CSCI 561 - Foundation for Artificial Intelligence

02. Problem Solving & Search

Professor Wei-Min Shen
University of Southern California

Outline

Problem Solving and Search

- Search Space

Formulation of "Problem Solving" and Search

Complexity of Problems

- Search Algorithms (uninformed vs informed)

- Uninformed Search

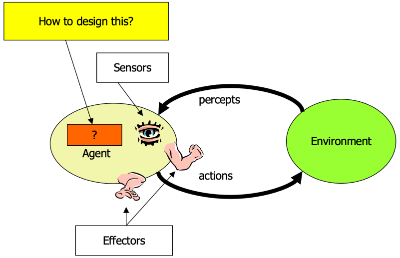
Search strategies: breadth-first, uniform-cost, depth-first, bi-directional, …

- Informed Search (next lecture)

Search strategies: best-first, A*

Heuristic functions

Review of General AI



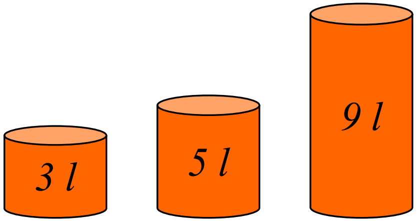
- **Definition of AI?**
- **Turing Test?**
- **Intelligent Agents:**
  - Anything that can be *viewed as* **perceiving** its **environment** through **sensors** and **acting** upon that environment through its **effectors** to maximize progress towards its **goals**.
  - PAGE (Percepts, Actions, Goals, Environment)
  - Described as a Perception (sequence) to Action Mapping: $f: \mathcal{P}^* \rightarrow \mathcal{A}$
  - Using look-up-table, closed form, etc.

- **Agent Types:** Reflex, state-based, goal-based, utility-based

- **Rational Action:** The action that maximizes the expected value of the performance measure <u>given the percept sequence to date</u>

PROBLEM SOLVING
WHAT IS A “PROBLEM”?
HOW TO DEFINE A “PROBLEM”?

An Example of “Problem” (Measuring Water)



Problem: Using these three buckets,
measure 7 liters of water.

Which solution do we prefer?

- **Solution 1:**

a	b	c	
0	0	0	start
3	0	0	
0	0	3	
3	0	3	
0	0	6	
3	0	6	
0	3	6	
3	3	6	
1	5	6	
0	5	**7**	**goal**

- **Solution 2:**

a	b	c	
0	0	0	start
0	5	0	
3	2	0	
3	0	2	
3	5	2	
3	**0**	**7**	**goal**

How to Define a "Problem"?

Measure 7 liters of water using a 3-liter, a 5-liter, and a 9-liter buckets.

1.

- **Formulate Problem:**
  1. ② States: space — amount of water in the buckets
  2. ③ Operators/Actions: — Fill bucket from source, empty bucket
  3. ① Initial State: — three empty buckets
  4. ④ Goal State: — Have 7 liters of water in 9-liter bucket

- **Find solution:** — a sequence of operators/actions that bring your agent from the initial/current state to the goal state

Problem-Solving Agent

```
function SIMPLE-PROBLEM-SOLVING-AGENT( p) returns an action
    inputs: p, a percept
    static: s, an action sequence, initially empty
            state, some description of the current world state
            g, a goal, initially null
            problem, a problem formulation

    state ← UPDATE-STATE(state, p)          // What is the current state?
    if s is empty then
        g ← FORMULATE-GOAL(state)  // From LA to San Diego (given curr. state)
        problem ← FORMULATE-PROBLEM(state, g)  // e.g., Gas usage
        s ← SEARCH( problem)
    action ← RECOMMENDATION(s, state)
    s ← REMAINDER(s, state)             // If fails to reach goal, update
    return action
```

Note: This is *offline* problem-solving. *Online* problem-solving involves acting w/o complete knowledge of the problem and environment

Remember: Environment Types

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Operating System	Yes	Yes	No	No	Yes
Virtual Reality	Yes	Yes	Yes/No	No	Yes/No
Office Environment	No	No	No	No	No
Mars	No	Semi	No	Semi	No

The environment types largely determine the design of agent

2. ## Types of "Problems"

- **Single-state problem:** deterministic, accessible (totally observable)
  Agent knows everything about world, thus can calculate optimal action sequence to reach goal state.

- **Multiple-state problem:** deterministic, inaccessible (partially observable)
  Agent must reason about sequences of actions and states assumed while working towards goal state.

- **Contingency problem:** nondeterministic, inaccessible
  - *Must use sensors during execution*
  - *Solution is a tree or policy*
  - *Often interleave search and execution*

- **Exploration problem:** unknown state space
  Discover and learn about environment while taking actions.

Problem types

(1) - **Single-state problem:** deterministic, accessible

  - Agent knows everything about world (the exact state),
  - Can calculate optimal action sequence to reach goal state.

  - E.g., playing chess. Any action will result in an exact state
  - Why is Chess or Go "accessible"?

Problem types

(2) - **Multiple-state problem:** deterministic, inaccessible

- Agent does not know the exact state (could be in any of the possible states)
  - May not have sensors at all

- Assume states while working towards goal state.

- E.g., walking in a dark room, or playing the poker game
  - If you are at the door, going straight will lead you to the kitchen
  - If you are at the kitchen, turning left leads you to the bedroom
  - …

Problem types

(3) • **Contingency problem:** nondeterministic, inaccessible

- Must use sensors during execution
- Solution is a tree or policy
- Often interleave search and execution

- E.g., a new skater in an arena
  - Sliding problem.
  - Many skaters around

Problem types

(4) • **Exploration problem:** unknown state space

Discover and learn about environment while taking actions.

- *E.g., Maze, or Mars*

Example 1: Vacuum world

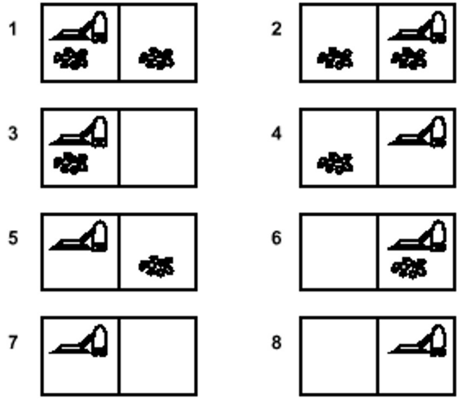
Simplified world: 2 locations, each may or not contain dirt, each may or not contain vacuuming agent.

Goal of agent: clean up the dirt.

Single-state, start in #5. Solution??

Multiple-state, start in $\{1, 2, 3, 4, 5, 6, 7, 8\}$
e.g., $Right$ goes to $\{2, 4, 6, 8\}$. Solution??

Contingency, start in #5
Murphy's Law: $Suck$ can dirty a clean carpet
Local sensing: dirt, location only.
Solution??



