CSE 433
Artificial Intelligence
4th year, Computer Engineering
Fall 2012, Lecture #2

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

September 24th, 2012

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Tutorials:
  - Starting this week (Eng. Amr).

Course Info:

- Website: http://www.hshehata.name.eg/courses/cse433

Adminstrivia

Notes

- Tutorials:
  - Starting this week (Eng. Amr).
- Office hours:
  - Need to be determined!

Course Info:

- Website: http://www.hshehata.name.eg/courses/cse433

Adminstrivia

Notes

- Tutorials:
  - Starting this week (Eng. Amr).
- Office hours:
  - Need to be determined!
- Assignment #1:
  - Programming assignment
  - To be released next week.
  - Work in groups of two.

Course Info:

- Website: http://www.hshehata.name.eg/courses/cse433

Outline

Outline

Problem solving by searching

Introduction

- Intelligent agents try to maximize their performance measure.

Problem solving by searching

Introduction

- Intelligent agents try to maximize their performance measure.
- An agent performs *actions* to get from its *initial* state to a *goal*.

Problem solving by searching

Introduction

- Intelligent agents try to maximize their performance measure.
- An agent performs *actions* to get from its *initial* state to a *goal*.
- The process of looking for a sequence of actions to reach a goal is called search.

Problem solving by searching

Introduction

- Intelligent agents try to maximize their performance measure.
- An agent performs *actions* to get from its *initial* state to a *goal*.
- The process of looking for a sequence of actions to reach a goal is called search.
- A search algorithm takes a *problem* as an input and returns a *solution* in the form of an action sequence.

Problem solving by searching

Introduction

- Intelligent agents try to maximize their performance measure.
- An agent performs *actions* to get from its *initial* state to a *goal*.
- The process of looking for a sequence of actions to reach a goal is called search.
- A search algorithm takes a *problem* as an input and returns a *solution* in the form of an action sequence.
- Search is central in many AI systems:
  - Theorem proving, VLSI layout, Game playing, Navigation, *etc*.

Problem solving by searching

Requirements of searching

- Define the problem:
  - Represent the search space by states.
  - Define the actions the agent can perform.
  - Define the costs associated with the defined actions.

Problem solving by searching

Requirements of searching

- Define the problem:
  - Represent the search space by states.
  - Define the actions the agent can perform.
  - Define the costs associated with the defined actions.
- Define a goal: What is the agent searching for?

Problem solving by searching

Requirements of searching

- Define the problem:
  - Represent the search space by states.
  - Define the actions the agent can perform.
  - Define the costs associated with the defined actions.
- Define a goal: What is the agent searching for?
- Define the solution:
  - The goal itself?
  - The path (i.e. sequence of actions) to get to the goal?

Problem solving by searching

Assumptions

- Goal-based agent.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.
  - Deterministic.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.
  - Deterministic.
  - Sequential.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.
  - Deterministic.
  - Sequential.
  - Static.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.
  - Deterministic.
  - Sequential.
  - Static.
  - Discrete.

Problem solving by searching

Assumptions

- Goal-based agent.
- Environment:
  - Fully observable.
  - Deterministic.
  - Sequential.
  - Static.
  - Discrete.
  - Single agent.

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**
  2. The possible actions available at each state.
     - ACTIONS : **STATE** $\rightarrow$ **ACTION SET**

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**
  2. The possible actions available at each state.
     - ACTIONS : **STATE** $\rightarrow$ **ACTION SET**
  3. The transition model describing what each action does.
     - RESULT : **STATE** $\times$ **ACTION** $\rightarrow$ **STATE**

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**
  2. The possible actions available at each state.
     - ACTIONS : **STATE** $\rightarrow$ **ACTION SET**
  3. The transition model describing what each action does.
     - RESULT : **STATE** $\times$ **ACTION** $\rightarrow$ **STATE**
  4. The goal test that detects the goal state.
     - GOAL-TEST : **STATE** $\rightarrow$ **BOOLEAN**

Problem solving by searching

Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**
  2. The possible actions available at each state.
     - ACTIONS : **STATE** $\rightarrow$ **ACTION SET**
  3. The transition model describing what each action does.
     - RESULT : **STATE** $\times$ **ACTION** $\rightarrow$ **STATE**
  4. The goal test that detects the goal state.
     - GOAL-TEST : **STATE** $\rightarrow$ **BOOLEAN**
  5. The path cost expressed in terms of step cost.
     - STEP-COST : **STATE** $\times$ **ACTION** $\rightarrow$ **REAL**$^+$.

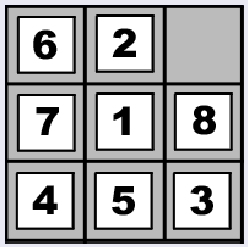
Problem solving by searching

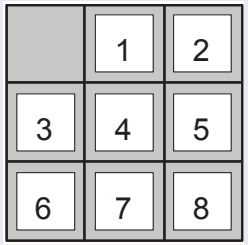
Formulating problems

- A problem can be defined formally by five components:
  1. The initial state that the agent starts in.
     - INITIAL-STATE : **STATE**
  2. The possible actions available at each state.
     - ACTIONS : **STATE** $\rightarrow$ **ACTION SET**
  3. The transition model describing what each action does.
     - RESULT : **STATE** $\times$ **ACTION** $\rightarrow$ **STATE**
  4. The goal test that detects the goal state.
     - GOAL-TEST : **STATE** $\rightarrow$ **BOOLEAN**
  5. The path cost expressed in terms of step cost.
     - STEP-COST : **STATE** $\times$ **ACTION** $\rightarrow$ **REAL**$^+$.
- This is known as a ***STATE-SPACE*** problem formulation.

Problem solving by searching

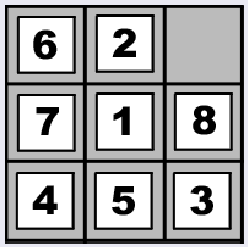
A puzzle example

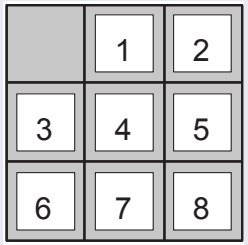




Problem solving by searching

A puzzle example

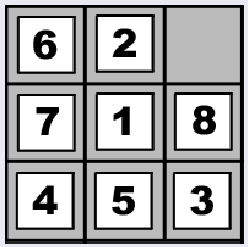


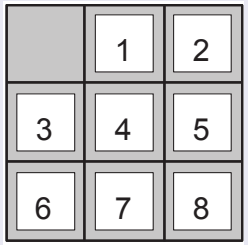


- **Initial state:** any arrangement of tiles.

Problem solving by searching

A puzzle example

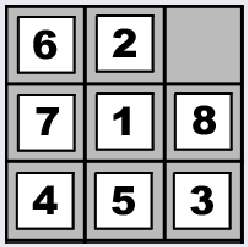


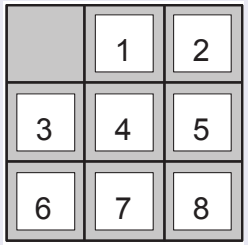


- **Initial state:** any arrangement of tiles.
- **Actions:** move blank left, right, up or down, provided it does not get out of the game.

Problem solving by searching

A puzzle example

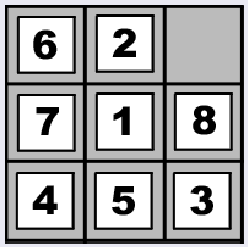


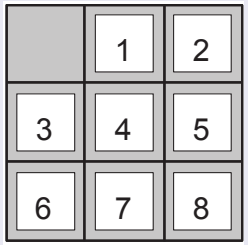


- **Initial state:** any arrangement of tiles.
- **Actions:** move blank left, right, up or down, provided it does not get out of the game.
- **Transition model:** given a state and action, return a new state by switching one tile with the blank.

Problem solving by searching

A puzzle example

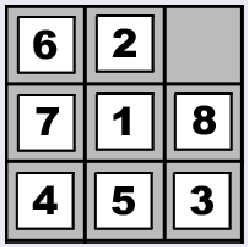


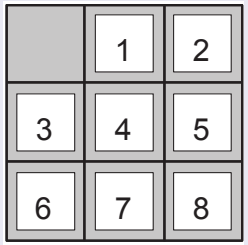


- **Initial state:** any arrangement of tiles.
- **Actions:** move blank left, right, up or down, provided it does not get out of the game.
- **Transition model:** given a state and action, return a new state by switching one tile with the blank.
- **Goal test:** are the tiles in the goal state order?

