COMP3620 / 6320 – S1 2023

This lecture will be recorded

Turn off your camera if you do not want to be in the recording

If you cannot see my whole slide, go to **View Options -> Zoom Ratio** and select **Fit to Window**

If you follow the class online and want to ask a question, either:

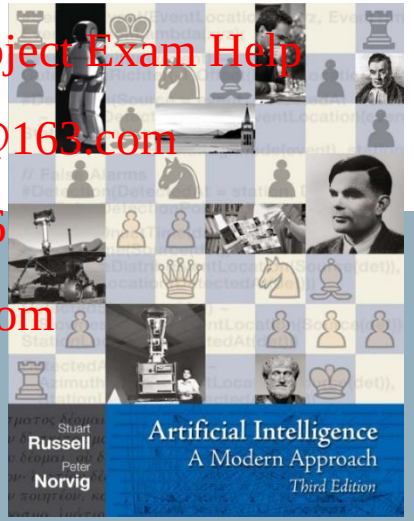
- Raise your hand and I will open your mic, or
- Type the question in the chat box
- Since the lecture is live in person, I cannot constantly monitor the online part, so please be patient

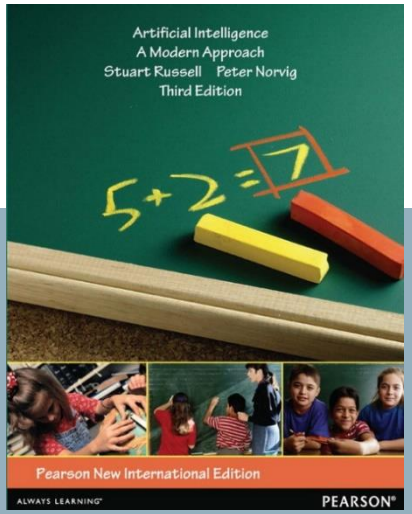
Reminders

- **Class representatives**
  - Please nominate yourself by sending a private message on piazza by today.
  - You are free to nominate yourself whether you are currently on-campus or studying remotely.
- **Labs and Tutorials**
  - Sign-up available on wattle. Please use the same on Mytimetable, wattle selection has preference.
  - No need to sign up for labs
  - First tutorial and lab: **14-20 March 2023 (Tuesday to Monday)**
  - **First tutorial question sheet is out now on wattle**

Intelligent Agents

Chapter 2





PEAS

To design a rational agent, we must specify the **task environment**

Consider, e.g., the task of designing a **driverless taxi**:

- **P**erformance measure:
  - safety, destination, profits, legality, comfort, …
- **E**nvironment:
  - streets/freeways, traffic, pedestrians, weather, …
- **A**ctuators:
  - steering, accelerator, brake, horn, blinkers, …
- **S**ensors:
  - GPS, video, accelerometers, gauges, engine sensors, …

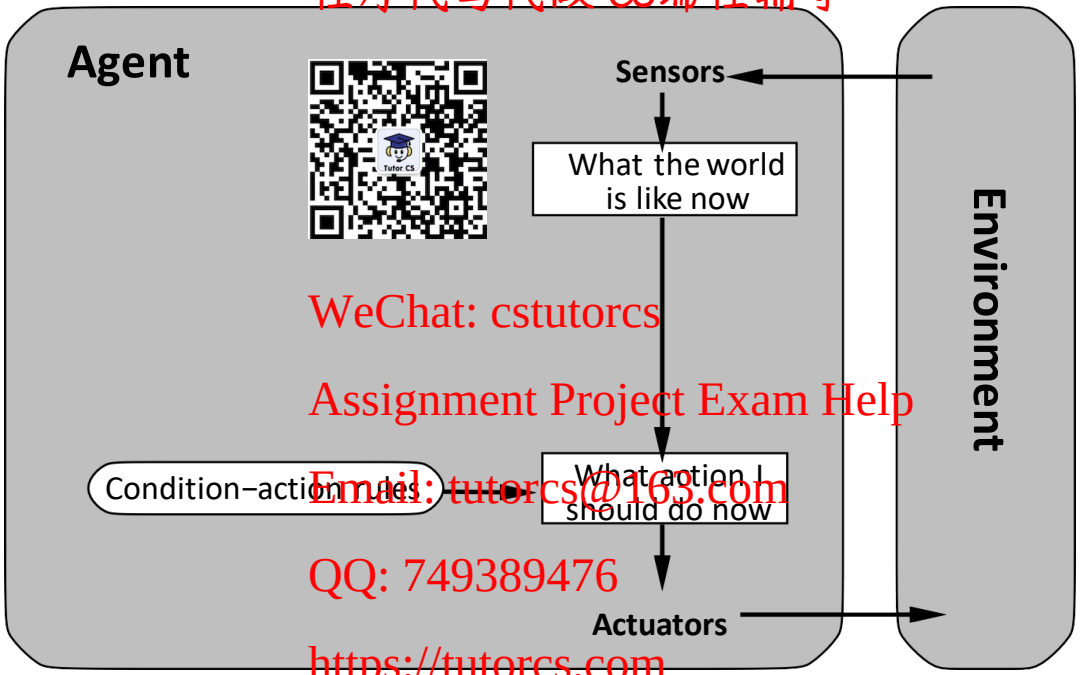
Properties of Task Environments

- **Fully vs partially observable**
  - do the agent sensors give acce[illegible]levant information about the environment state?
- **Deterministic vs stochastic**
  - is the next state completely de[illegible]t by the current state and executed action?
- **Known vs unknown**
  - does the agent know the environment's laws of physics?
- **Episodic vs sequential**
  - is the next decision independent of the previous ones?
- **Static vs dynamic**
  - can the environment change whilst the agent is deliberating?
  - **Semi-dynamic:** only the performance score changes.
- **Discrete vs continuous**
  - can time, states, actions, percepts be represented in a discrete way?
- **Single vs multi-agent**
  - is a single agent making decisions, or do multiple agents need to compete or cooperate to maximise interdependent performance measures?

Agent types

- Four basic types of agents in order of increasing generality:
  - simple reflex agents
  - reflex agents with state
  - goal-based agents
  - utility-based agents
- All these can be turned into learning agents

Simple reflex agents

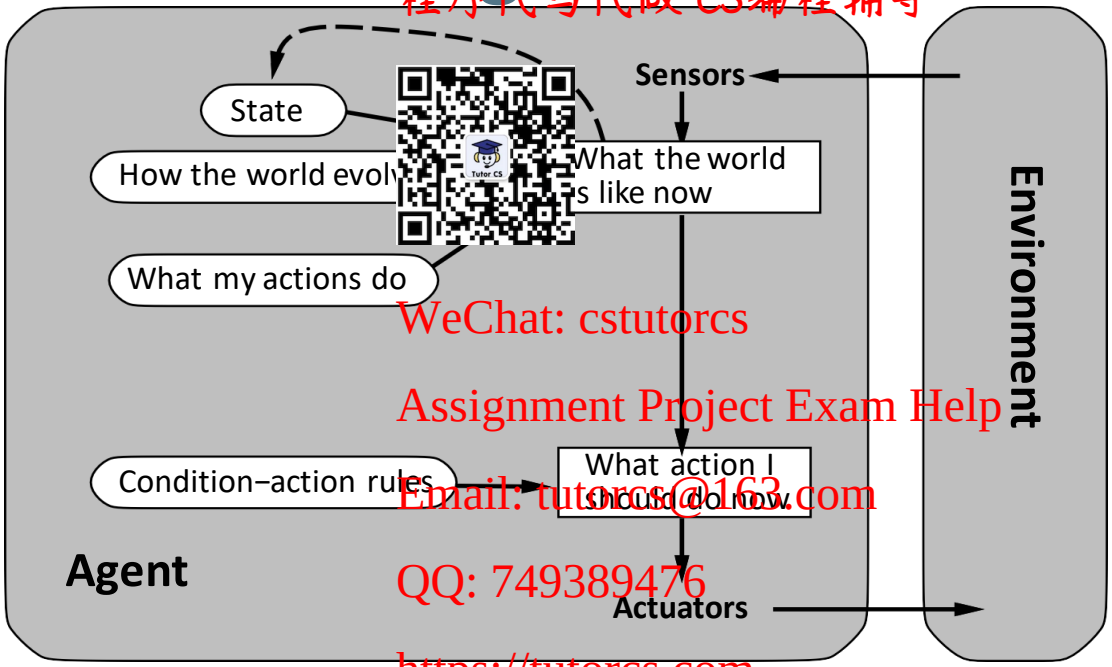


Decisions are made based on the current percept only. Raises issues for partially observable environments.