Example 2: Traveling in Romania

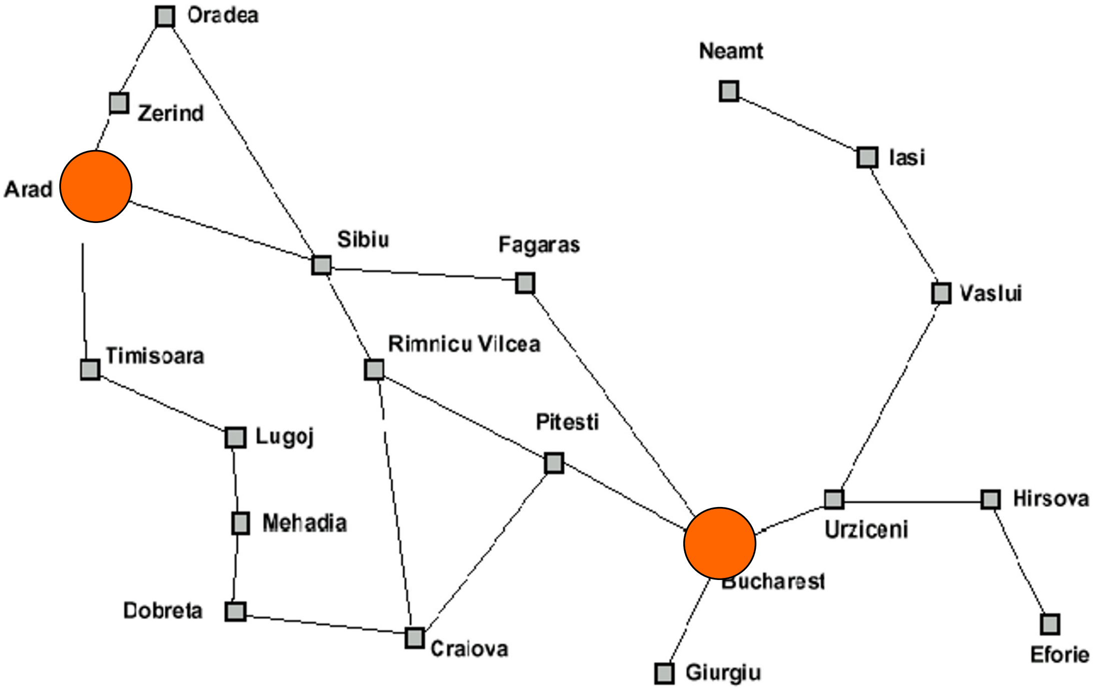
- In Romania, on vacation. Currently in Arad.
- Flight leaves tomorrow from Bucharest.

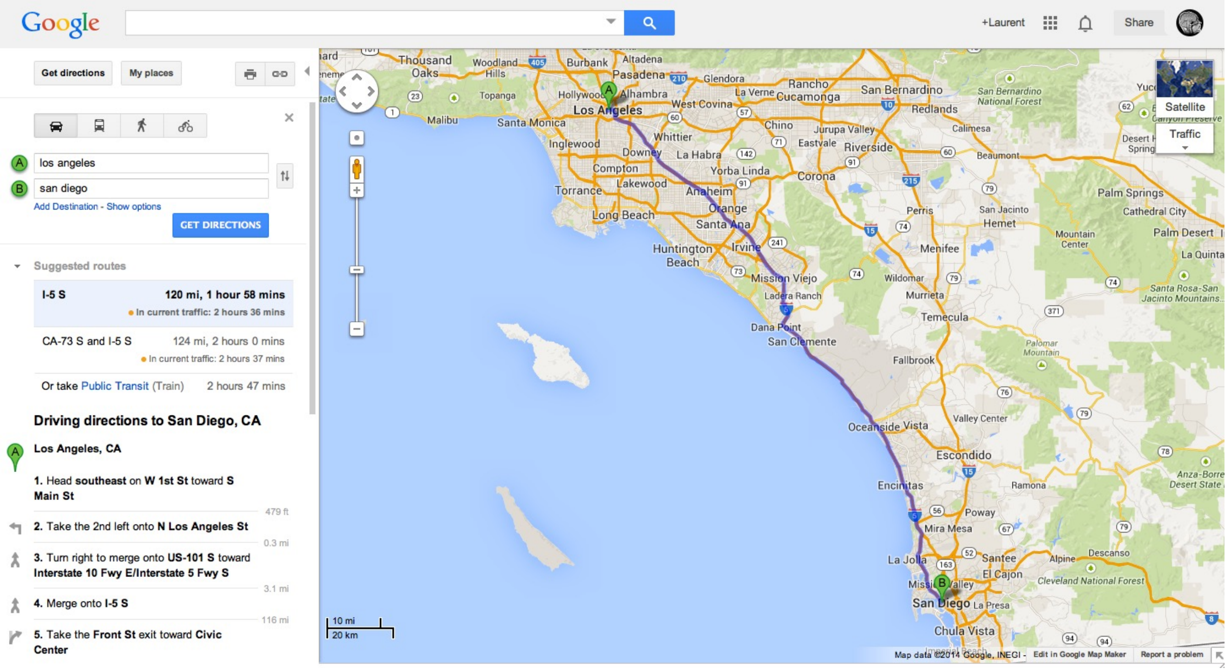
- **Formulate goal:**
    - Be in Bucharest

- **Formulate problem:**
    - States: various cities
    - Operators: drive between cities

- **Find solution:**
    - Sequence of cities, such that total driving distance is minimized.

Example: Traveling from Arad To Bucharest




Google
+Laurent
Share
Get directions
My places
los angeles
san diego
Add Destination - Show options
GET DIRECTIONS
Suggested routes
I-5 S 120 mi, 1 hour 58 mins
In current traffic: 2 hours 36 mins
CA-73 S and I-5 S 124 mi, 2 hours 0 mins
In current traffic: 2 hours 37 mins
Or take Public Transit (Train) 2 hours 47 mins
Driving directions to San Diego, CA
Los Angeles, CA
1. Head southeast on W 1st St toward S Main St
479 ft
2. Take the 2nd left onto N Los Angeles St
0.3 mi
3. Turn right to merge onto US-101 S toward Interstate 10 Fwy E/Interstate 5 Fwy S
3.1 mi
4. Merge onto I-5 S
116 mi
5. Take the Front St exit toward Civic Center
Satellite
Traffic
Los Angeles
Pasadena
Santa Monica
Long Beach
Anaheim
Santa Ana
Irvine
Mission Viejo
San Clemente
Oceanside
Escondido
La Jolla
San Diego
Chula Vista
Riverside
San Bernardino
Temecula
Palm Springs
10 mi
20 km
Map data ©2014 Google, INEGI
Edit in Google Map Maker
Report a problem

The General Formulation of "Problem"

A *problem* is defined by four items:

<u>*initial state*</u> e.g., "at Arad"

<u>*operators*</u> (or *successor function* $S(x)$)
e.g., Arad $\rightarrow$ Zerind Arad $\rightarrow$ Sibiu etc.