Problem solving by searching

A puzzle example

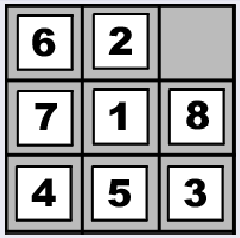




- **Initial state:** any arrangement of tiles.
- **Actions:** move blank left, right, up or down, provided it does not get out of the game.
- **Transition model:** given a state and action, return a new state by switching one tile with the blank.
- **Goal test:** are the tiles in the goal state order?
- **Path cost:** each step costs 1, so path cost is the number of steps along the path.

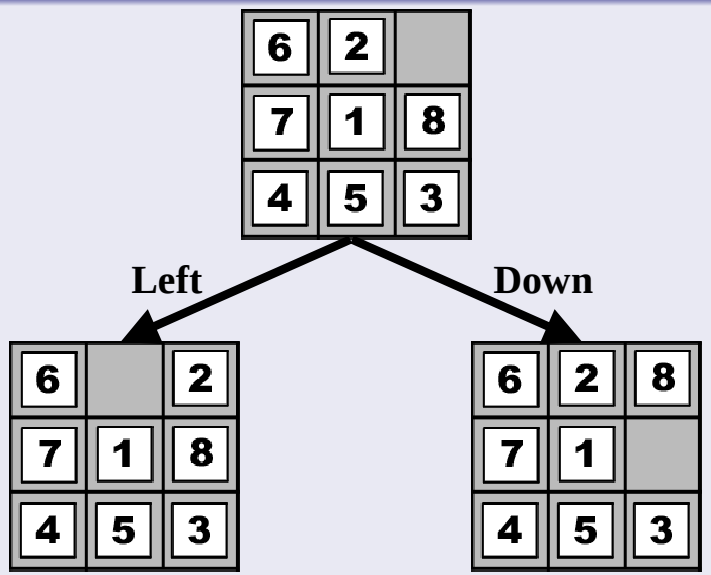
Problem solving by searching

A puzzle search tree



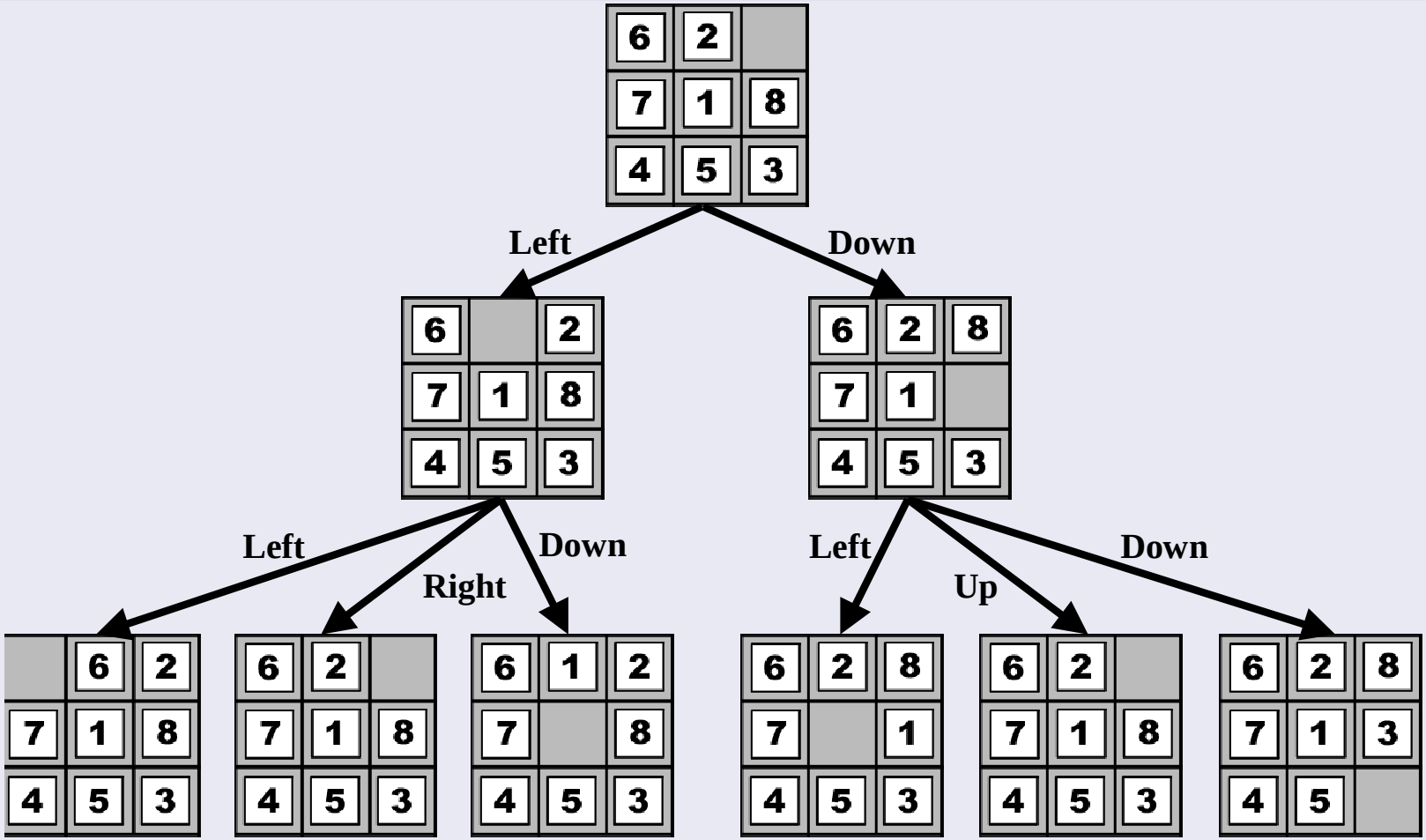
Problem solving by searching

A puzzle search tree



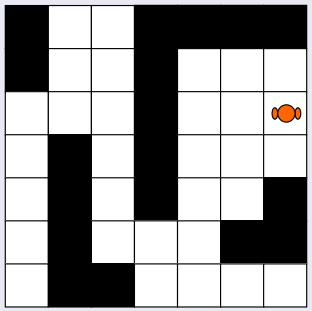
Problem solving by searching

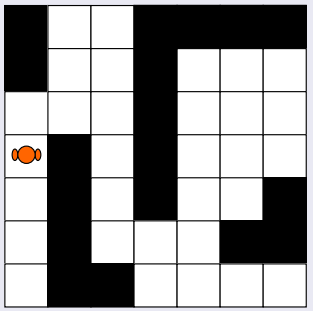
A puzzle search tree

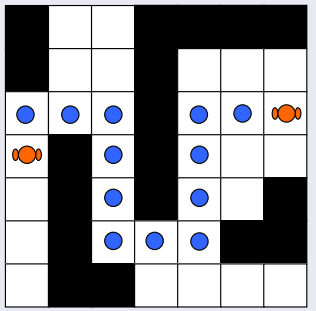


Problem solving by searching

A robot navigation example

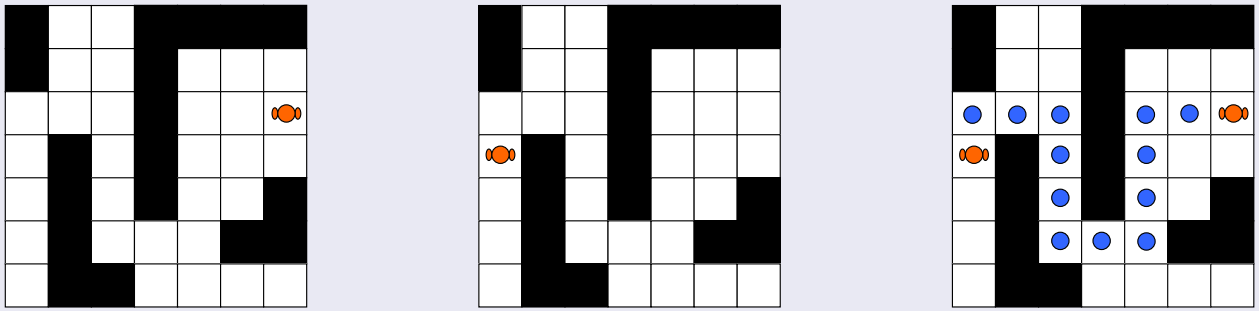






Problem solving by searching

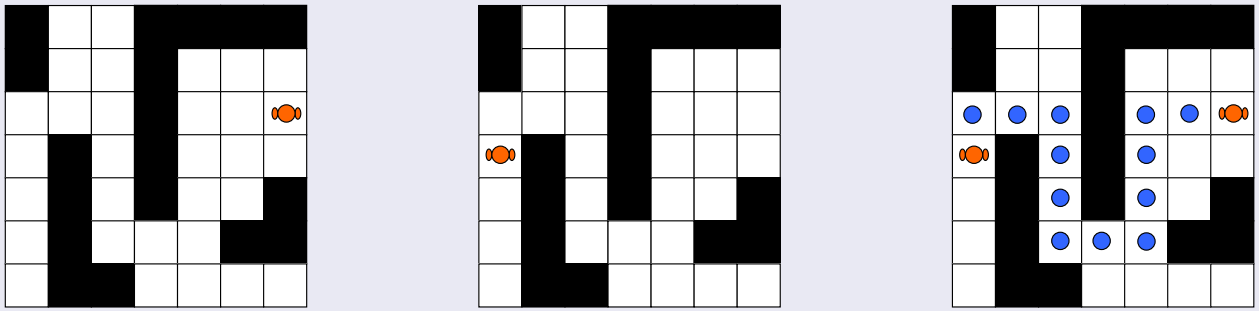
A robot navigation example



- **Initial state:** the robot is at a certain location.

Problem solving by searching

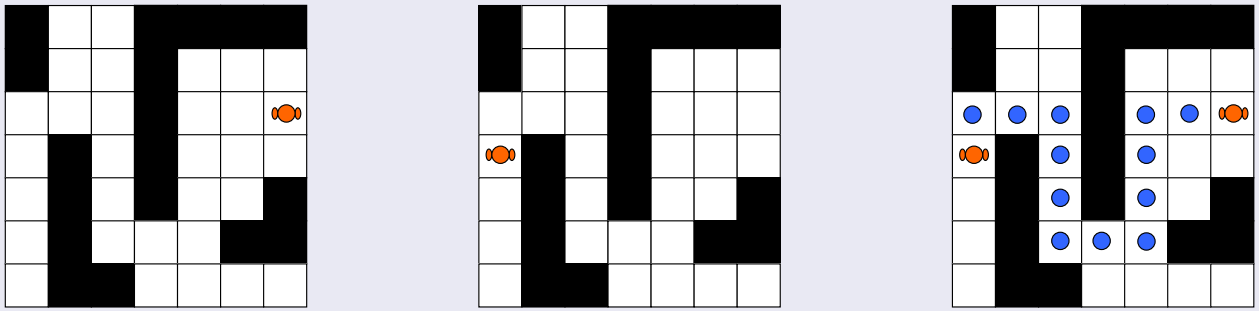
A robot navigation example



- **Initial state:** the robot is at a certain location.
