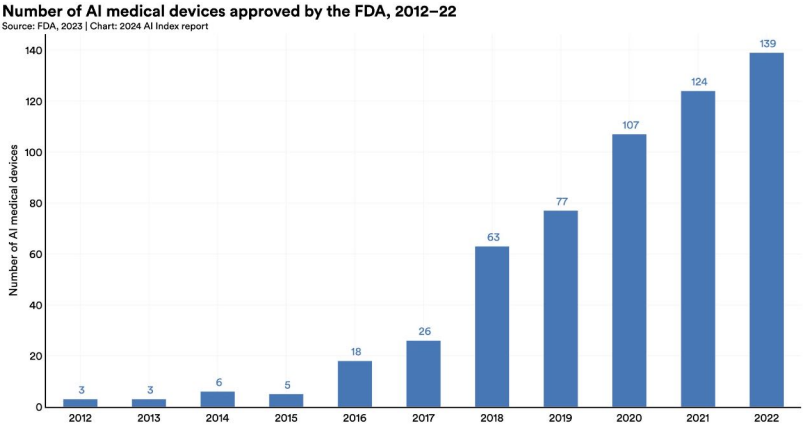


Sensor Fusion

Real Examples of AI: Healthcare

- Chest X-ray screening with AI: `https://www.youtube.com/embed/Im4L1lH-HAQ`
- Medical imaging segmentation: `https://github.com/bowang-lab/MedSAM`
  - Identifying and delineating regions of interest (RoI), e.g., tumor, brain tissues, and lung.
  - [Live demo of MedSAM]
- AI for drug discovery: `https://www.youtube.com/embed/xDMzOUUnNzw`

Real Examples of AI: Healthcare

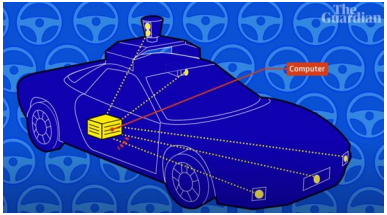


The FDA approved 139 AI-related devices in 2022, a 12.1% increase from 2021.[1]

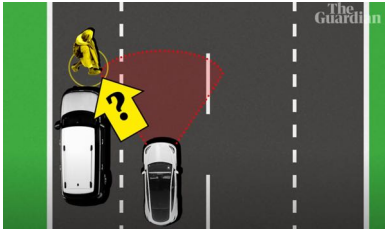
[1] Figure from: https://aiindex.stanford.edu/report/

Real Examples of AI: Autonomous Vehicles

- Why self-driving cars have stalled: `https://www.youtube.com/embed/4sCK-a33Nkk`
- ML tasks of autonomous vehicle development: `https://wandb.ai/av-team/av-tasks/reports/The-ML-Tasks-Of-Autonomous-Vehicle-Development--VmlldzoyNTc2Nzkx`



Core: Perception, Understanding and Decision-making



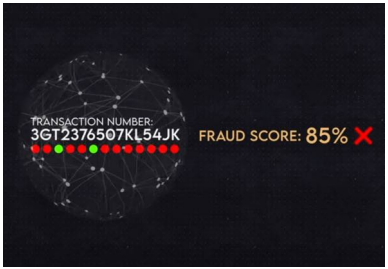
Example: Pedestrian Detection

Example: Intention and Trajectory Prediction

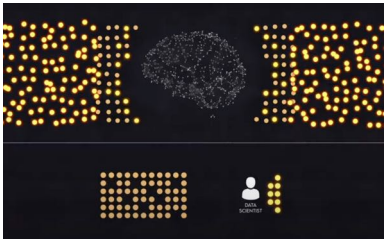
Real Examples of AI: Fraud Detection

Fraud Detection: `https://www.youtube.com/embed/QFyM3w95fXI`

Rule-based method: Lacking adaptability



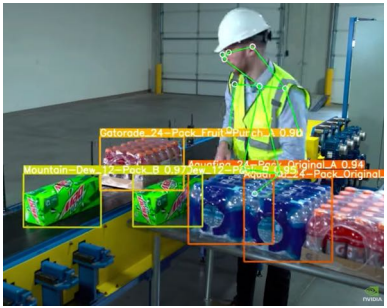
Learning-based method: Finding patterns and make predictions



Deep learning method: Finding patterns without manually selecting features

Real Examples of AI: Supply Chain Optimization

Supply Chain Optimization: `https://www.youtube.com/embed/he5I6ByoaB4`



Object Detection

Quality control: damage classification

Route and path planning

Real Examples of AI: Large Language Models

ChatGPT: `https://chatgpt.hkbu.edu.hk/`
What are LLMs: `https://www.youtube.com/embed/iR2O2GPbB0E`

Capturing general knowledge: *large models* using *huge amount of data*.

Other potential applications:

- Programming code is the language of computers:
  Malware analysis: `https://www.youtube.com/embed/OBnylC91RCk`
- Protein and molecular sequences are the language of biology:
  Drug discovery: `https://www.youtube.com/embed/8z_iIHXkAJk`

Real Examples of AI: Image and Video Generation

- DALL·E 2
  - https://www.youtube.com/embed/qTgPSKKjfVg
- Midjourney
  - showcases: https://www.midjourney.com/showcase/top/
- Latent Diffusion Model
  - [live demo of stable diffusion]
  - Trained using over 5 billion image-caption pairs.
  - Trained using 256 Nvidia A100 GPUs (a total of 150,000 GPU-hours).
- Sora
  - https://openai.com/index/sora/

Back to the question: What is AI?

Different Aspects of AI

Two dimensions: (*Human vs. Rational*) and (*Thought vs. Behavior*)

Acting humanly

The Turing test approach: if a machine exhibits intelligent behavior indistinguishable from that of a human.

Thinking humanly

The cognitive modeling approach: if a machine's input-output behavior matches corresponding human behavior.

Acting rationally

The rational agent approach: a rational agent acts so as to achieve the best outcome.

Thinking rationally

The "laws of thought" approach: representing knowledge and making inferences using formal logic systems.

