UIT2504 Artificial Intelligence

Problem Solving Agents

C. Aravindan
<AravindanC@ssn.edu.in>

Professor of Computing
SSN College of Engineering

August 05, 2024

Introduction

- Consider problems such as the following:

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle
  - Place 8 queens on a chess board such that no queen is attacked

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle
  - Place 8 queens on a chess board such that no queen is attacked
  - From a place called Arad in Romania, find the best road route to reach Bucharest to catch a flight

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle
  - Place 8 queens on a chess board such that no queen is attacked
  - From a place called Arad in Romania, find the best road route to reach Bucharest to catch a flight
  - Given a map of cities, find a circuit starting from a city, visiting each city only once, and return to the start city (Hamiltonian Circuit)
  - Given a map of cities, find a shortest Hamiltonian Circuit (Traveling Salesperson Problem)

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle
  - Place 8 queens on a chess board such that no queen is attacked
  - From a place called Arad in Romania, find the best road route to reach Bucharest to catch a flight
  - Given a map of cities, find a circuit starting from a city, visiting each city only once, and return to the start city (Hamiltonian Circuit)
  - Given a map of cities, find a shortest Hamiltonian Circuit (Traveling Salesperson Problem)
  - Given n objects of known weights $w_1, w_2, \cdots, w_n$ and values $v_1, v_2, \cdots, v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack (Knapsack Problem)

Introduction

- Consider problems such as the following:
  - Solve a SuDoKu puzzle
  - Place 8 queens on a chess board such that no queen is attacked
  - From a place called Arad in Romania, find the best road route to reach Bucharest to catch a flight
  - Given a map of cities, find a circuit starting from a city, visiting each city only once, and return to the start city (Hamiltonian Circuit)
  - Given a map of cities, find a shortest Hamiltonian Circuit (Traveling Salesperson Problem)
  - Given n objects of known weights $w_1, w_2, \cdots, w_n$ and values $v_1, v_2, \cdots, v_n$ and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack (Knapsack Problem)
- Is there any common approach to solve such problems? — general purpose problem solving approach

State-Space Approach

- "Formulate" the problem in "state space" and "search" for a "solution"

State-Space Approach

- "Formulate" the problem in "state space" and "search" for a "solution"
- States encapsulate certain properties of a system

State-Space Approach

- "Formulate" the problem in "state space" and "search" for a "solution"
- States encapsulate certain properties of a system
- System changes from one state to another based on action / event