- **Actions:** the robot moves left, right, up or down, while avoiding obstacles.

Problem solving by searching

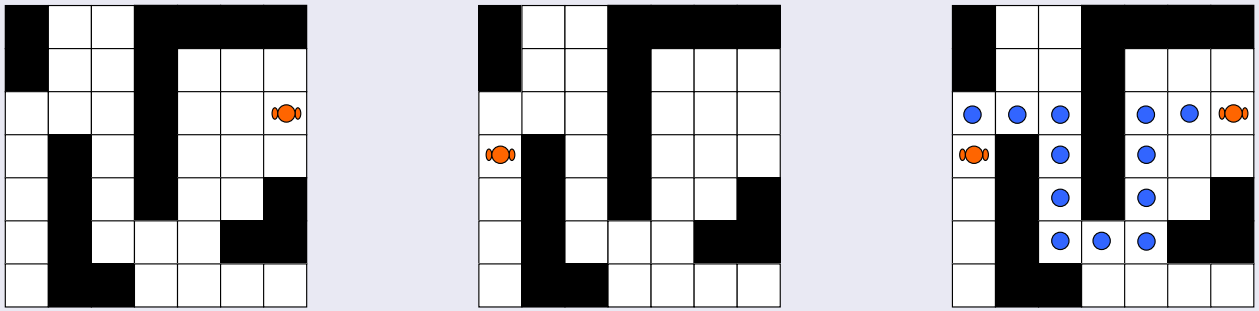
A robot navigation example



- **Initial state:** the robot is at a certain location.
- **Actions:** the robot moves left, right, up or down, while avoiding obstacles.
- **Transition model:** given a state and action, return a state representing a neighboring location.

Problem solving by searching

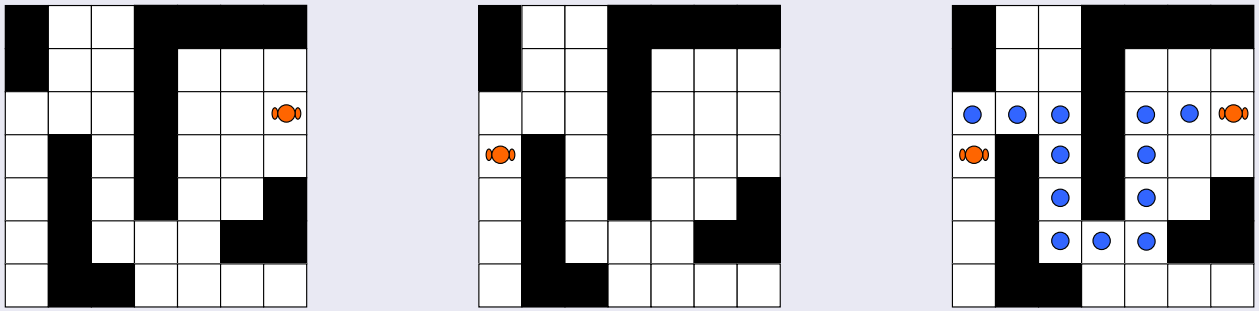
A robot navigation example



- **Initial state:** the robot is at a certain location.
- **Actions:** the robot moves left, right, up or down, while avoiding obstacles.
- **Transition model:** given a state and action, return a state representing a neighboring location.
- **Goal test:** is the robot at the final location?

Problem solving by searching

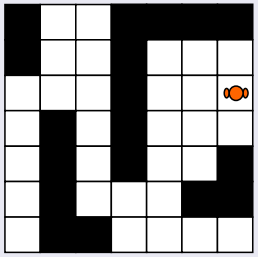
A robot navigation example



- **Initial state:** the robot is at a certain location.
- **Actions:** the robot moves left, right, up or down, while avoiding obstacles.
- **Transition model:** given a state and action, return a state representing a neighboring location.
- **Goal test:** is the robot at the final location?
- **Path cost:** each step costs 1, so path cost is the number of steps along the path.