- *Rational: maximally achieving pre-defined goals.*

What is AI?

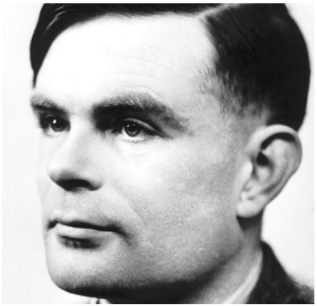
Artificial Intelligence

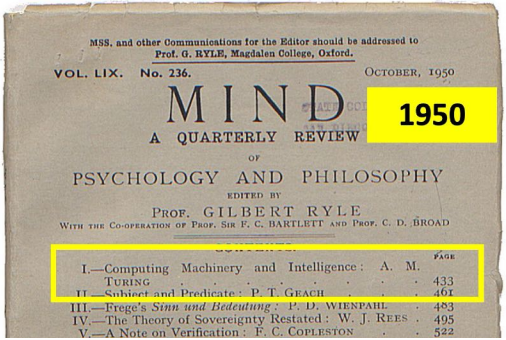
Computer systems able to perform tasks typically requiring human intelligence, including **problem-solving** and **decision-making**.

Achieving them requires tackling tasks such as visual perception, speech recognition, natural language recognition and generation.

1943–1956: Inception of AI

- Knowledge of the basic physiology and function of neurons in the brain;
- A formal analysis of propositional logic due to Russell and Whitehead;
- 1950: Alan Turing's theory of computation.





MSS. and other Communications for the Editor should be addressed to
Prof. G. RYLE, Magdalen College, Oxford.

VOL. LIX. No. 236. OCTOBER, 1950

MIND

1950

A QUARTERLY REVIEW

OF

PSYCHOLOGY AND PHILOSOPHY

EDITED BY

PROF. GILBERT RYLE

WITH THE CO-OPERATION OF PROF. SIR F. C. BARTLETT AND PROF. C. D. BROAD

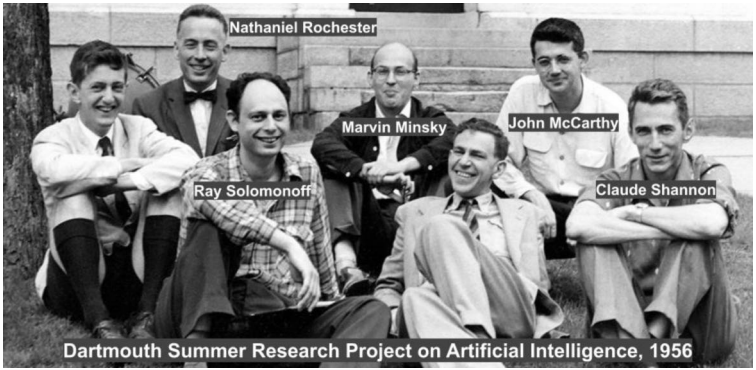
CONTENTS.

PAGE

I.—Computing Machinery and Intelligence: A. M. Turing 433
II.—Subject and Predicate: P. T. Geach 461
III.—Frege's *Sinn und Bedeutung*: P. D. Wienpahl . 483
IV.—The Theory of Sovereignty Restated: W. J. Rees . 495
V.—A Note on Verification: F. C. Copleston . . . 522

1952–1969: Early Enthusiasm, Great Expectations

- 1950s: Focus on tasks indicative of human intelligence, *e.g.*, games, puzzles, and IQ tests;
- 1956: Dartmouth meeting: introduce the term "Artificial Intelligence";
- 1965: complete algorithm for logical reasoning by J. A. Robinson.



1969–1986: Expert Systems

- Dendral program (Buchanan *et al.*, 1969): inferring molecular structure from the information provided by a mass spectrometer.
- Expertise derived from large numbers of special-purpose rules.
- Mycin system for diagnosing blood infections: with around 450 rules, it could perform as well as some experts.

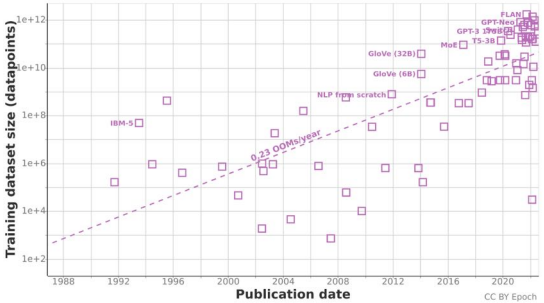
- 1980–1988: AI industry boomed from a few million dollars to billions,
- After 1988: "AI Winter", difficult to build expert systems for complex domains.
  - reasoning methods break down in face of *uncertainty*.
  - the system *could not learn from experience*.

1987–Now: Statistical Machine Learning

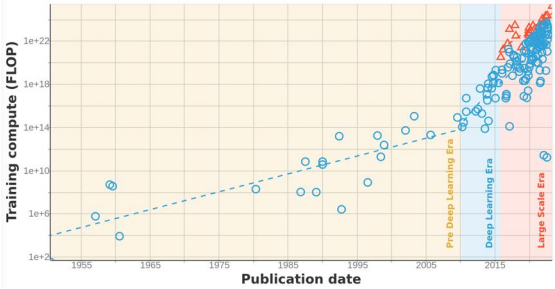
- A particular focus on uncertainty.
- *Probabilistic Reasoning in Intelligent Systems* (Judea Pearl, 1988)
  - Led to a new acceptance of probability and decision theory in AI.
- *Bayesian networks*:
  - A rigorous and efficient formalism for representing uncertain knowledge.
  - Practical algorithms for probabilistic reasoning.
- AI's newfound appreciation for data, statistical modeling, optimization, and ML
  - Reunification of subfields, *e.g.*, CV, robotics, speech recognition, and NLP.