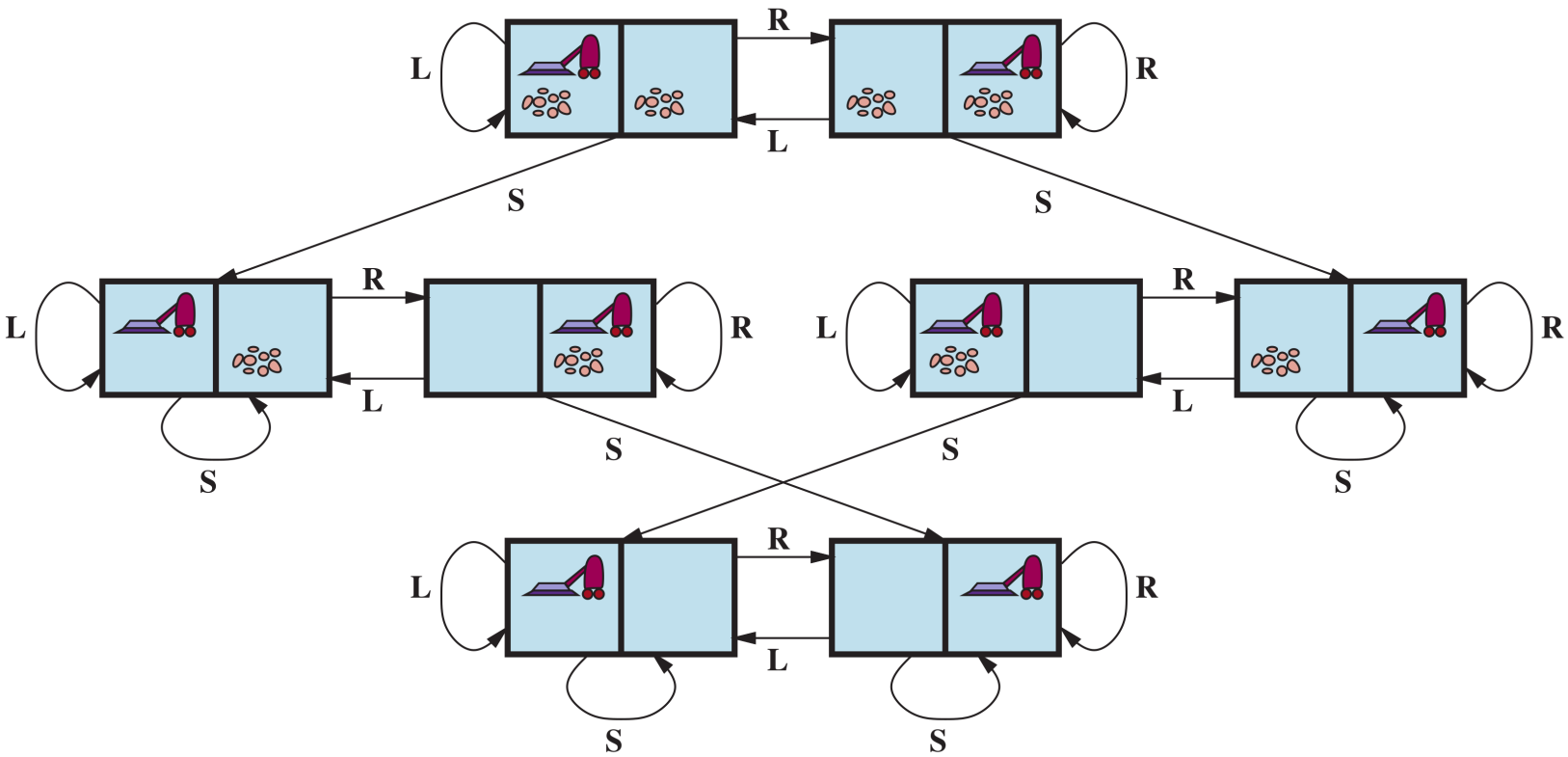
State-Space Approach

- "Formulate" the problem in "state space" and "search" for a "solution"
- States encapsulate certain properties of a system
- System changes from one state to another based on action / event
- Only certain actions may be permitted in a given state

State-Space Approach

- "Formulate" the problem in "state space" and "search" for a "solution"
- States encapsulate certain properties of a system
- System changes from one state to another based on action / event
- Only certain actions may be permitted in a given state
- There is an "initial" state for the system and some "goal" states to terminate