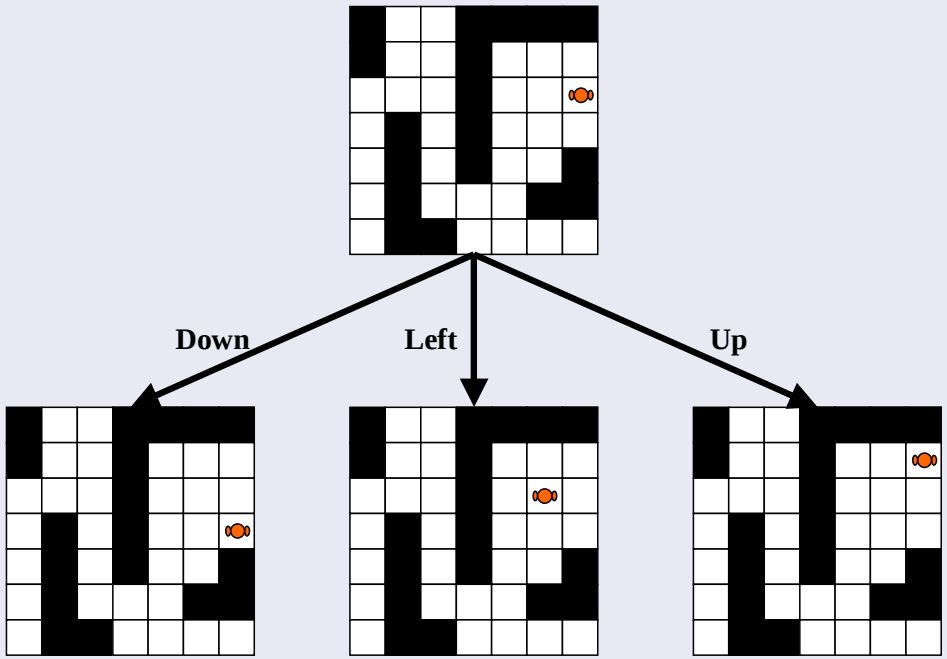
Problem solving by searching

A robot navigation search tree



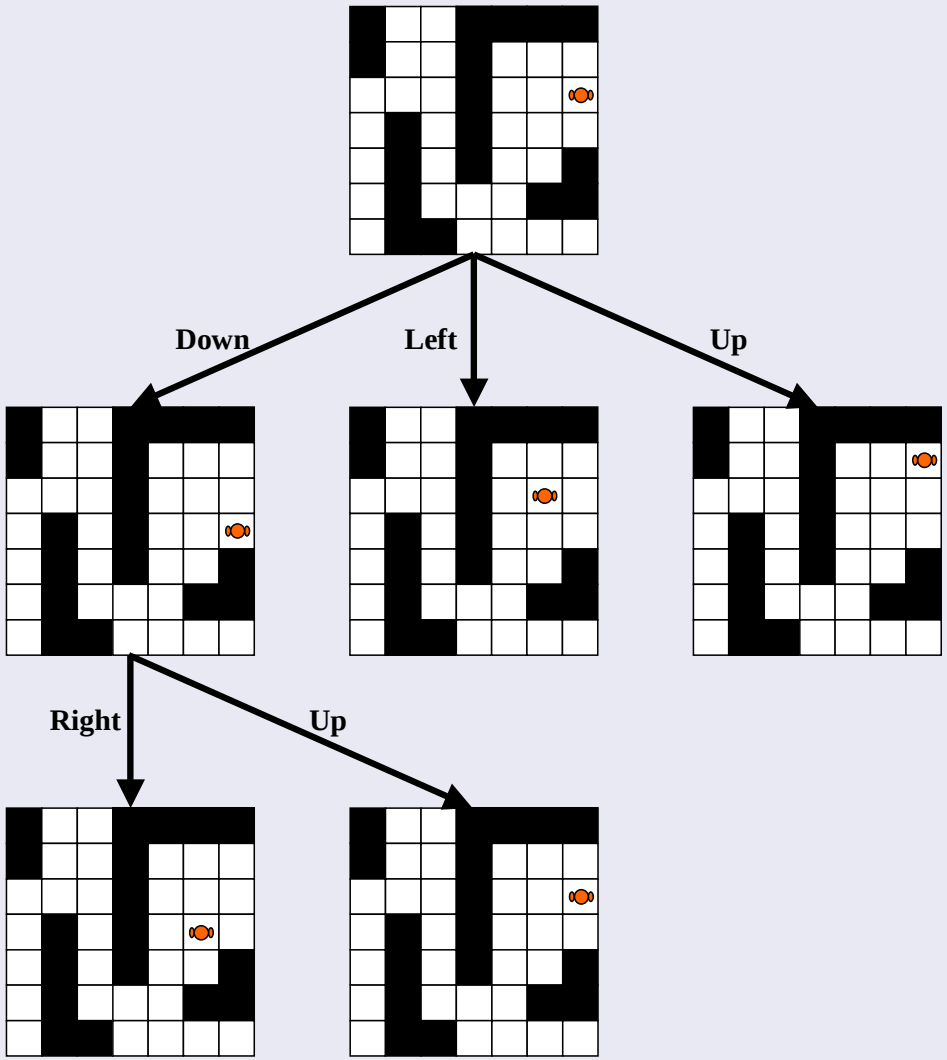
Problem solving by searching

A robot navigation search tree



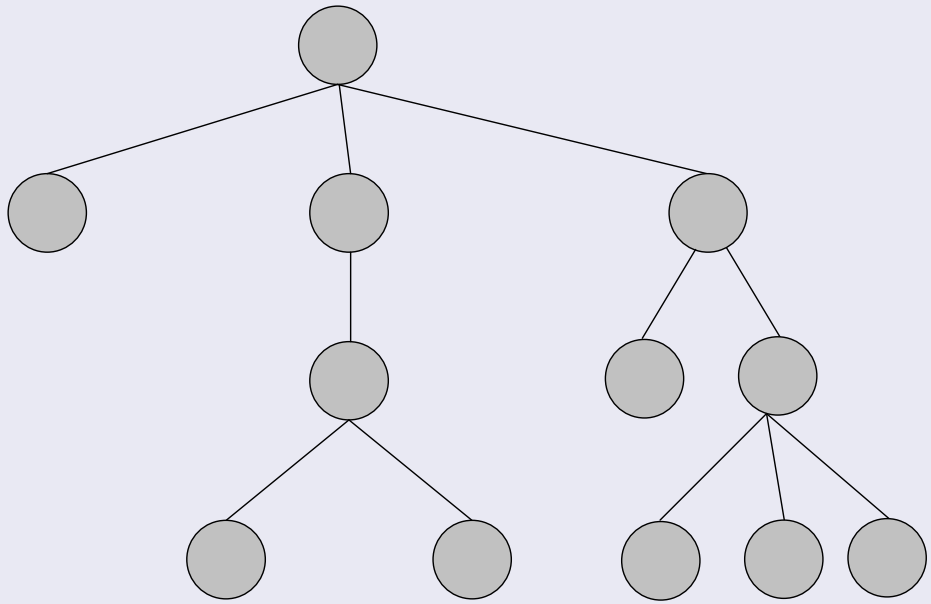
Problem solving by searching

A robot navigation search tree



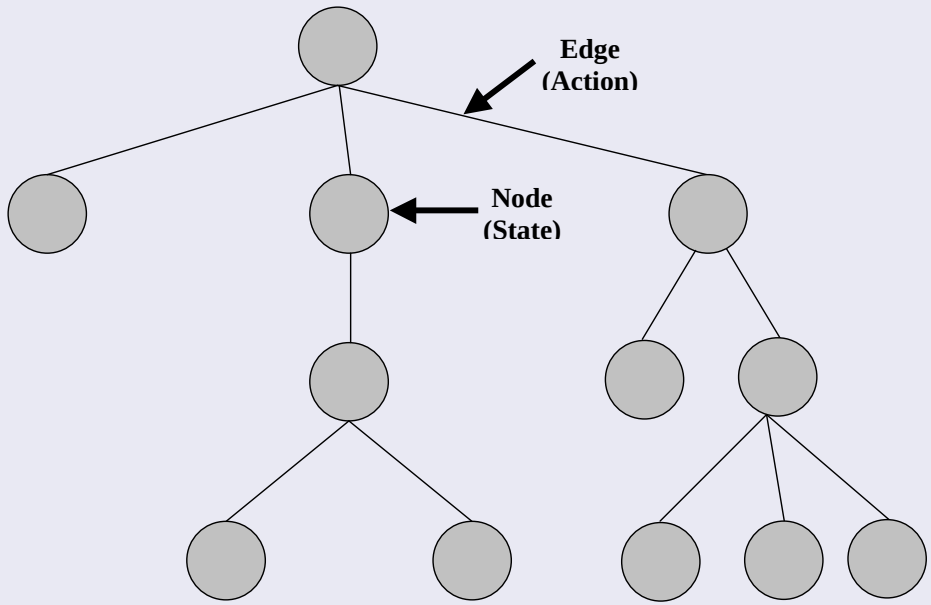
Problem solving by searching

A search tree



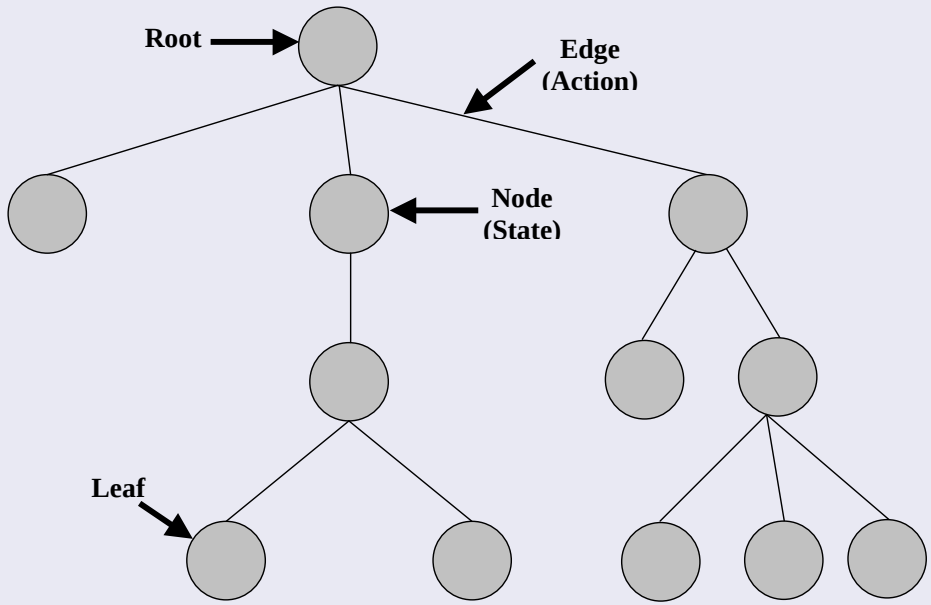
Problem solving by searching

A search tree



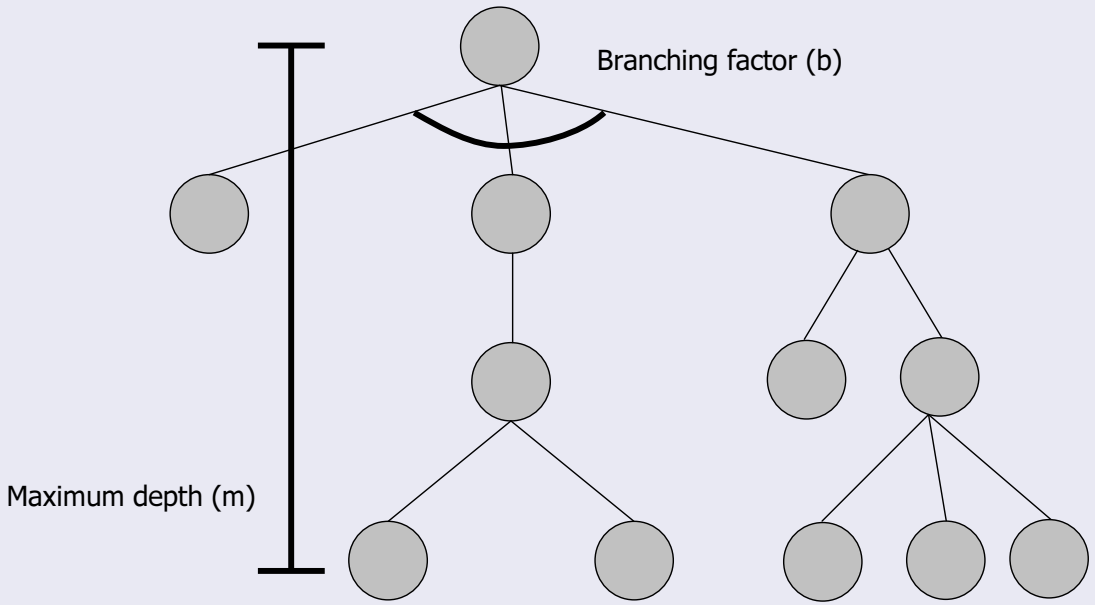
Problem solving by searching

A search tree



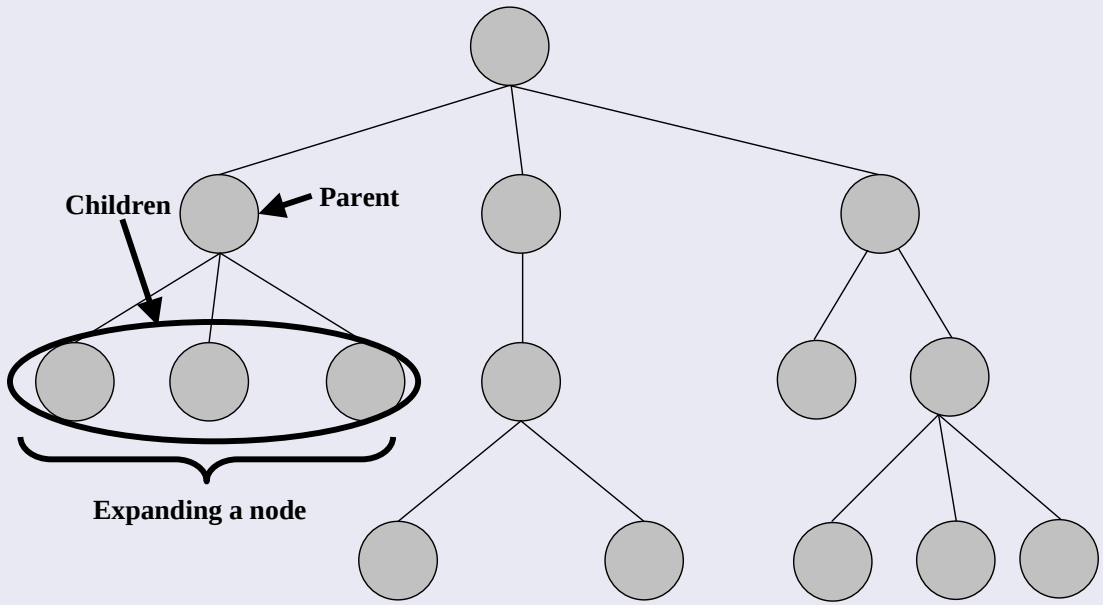
Problem solving by searching

