ENCS3340 - Artificial Intelligence

Search
(Problem Formulation)

Search as Problem-Solving Strategy

- Many problems can be viewed as reaching a goal from a given starting point
  - often there is an underlying state space that defines the problem and its possible solutions in a more formal way
  - the space can be traversed by applying a successor function (operators, actions, state transitions) to proceed from one state to the next
  - if possible, information about the specific problem or the general domain is used to improve the search
    - experience from previous instances of the problem
    - strategies expressed as heuristics
    - simpler versions of the problem
    - constraints on certain aspects of the problem

Examples

- Loading a moving truck
  - **start**: apartment full of boxes and furniture
  - **goal**: empty apartment, all boxes and furniture in the truck
  - **actions**: select item, carry item from apartment to truck, load item
- Getting settled after moving
  - **start**: items randomly distributed over the place
  - **goal**: satisfactory arrangement of items
  - **actions**: select item, move item
- Repairing a flat tire on your bike
  - **start**: bike with a flat tire
  - **goal**: bike with two properly inflated tires
  - **actions**: remove wheel, remove tire, remove tube, fix tube, return tube, return tire, partially inflate tube, return wheel, fully inflate tube

Problem-Solving Agents

- Agents whose task it is to solve a particular problem
  - problem formulation
    - what are the possible states of the world relevant for solving the problem
    - what information is accessible to the agent
    - how can the agent progress from state to state
  - goal formulation
    - what is the goal state
    - what are important characteristics of the goal state
    - how does the agent know that it has reached the goal
    - are there several possible goal states
      - are they equal or are some more preferable
      - if necessary, a utility function is required to determine priorities among goals

Problem Types

- single-state problems
  - accessible world and knowledge of its actions allow the agent to know which state it will be in after a sequence of actions
  - Ex: playing chess
- multiple-state problems
  - the world is only partially accessible, and the agent has to consider several possible states as the outcome of a sequence of actions
  - Ex: walking in a dark room
- contingency problems
  - at some points in the sequence of actions, sensing may be required to decide which action to take; this leads to a tree of sequences
  - Ex: a new skater in a ring
- exploration problems
  - the agent doesn't know the outcome of its actions, and must experiment to discover states of the world and outcomes of actions
  - Ex: Mars Exploration Rovers

Well-Defined Problems

- initial state
  - starting point from which the agent sets out
- actions (operators, successor functions)
  - describe the set of possible actions, and transitions from one state to another
- state space
  - set of all states reachable from the initial state by any sequence of actions
- goal state
  - terminal state that the agent wants to achieve
- goal test
  - determines if a given state is the goal state
- Path
  - sequence of actions leading from one state in the state space to another
- path cost
  - determines the expenses of the agent for executing the actions in a path
- Solution
  - path from the initial state to a goal state

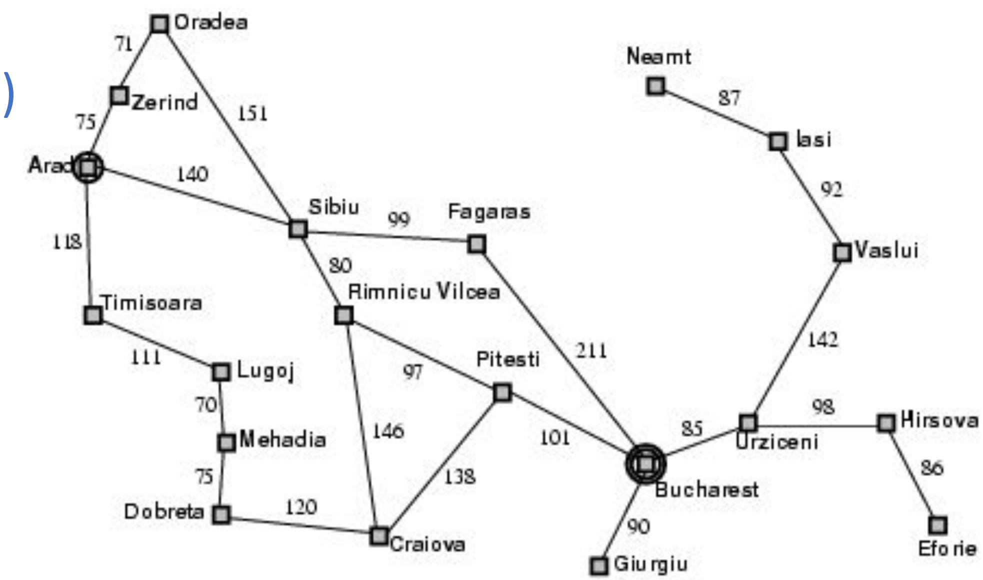
Selecting States and Actions

- states describe distinguishable points or periods during the problem-solving process
  - dependent on the task and domain
- actions move the agent from one state to another one
  - an action is applied to the current state and takes the agent to the successor state
  - dependent on states, capabilities of the agent, and properties of the environment
- choice of suitable states and actions
  - can make the difference between a problem that can or cannot be solved
  - level of abstraction
    - high: smaller state space, complex actions
    - low: simple actions, larger state space