Example


```
function REFLEX-VACUUM-AGENT ([location,status]) returns an action
    if status = Dirty then return Suck
    else if location = A then return Right
    else if location = B then return Left
```

Reflex agents with state



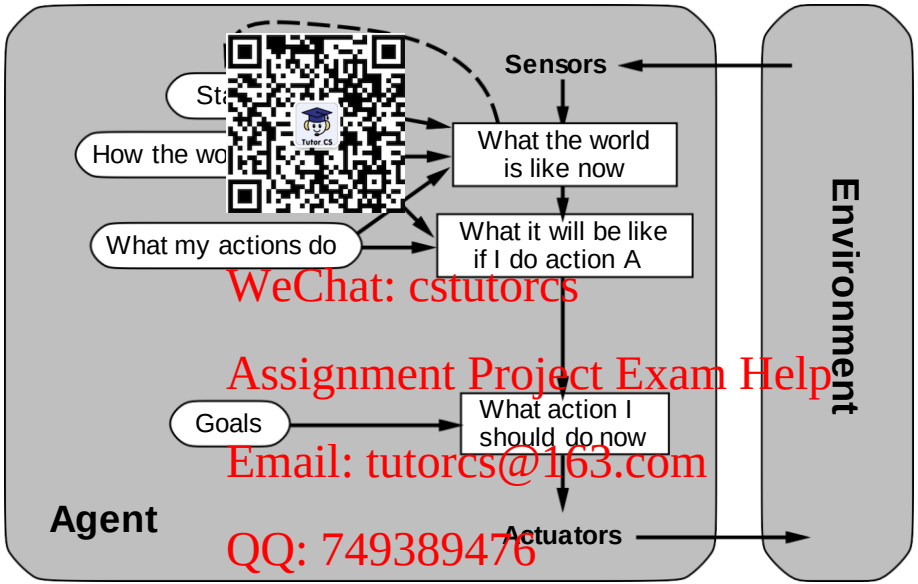
The internal state keeps track of relevant unobservable aspects of the environment. The environment model describes how the environment works (how the environment state is affected by actions)

Example


```
function VACUUM-AGENT-WITH-STATE( (location,status)) returns an action
static: last_A, last_B, n                                    lly ∞
    increment last_A and
    if location = A then
    else last_B = 0
    case
        status = Dirty:
            return Suck
        location = A:
            if last_B > 3 then return Right
            else return NoOp
        location = B:
            if last_A > 3 then return Left
            else return NoOp
```

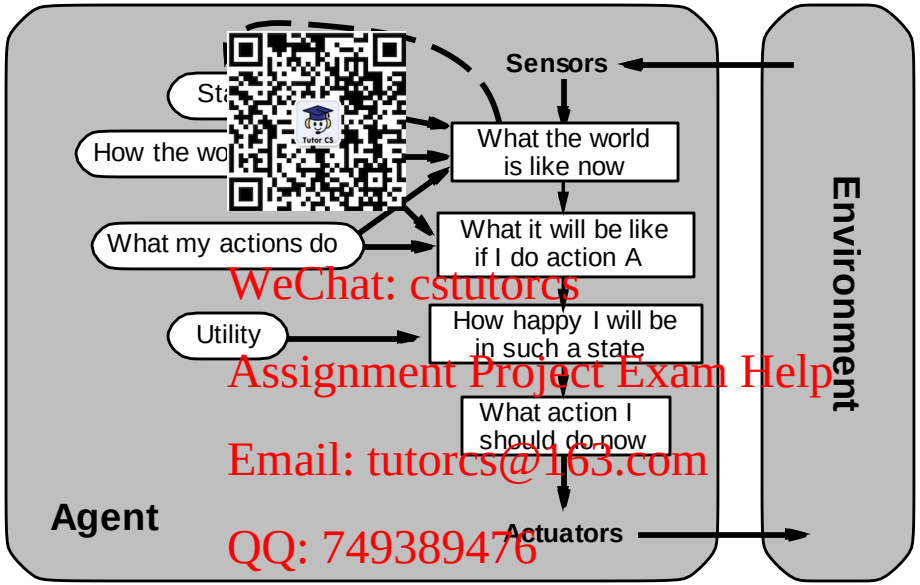

The time passed since a location was visited is a proxy for the likelihood of this location's status changing from clean to dirty.

Goal-Based agents



- The goal describes desirable situations.
- The agent combines goal and environment model to choose actions.
- Planning and search are AI subfields devoted to building goal-based agents.

Utility-based agents



- The utility function internalises the performance measure.
- Under uncertainty, the agent chooses actions that maximise the expected utility.

Utility-based agents

Rational agent: chooses the action that maximises expected utility:

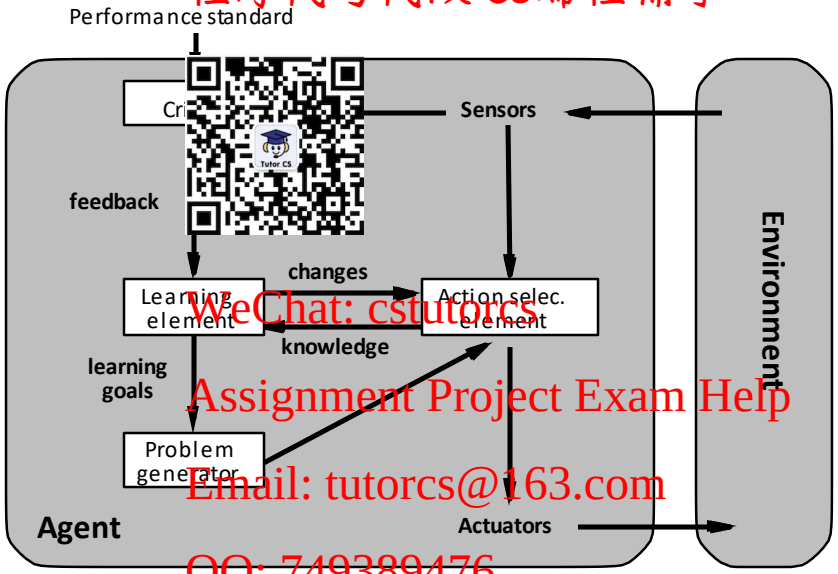
Utility function:
1 per clean room
-0.5 per action

Expected utility of *Suck*:

$$p1 \times u1 + p2 \times u2 = 0.7 \times 2.5 + 0.3 \times 1.5 = 2.2$$

- *Suck* has an expected utility of 2.2
- *NoOp* has an expected utility of 2
- *Left* has an expected utility of 1.5

Learning agents



- The action selection element is what we described earlier.
- The learning element uses feedback from the critic to modify the action selection.
- The problem generator suggests actions that lead to new informative experience.