<u>*goal test*</u>, can be
explicit, e.g., $x =$ "at Bucharest"
implicit, e.g., $NoDirt(x)$

<u>*path cost*</u> (additive)
e.g., sum of distances, number of operators executed, etc.

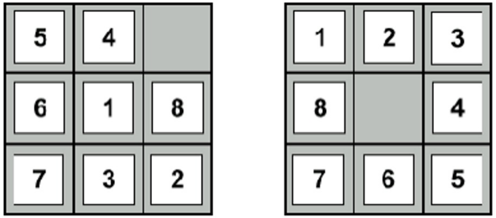
A *solution* is a sequence of operators
leading from the initial state to a goal state

Selecting a Space of States

- Real world is absurdly complex; some abstraction is necessary to allow us to reason on it…
- Selecting the correct abstraction and resulting state space is a difficult problem!
- Abstract states ⇔ real-world states
- Abstract operators ⇔ sequences or real-world actions

  (e.g., going from city i to city j costs Lij ⇔ actually drive from city i to j)
- Abstract solution ⇔ set of real actions to take in the real world such as to solve problem

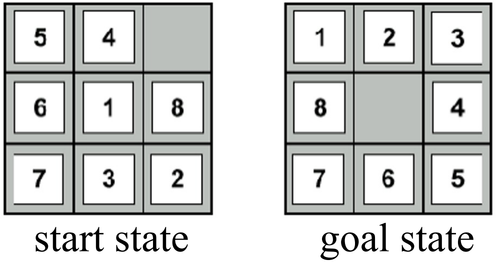
Example: 8-puzzle



start state   goal state

- State: integer location of tiles (ignore intermediate locations)
- Operators: moving blank left, right, up, down (ignore jamming)
- Goal test: does state match goal state?
- Path cost: 1 per move

Example: 8-puzzle

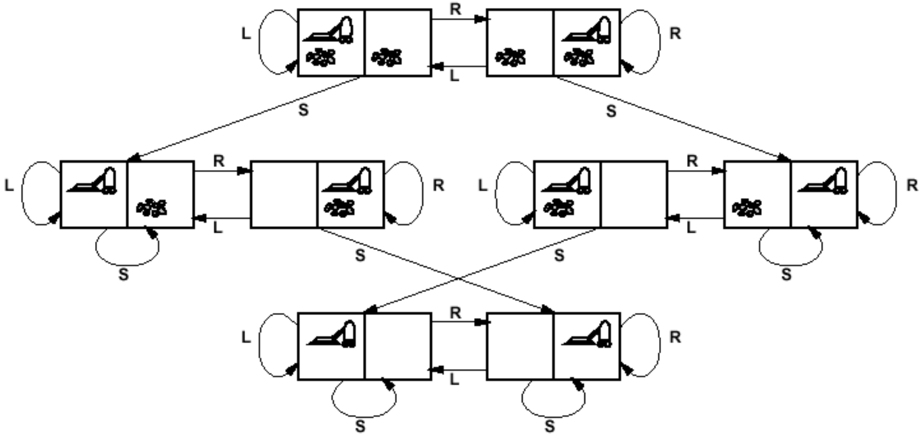


Why do we need search algorithms?

- 8-puzzle has 362,880 states
- 15-puzzle has 10^12 states
- 24-puzzle has 10^25 states

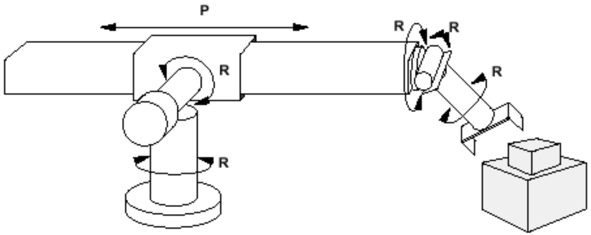
So, we need a principled way to look for a solution in these huge search spaces…

Back to Vacuum World



states??: integer dirt and robot locations (ignore dirt *amounts*)
operators??: *Left*, *Right*, *Suck*
goal test??: no dirt
path cost??: 1 per operator

Example: Robotic Assembly



states??: real-valued coordinates of
robot joint angles
parts of the object to be assembled