State-Graph



Recall: Granularity of state definition

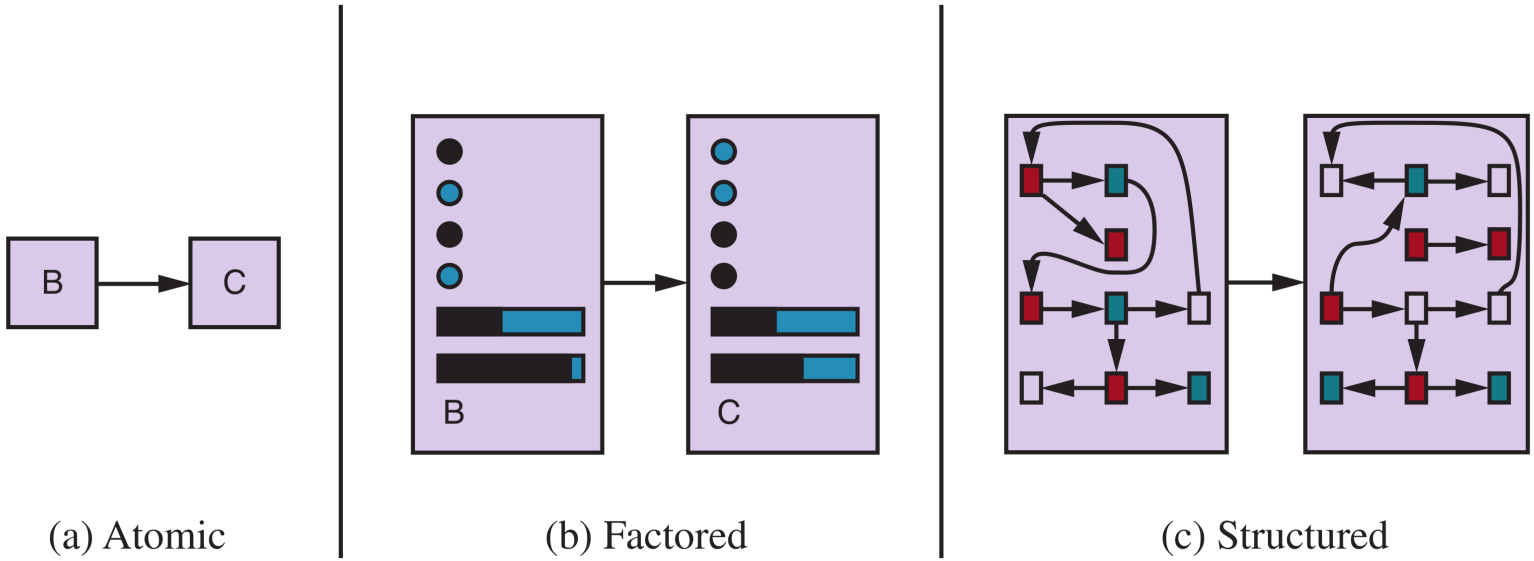


Illustration: 8-Puzzle

3	2	7
5	8	
1	4	6

Illustration: 8-Puzzle

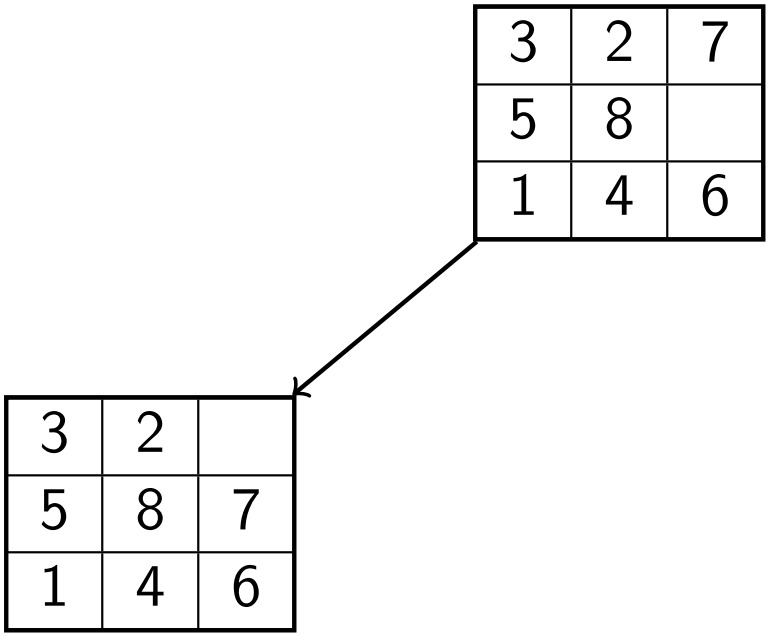


Illustration: 8-Puzzle

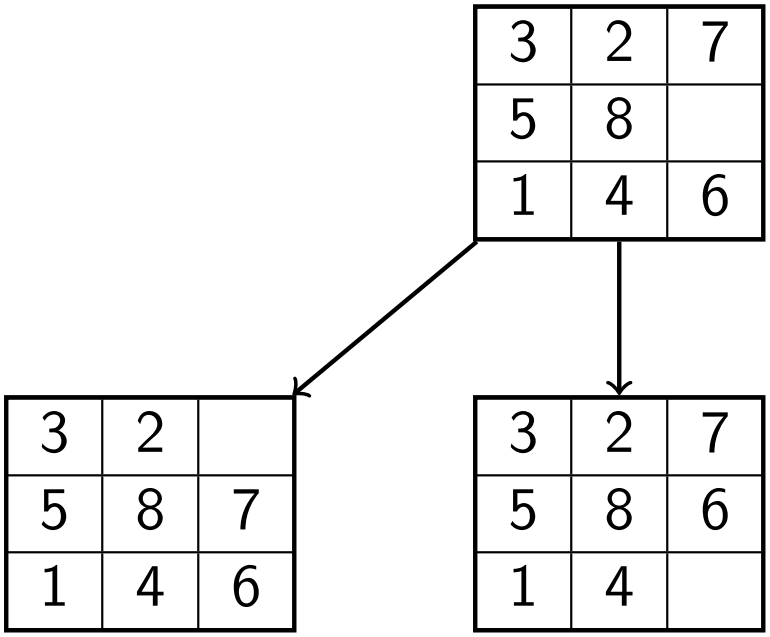


Illustration: 8-Puzzle

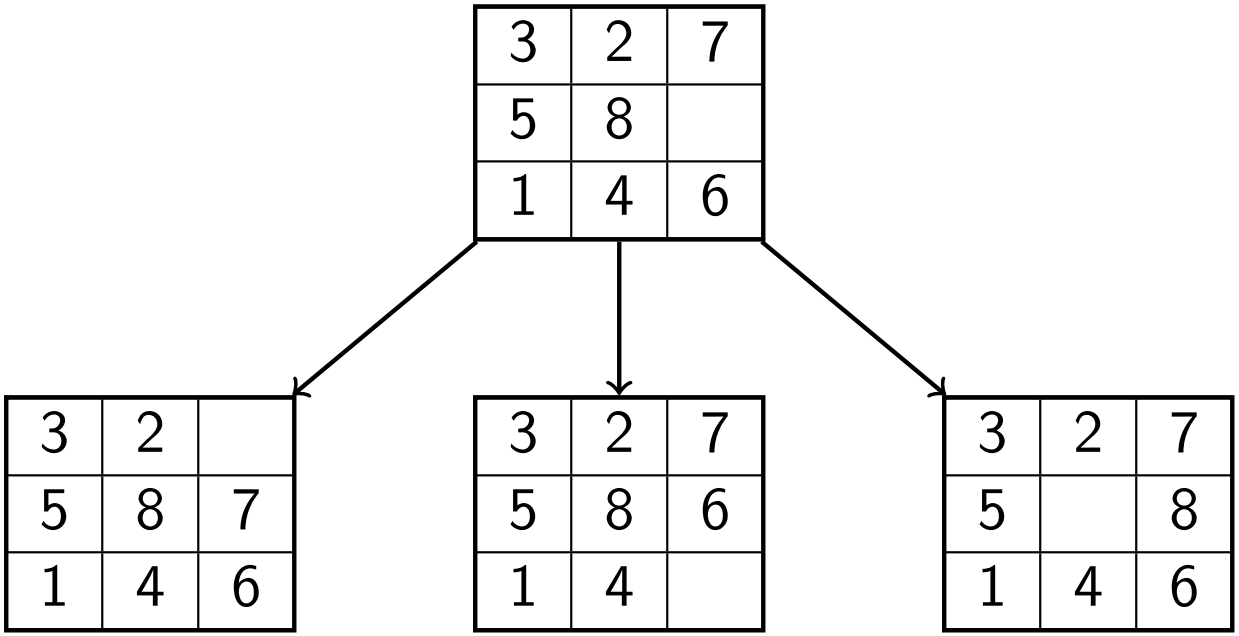


Illustration: 8-Puzzle

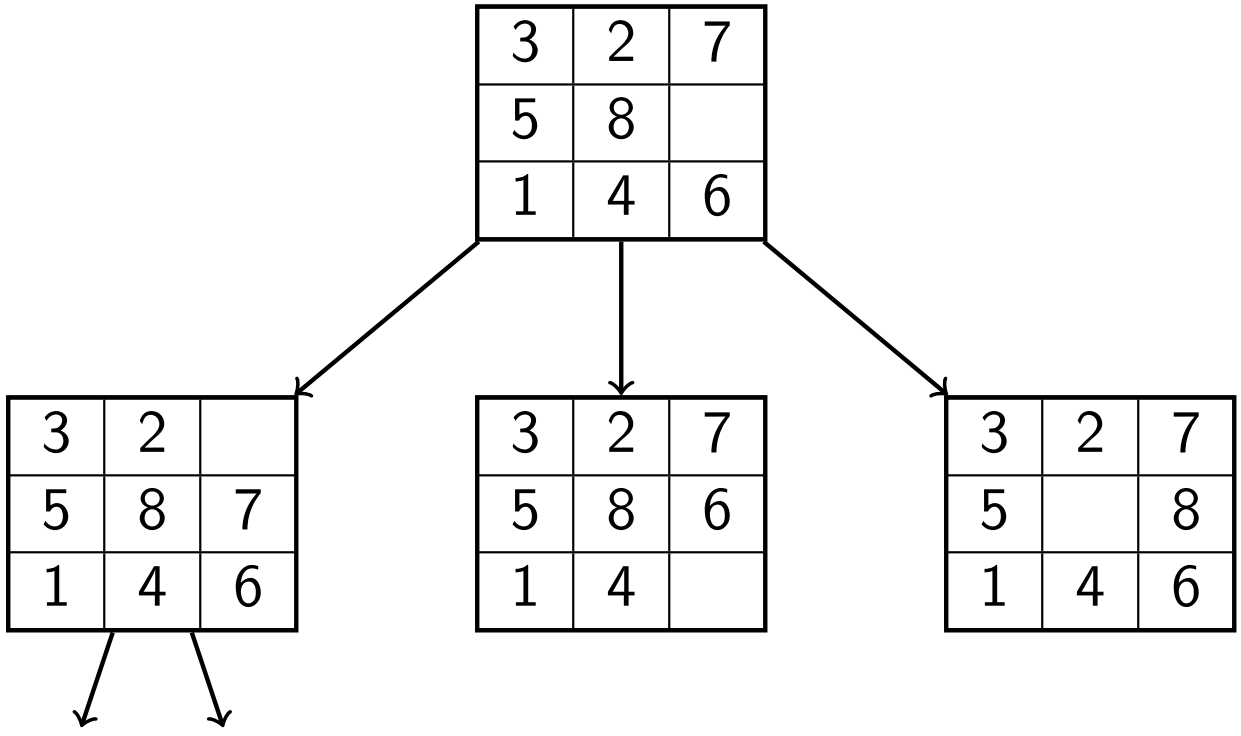


Illustration: 8-Puzzle