Exploration vs Exploitation

A fundamental dilemma for learning agents:

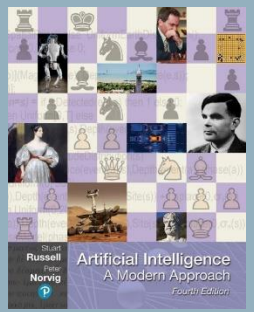
- **Exploitation**: greedily uses what the agent has learnt to select the action that will, in the light of the current knowledge, have the best outcome
- **Exploration**: taking some other (possibly random) action to learn more, hoping to find something even better than what is currently known

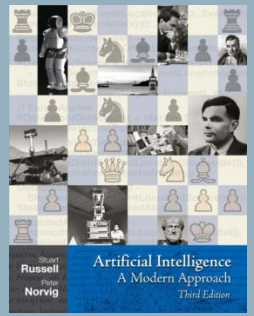
In practice, agents must explore to avoid getting stuck in severely sub-optimal behaviour, but exploration has a cost.

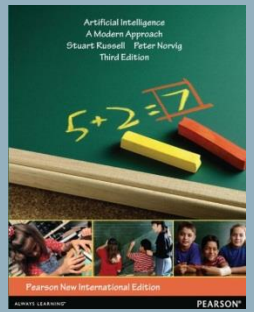
Typically, a smart agent explores more in early stages than later on

Goal-based Agents: Solving Problems by Searching

Chapter 3, Sections 1-4







Search for Atari Games

Lipovetzky, Nir, Miquel Ramirez, and Hector Geffner. **"Classical planning with simulators: Results on the atari video games."** In 24th International Joint Conference on Artificial Intelligence. 2015. https://www.youtube.com/watch?v=P-603qPMkSg

Problem solving agents

- Form of goal-based agent that formulates the problem of reaching a goal in its environment, searches for a sequence of actions solving the problem, and executes it.

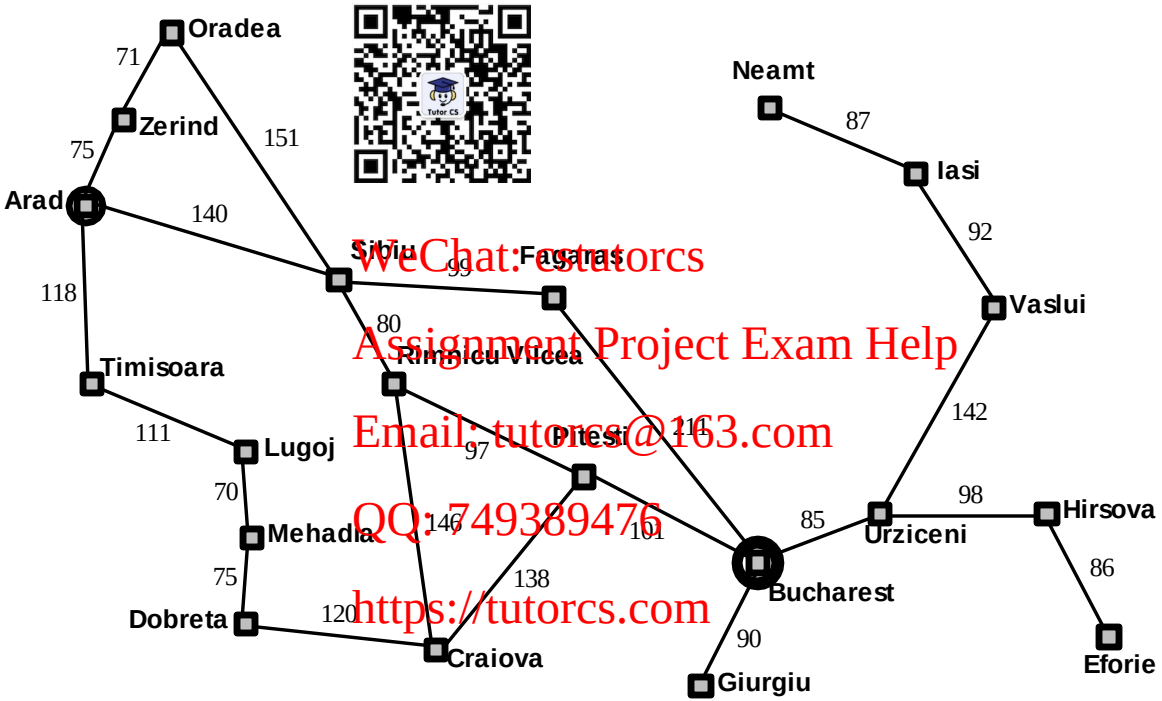
- Assumptions about the task environment:
  - fully observable
  - deterministic
  - static
  - single agent

- Offline (or open-loop) problem solving is suitable under those assumptions; the entire sequence solution can be executed “eyes closed.”

Outline

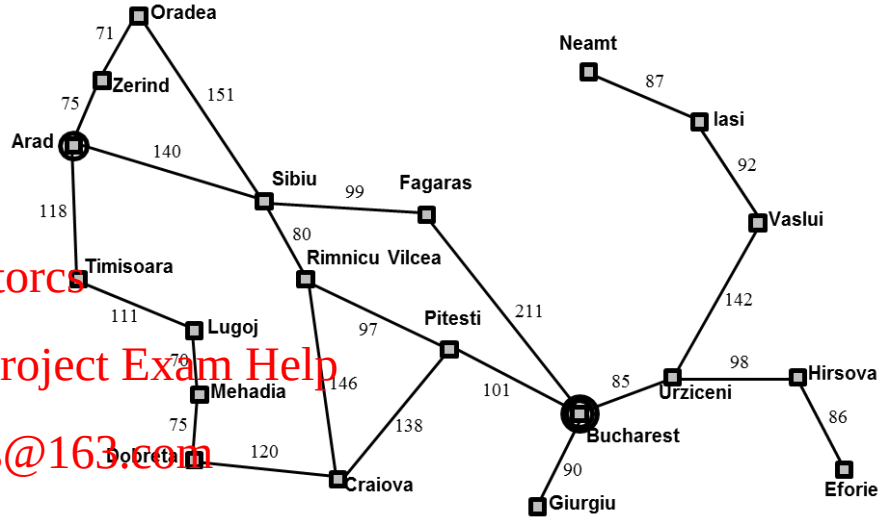
- Problem formulation
- Problem formulation examples
- Tree search algorithm
- Uninformed search strategies
- Informed search strategies

Example: Romania



Example: Romania

- On holiday in Romania
  - currently in Arad
  - flight leaves tomorrow from Bucharest



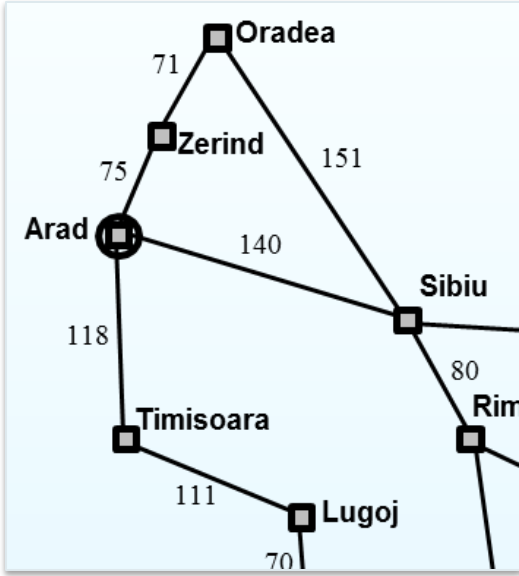
- **Formulate problem**:
  - **initial state**: in Arad
  - **goal:** be in Bucharest
  - **states:** various cities
  - **actions:** drive between cities