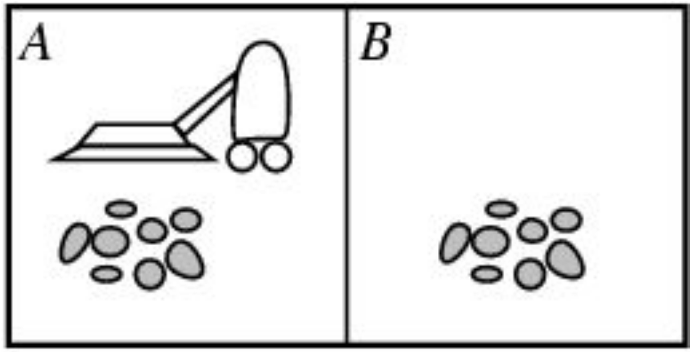
Example Problem: Romania Map

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

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

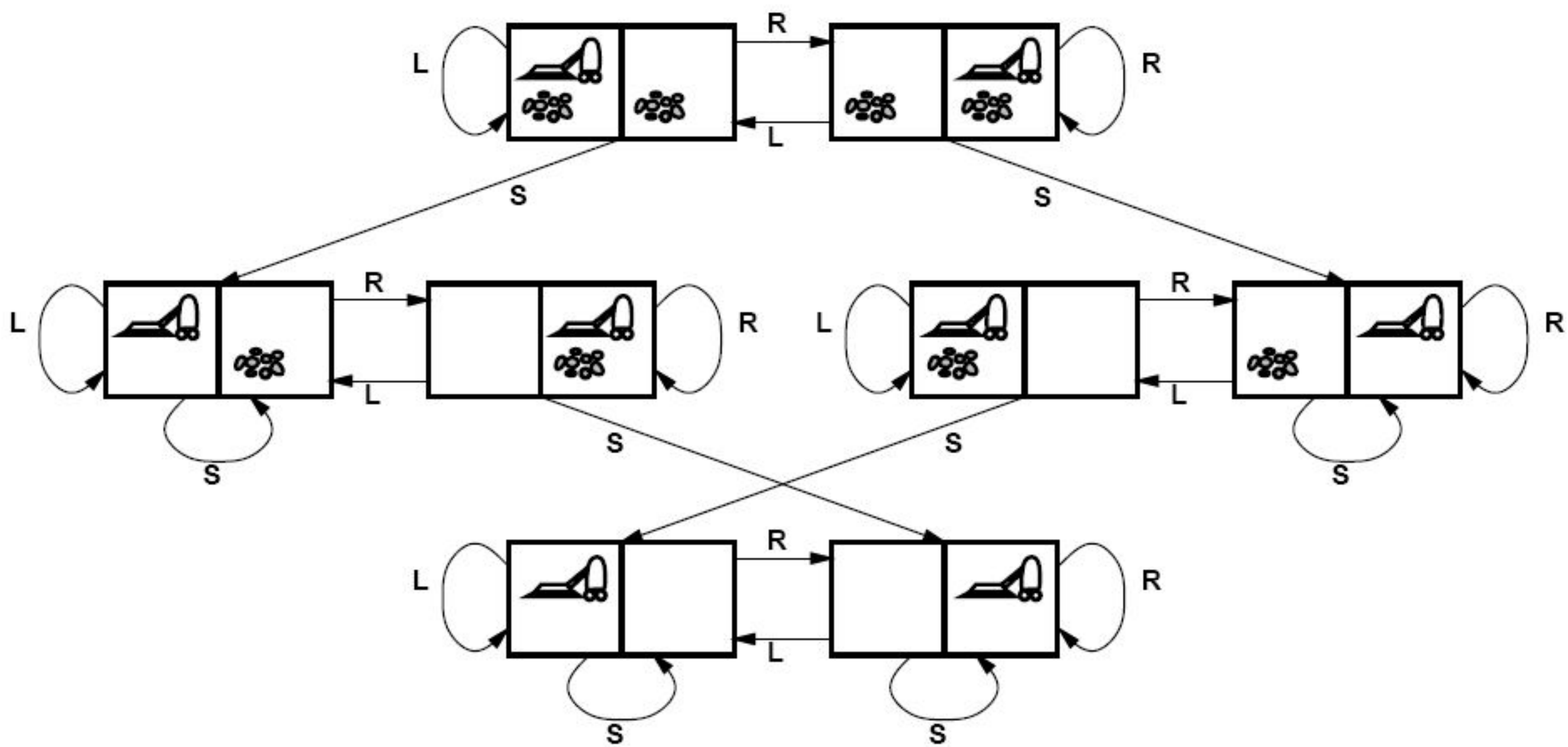


Example Problem: Vacuum world



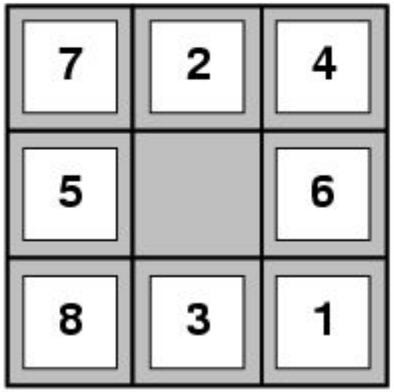
- States
  - Agent location and dirt location
  - How many possible states?
  - What if there are n possible locations?
- Actions
  - Left, right, suck
- goal test
  - all squares clean
- path cost
  - one unit per action