operators??: continuous motions of robot joints

goal test??: complete assembly *with no robot included!*

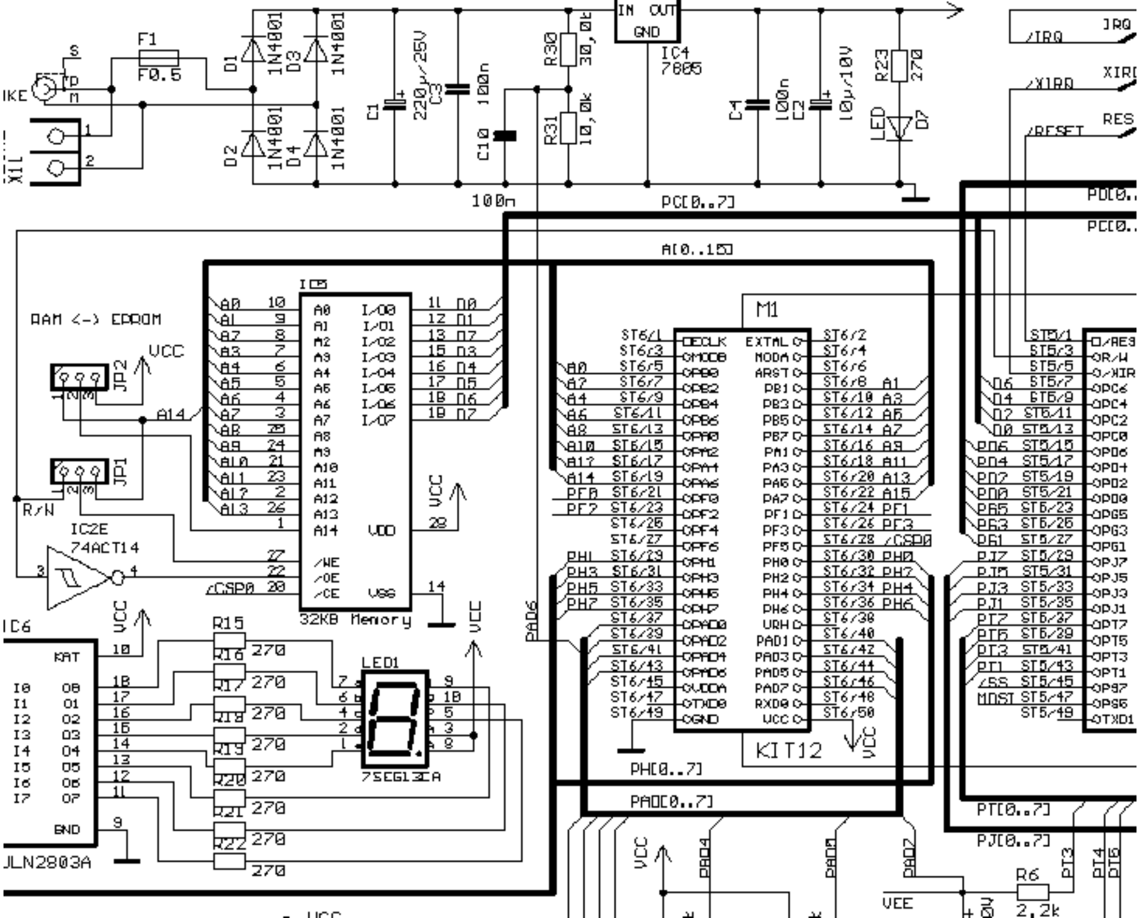
path cost??: time to execute

A Real-life Example: Circuit Board Layout

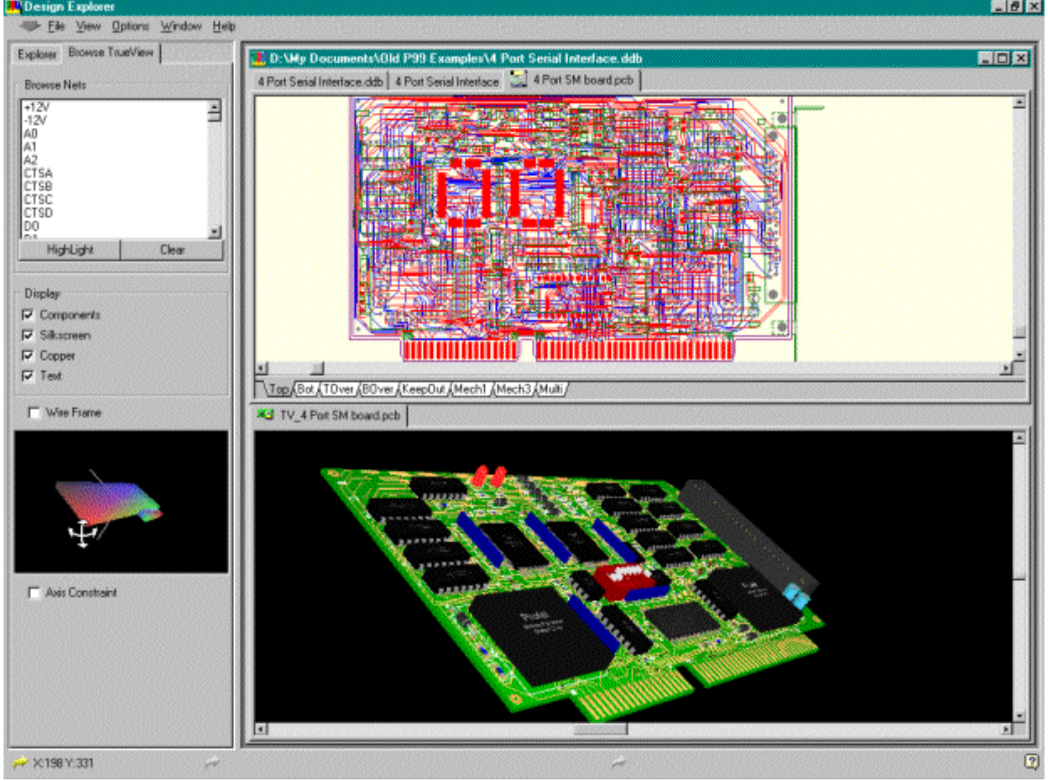
- Given schematic diagram comprising components (chips, resistors, capacitors, etc) and interconnections (wires), find optimal way to place components on a printed circuit board, under the constraint that only a small number of wire layers are available (and wires on a given layer cannot cross!)

- "optimal way"??

- ➢ minimize surface area
- ➢ minimize number of signal layers
- ➢ minimize number of vias (connections from one layer to another)
- ➢ minimize length of some signal lines (e.g., clock line)
- ➢ distribute heat throughout board
- ➢ etc.



Use automated tools to place components and route wiring.



Protel 99 SE's unique 3D visualization feature lets you see your finished board before it leaves your desktop. Sophisticated 3D modeling and extrusion techniques render your board in stunning 3D without the need for additional height information. Rotate and zoom to examine every aspect of your board.

SEARCH FOR SOLUTIONS

Search Algorithms

Basic idea:

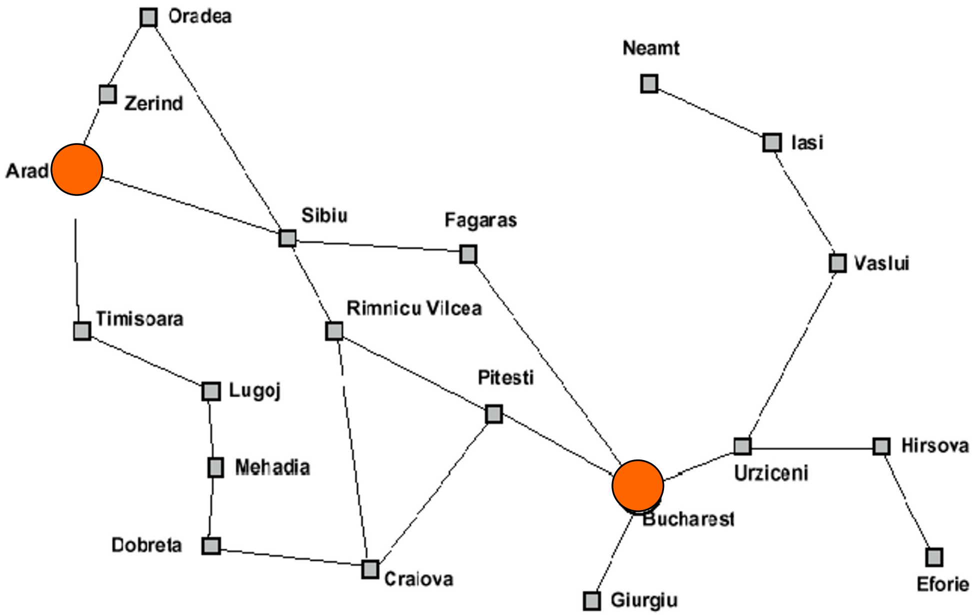
offline, systematic exploration of simulated state-space by generating successors of explored states (expanding)

Function General-Search(*problem*, *strategy*) returns a *solution*, or failure

    initialize the search tree using the initial state problem
    loop do
        if there are no candidates for expansion **then return** failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state **then**
            return the corresponding solution
        else expand the node and add resulting nodes to the search tree
    end

Example: A micro_mouse searches in a maze

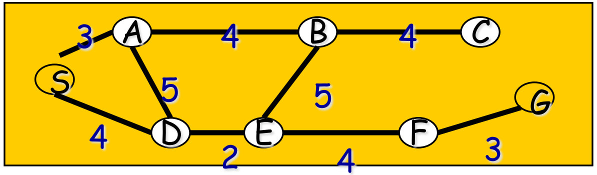
Example: Traveling from Arad To Bucharest