- **Search for a solution**:
  - sequence of drive actions or equivalently (in this case) sequence of cities, e.g. Arad, Sibiu, Fagaras, Bucharest

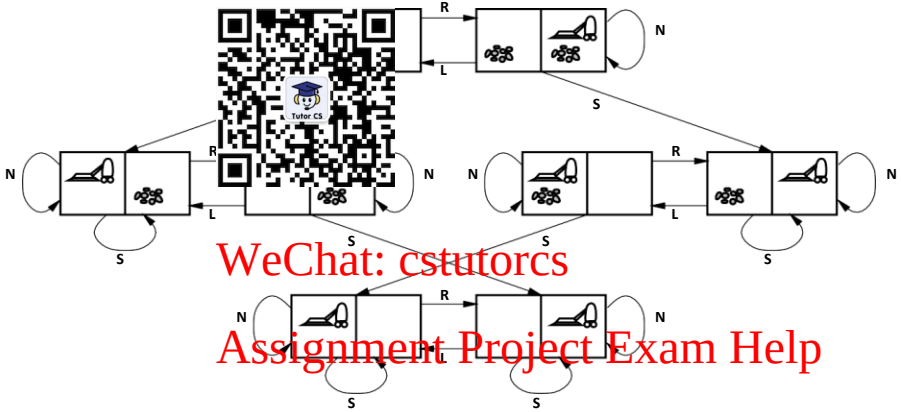
Problem formulation

A problem is defined by four items:

- initial state: e.g., "at Arad"
- successor function: $S(x) =$ set of action–state pairs
  e.g., $S(Arad) = \{\langle Arad \rightarrow Zerind, Zerind\rangle, \langle Arad \rightarrow Sibiu, Sibiu\rangle,$
  $\langle Arad \rightarrow Timisoara, Timisoara\rangle\}$
- goal test, can be
  - explicit, e.g., if $x =$ "at Bucharest"
  - implicit, e.g., if $HasAirport(x)$ is True
- path cost (additive)
  - e.g., sum of distances, number of actions executed, etc.
  - $c(x, a, y) \geq 0$ is the step cost of going from state x to state y via action a.
- A solution is a sequence of actions leading from the initial state to a goal state

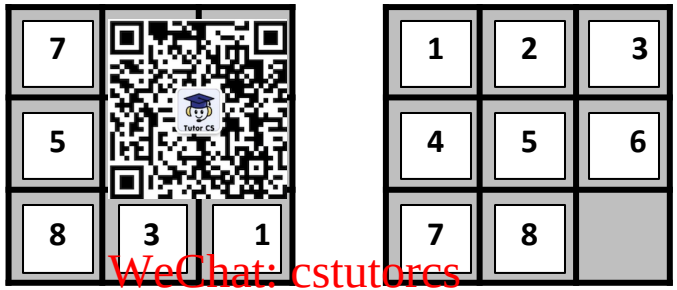


Example: Vacuum world



- **states?** dirt presence in each room and robot location (ignore dirt amounts)
- **actions?** *Left*, *Right*, *Suck*, *NoOp*
- **goal test?** no dirt
- **path cost?** 1 per action (0 for *NoOp*)

Example: The 8-puzzle



Start State Goal State

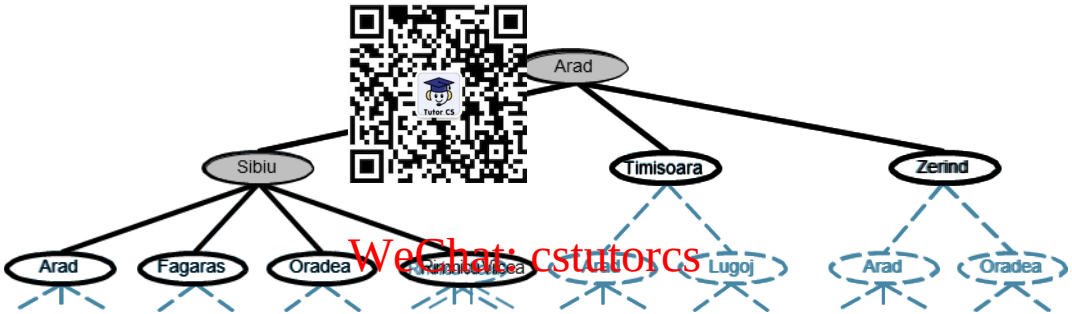
- **states?** Integer locations of tiles (ignore intermediate positions)
- **actions?** move blank left, right, up, down (ignore unjamming etc.)
- **goal test?** goal state (given)
- **path cost?** 1 per move

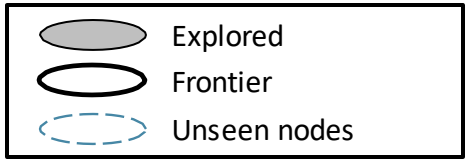
[Note: optimal solution of n-Puzzle family is NP-hard]

Selecting a state space

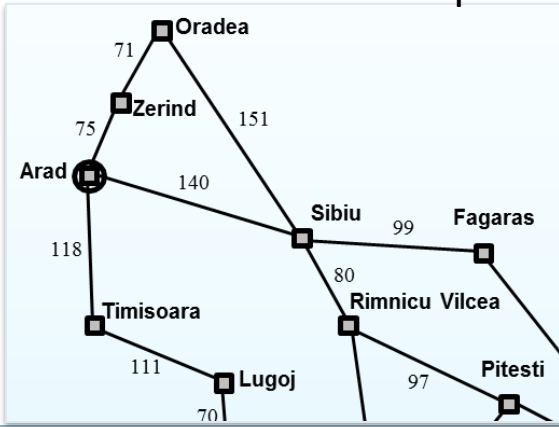
- Real world is absurdly complex
  ⇒ state space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., "*Arad* → *Zerind*" represents a complex set of possible routes, detours, rest stops, etc.
  - For guaranteed realizability, any real state "in Arad" must get to some real state in "in Zerind".
- (Abstract) solution = set of real paths that are solutions in the real world
- **Abstraction should be "easier" than the original problem!**

Tree search example





Romania Map:



Tree search algorithm

Basic idea:

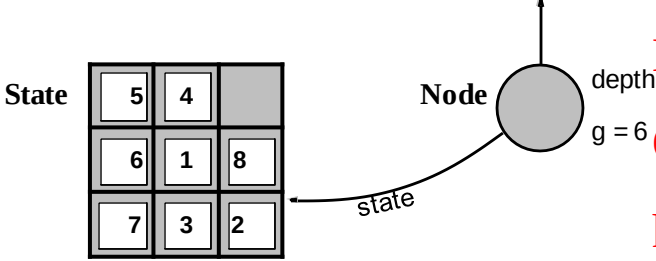
- offline, simulated exploration of state space by generating successors of already-explored nodes (a.k.a. expanding nodes)


```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion on the frontier then return failure
        choose a frontier node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the frontier of the tree
    end
```

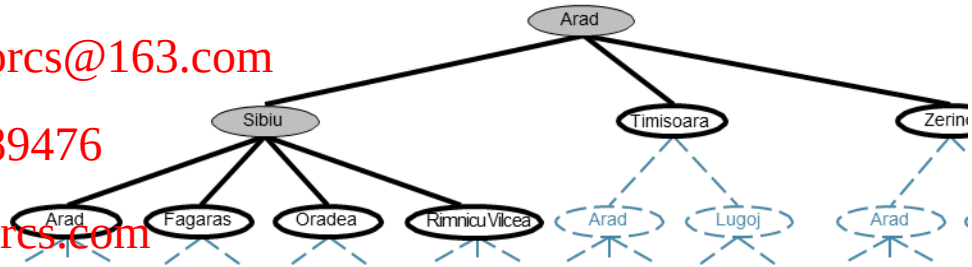
Implementation: states vs. nodes

- A **state** is a (representation of) a physical configuration
- A **node** is a data structure constituting part of a search tree
  - includes: state, parent, children, depth, **path cost** $g(n)$
- States do not have parents, children, depth, or path cost!