Vacuum world state space graph

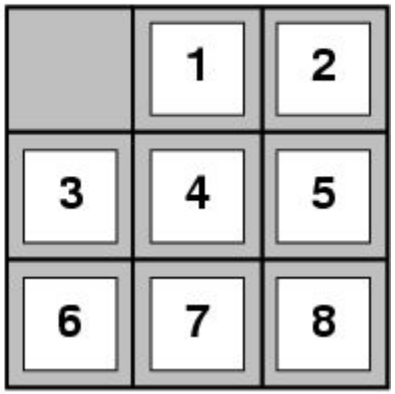


Example Problem: The 8-puzzle

- States
  - location of tiles (including blank tile)
  - 9!/2 = 181,440 reachable states
- Actions
  - Move blank left, right, up, down
- Path cost
  - 1 per move
- <u>Finding the optimal solution of n-Puzzle is NP-hard</u>

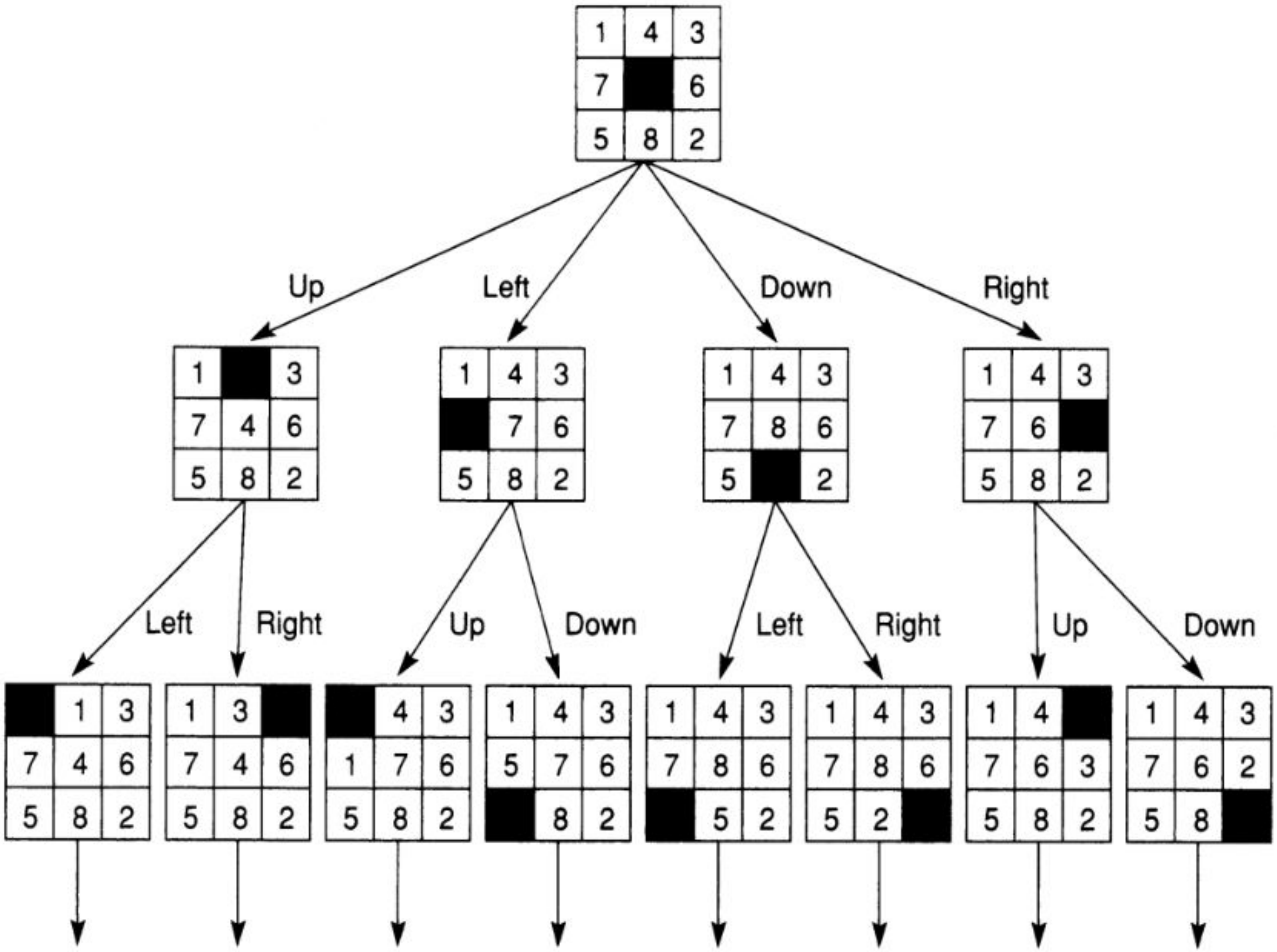


Start State

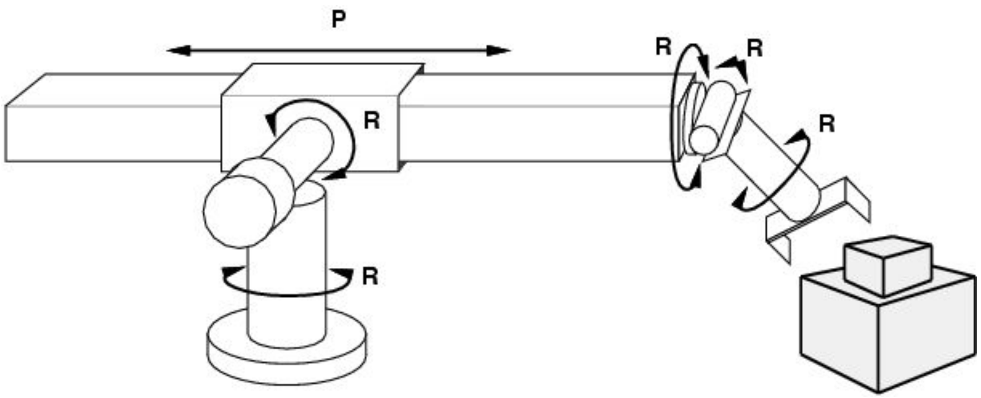


Goal State

State Space for the 8-puzzle