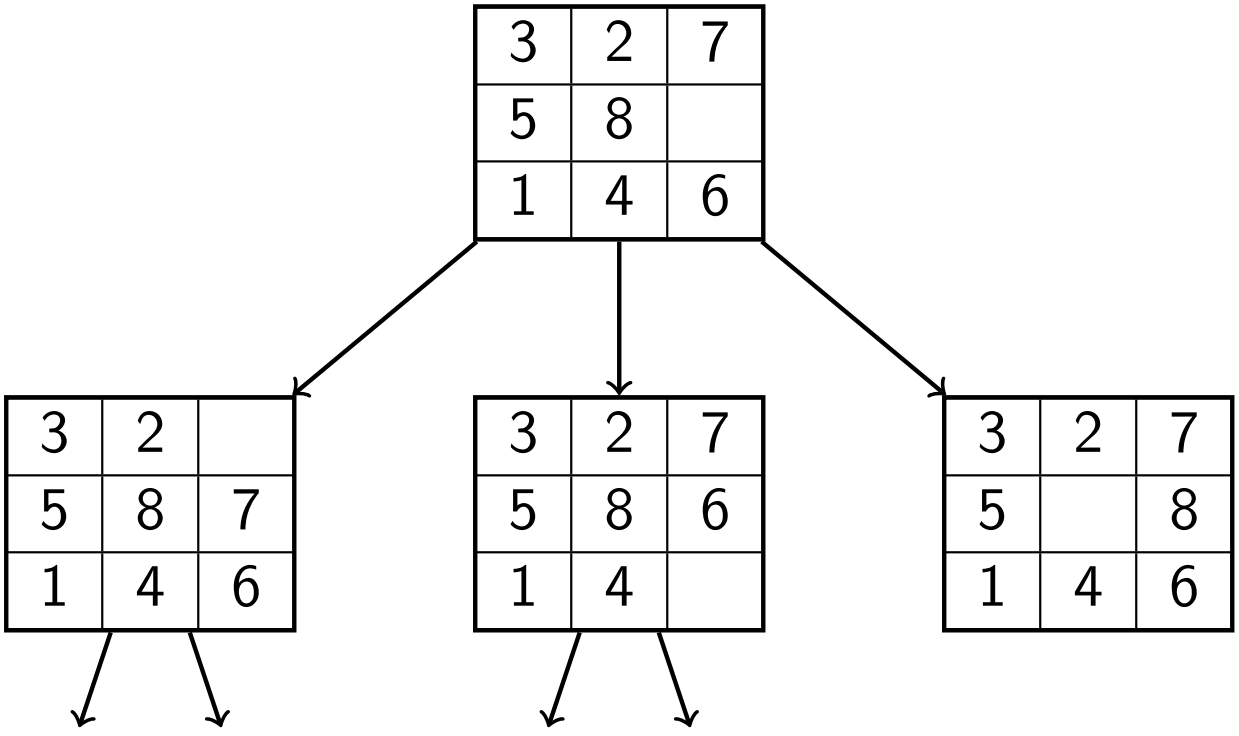


Illustration: 8-Puzzle

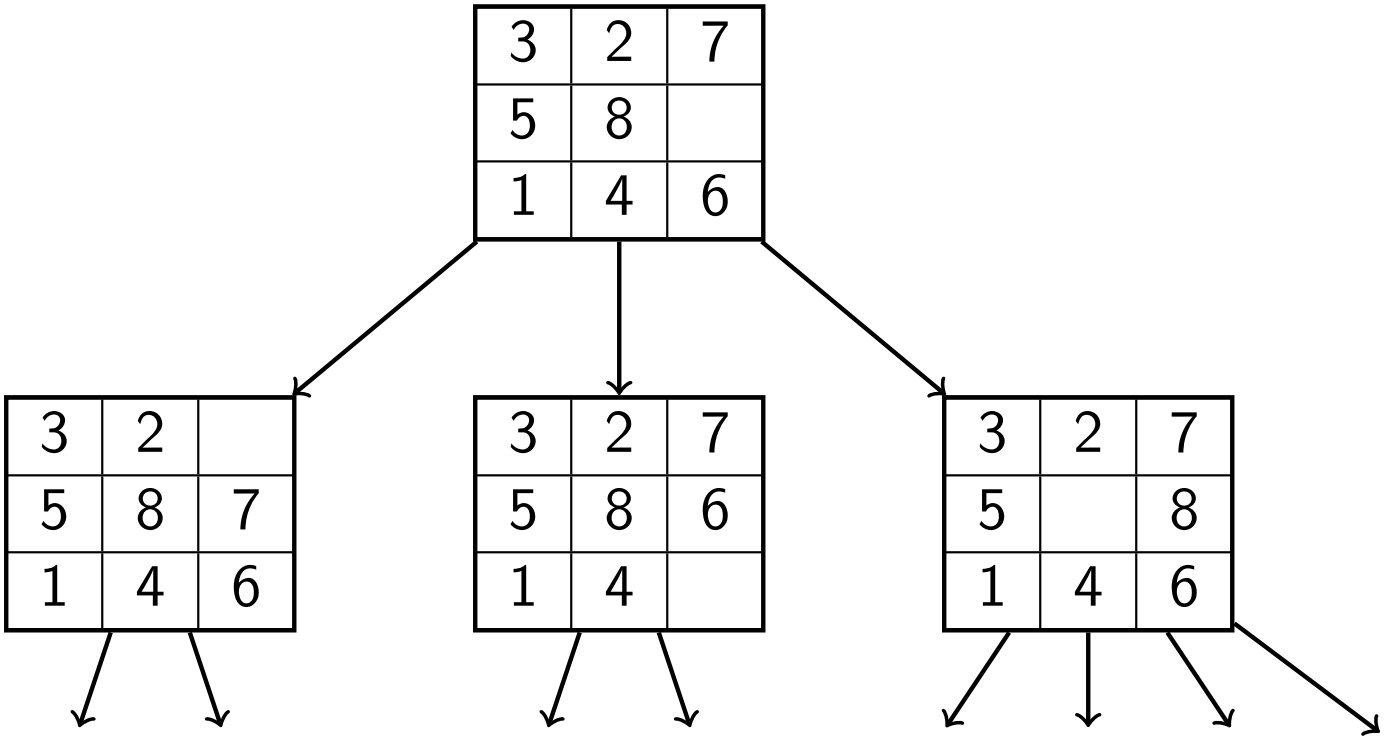
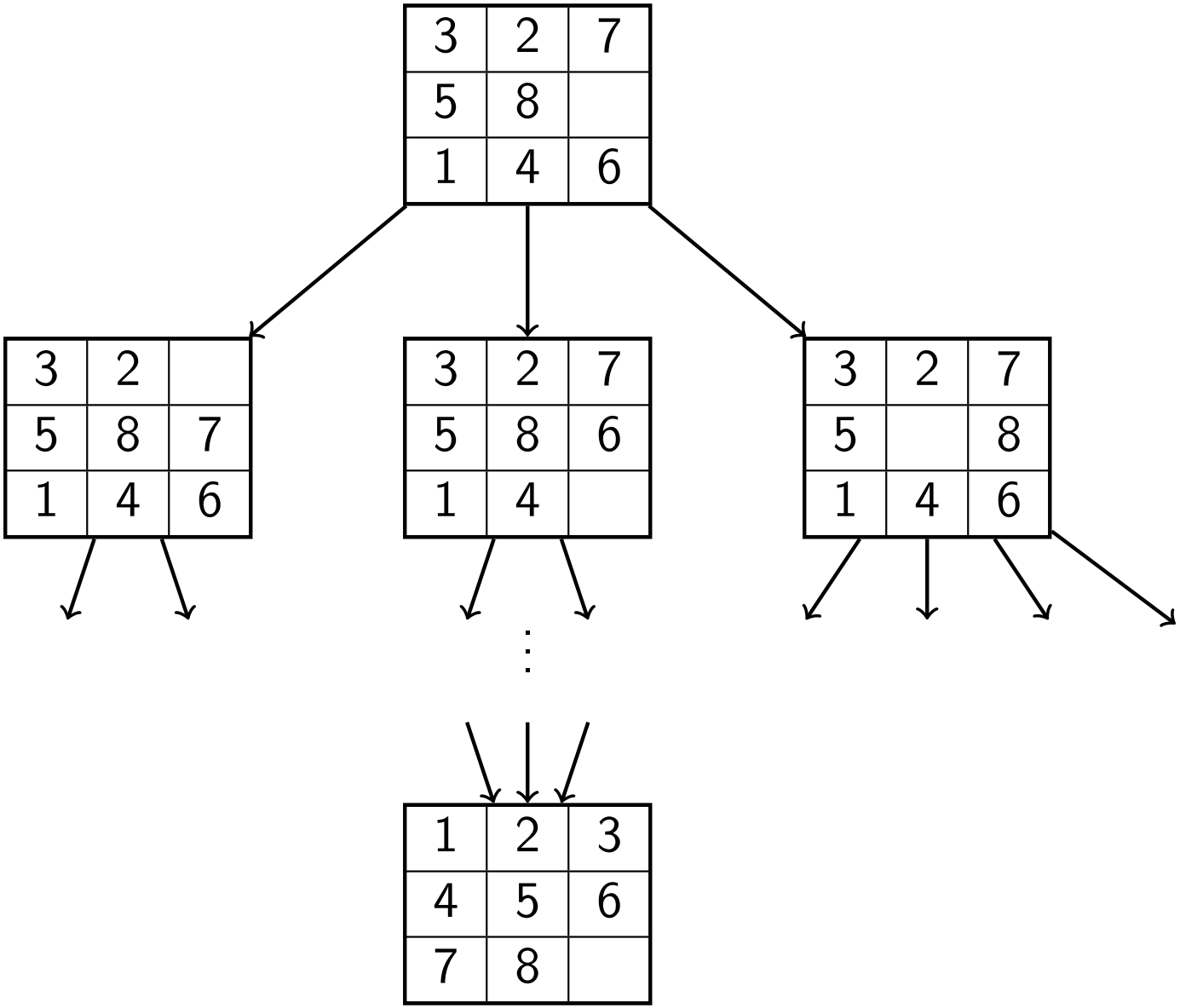


Illustration: 8-Puzzle



State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree?

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes — there can also be "dead-end" states where no action is possible

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes — there can also be "dead-end" states where no action is possible
- Where is the solution in the search tree?

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes — there can also be "dead-end" states where no action is possible
- Where is the solution in the search tree? — normally a path from initial state to a goal state

State-Space Approach

- Given an initial state, and a set of actions, "search tree" can be visualized
- The number of possible actions determine the branching factor of the tree
- State space is the set of all states that are reachable from the initial state using a sequence of actions
- Which states will be leaf nodes in the search tree? — all goal states will be leaf nodes — there can also be "dead-end" states where no action is possible
- Where is the solution in the search tree? — normally a path from initial state to a goal state
- In some cases, goal state itself is a solution!