8-puzzle:



Romania:



Implementation: Expand and Frontier

- The **EXPAND** function creates new nodes, filling in the various fields and using the **SUCCESSORFN** of the problem to create the corresponding states.
  - Example: **SUCCESSORFN**(Arad) → $\{\langle Arad \rightarrow Zerind, Zerind\rangle, \langle Arad \rightarrow Sibiu, Sibiu\rangle, \langle Arad \rightarrow Timisoara, Timisoara\rangle\}$

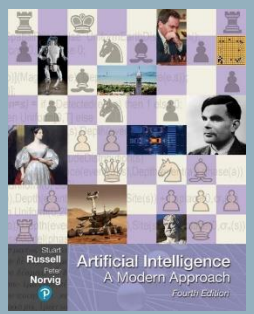
- Frontier implemented as **priority queue** of nodes ordered according to strategy

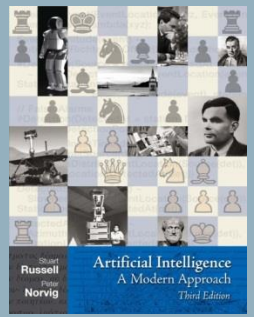
Implementation: general tree search

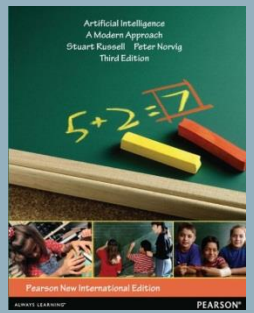

```
function EXPAND(node, problem) returns a set of nodes
    successors ← the empty set
    for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
        s ← a new NODE
        PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
        PATH-COST[s] ← PATH-COST[node] + STEP-COST(STATE[node], action, result)
        DEPTH[s] ← DEPTH[node] + 1
        add s to successors
    return successors
```

Uniform Search Strategies

Chapter 3, Section 4





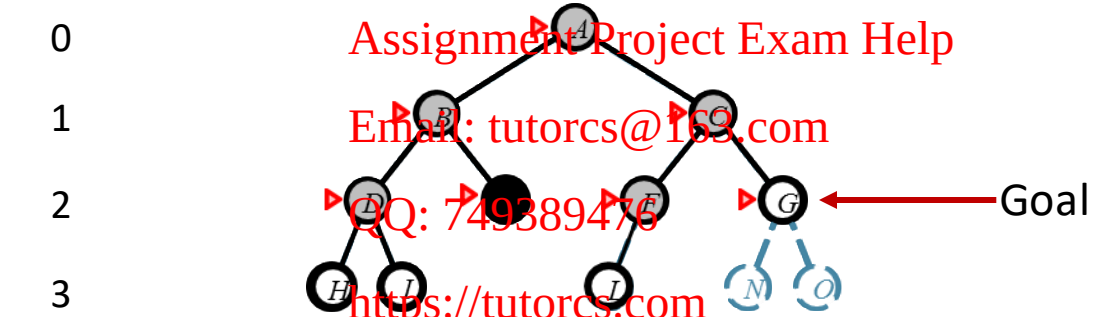


Uninformed search strategies

- A strategy is defined by picking the order of node expansion.
- This is the order used for the priority queue implementing the frontier.
- Uninformed strategies use only the information available in the definition of the problem:
  - Breadth-first search
  - Uniform-cost search
  - Depth-first search
  - Depth-limited search
  - Iterative deepening search

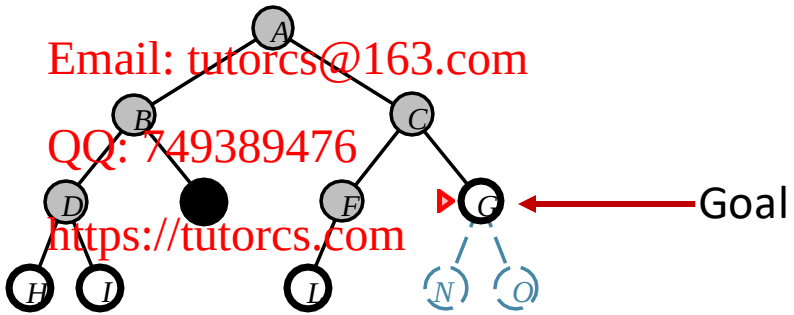
Breadth-first search

- **Strategy**: expand shallowest unexpanded node
- **Implementation**:
  - ***frontier*** is a FIFO queue, i.e., new successors go at end



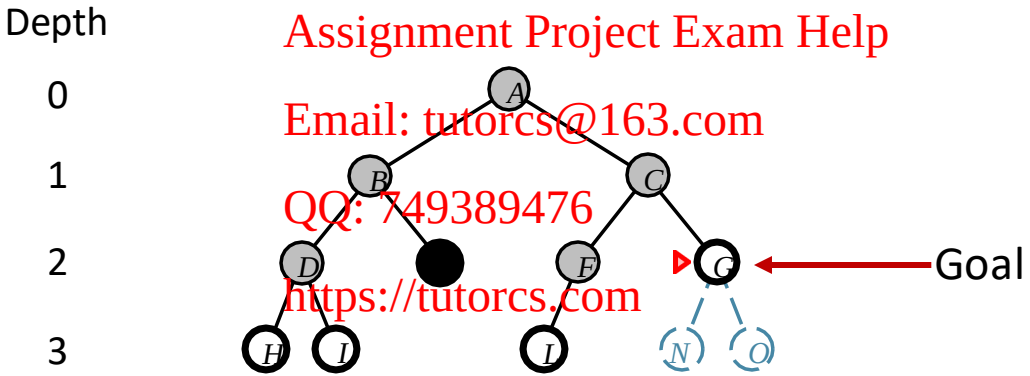
Evaluating Search Strategies

- Strategies are evaluated along the following dimensions:
  - completeness - does it always find a solution if one exists?
  - solution optimality - does it always find a least-cost solution?
  - time complexity - number of nodes generated (or expanded)
  - space complexity - maximum number of nodes in memory



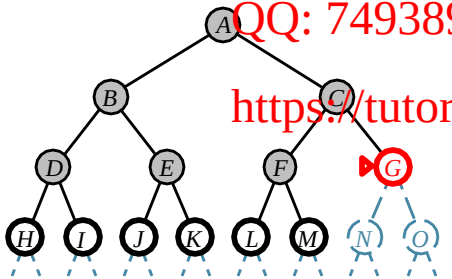
Time and Space Complexity

- Time and space complexity are measured using big-O notation in terms of
  - b: maximum **branching** fa[illegible]e search tree
  - d: **depth** of the shallowest [illegible]
  - m: **maximum** depth of the state space (may be ∞)