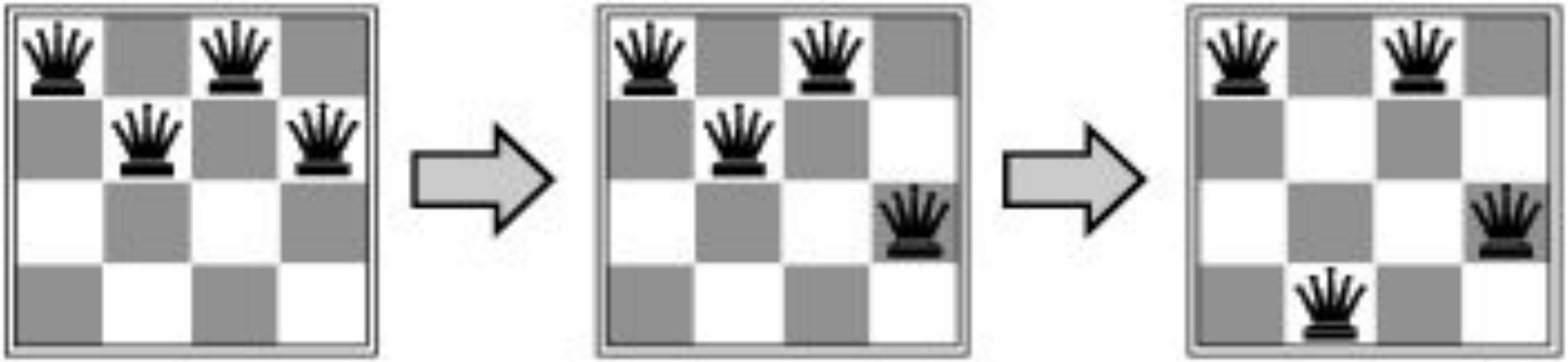
Example Problem: Robot motion planning



- States
  - Real-valued coordinates of robot joint angles
- Actions
  - Continuous motions of robot joints
- Goal state
  - Desired final configuration (e.g., object is grasped)
- Path cost
  - Time to execute, smoothness of path, etc.

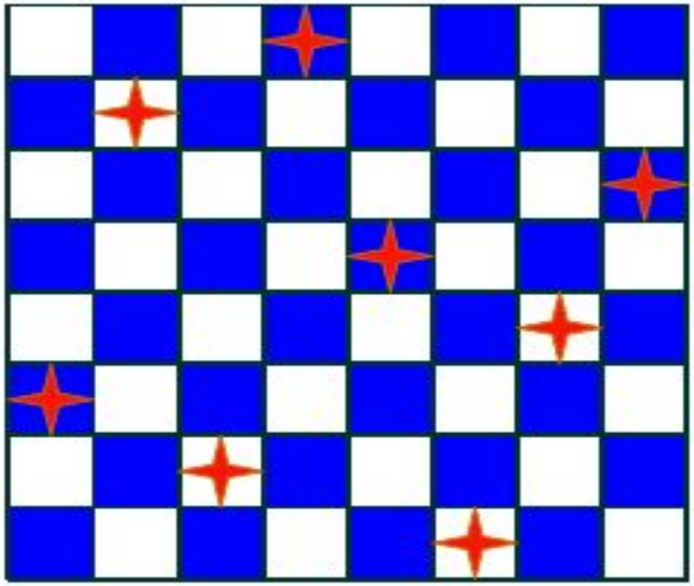
Example: n-queens

- Put n queens on a n × n board with no two queens on the same row, column, or diagonal