2011–Now: Deep Learning

Big data and advanced computing power



Evolution of Language Datasets[1]



Trend of Training Computes (FLOP)[2]

[1] Image from: https://epochai.org/blog/trends-in-training-dataset-sizes

[2] Image from: https://epochai.org/blog/compute-trends

2011–Now: Deep Learning

Some non-exhaustive milestones

- 2011: IBM's Watson defeats human champions in the quiz show.
- 2012: AlexNet shows that Convolutional Neural Networks works.
  Commonly considered as what brought Deep Learning in to the mainstream
- 2016: Google's AlphaGo beats human champion in Go.
- 2018 and 2021: AlphaFold predicts protein structure with remarkable accuracy.
- 2022: OpenAI released ChatGPT.

Discussion: Risks of AI

What do you think are the potential risks of AI?

We will discuss more on the risks of AI in November.

What are we going to learn in this course?

AI covers a broad range of models and complementary approaches.

**subject to change according to the learning progress.*

Weeks 1–4	Introduction to AI Solving problems by searching Knowledge representation and reasoning
Weeks 5–8	Statistical Machine Learning Public Holiday (week 6) In-class Quiz (week 7)
Weeks 9–13	Deep Learning Reinforcement Learning Generative AI

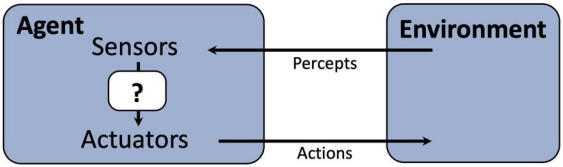
Solving Problems by Searching I

- Problem-Solving Agents
- Formulating A Search Problem
- The General Search Algorithm
- Uninformed (Blind) Search

Agents and Environments

Agent

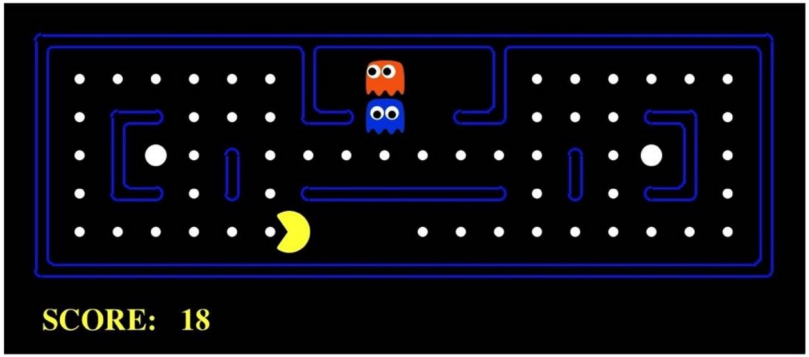
An **agent** perceives the **environment** through sensors and **acts** upon that environment.



Rational Agents

A **rational agent** selects actions that maximize its (expected) utility.

Agents and Environments: PacMan



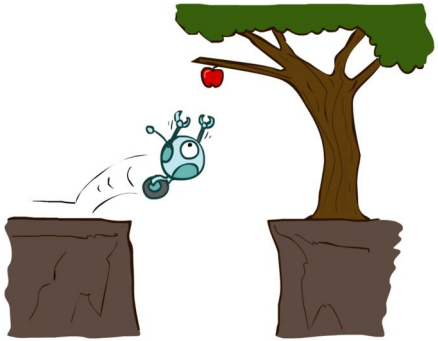
Play a few rounds here: `https://g.co/kgs/DxZ3dS`

Pac-Man is a registered trademark of Namco-Bandai Games, used here for educational purposes

The PacMan AI projects were developed at UC Berkeley: `http://ai.berkeley.edu/project_overview.html`

Reflex Agents

- Choose actions based on current percept
- May have **memory** that reflects some unobserved aspects of the current state
- May have a **model** of the world's current state
- **Do not consider the future consequences of the actions**



Question

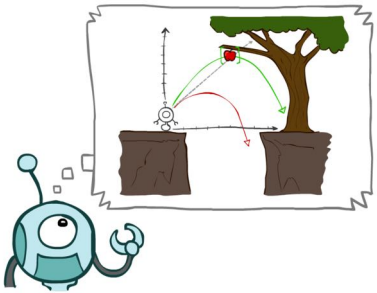
Can a reflex agent be rational?

Image credits: CS188@UCB

Agents Planning Ahead

When the correct action is not immediately obvious...

- Need to consider the consequences of actions
- Must have a **model** of how the environment changes in response to actions
- Must have a **goal formulation**
  (*a function that checks if the goal is achieved*)



📖 Problem-solving agent

An agent that considers a sequence of actions that form a path to a goal state.

📖 Search

The computational process a problem-solving agent undertakes.

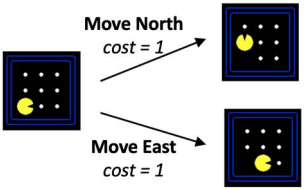
Image credits: CS188@UCB

Elements of Search Problems

- A **state space**: a set of all possible configurations of the world.

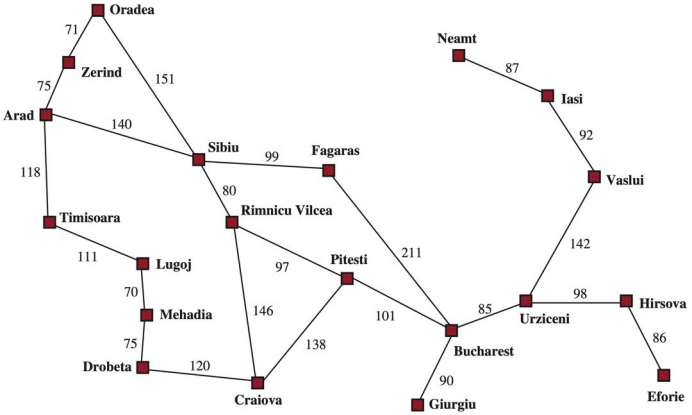
  - The **initial state**: the state that the agent starts in.
  - A set of **goal states**: can be one or many. (Need a *goal test*)

- A **successor function**: returns the actions available and the corresponding costs.