A search tree



Problem solving by searching

A search tree



Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure

Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure
initialize the frontier using the initial state of a *problem*

Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  loop do
    if the frontier is empty **then return** failure

Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier

Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution

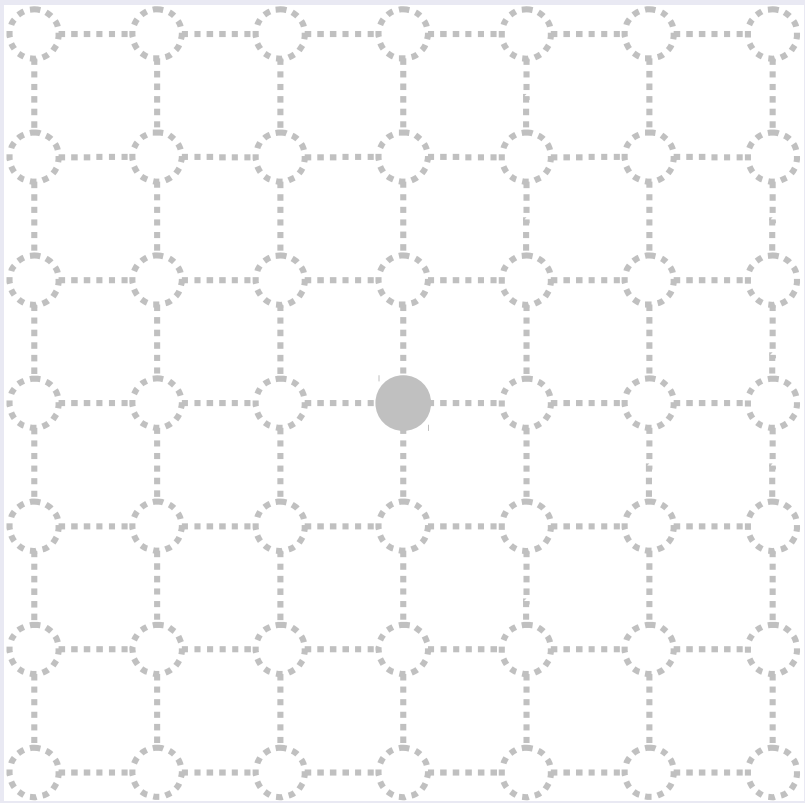
Search Algorithms

General tree-search algorithm

TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting node to the frontier
  end