Properties of breadth-first search

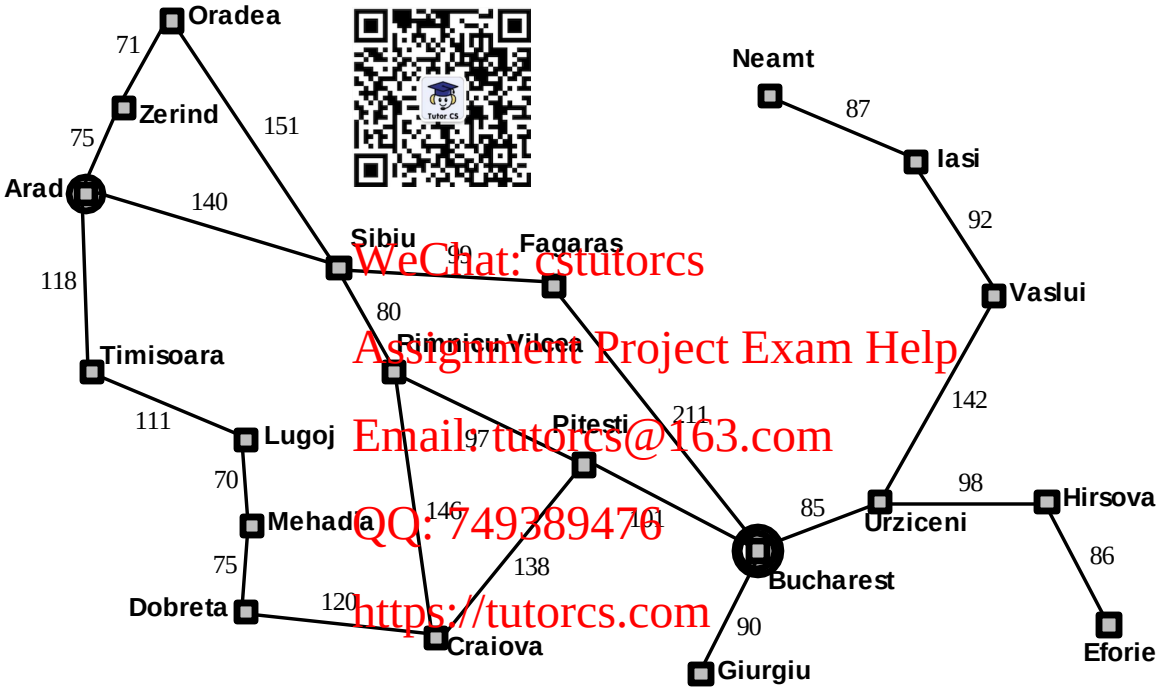
- Complete? Yes (if b and d are finite)
- Time? $1 + b + b^2 + b^3 + \ldots + (b^{d+1} - b) = O(b^{d+1})$, i.e., exp. in d
- Space? $O(b^{d+1})$
  - if b and d are finite $b = 10$, 1 million node/sec, 1Kb/node, $d = 12$ would take 13 days and 1 petabyte of memory.
- Optimal? Yes if cost = 1 per step; not optimal in general



depth	#nodes
0	b^0
1	b^1
2=d	b^2 = b^d
3=d+1	$b^3 - b = b^{d+1} - b$

b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

Example: Romania



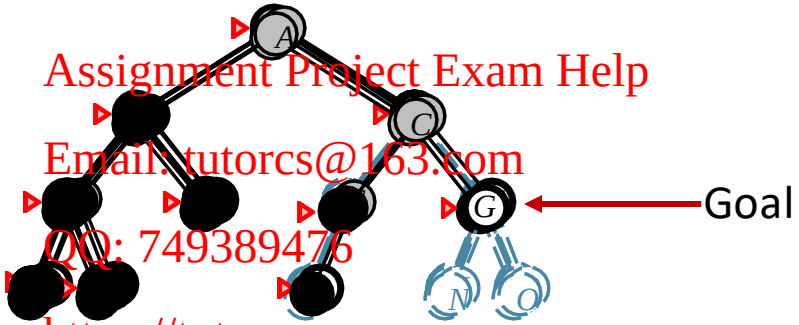
Uniform-cost search

- **Strategy:** Expand least-cost unexpanded node
- **Implementation:**
  - *frontier* = queue ordered by path cost (i.e., $g(n)$), lowest first
- Equivalent to breadth-first if step costs all equal
- Complete? Yes, if step cost $\geq \epsilon$ (for $\epsilon > 0$)
- Time? # of nodes with $g \leq C^*$, $O(b^{1+\lceil C^*/\epsilon \rceil})$
  where C^* is the cost of the optimal solution
- Space? # of nodes with $g \leq C^*$, $O(b^{1+\lceil C^*/\epsilon \rceil})$
- Optimal? Yes—nodes expanded in increasing order of $g(n)$

b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

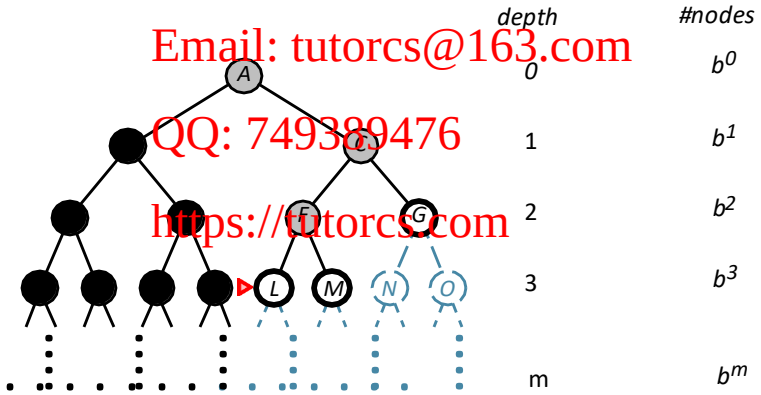
Depth-first search

- **Strategy:** Expand deepest unexpanded node
- **Implementation**:
  - *frontier* = LIFO queue, i.e., put successors at front



Properties of depth-first search

- Complete? No: fails in infinite-depth spaces, spaces with loops
  ⇒ complete in finite spaces if we modify to avoid repeated states along path
- Time? $O(b^m)$: terrible if m is much larger than d
  ⇒ if solutions are dense, may be much faster than breadth-first
- Space? $O(bm)$, i.e., linear space! (deepest node + ancestors + their siblings)
- Optimal? No



b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

Breadth-first versus depth-first search

- Use breadth-first search when there exists short solutions.
- Use depth-first search when there exists many solutions.

Eternity II Puzzle

2 million dollar prize! Few deep solutions (presumably…)

Iterative deepening search

- Combines advantages of breadth-first and depth-first search:
  - is complete
  - returns shallowest so
  - use linear amount of memory

- Performs a series of depth limited depth-first searches


```
function ITERATIVE-DEEPENING-SEARCH( problem) returns solution/failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

Depth-limited search

- Depth-first search with depth limit l, i.e., nodes at depth l have no successors
- **Recursive implementation**:

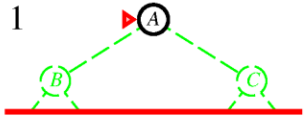
```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST(problem, STATE[node]) then return node
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

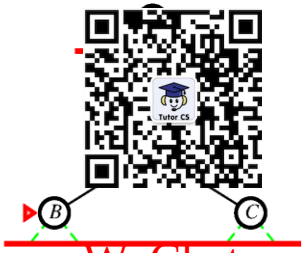

- **solution**: solution found within depth limit
- **cutoff**: no solution within the depth limit
- **failure**: the problem has no solution