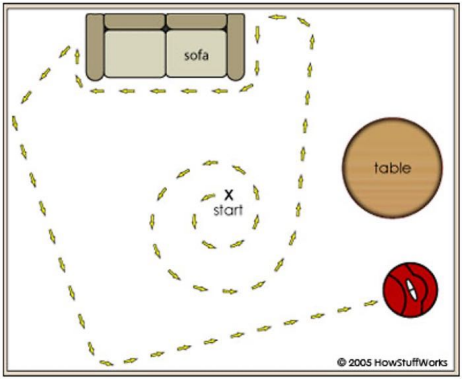
- A **solution**: a sequence of actions that starts from the initial state and reaches a goal state.

Example: Route Finding



- **State spaces**:
  - Cities in the Romania map
- **Successor function**:
  - Actions: Go to adjacent cities
  - Cost: distance / travel time
- **Initial state**:
  - Arad
- **Goal test**:
  - Is state == Bucharest?

Example: Cleaning Robots



- **State spaces**:
  - All possible locations in the room with different orientations.
- **Successor function**:
  - Actions: Move forward, rotate (degrees)
  - Cost: Time / energy consumption
- **Initial state**:
  - The charging dock
- **Goal test**:
  - Is location == somewhere specified?
  - *or* Is room cleaned?

How to Formulate A Search Problem?

1. **Defining the objective**:
   - The objective defines the *goal test* and limits the actions to consider.
   - Examples:
     - Arrive at Bucharest with shortest distance
     - Arrive at Bucharest with least fuel consumption
     - Arrive at Bucharest with shortest travel time

2. **Formulating the Search Problem**:

The state of the world includes many things:

- Radio program while driving, condition of the road, color of the car,...

🤔 Question

Do they all matter for the problem formulation?

Level of Abstraction

📖 Abstraction

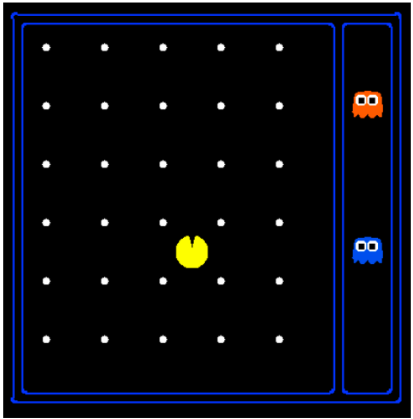
The state space only needs to keep details necessary for planning. The process of removing details from a representation is called abstraction.

- The right level of abstraction **depends on the problem and the objective**.
- For example, when searching for shortest travel time from Arad to Bucharest:
  - Driving a red car vs. driving a blue car: no difference
  - Means of transportation (e.g., driving 🚗, bus 🚌, train 🚄, rocket 🚀, or 🛸): huge difference

✍ A rule of thumb

Remove as much detail as possible and make only those distinctions necessary to ensure a valid solution.

Sizes of The State Space



Example from CS188@UCB

- A 10×12 grid world: 120 possible agent positions
- Food count: 30
- Ghost position: 12
- Agent facing: N, S, E, W

How many world states in total?

$120 \times 2^{30} \times 12^2 \times 4$

State space size for path finding?

120

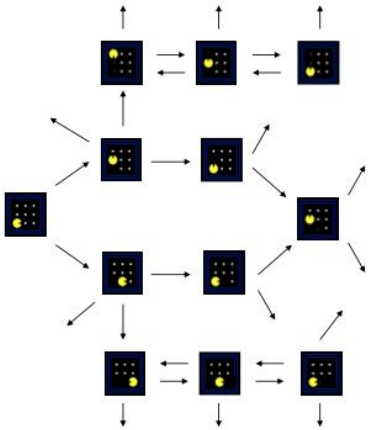
State space size for eating all dots?

120×2^{30}

State Space Graphs

The **state space graph** is a mathematical representation of a search problem, where

- Nodes are states (abstracted world configurations)
- Arrows are actions (pointing to their consequences)
- One of the nodes is the initial state
- The goal test is a set of goal nodes (one or many)
- Each state appear only once in a state space graph
- Often too big to build the full graph in memory



Example from CS188@UCB

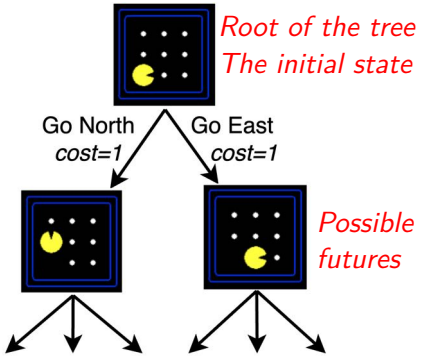
State Space Graphs → Search Trees

We can solve the search problems by constructing search trees.

- A "what if" tree of plans and their consequences
- Nodes are states
- Starting from the initial state (the root of the tree)

- Each state may appear multiple times in a search tree
- Each node has a unique path back to the root.

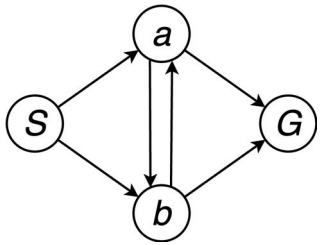
- **Solution to the search problem**:
  A path from the root note (initial state) to a goal state.