Questions?

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does Actions and transition model may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using some action

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does Actions and transition model may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using some action
  - A set of one or more goal states — this may be facilitated by defining a Goal test

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does Actions and transition model may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using some action
  - A set of one or more goal states — this may be facilitated by defining a Goal test — a function that returns true if a given state is a goal state

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does Actions and transition model may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using some action
  - A set of one or more goal states — this may be facilitated by defining a Goal test — a function that returns true if a given state is a goal state
  - An action cost function that gives cost of applying an action in a given state

Problem Formulation

- A problem may be formulated in the state-space approach by defining the following components:
  - Set of possible states of the environment, called as the state space
  - An unique state of the environment, called as the initial state, in which the agent starts
  - Set of possible actions — may be abstracted by a function that, given a state, returns possible actions in that state
  - A transition model which describes what each action does Actions and transition model may be abstracted by a successor function S. For any state x, $S(x)$ is the set of states reachable from x using some action
  - A set of one or more goal states — this may be facilitated by defining a Goal test — a function that returns true if a given state is a goal state
  - An action cost function that gives cost of applying an action in a given state Path cost — sum of all the costs of actions from initial state to a goal state

Example: 8-Puzzle

- *State*:

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*:

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4
- *Goal test*:

Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4
- *Goal test*: state matches the given target configuration

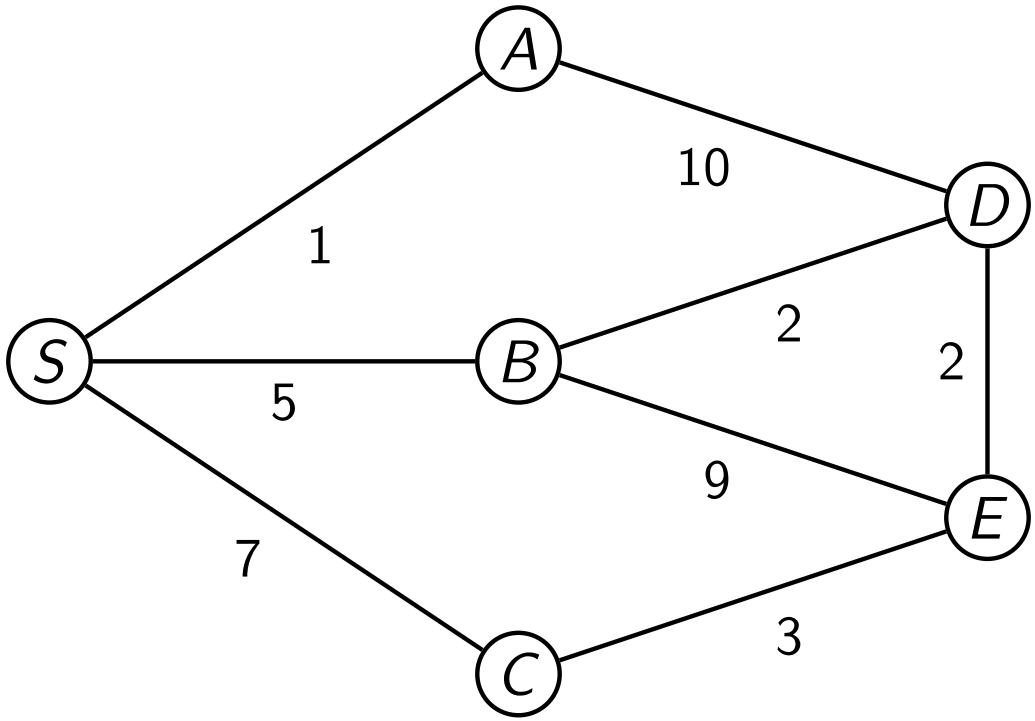
Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4
- *Goal test*: state matches the given target configuration
- *Action cost*:

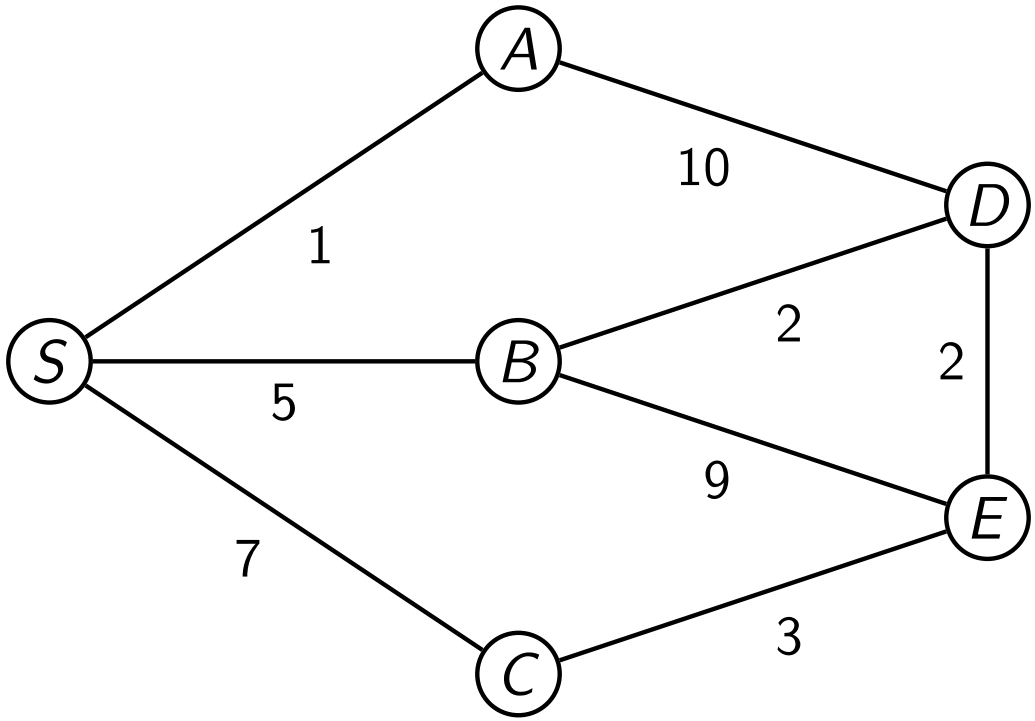
Example: 8-Puzzle

- *State*: Arrangement of eight tiles on a 3×3 grid — *Initial state* is some random arrangement of eight tiles on nine squares
- *Actions*: blank moves left, right, up, or down — branching factor of 4
- *Goal test*: state matches the given target configuration
- *Action cost*: unit cost for each action

Example: Route finding problem



Example: Route finding problem



Find a route from S to E

Example: Route finding problem

- *State*:

Example: Route finding problem

- *State*: Being in a particular city x

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*:

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road — optionally, avoid the city from where we came to x

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road — optionally, avoid the city from where we came to x
- *Goal test*:

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road — optionally, avoid the city from where we came to x
- *Goal test*: Current city matches the destination

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road — optionally, avoid the city from where we came to x
- *Goal test*: Current city matches the destination
- *Action cost*:

Example: Route finding problem

- *State*: Being in a particular city x — start city is the *Initial state*
- *Actions*: move to a city y that is directly connected by a road — optionally, avoid the city from where we came to x
- *Goal test*: Current city matches the destination
- *Action cost*: Cost associated with the road taken

Example: 8-Queens Problem

- *Goal test*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board — *Initial state* is board with 0 queens

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board — *Initial state* is board with 0 queens
- *Actions*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board — *Initial state* is board with 0 queens
- *Actions*: Add a queen to any free square

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board — *Initial state* is board with 0 queens
- *Actions*: Add a queen to any free square
- How many paths are there in the search tree?

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board — *Initial state* is board with 0 queens
- *Actions*: Add a queen to any free square
- How many paths are there in the search tree? — $O(64^8)$

Example: 8-Queens Problem

- *Goal test*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board with none attacked

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board with none attacked — *Initial state* is board with 0 queens

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board with none attacked — *Initial state* is board with 0 queens
- *Actions*:

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board with none attacked — *Initial state* is board with 0 queens
- *Actions*: Add a queen to the left-most empty column s.t. it is not attacked

Example: 8-Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Any arrangement of 0 to 8 queens on a board with none attacked — *Initial state* is board with 0 queens
- *Actions*: Add a queen to the left-most empty column s.t. it is not attacked
- Only 2057 possible sequences!!!

Example: 8 Queens Problem

- *Goal test*:

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*:

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*:

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Arrangement of 8 queens, one in each column

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Arrangement of 8 queens, one in each column — *Initial state* may be some random arrangement of 8 queens, one in each column

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Arrangement of 8 queens, one in each column — *Initial state* may be some random arrangement of 8 queens, one in each column
- *Actions*:

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Arrangement of 8 queens, one in each column — *Initial state* may be some random arrangement of 8 queens, one in each column
- *Actions*: Move an attacked queen to another square in the same column

Example: 8 Queens Problem

- *Goal test*: 8-queens on a chess board — none attacked
- *Action cost*: None!
- *State*: Arrangement of 8 queens, one in each column — *Initial state* may be some random arrangement of 8 queens, one in each column
- *Actions*: Move an attacked queen to another square in the same column
- Complete state formulation — only one state is kept in memory — and it is iteratively improved

Questions?

Searching for a solution

- Start with initial state in the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution

Searching for a solution

- Start with initial state in the working set
- Iterate:
  - Return failure if the working set is empty
  - Choose and remove a state x from the working set
  - If it is a goal state, return solution
  - Else, expand x and add the successor states $S(x)$ to the working set

Search Strategies

- Uninformed:
  - Depth-First
  - Breadth-First
  - Depth-Limited
  - Bi-Directional Search
- Informed (Heuristics):
  - Best-first Greedy
  - A^*
  - Local Search Strategies
- Constraint Satisfaction

Performance Measures

- Completeness
- Time Complexity
- Space Complexity
- Optimality

Questions?

- Read Chapter 3