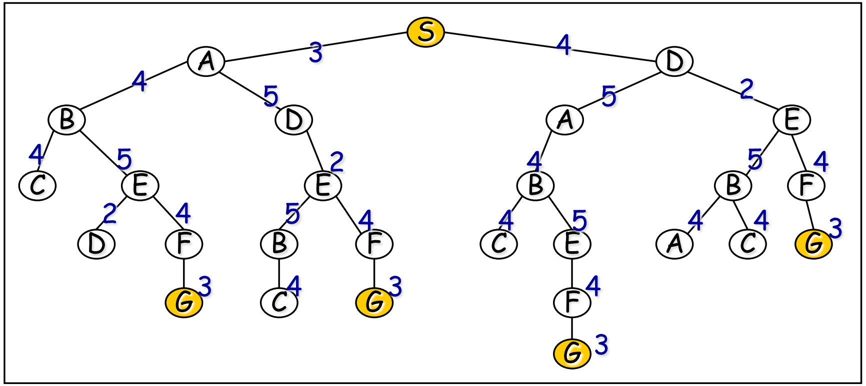
From Problem Space to Search Tree

- Some material in this and following slides is from http://www.cs.kuleuven.ac.be/~dannyd/FAI/ check it out!

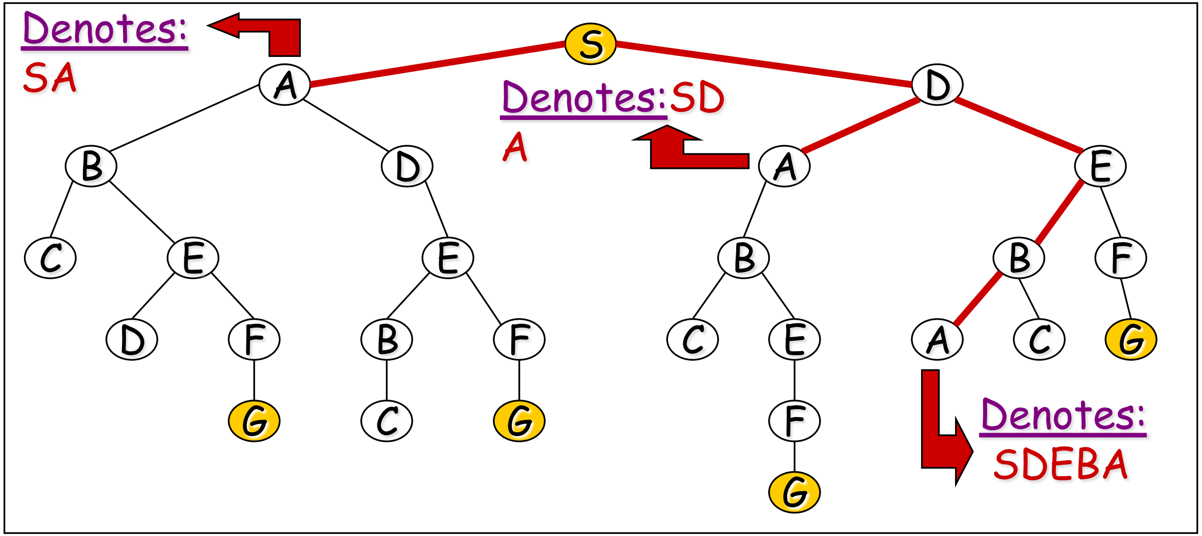
Problem space



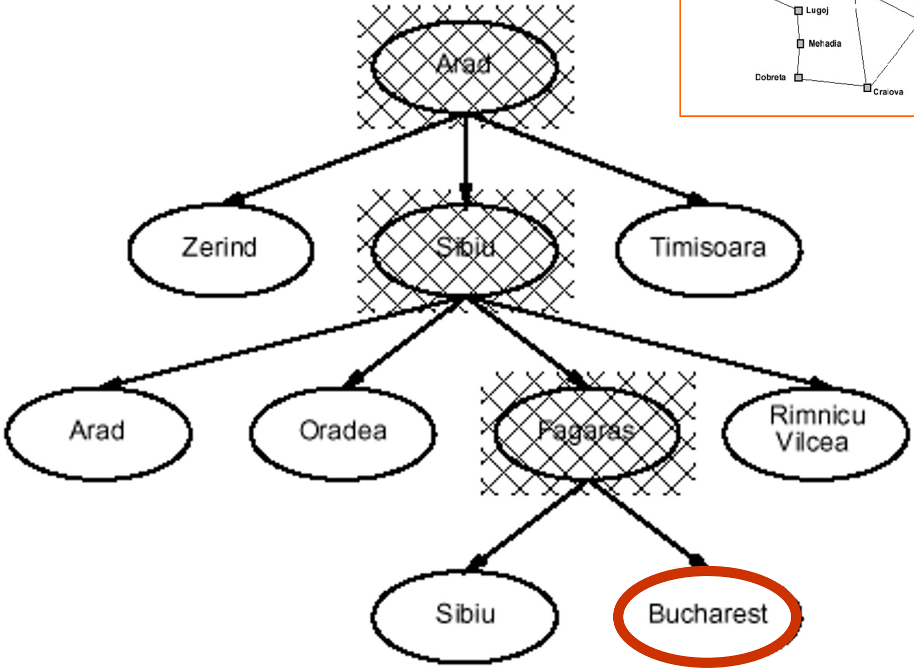
Associated loop-free search tree

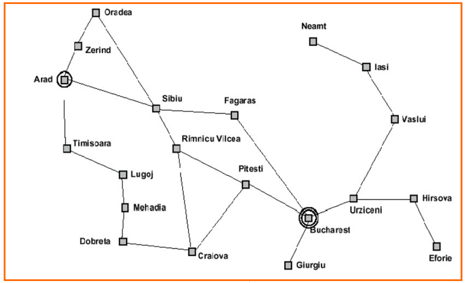


Paths in Search Trees



A General Search Example





An Implementation of Search Algorithms

```
Function General-Search(problem, Queuing-Fn) returns a solution, or failure
    nodes ← make-queue(make-node(initial-state[problem]))
    loop do
            if nodes is empty then return failure
            node ← Remove-Front(nodes)
            if Goal-Test[problem] applied to State(node) succeeds then return node
            nodes ← Queuing-Fn(nodes, Expand(node, Operators[problem]))
    end
```

Queuing-Fn(*queue, elements*) is a queuing function that inserts a set of elements into the queue and <u>determines the order of node expansion</u>. Varieties of the queuing function produce varieties of the search algorithm.