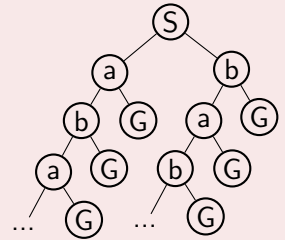
Example from CS188@UCB

State Space Graphs vs. Search Trees

Consider this four-state space graph:



🤔 How big is the search tree (initial state: S)



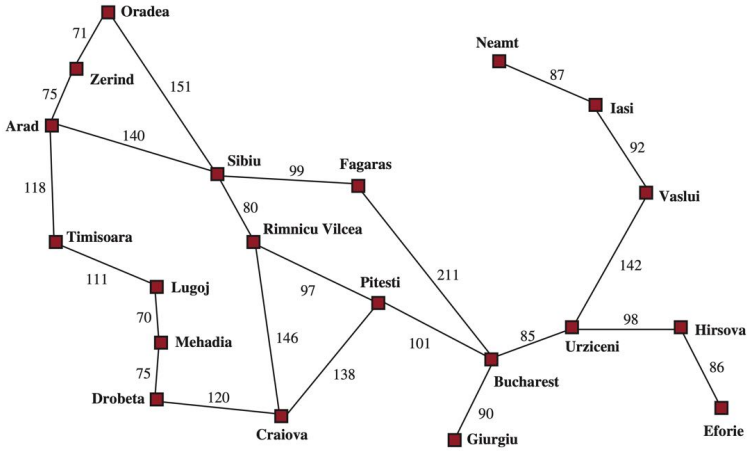
size of the tree:∞

- There are many repeated (redundant) structure in the search tree!
- We can barely build the full search tree.
- Construct search trees on the fly, and construct as little as possible.

Example from CS188@UCB

The General Search Algorithm

Let's start from an example: finding a path from Arad to Bucharest.



The General Search Algorithm: Building The Search Trees

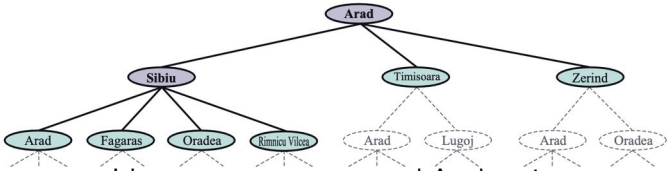
Tree-Like Search

```
1  frontierList ← [initialState];
2  while frontierList is not empty do
3  |  select a node S according to the search strategy;
4  |  if node S is a goal state then
5  |  |  return the corresponding solution;
6  |  else
7  |  |  expand node S;
8  |  |  add all children to frontierList;
9  |  end if
10 end
11 return failure;
```

- Frontier List: A list containing all nodes *to be expanded*.
- *The search strategy*: How to choose the node to expand? (The key question)
- **Expand**: Use the *successor function* to get available actions and costs.

Redundant Paths

Search trees may have repeated states (*e.g.*, Arad appear twice in the following tree)



It's not necessary to expand Arad again.

- The purpose of having "visitedList": check and avoid redundant paths.

> **Quote**
>
> Algorithms that cannot remember the past are doomed to repeat it.

- Price to pay: more memory usage (to store the visited nodes).
- On the other hand, we can save memory if we do not check redundant paths (*e.g.*, DFS).

The General Search Algorithm: Building The Search Tree

Graph Search

```
1  frontierList ← [initialState];
2  visitedList ← [ ];
3  while frontierList is not empty do
4  |   select a node S according to the search strategy;
5  |   if node S is a goal state then
6  |   |   return the corresponding solution;
7  |   else
8  |   |   expand node S;
9  |   |   add children (not already in frontierList and visitedList) to frontierList;
10 |   |   add node S to visitedList;
11 |   end if
12 end
13 return failure;
```

The General Search Algorithm

```
1  frontierList ← [initialState];
2  visitedList ← [ ];
3  while frontierList is not empty do
4  |  select a node S according to the search strategy;
5  |  if node S is a goal state then
6  |  |  return the corresponding solution;
7  |  else
8  |  |  expand node S;
9  |  |  add children (not already in frontierList and visitedList) to frontierList;
10 |  |  add node S to visitedList;
11 |  end if
12 end
13 return failure;
```

- Frontier List: A list containing all nodes *to be expanded*.
- Visited List: A set of all nodes that *have been expanded*. *(Not used for tree-like search)*
- *The search strategy*: How to choose the node to expand? (The key question)
- Expand: Use the *successor function* to get available actions and costs.

Graph Search vs. Tree-Like Search

- When an algorithm checks redundant paths, we call it "graph search"
- When we do not check redundant paths, it's "tree-like search"

Graph Search

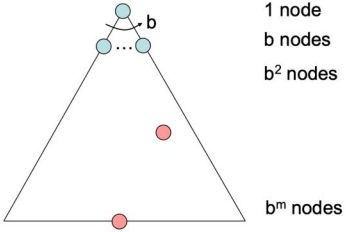
```
1  frontierList ← [initialState];
2  visitedList ← [ ];
3  while frontierList is not empty do
4      select a node S according to the search strategy;
5      if node S is a goal state then
6          return the corresponding solution;
7      else
8          expand node S;
9          add children (not already in frontierList and
             visitedList) to frontierList;
10         add node S to visitedList;
11     end if
12 end
13 return failure;
```

Tree-Like Search

```
1  frontierList ← [initialState];
2  while frontierList is not empty do
3      select a node S according to the search strategy;
4      if node S is a goal state then
5          return the corresponding solution;
6      else
7          expand node S;
8          add all children to frontierList;
9      end if
10 end
11 return failure;
```

Properties of Search Algorithms (Performance Measure)

- **Completeness**: Is the algorithm guaranteed to find a solution if one exists?
- **Cost optimality**: Is the algorithm guaranteed to find a solution with lowest cost?
- **Time complexity**: How much time it takes? ("Big-O" notation)
- **Space complexity**: How much memory is needed? ("Big-O" notation)



- b: average number of children (*branching factor*)
- m: maximum depth
- Solutions at various depths; worst case at depth of m

🤔 How many nodes in the entire tree?

$1 + b + b^2 + ... + b^m = O(b^m)$ ◁ "Big-O": how it grows as b and m grows.