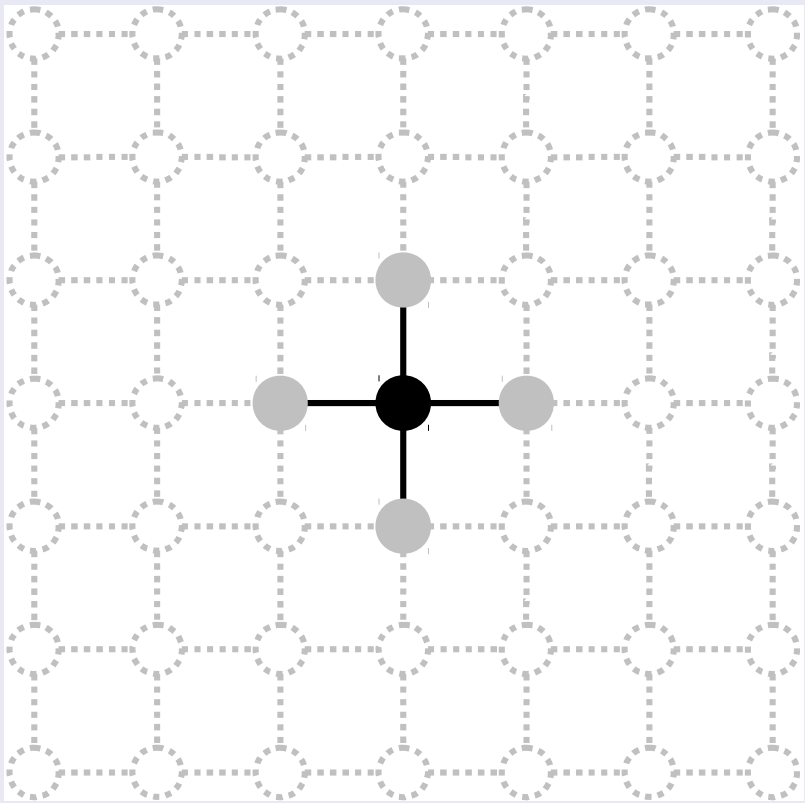
Search Algorithms

Tree-Search on a rectangular grid problem



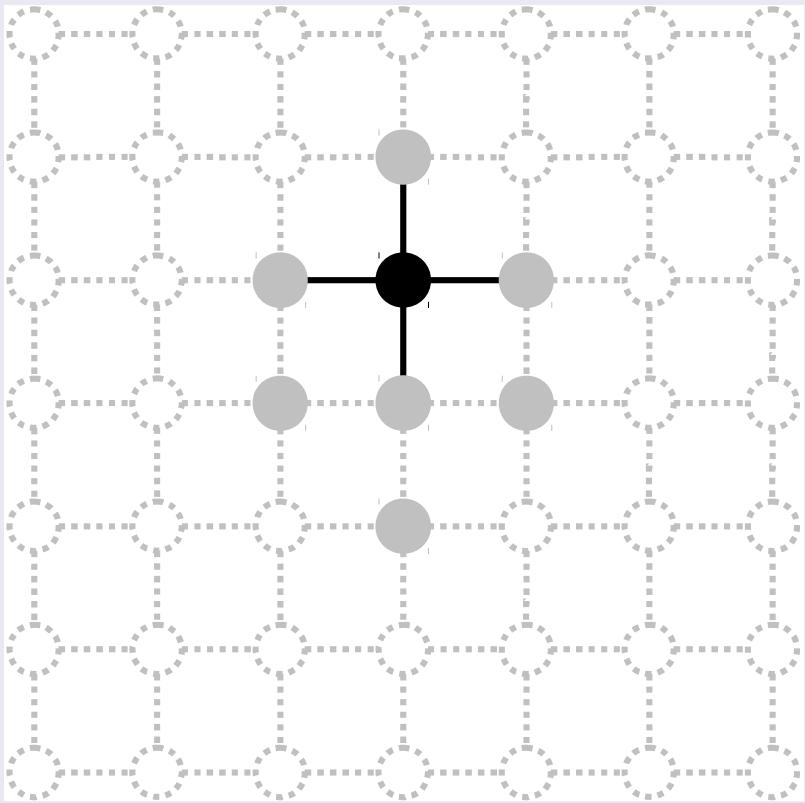
Search Algorithms

Tree-Search on a rectangular grid problem



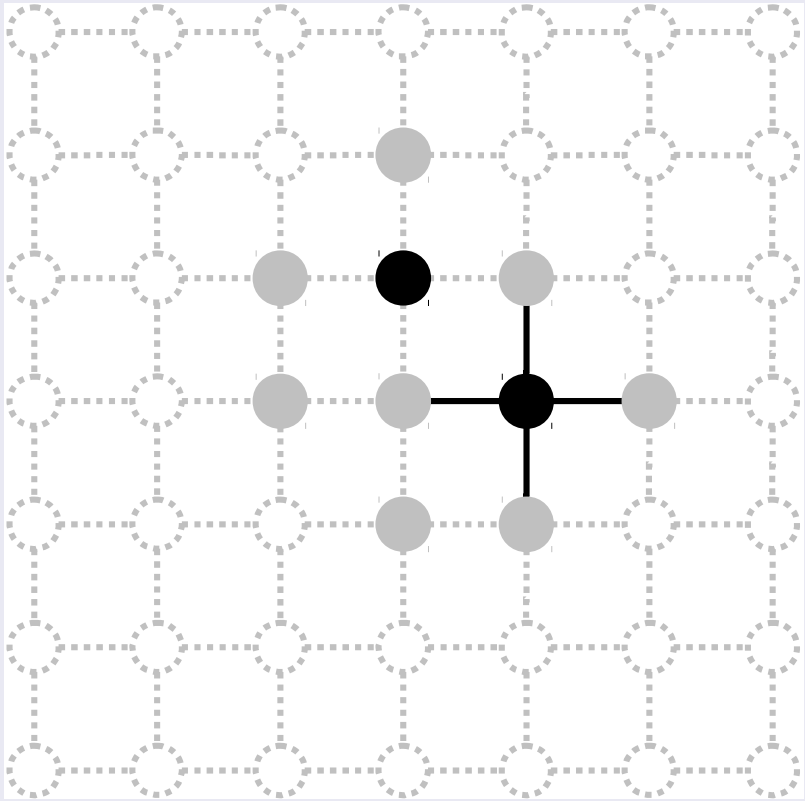
Search Algorithms

Tree-Search on a rectangular grid problem



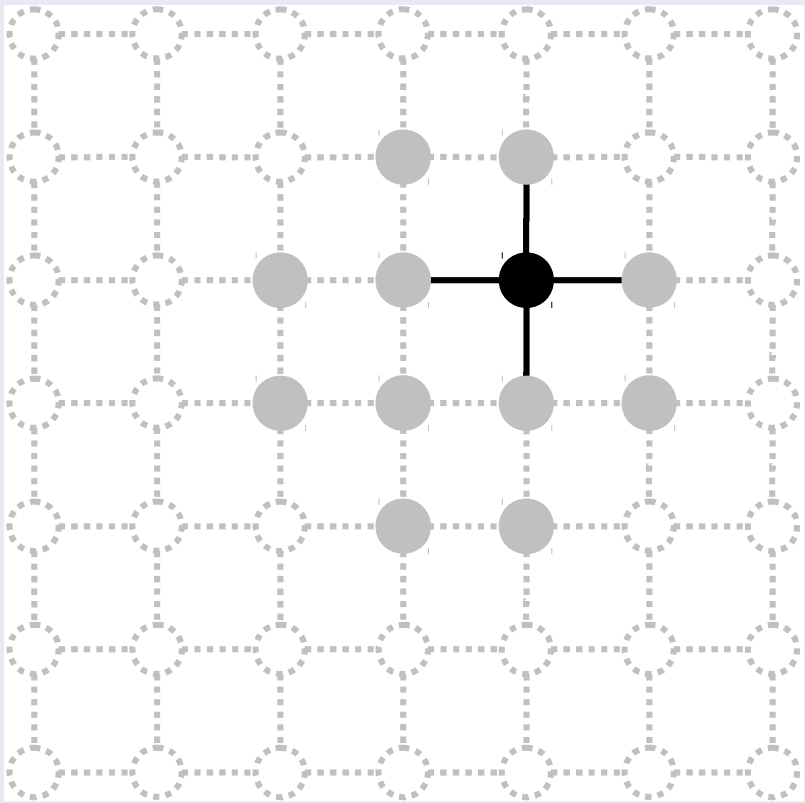
Search Algorithms

Tree-Search on a rectangular grid problem



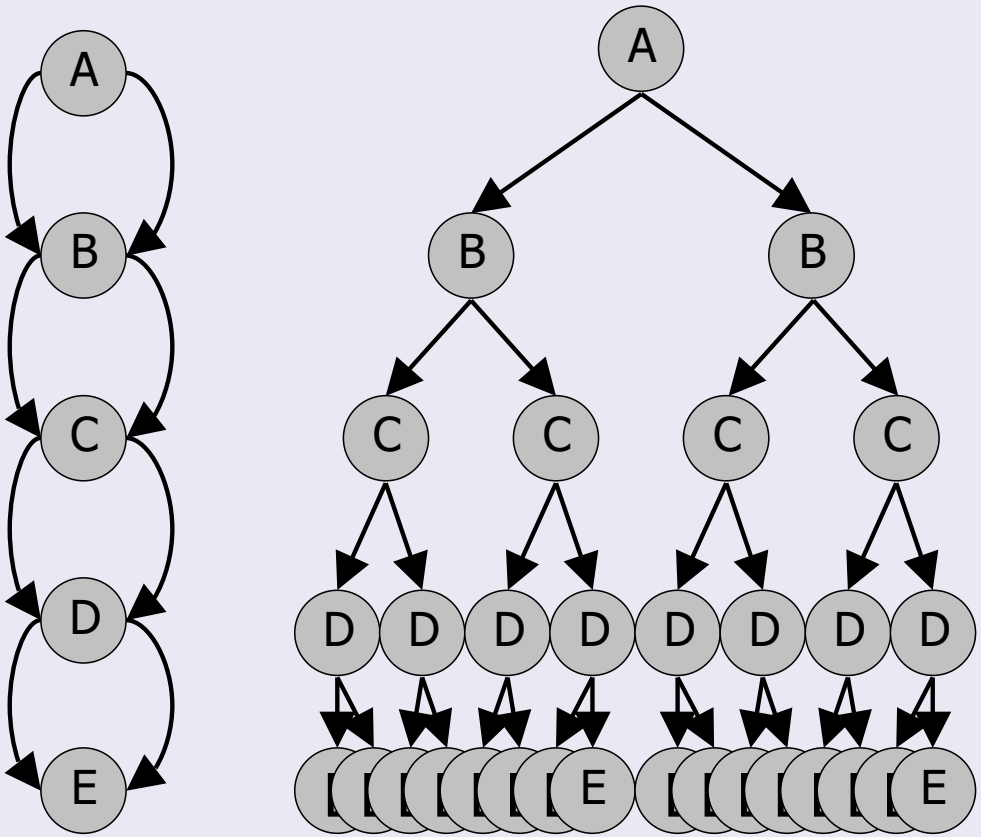
Search Algorithms

Tree-Search on a rectangular grid problem



Search Algorithms

Multiple paths to the same state



Search Algorithms

Repeated states

- Unavoidable in problems where:
  - Actions are reversible (*e.g.,* , rectangular grid problems).
  - Multiple paths to the same state are possible.

Search Algorithms

Repeated states

- Unavoidable in problems where:
  - Actions are reversible (*e.g.,* , rectangular grid problems).
  - Multiple paths to the same state are possible.
- Can greatly increase the number of nodes in a tree or even make a finite tree infinite.

Search Algorithms

Repeated states

- Unavoidable in problems where:
  - Actions are reversible (*e.g.,* , rectangular grid problems).
  - Multiple paths to the same state are possible.
- Can greatly increase the number of nodes in a tree or even make a finite tree infinite.
- Can be solved by augmenting the tree search algorithm with a data structure called the explored set (a.k.a., the closed list) to keep track of the explored states.