8-Queens: Incremental Approach

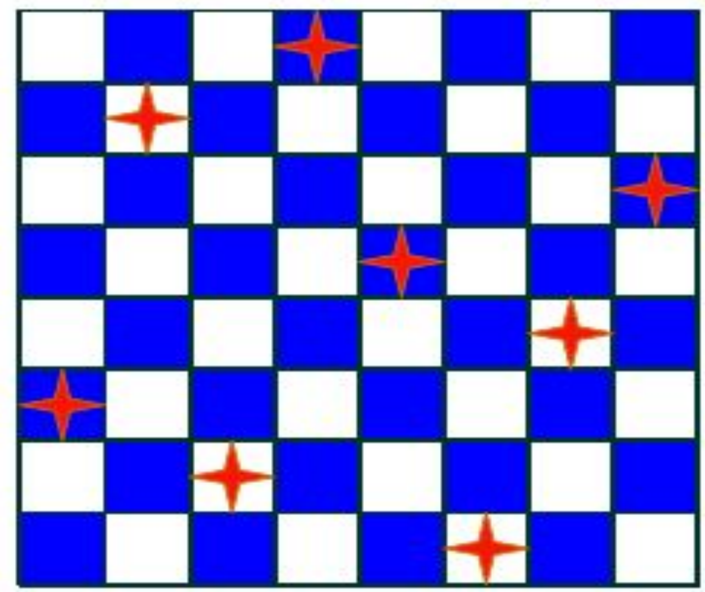
- start with an empty board
- add queens one by one (no violation of constraints)
- incremental formulation
  - states
    - arrangement of up to 8 queens on the board
  - initial state
    - empty board
  - successor function (actions)
    - add a queen to any square
  - goal test
    - all queens on board
    - no queen attacked
  - path cost
    - irrelevant (all solutions equally valid)



A solution

8-Queens: Complete-State Approach

- start with a full board (all n queens placed on the board, conflicts are to be expected)
- try to find a better configuration (reduced number of conflicts)
- Complete-state formulation
  - states
    - arrangement of the 8 queens on the board
  - initial state
    - all 8 queens on board
  - successor function (actions)
    - move a queen to a different square
  - goal test
    - no queen attacked
  - path cost
    - irrelevant (all solutions equally valid)



A solution

Example Problem: VLSI Layout

- States
  - positions of components, wires on a chip
- Initial state
  - incremental: no components placed
  - complete-state: all components placed (e.g. randomly, manually)
- Successor function (actions)
  - incremental: place components, route wire
  - complete-state: move component, move wire
- Goal test
  - all components placed
  - components connected as specified
- Path cost
  - may be complex: distance, capacity, number of connections per component

Searching for Solutions

- Given
  - Initial state
  - Actions
  - Transition model
  - Goal state
  - Path cost

- How do we find the optimal solution?
  - How about building the state space and then using Dijkstra’s shortest path algorithm?
    - The state space may be huge!
    - Complexity of Dijkstra’s is $O(E + V \log V)$, where V is the size of the state space

Searching for Solutions

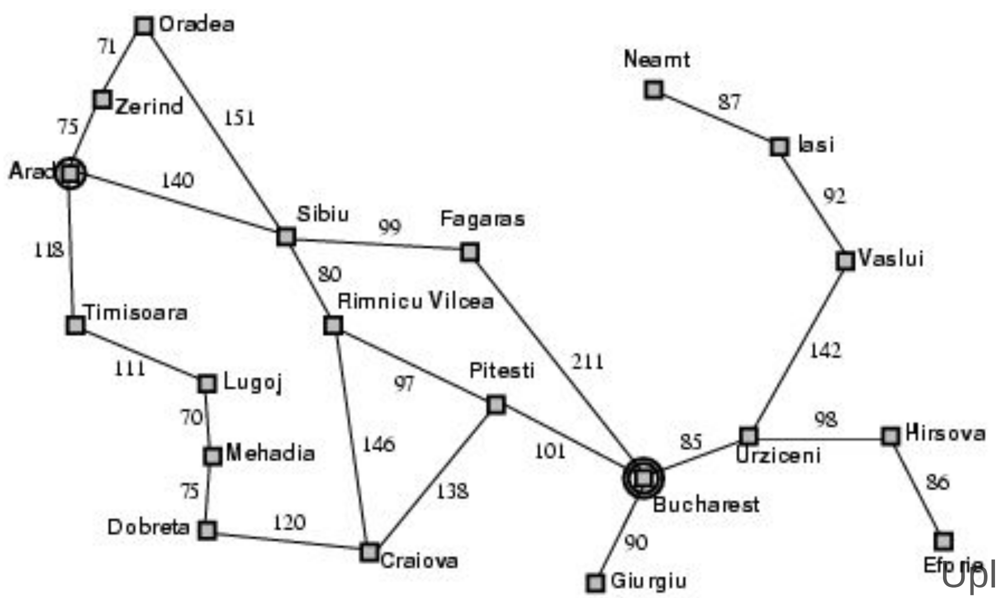
- traversal of the search space
  - from the initial state to a goal state
  - legal sequence of actions as defined by successor function (actions, operators, state transitions)
- general procedure
  - check for goal state
  - expand the current state
    - determine the set of reachable states
    - return “failure” if the set is empty
  - select one from the set of reachable states
  - move to the selected state
- a search tree is generated
  - nodes are added as more states are visited

Search Terminology

- search tree
  - generated as the search space is traversed
    - the search space itself is not necessarily a tree, frequently it is a graph
    - the tree specifies possible paths through the search space
  - expansion of nodes
    - as states are explored, the corresponding nodes are expanded by applying the successor function
      - this generates a new set of (child) nodes
    - the fringe (frontier) is the set of nodes not yet visited
      - newly generated nodes are added to the fringe
  - search strategy
    - determines the selection of the next node to be expanded
    - can be achieved by ordering the nodes in the fringe
      - e.g. queue (FIFO), stack (LIFO), “best” node w.r.t. some measure (cost)