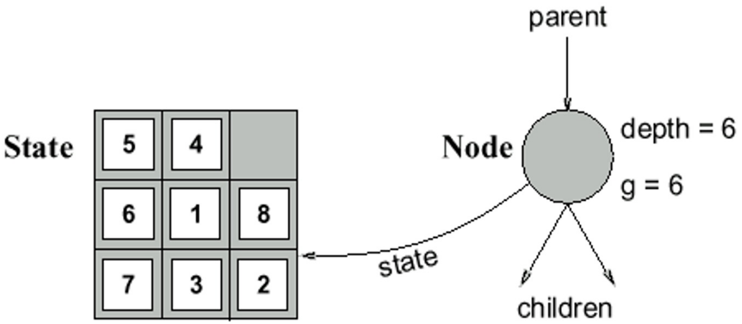
Encapsulating *state* information in *nodes*

A *state* is a (representation of) a physical configuration

A *node* is a data structure constituting part of a search tree

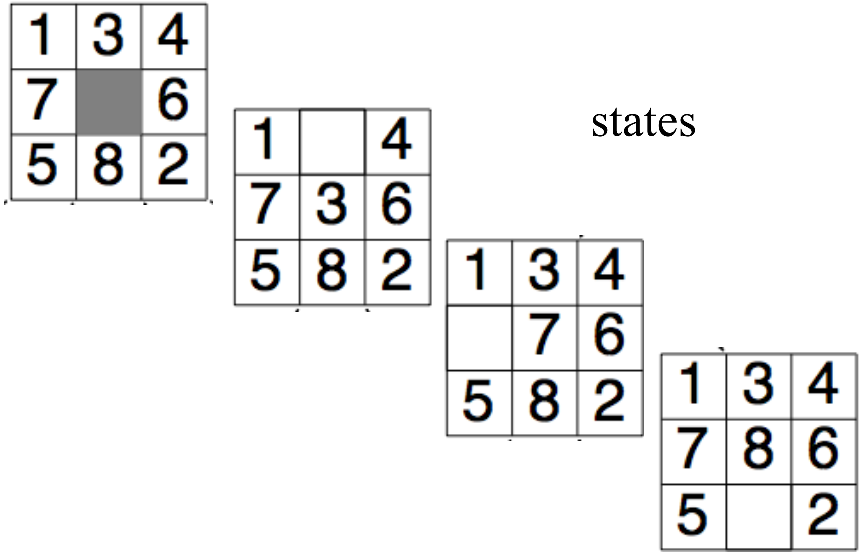
includes *parent*, *children*, *depth*, *path cost* $g(x)$

States do not have parents, children, depth, or path cost!



The EXPAND function creates new nodes, filling in the various fields and using the OPERATORS (or SUCCESSORFN) of the problem to create the corresponding states.

Paths in search trees

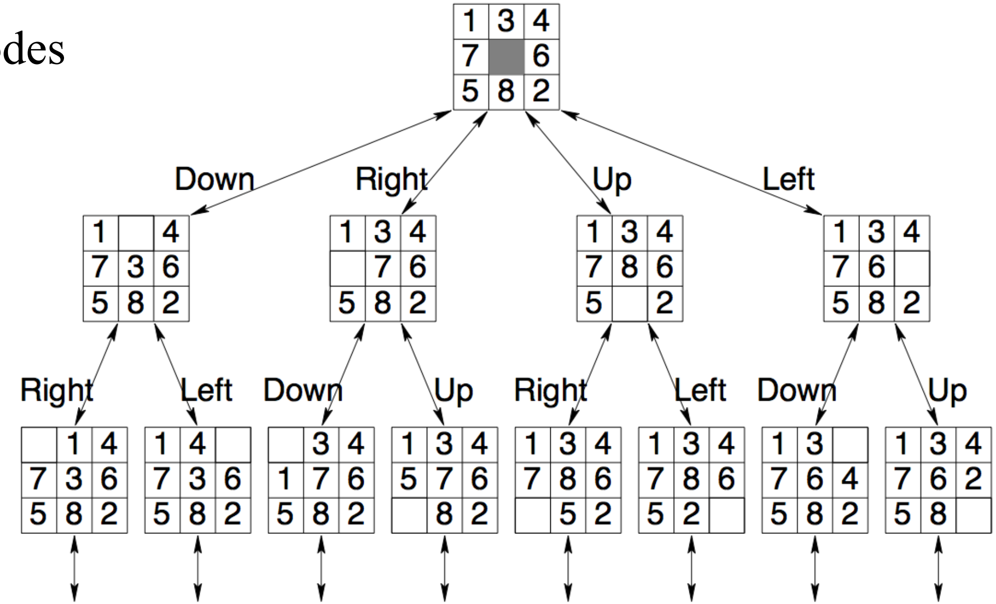


states

…

Paths in search trees

Search tree nodes



1. **Evaluation of search strategies**

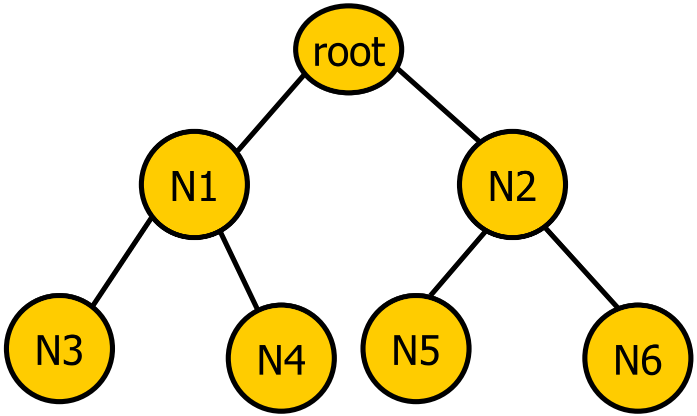
- A search strategy is defined by picking the order of node expansion
- Search algorithms are commonly evaluated according to the following four criteria:
  - **Completeness**: does it always find a solution if one exists?
  - **Time Complexity**: how long does it take as function of num. of nodes?
  - **Space Complexity**: how much memory does it require?
  - **Optimality**: does it guarantee the least-cost solution?
- Time and space complexity are measured in terms of:
  - b – max branching factor of the search tree
  - d – depth of the least-cost solution
  - m – max depth of the search tree (may be infinity)

Binary Tree Example

Depth = 0

Depth = 1

Depth = 2



Number of nodes at max depth: $n = 2^{\text{max depth}}$

Number of levels (given n at max depth) = log2(n)

COMPLEXITY OF PROBLEMS

Complexity

- Why worry about complexity of algorithms?