Search Algorithms

Repeated states

- Unavoidable in problems where:
  - Actions are reversible (*e.g.,* , rectangular grid problems).
  - Multiple paths to the same state are possible.
- Can greatly increase the number of nodes in a tree or even make a finite tree infinite.
- Can be solved by augmenting the tree search algorithm with a data structure called the explored set (a.k.a., the closed list) to keep track of the explored states.
- The tree search algorithm becomes a graph search algorithm.

Search Algorithms

General graph-search algorithm

GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  ...
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    ...
    expand the chosen node, adding the resulting node to the frontier
      ...
  end

Search Algorithms

General graph-search algorithm

GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    ...
    expand the chosen node, adding the resulting node to the frontier
      ...
  end

Search Algorithms

General graph-search algorithm

GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting node to the frontier
      ...
  end

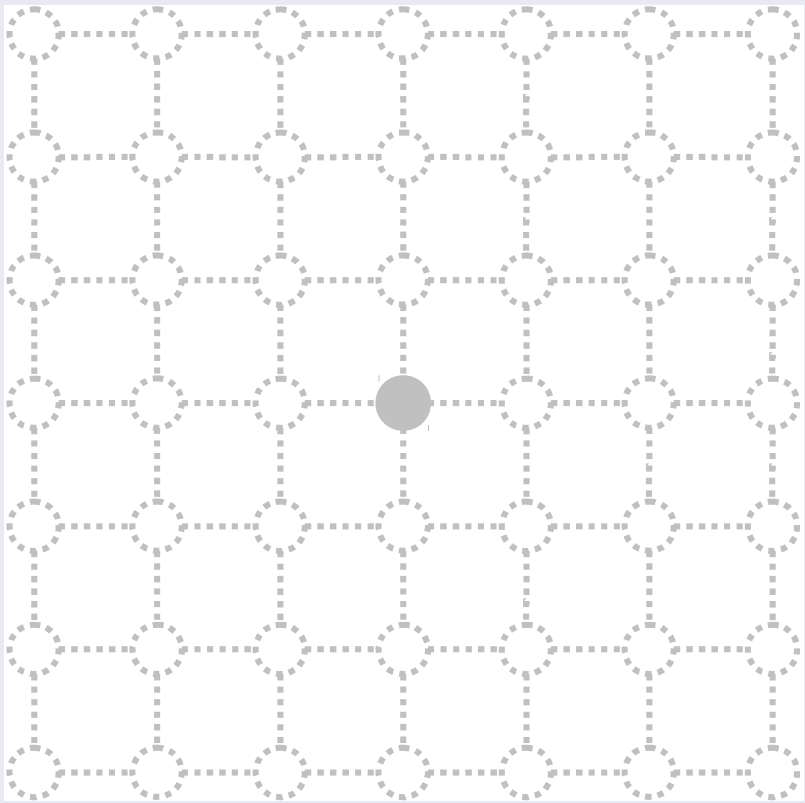
Search Algorithms

General graph-search algorithm

GRAPH-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of a *problem*
  initialize the explored set to be empty
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting node to the frontier
      only if not in the frontier and not in the explored set
  end