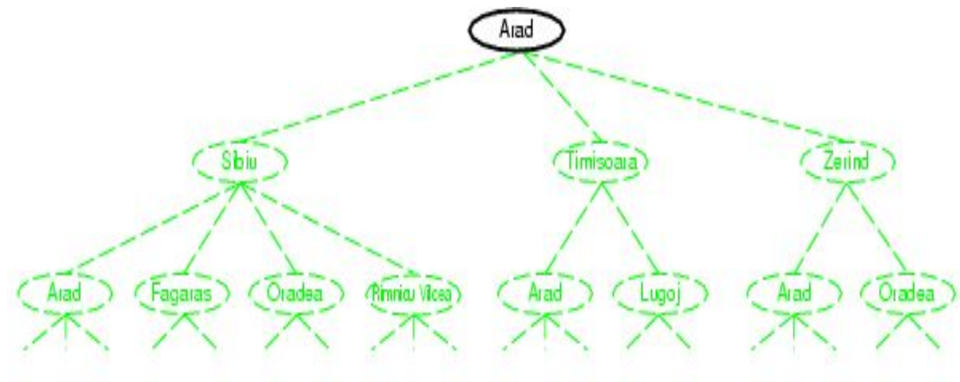
Example: Graph Search

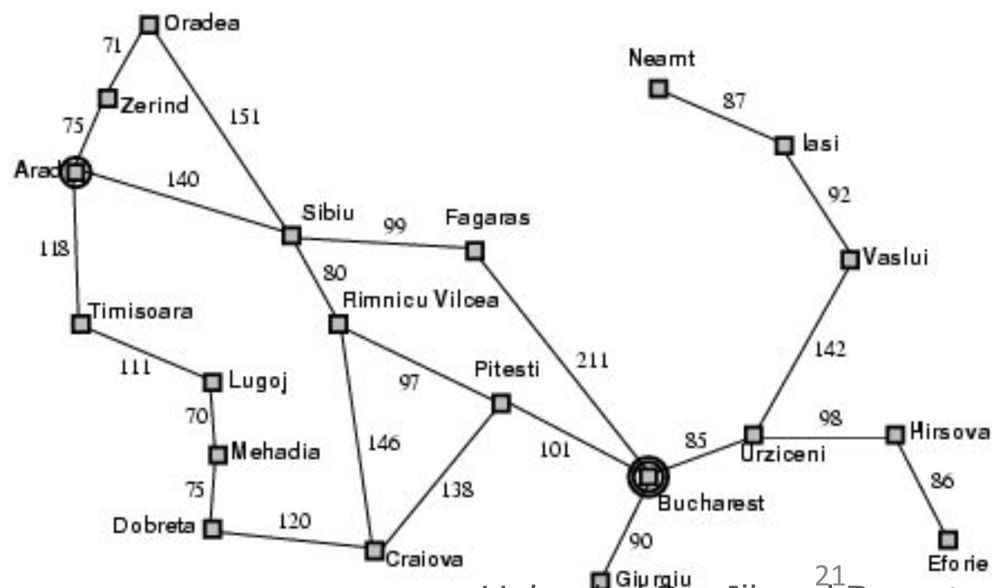
- describes the search (state) space
  - each node represents one state in the search space
  - e.g. a city to be visited in a routing or touring problem
- additional information
  - names and properties for the states
  - links between nodes, specified by the successor function
  - properties for links (distance, cost, name, ...)



Graph and Tree

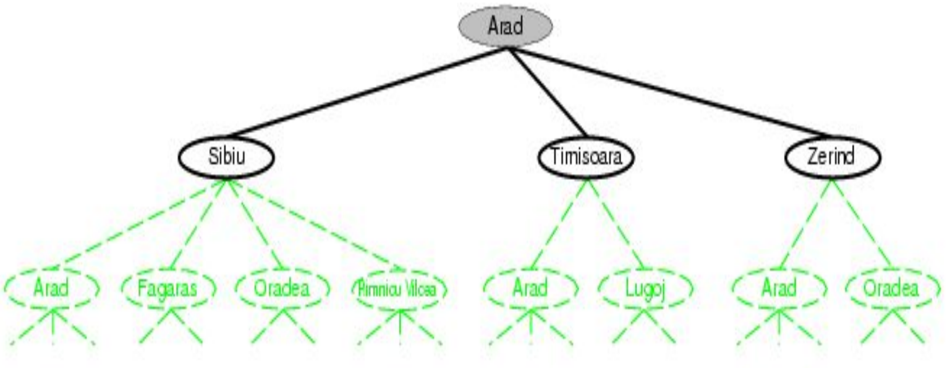
- the tree is generated by traversing the graph
- the same node in the graph may appear repeatedly in the tree
- the arrangement of the tree depends on the traversal strategy (search method)
- the initial state becomes the root node of the tree
- in the fully expanded tree, the goal states are the leaf nodes
- cycles in graphs may result in infinite branches