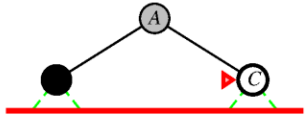
Iterative deepening search – Example

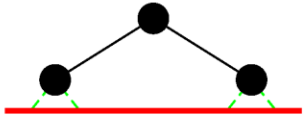
Limit = 0

Limit = 1

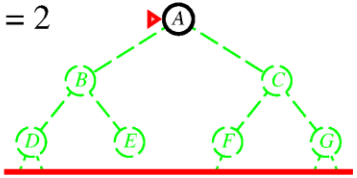


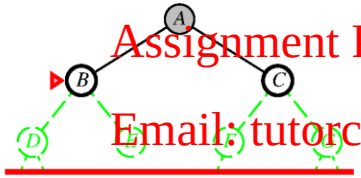


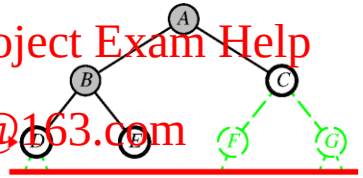


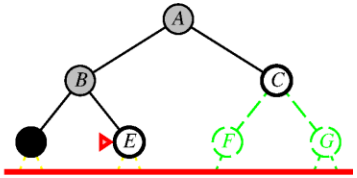


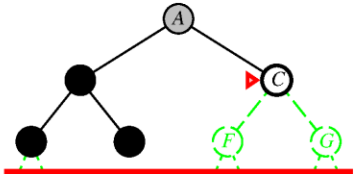
Limit = 2

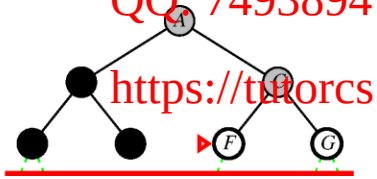


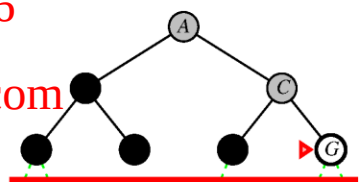












Properties of iterative deepening search

b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

- Complete? Yes (if b and d are finite)
- Time? $(d+1)b^0 + db^1 + (d-1)b^2 + \cdots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1 (can be modified to explore uniform-cost tree)

Numerical comparison for $b = 10$ and $d = 5$, solution at far-right leaf:

Time: IDS doesn't do much worse than BFS

$$N(BFS) = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$$
$$N(IDS) = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$$

Assumes BFS applies the goal test when a node is generated

Space: IDS does much better $N(IDS) = 50$, $N(BFS) \simeq 1{,}000{,}000$

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes[1]	Yes	No	Yes, if $l \geq d$	Yes[1]
Time	b^{d+1}	$b^{1+\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{1+\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes[2]	Yes	No	No	Yes[2]

1. Only if b and d are finite.
2. Only if all step costs are identical.

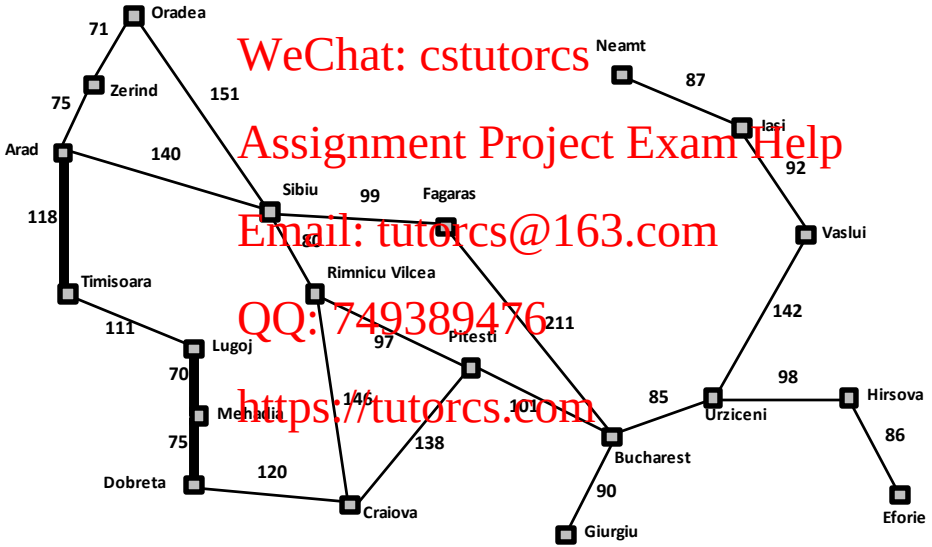
b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

Informed search algorithms

Chapter 3, Sections 5–6

Heuristic Function

- Estimates the cost from a given state to the goal
  - Estimate for the Travel in Romania example?
    - **Straight Line Distance**



Straight-line dist. to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Outline

- Evaluation functions (use heuristics)
- Greedy search
- A* search

Review: Tree search

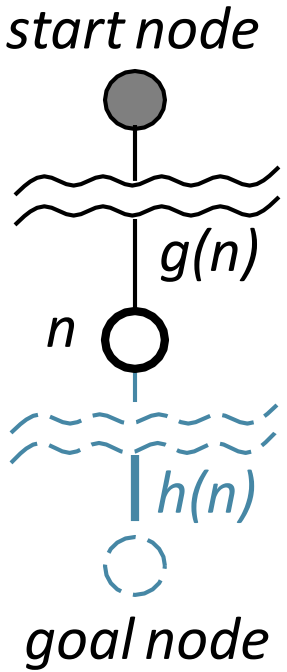

```
function TREE-SEARCH( ... frontier) returns a solution, or failure
    frontier ← INSERT(MA ... NITIAL-STATE[problem]), frontier)
    loop do
        if frontier is empty ... rn failure
        node ← REMOVE-FRONT(frontier)
        if GOAL-TEST(problem, STATE(node)) then return node
        frontier ← INSERTALL(EXPAND(node, problem), frontier)
```


- The goal test is performed when the node is popped from the frontier, NOT when it is generated during expansion.
  - This is important when looking for optimal solutions.
- A strategy is defined by picking the order of node expansion

Evaluation function

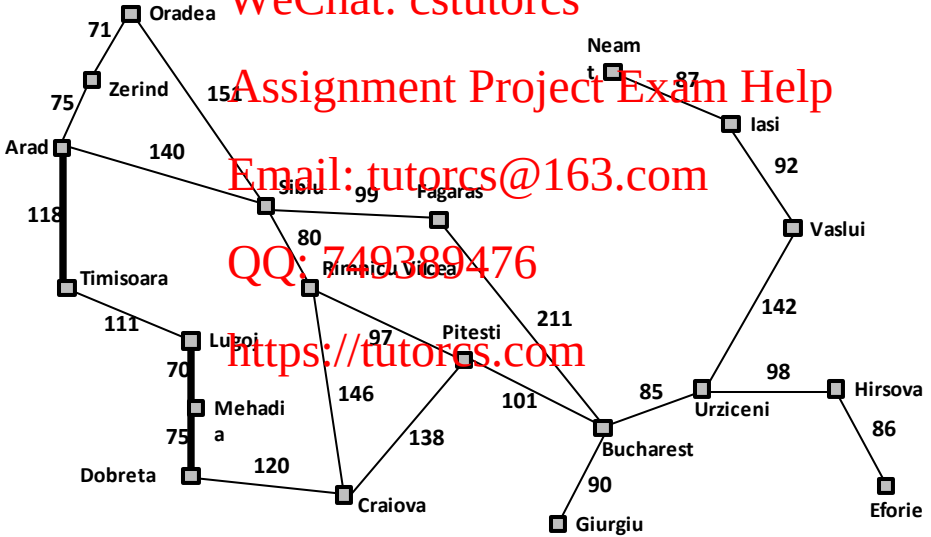
- Evaluation function $f(n) = g(n) + h(n)$
  - estimate of "desirability", u[illegible]oblem-specific.
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to the closest goal (heuristic)
  - $f(n)$ = estimated total cost of path through n to goal
    - The lower $f(n)$, the more desirable n is

- Implementation:
  - *frontier* is a queue sorted in ascending value of $f(n)$
- Special cases:
  - uniform cost search (uninformed): $f(n) = g(n)$
  - greedy search (informed): $f(n) = h(n)$
  - A* search (informed): $f(n) = g(n) + h(n)$