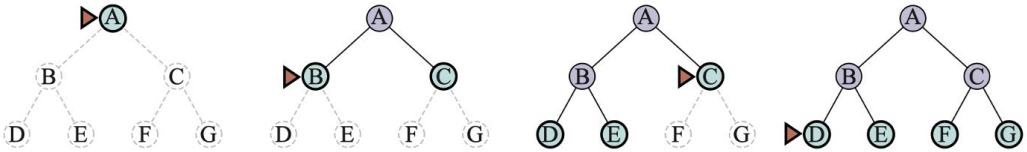
Road Map of Search Algorithms

- Uninformed search (blind search; no clue about how close a state is to the goal state)
  - Breadth-first Search (BFS)
  - Uniform-cost Search (UCS)
  - Depth-first Search (DFS)
  - Depth-limited Search and Iterative deepening Search
- Informed search (heuristic search; we have some hints about the location of goals)
  - Greedy Search
  - A* Search

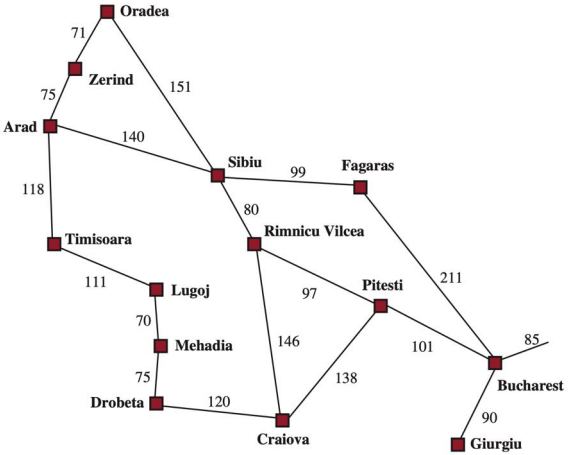
Breadth-First Search (BFS)

- Search strategy: Expand a *shallowest* node first.
  i.e., always expanding the nodes at depth $d-1$ before expanding nodes at depth d.

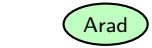
- Often implemented as graph search (check redundant paths).



Example: Use BFS to Find A Route From Arad to Fagaras

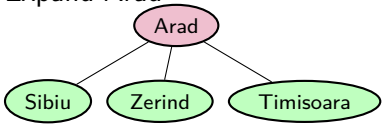


Step 1: Initialization



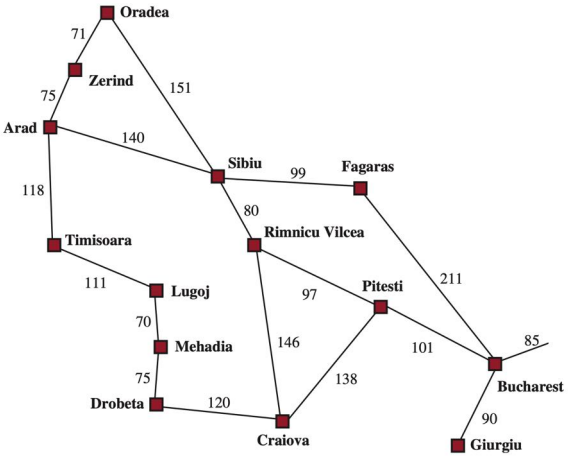

```
frontiers = [Arad]
visited = []
```

Step 2: Expand Arad

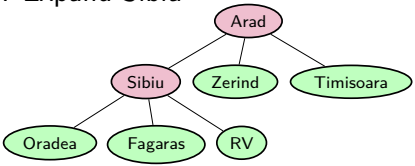



```
frontiers = [Sibiu, Zerind, Timisoara]
visited = [Arad]
```

Example: Use BFS to Find A Route From Arad to Fagaras

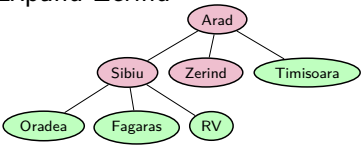


Step 3: Expand Sibiu



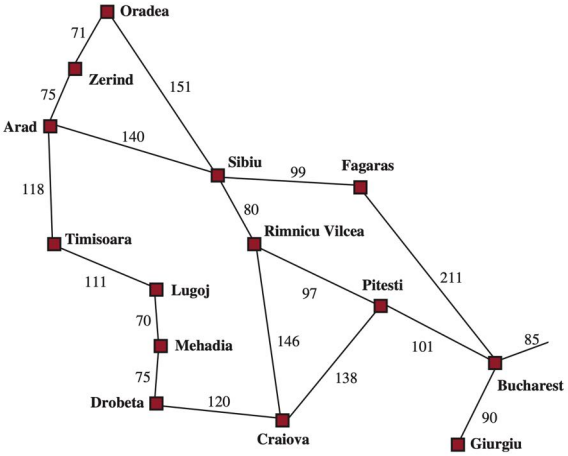

```
frontiers = [Zerind, Timisoara, Oradea, Fagaras,
RV]
visited = [Arad, Sibiu]
```

Step 4: Expand Zerind

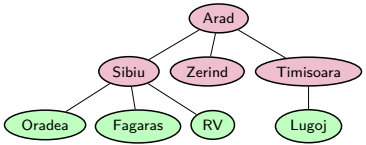



```
frontiers = [Timisoara, Oradea, Fagaras, RV]
visited = [Arad, Sibiu, Zerind]
```

Example: Use BFS to Find A Route From Arad to Fagaras

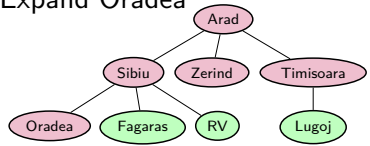


Step 5: Expand Timisoara



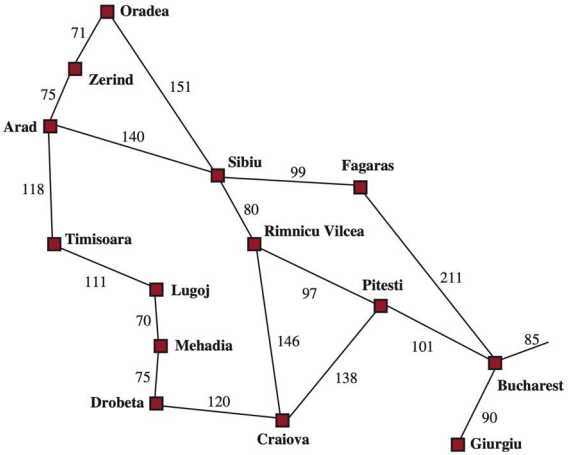

```
frontiers = [Oradea, Fagaras, RV, Lugoj]
visited = [Arad, Sibiu, Zerind, Timisoara]
```