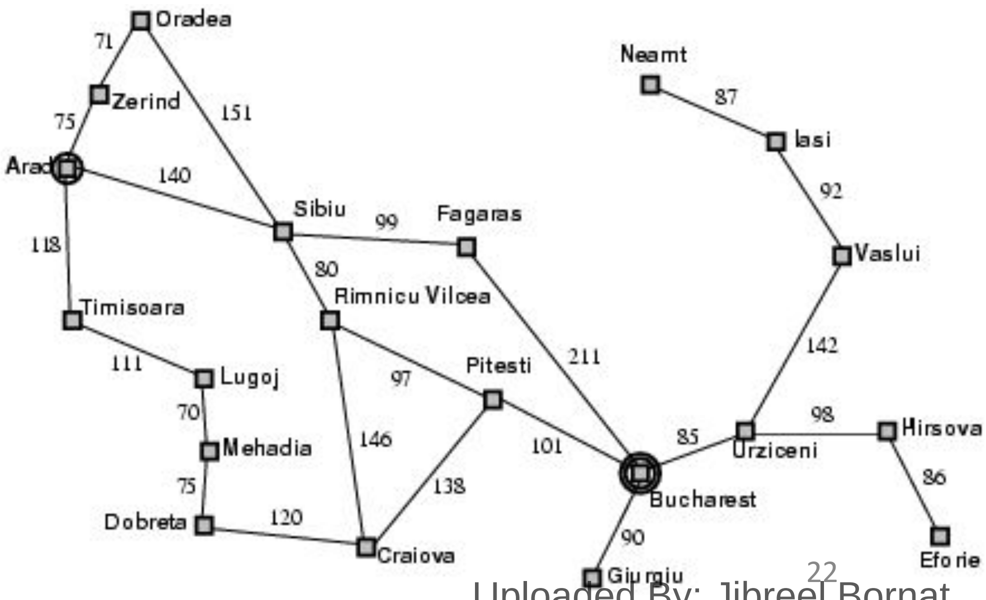




Graph and Tree

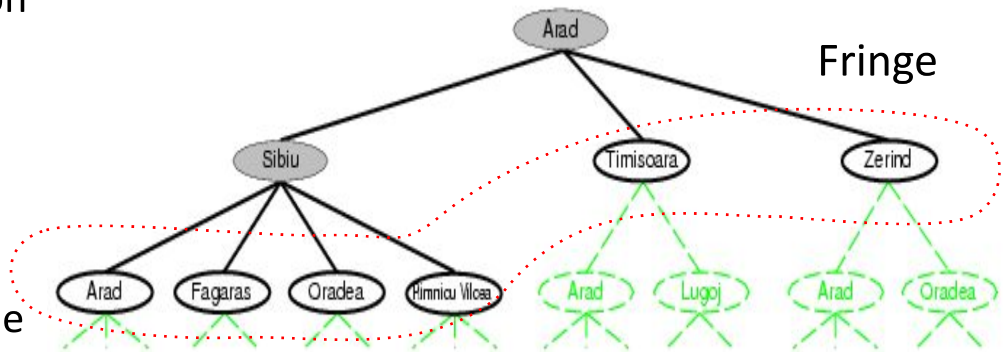
- the tree is generated by traversing the graph
- the same node in the graph may appear repeatedly in the tree
- the arrangement of the tree depends on the traversal strategy (search method)
- the initial state becomes the root node of the tree
- in the fully expanded tree, the goal states are the leaf nodes
- cycles in graphs may result in infinite branches

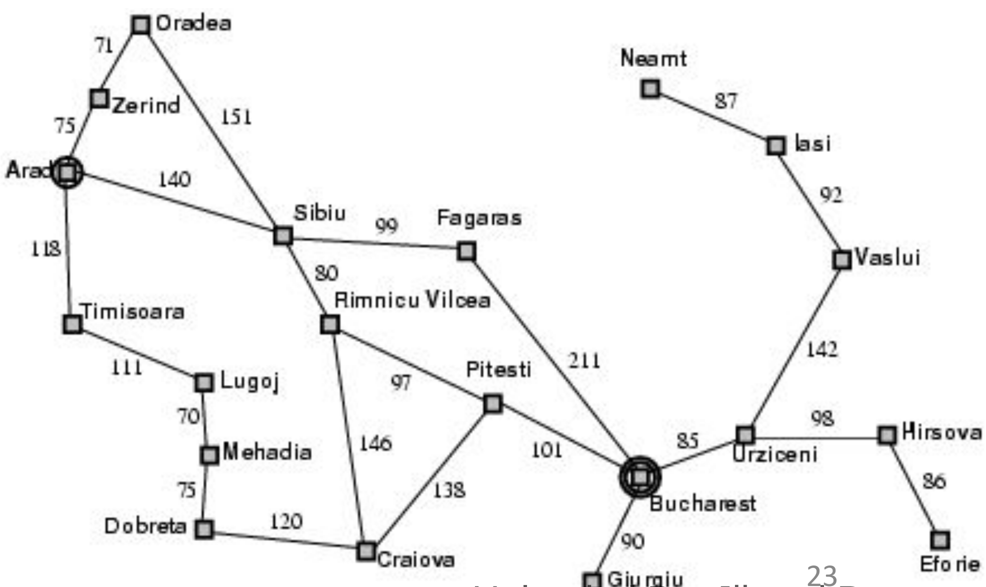




Graph and Tree

- the tree is generated by traversing the graph
- the same node in the graph may appear repeatedly in the tree
- the arrangement of the tree depends on the traversal strategy (search method)
- the initial state becomes the root node of the tree
- in the fully expanded tree, the goal states are the leaf nodes
- cycles in graphs may result in infinite branches