Greedy search

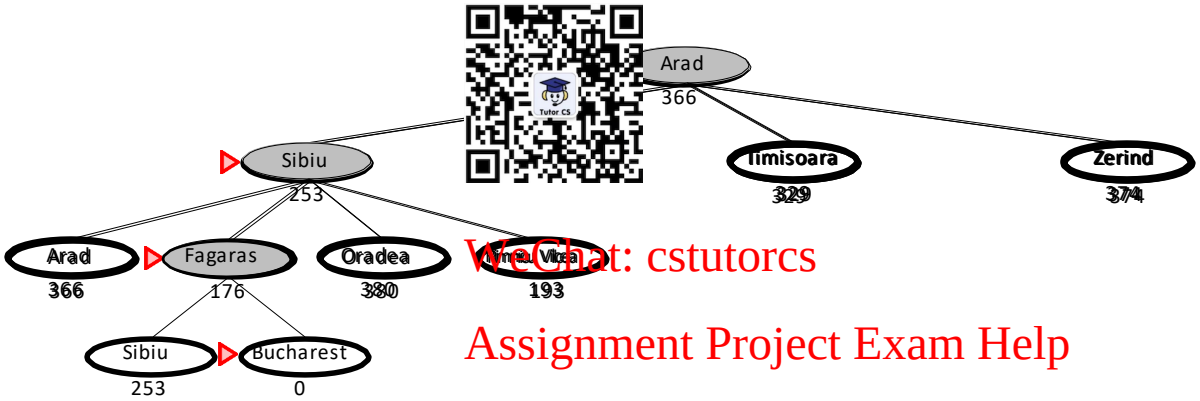
- Evaluation function $f(n) = h(n)$ (entirely heuristic)
  = estimate of cost from n to the closest goal
- E.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
- Greedy search expands the node that **appears** to be closest to goal



Straight-line dist to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy Search Example



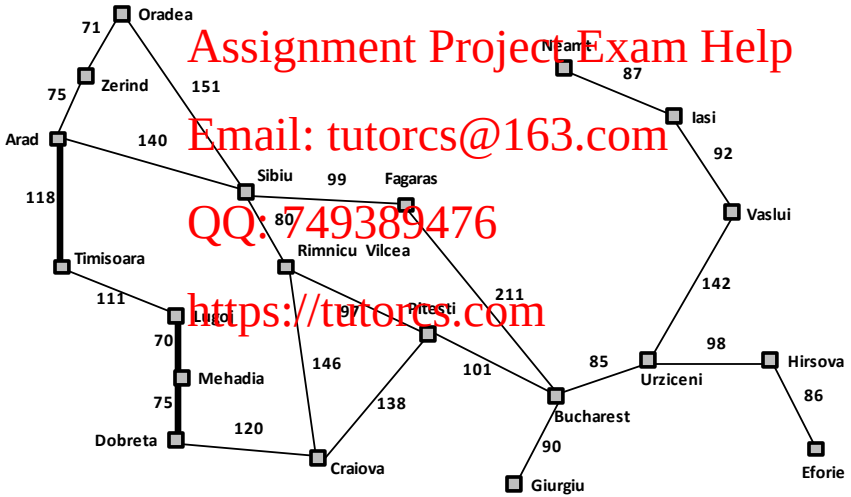
Straight-line dist. to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy search

- Complete? No. It can get stuck in loops, e.g., with going from Iasi to Fagaras:

  Iasi → Neamt → Iasi → Neamt →…

  - **Complete** in finite space with repeated-state checking
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$
- Optimal? No

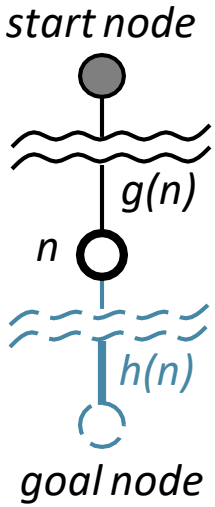


b: max. branching factor
d: depth of the shallowest solution
m: max. depth of the state space

A* Search

- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to the closest goal
  - $f(n)$ = estimated total cost of path through n to goal

- Admissible heuristic:
  - $\forall n\ h(n) \le h^*(n)$ where $h^*(n)$ is the true cost from n
  - Also require $h(n) \ge 0$, so $h(G) = 0$ for any goal G
- E.g., $h_{SLD}(n)$ never overestimates the actual road distance
- When $h(n)$ is admissible, $f(n)$ never overestimates the total cost of the optimal path through n to the goal
- Theorem: if h is admissible, A* search finds the optimal solution



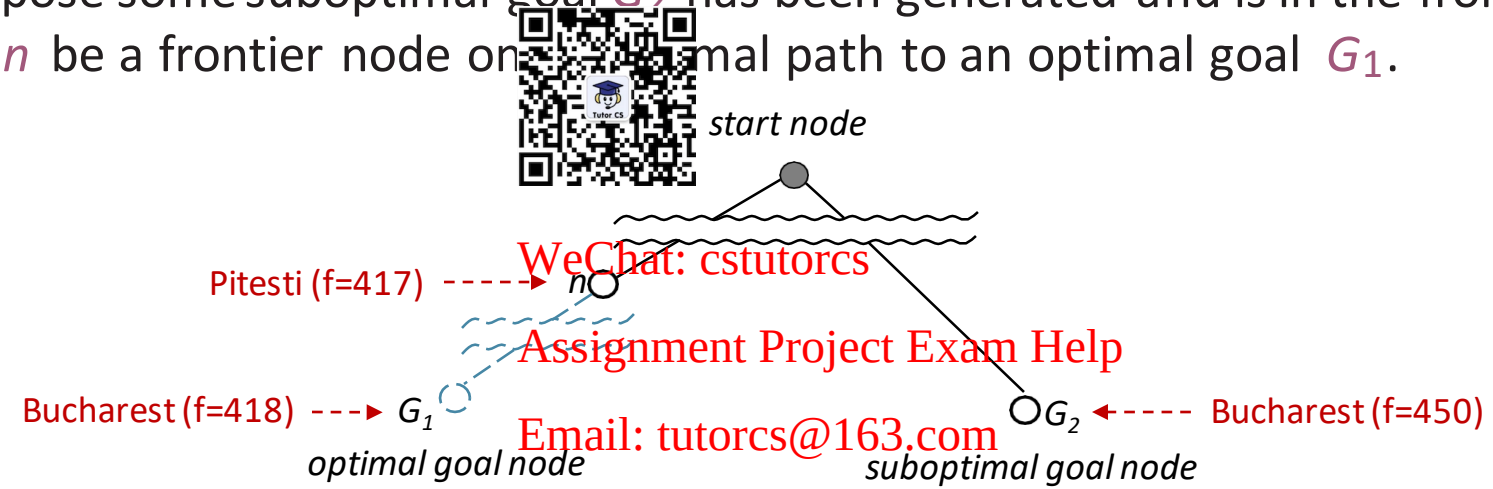
A* Search Example

$f(n) = g(n) + h(n)$

Straight-line dist. to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Optimality of A* (based on admissibility)

- Suppose some suboptimal goal G_2 has been generated and is in the frontier. Let n be a frontier node on a shortest path to an optimal goal G_1.



$f(G_2) = g(G_2)$ since G_2 is a goal node, hence, $h(G_2) = 0$

$> g(G_1)$ since G_2 is suboptimal

$\geq f(n)$ since h is admissible: $f(n) = g(n) + h(n) \leq g(n) + h^*(n) = g(G_1)$

- Since $f(G_2) > f(n)$, A* will never select G_2 for expansion