➢ because a problem may be solvable in principle but may take too long to solve in practice

Complexity: Tower of Hanoi

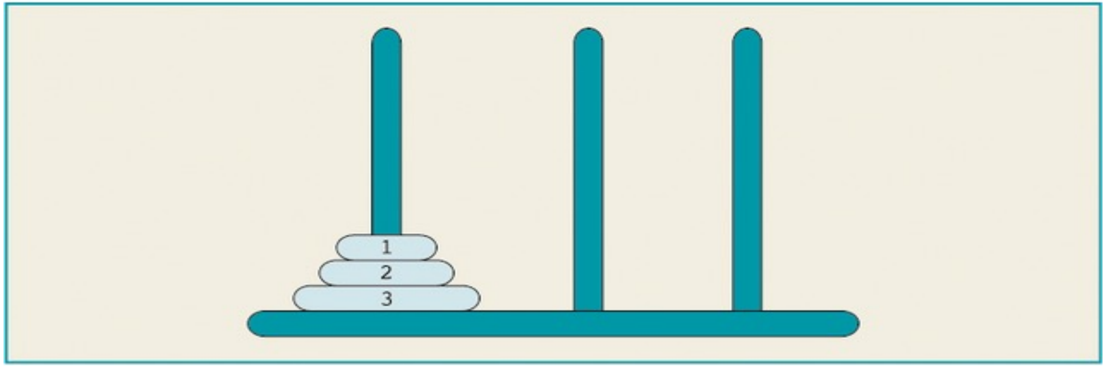


Figure 11-6 Tower of Hanoi problem with three disks

Complexity: Tower of Hanoi

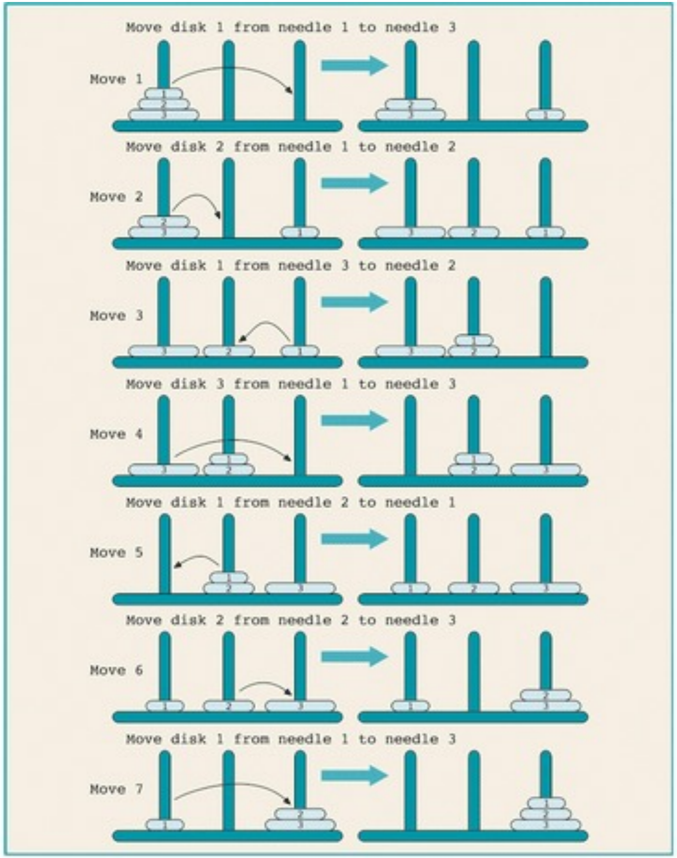


Figure 11-7 Solution of Tower of Hanoi problem with three disks

Complexity: Tower of Hanoi

- 3-disk problem: $2^3 - 1 = 7$ moves
- 64-disk problem: $2^{64} - 1$.
  - $2^{10} = 1024 \approx 1000 = 10^3$,
  - $2^{64} = 2^4 * 2^{60} \approx 2^4 * 10^{18} = 1.6 * 10^{19}$
- One year $\approx 3.2 * 10^7$ seconds

Complexity: Tower of Hanoi

- The wizard's speed = one disk / second

$$1.6 * 10^{19} = 5 * 3.2 * 10^{18} =$$

$$5 * (3.2 * 10^{7}) * 10^{11} =$$

$$(3.2 * 10^{7}) * (5 * 10^{11})$$

500 billion years

Complexity: Tower of Hanoi

- The time required to move all 64 disks from needle 1 to needle 3 is roughly $5 * 10^{11}$ years.

- It is estimated that our universe is about 15 billion = $1.5 * 10^{10}$ years old.

$$5 * 10^{11} = 50 * 10^{10} \approx 33 * (1.5 * 10^{10}).$$

Complexity: Tower of Hanoi

- Assume: a computer with 1 billion = 10^9 moves/second.
  - Moves/year=$(3.2 * 10^7) * 10^9 = 3.2 * 10^{16}$
- To solve the problem for 64 disks:
  - $2^{64} \approx 1.6 * 10^{19} = 1.6 * 10^{16} * 10^3 =$

    $(3.2 * 10^{16}) * 500$
  - 500 years for the computer to generate 2^{64} moves at the rate of 1 billion moves per second.

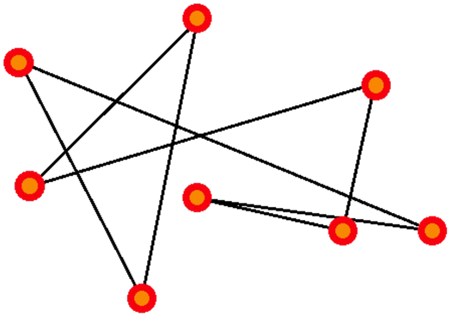
Complexity

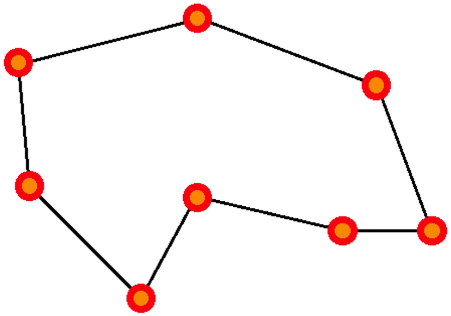
- Why worry about complexity of algorithms?
   - because a problem may be solvable in principle but may take too long to solve in practice

- How can we evaluate the complexity of algorithms?
   - through asymptotic analysis, i.e., estimate time (or number of operations) necessary to solve an instance of size n of a problem when n tends towards infinity
   - See AIMA, Appendix A.

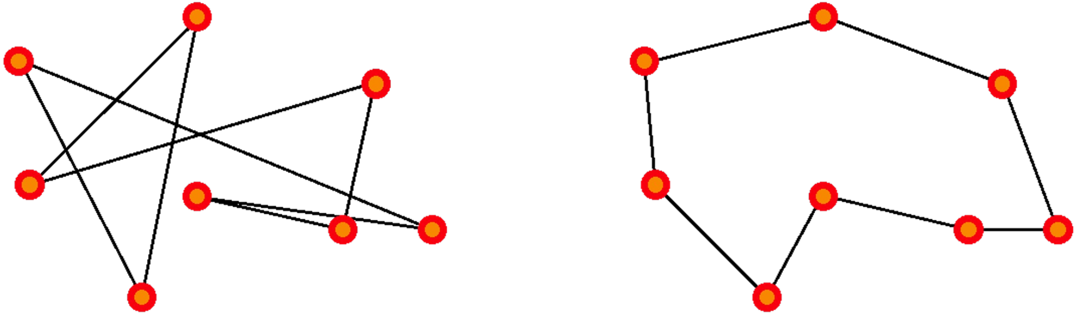
Complexity example: Traveling Salesman Problem

- There are n cities, with a road of length L_{ij} joining city i to city j.
- The salesman wishes to find a way to visit all cities that is optimal in two ways:
  each city is visited only once, and
  the total route is as short as possible.