General Tree Search Algorithm

- generate the first node from the initial state of the problem

- Repeat
  - return failure if there are no more nodes in the fringe
  - examine the current node; if it's a goal, return the solution
  - expand the current node, and add the new nodes to the fringe

Terminology:

Fringe: Set of "visible" but unexplored notes

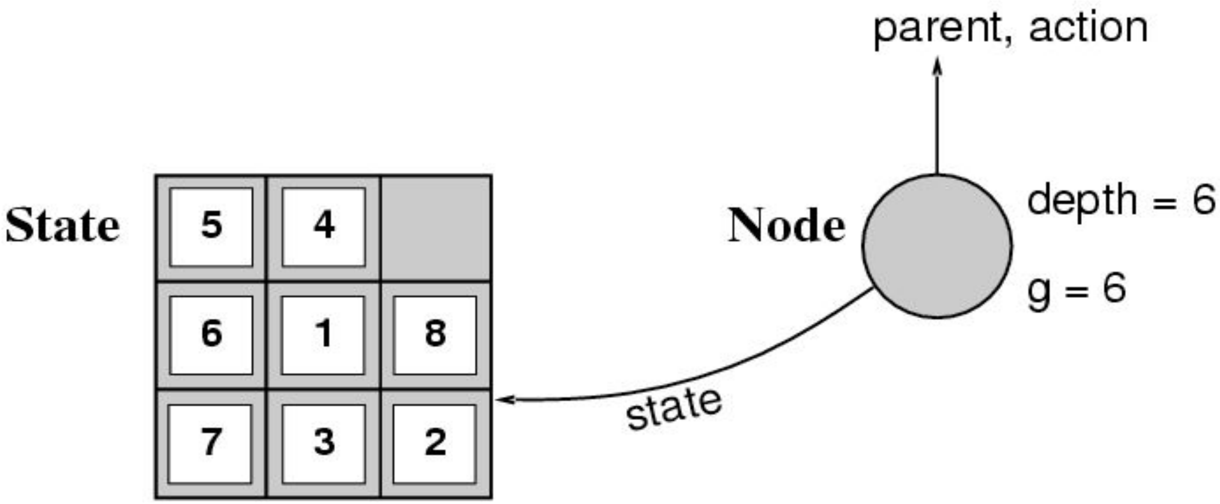
General Search Algorithm

```
function GENERAL-SEARCH(problem, QUEUING-FN) returns solution
    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,
    ActionS[problem]))
     end
```

Note: QUEUING-FN is customizable which will be used to specify the search method

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 node, action, path cost g(x), depth



- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

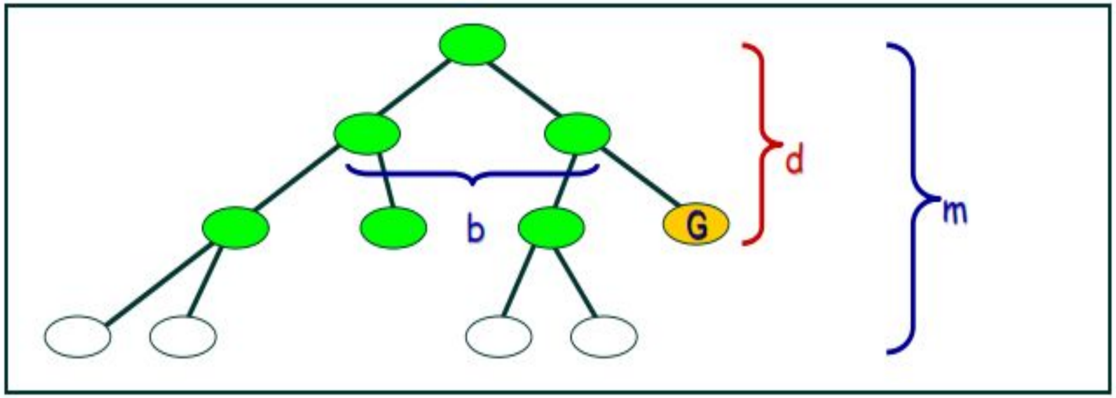
Search strategies

- A search strategy is defined by picking the order of node expansion

- Strategies are evaluated along the following dimensions:
  - Completeness: does it always find a solution if one exists?
  - Optimality: does it always find a least-cost solution?
  - Time complexity: time it takes to find the solution (number of nodes generated)
  - Space complexity: maximum number of nodes in memory

Search strategies

- Time and space complexity are measured in terms of
  - b: maximum branching factor of the search tree
  - d: depth of the least-cost solution
  - m: maximum length of any path in the state space (may be infinite)



Search Cost and Path Cost

- the search cost indicates how expensive it is to generate a solution
  - time complexity (e.g. number of nodes generated) is usually the main factor
  - sometimes space complexity (memory usage) is considered as well
- path cost indicates how expensive it is to execute the solution found in the search
  - distinct from the search cost, but often related
- total cost is the sum of search and path costs

Search strategies

- Uninformed Search
  - breadth-first
  - depth-first
  - uniform-cost search
  - depth-limited search
  - iterative deepening
  - bi-directional search
- Informed Search
  - best-first search
  - search with heuristics
  - memory-bounded search
  - iterative improvement search
- Local Search and Optimization
  - hill-climbing
  - simulated annealing
  - local beam search
  - genetic algorithms
  - constraint satisfaction
- Others