Step 6: Expand Oradea

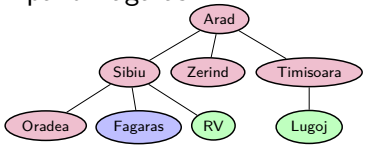



```
frontiers = [Fagaras, RV, Lugoj]
visited = [Arad, Sibiu, Zerind, Timisoara, Oradea]
```

Example: Use BFS to Find A Route From Arad to Fagaras



Step 7: Expand Fagaras

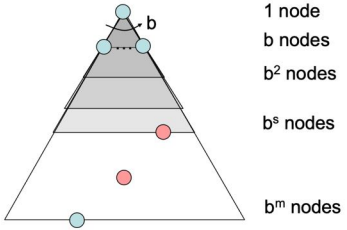


`Solution: Arad → Sibiu → Fagaras`

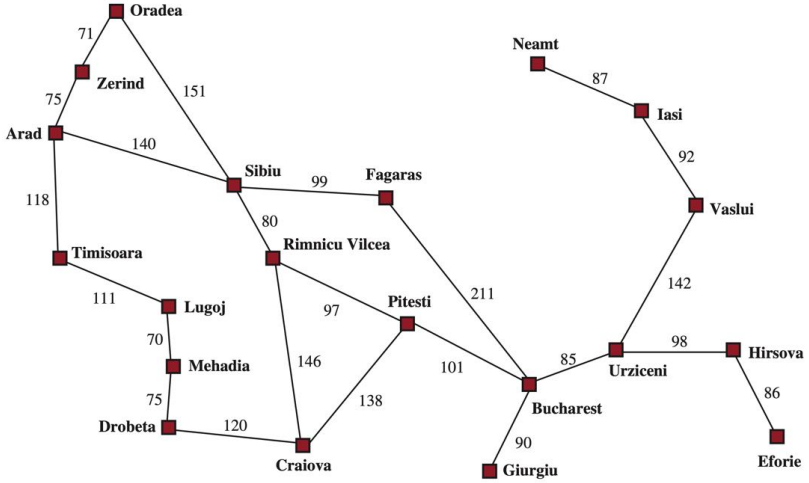
Properties of Breadth-First Search

Suppose the depth of the shallowest solution is s.

- **Nodes expanded?** All nodes above the shallowest solution.
- **Is BFS Complete?** Yes, s is finite if a solution exists.
- **Is BFS Cost-Optimal?**
  BFS finds the *shallowest* solution.
  Cost-optimal only if all actions cost the same.
- **Time complexity?** Takes time $O(b^s)$
- **Space needed?**
  visitedList: roughly $1 + b + b^2 + ... + b^{s-1}$
  frontierList: roughly b^s
  Total: $O(b^s)$

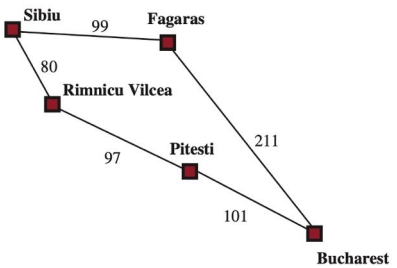


Exericise: Use BFS to Find A Route From Arad to Bucharest



Uniform-Cost Search (UCS)

- Consider the cost: also store the cost from root to nodes in frontierList.
- *Search strategy*: Expand the node with the smallest cost from root to the node.



Route-finding from Sibiu to Bucharest

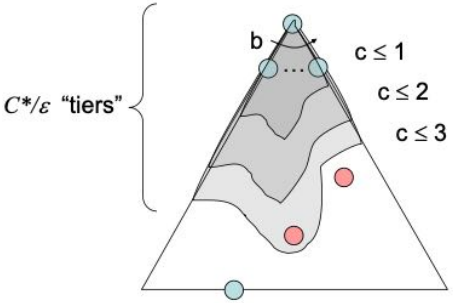
Numbers are distances

1. Expand Sibiu:
   `frontiers=[Fagaras(99), RV(80)]` `visited=[Sibiu]`
2. Expand Fagaras or RV? (RV has the smallest cost)
   `frontiers=[Fagaras(99), Pitesti(177)]` `visited=[Sibiu, RV]`
3. Expand Fagaras or Pitesti? (Fagaras now has the smallest cost)
   `frontiers=[Pitesti(177), Bucharest(310)]`
   `visited=[Sibiu, RV, Fagaras]`
4. Expand Pitesti or Bucharest? (Pitesti has the smallest cost)
   `frontiers=[Bucharest(~~310~~ 278)]`
   `visited=[Sibiu, RV, Fagaras, Pitesti]`
5. Expand Bucharest: *goal state reached*
   Solution: Sibiu → RV → Pitesti → Bucharest (cost=278)

Properties of Uniform-Cost Search

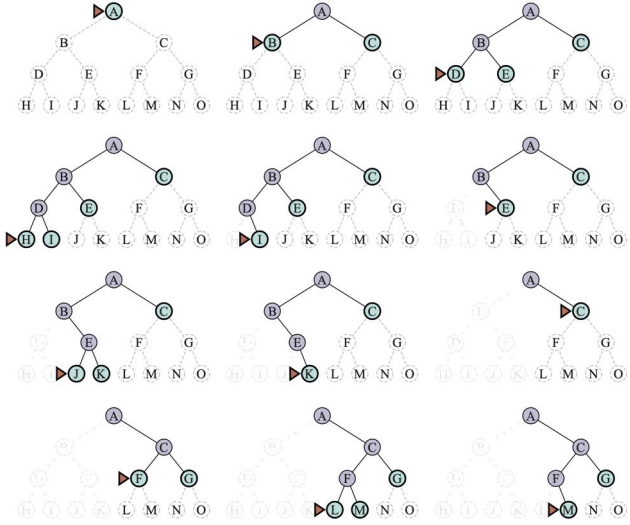
Suppose the solution costs C^* and the lower bound of the cost is $\epsilon > 0$.