Complexity example: Traveling Salesman Problem



This is a *hard* problem: the only known algorithms (so far) to solve it have exponential complexity, that is, the number of operations required to solve it grows as *exp(n)* for *n* cities.

Why is exponential complexity "hard"?

It means that the number of operations necessary to compute the exact solution of the problem grows exponentially with the size of the problem (here, the number of cities).

- exp(1) = 2.72
- exp(10) = 2.20 10^4 (daily salesman trip)
- exp(100) = 2.69 10^{43} (monthly salesman planning)
- exp(500) = 1.40 10^{217} (music band worldwide tour)
- exp(250,000) = $10^{108,573}$ (fedex, postal services)

- Fastest computer = 10^{12} operations/second

So…

In general, exponential-complexity problems *cannot be solved for any but the smallest instances!*

Complexity

- Polynomial-time (P) problems: we can find algorithms that will solve them in a time (=number of operations) that grows polynomially with the size of the input.

➢ for example: sort n numbers into increasing order: poor algorithms have *n^2* complexity, better ones have *n log(n)* complexity.

Complexity

- Since we did not state what the order of the polynomial is, it could be very large! Are there algorithms that require more than polynomial time?

- Yes (until proof of the contrary); for some algorithms, we do not know of any polynomial-time algorithm to solve them. These belong to the class of nondeterministic-polynomial-time (NP) algorithms (which includes P problems as well as harder ones).

➢ for example: traveling salesman problem.

- In particular, exponential-time algorithms are believed to be NP.

Note on NP-hard problems

- The formal definition of NP problems is:

A problem is nondeterministic polynomial if there exists some algorithm that can guess a solution and then verify whether or not the guess is correct in polynomial time.

(one can also state this as these problems being solvable in polynomial time on a nondeterministic Turing machine.)

In practice, until proof of the contrary, this means that known algorithms that run on known computer architectures will take more than polynomial time to solve the problem.

Complexity: *O* () and *o*() measures (Landau symbols)

- How can we represent the complexity of an algorithm?

- Given: Problem input (or instance) size: *n*
  Number of operations to solve problem: *f(n)*

- If, for a given function g(n), we have: // for some *k*

$$\exists k \in \Re, \exists n_0 \in \mathrm{N}, \forall n \in \mathrm{N}, n \geq n_0, f(n) \leq kg(n)$$

then $f \in O(g)$ f is dominated by g

- If, for a given function g(n), we have: // for all *k*

$$\forall k \in \Re, \exists n_0 \in \mathrm{N}, \forall n \in \mathrm{N}, n \geq n_0, f(n) \leq kg(n)$$

then $f \in o(g)$ f is negligible compared to g

Landau symbols

$$f \in O(g) \Leftrightarrow \underline{\exists k}, f(n) \underset{n\to\infty}{\leq} kg(n) \Leftrightarrow \frac{f}{g} \text{ is bounded}$$

$$f \in o(g) \Leftrightarrow \underline{\forall k}, f(n) \underset{n\to\infty}{\leq} kg(n) \Leftrightarrow \frac{f(n)}{g(n)} \underset{n\to\infty}{\longrightarrow} 0$$

Examples, Properties

- $f(n)=n$, $g(n)=n^2$:

  n is $o(n^2)$, because $n/n^2 = 1/n \to 0$ as $n \to$ infinity

  similarly, $\log(n)$ is $o(n)$

  n^C is $o(\exp(n))$ for any C

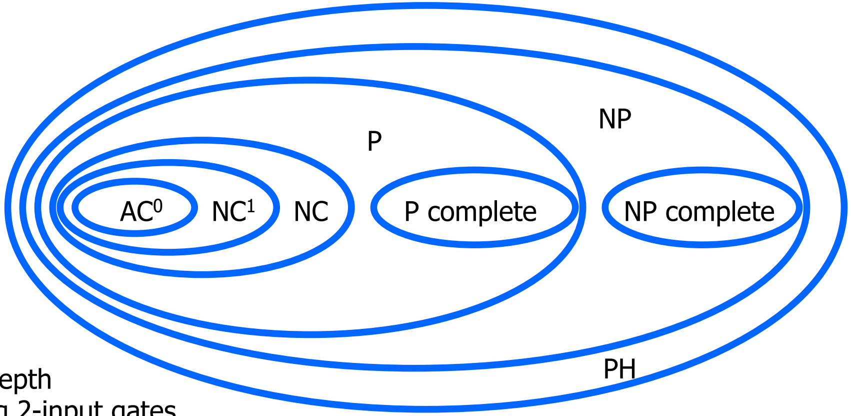
- if f is $O(g)$, then for any K, K*f is also O(g); idem for o()
- if f is $O(h)$ and g is O(h), then for any K, L: (K*f + L*g) is O(h)

  idem for o()

- if f is O(g) and g is O(h), then f is O(h)
- if f is O(g) and g is o(h), then f is o(h)
- if f is o(g) and g is O(h), then f is o(h)

Polynomial-time hierarchy

See Handbook of Brain Theory & Neural Networks
(Arbib, ed.; MIT Press 1995).



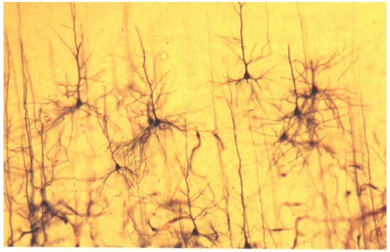
AC^0: can be solved using gates of constant depth
NC^1: can be solved in logarithmic depth using 2-input gates
NC: can be solved by small, fast parallel computer
P: can be solved in polynomial time
P-complete: hardest problems in P; if one of them can be proven to be NC, then P = NC
NP: nondeterministic-polynomial algorithms
NP-complete: hardest NP problems; if one of them can be proven to be P, then NP = P
PH: polynomial-time hierarchy

Complexity and Human Brain

- Are computers close to human brain power?

- Current computer chip (CPU):
  - 10^3 inputs (pins)
  - 10^7 processing elements (gates)
  - 2 inputs per processing element (fan-in = 2)
  - processing elements compute boolean logic (OR, AND, NOT, etc)

- Typical human brain:
  - 10^7 inputs (sensors)
  - 10^10 processing elements (neurons)
  - fan-in = 10^3
  - processing elements compute complicated functions



Still a lot of improvement needed for computers; but computer clusters come close!

Summary

- This Week:
- Problem formulation usually requires abstracting away real-world details to define a state space that can be explored using computer algorithms.
- Once problem is formulated in abstract form, complexity analysis helps us picking out best algorithm to solve problem.

- Next Week:
- Variety of uninformed search strategies; difference lies in method used to pick node that will be further expanded.
- Iterative deepening search only uses linear space and not much more time than other uniformed search strategies.