- **Nodes expanded?**
  All nodes with cost less than the optimal solution.
- **Is UCS Complete?** Yes with finite cost.
- **Is UCS Cost-Optimal?** Yes!
- **Worse-Case Time complexity?** $O(b^{C^*/\epsilon})$
- **Worse-Case Space needed?** $O(b^{C^*/\epsilon})$



Depth-First Search (DFS)

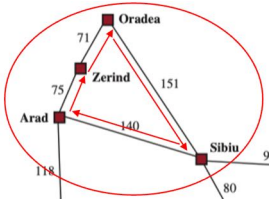
- Search strategy:
  Expand a *deepest* node first.
- Often implemented as tree-like search (does not record visited nodes).



👍 Memory efficient.

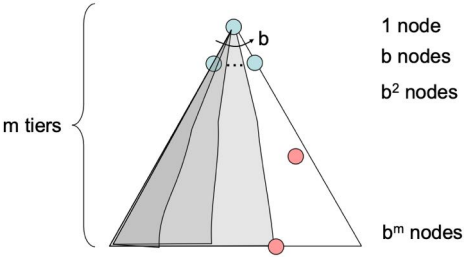
👎 Expand some nodes multiple times.

👎 Can get stuck in infinite loops.



Properties of Depth-First Search

- **Nodes expanded?** Some left prefix of the tree.
  Worst case: the whole tree
- **Is DFS Complete?** No! It can get stuck.
- **Is DFS Cost-Optimal?**: No, it returns the first solution it finds, be it optimal or not.
- **Time complexity?** $O(b^m)$
- **Space needed?**
  frontiers: $b + b + ... + b = O(bm)$
  visited: 0 (not stored)

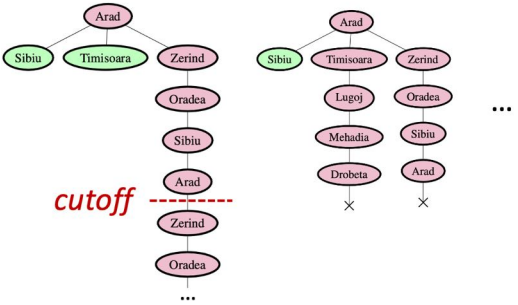


Depth-Limited Search

🤔 Can we avoid getting stuck in wrong path or infinite loops while same memory?

A simple idea: Set a depth limit l and ignore all nodes exceeding this depth.

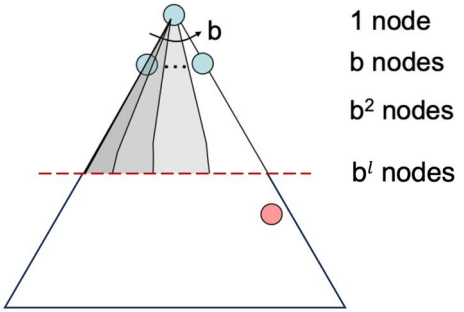
E.g., if we set $l = 4$:



- If the goal is beyond depth l, we cannot find it.
- How to set a good l?
  We need **knowledge** about the problem.
  - There are only 20 cities in the Romania map: $l = 19$?
  - A closer look at the map: from any city, we can reach any other city in at most 9 actions. $l = 9$?

Properties of Depth-Limited Search

- **Nodes expanded?** Some left prefix above depth l.
- **Is Depth-Limited Search Complete?**
  No! Cannot find solution beyond depth l.
- **Is Depth-Limited Search Cost-Optimal?**:
  No, it returns the first solution it finds.
- **Time complexity?** $O(b^l)$
- **Space needed?**
  frontiers: $b + b + ... + b = O(bl)$
  visited: 0 (not stored)

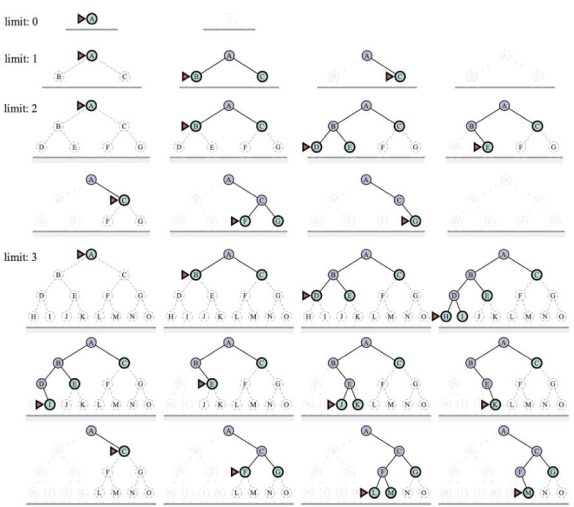


Iterative Deepening Search

What if we cannot know a good l?

Idea: We can try different values of l (from 0 up to infinity) iteratively.

- **Completeness?**
  Yes, guaranteed to find a solution if one exists.
- **Cost-optimality?** Finds the shallowest solution, cost-optimal if all costs are identical.
- **Space complexity?** $O(bd)$ (d: depth of the shallowest solution).
- **Time complexity?**
  - Repeating upper levels takes more time.
  - But upper levels do not have much nodes.
  - Complexity: $O(b^d)$



Exercise: Comparing Uninformed Search Algorithms

Try to summarize and complete the following table:

Property	Breadth-Frist	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?					
Cost-optimal?					
Time					
Space					

Summary for Today

- What is AI? What are the applications of AI?
- Agents, search problems, applications of searching.
- How to formulate a search problem?
- State space graphs and search trees.
- The general search algorithms.
- Graph search vs. tree-like search: definitions? pros and cons?
- How to measure the performance of search algorithms?
  Completeness, optimality, time, and space.
- Uninformed search algorithms: BFS, UCS, DFS, depth-limited, and iterative deepening search.

After-Class Exericises

Practice different search algorithms by finding routes from Rimnicu Vilcea to Vaslui.