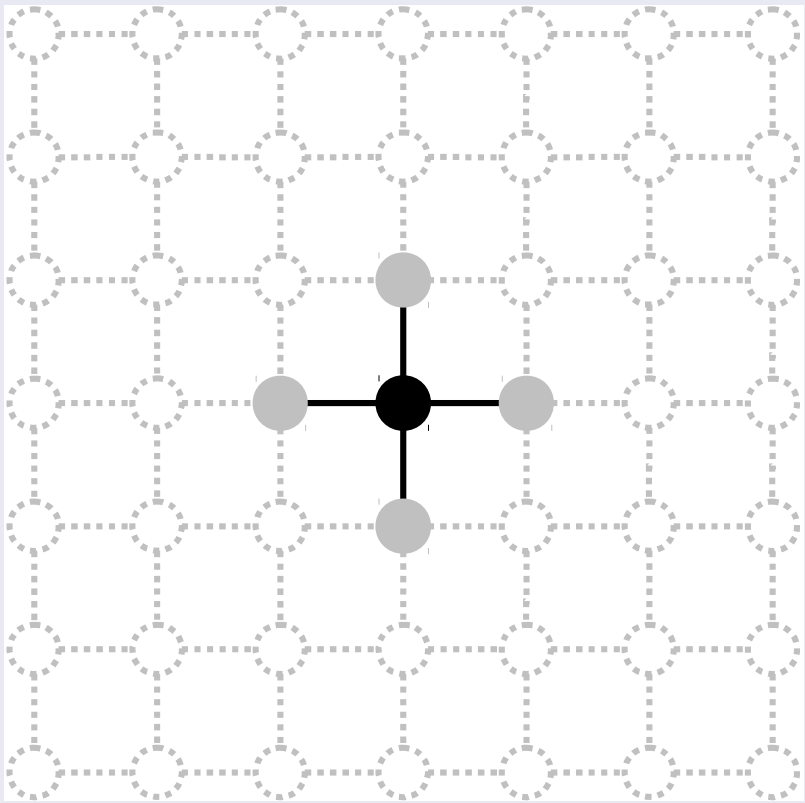
Search Algorithms

Graph-search on a rectangular grid problem



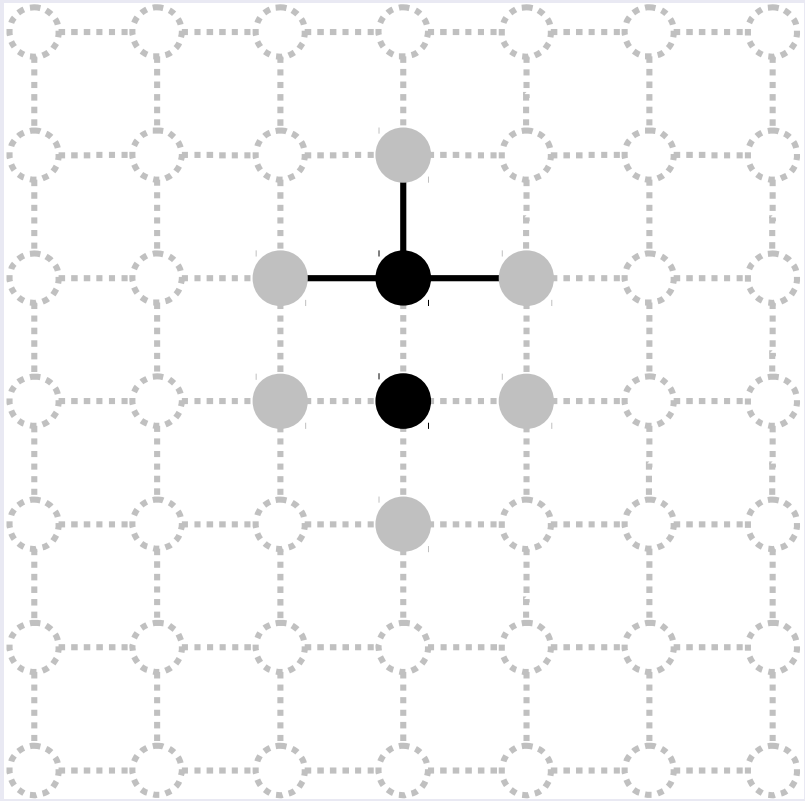
Search Algorithms

Graph-search on a rectangular grid problem



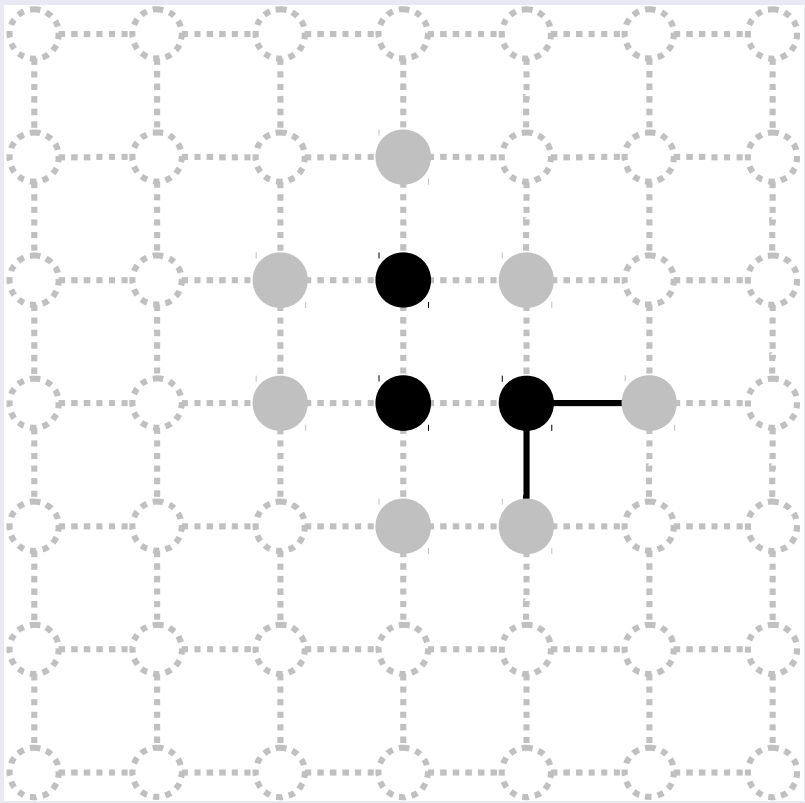
Search Algorithms

Graph-search on a rectangular grid problem



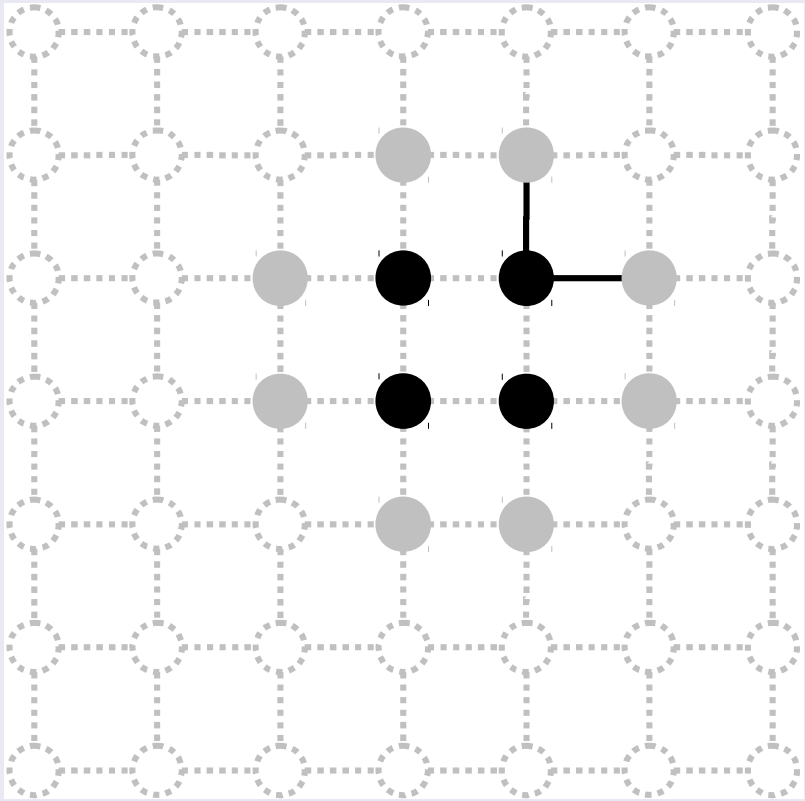
Search Algorithms

Graph-search on a rectangular grid problem



Search Algorithms

Graph-search on a rectangular grid problem



Search Algorithms

Properties of search algorithms

- The method in which a search algorithm traverses the tree is known as the ***search strategy***.

Search Algorithms

Properties of search algorithms

- The method in which a search algorithm traverses the tree is known as the ***search strategy***.
- The ***search strategy*** is defined by the method used by the algorithm for ***choosing*** the next node from the frontier (a.k.a., fringe or open list).

Search Algorithms

Properties of search algorithms

- **Completeness:**
  Is the algorithm guaranteed to find a goal node, if one exists?

Search Algorithms

Properties of search algorithms

- **Completeness:**
  Is the algorithm guaranteed to find a goal node, if one exists?
- **Optimality:**
  Is the algorithm guaranteed to find the best goal node, i.e. the one with the cheapest path cost?

Search Algorithms

Properties of search algorithms

- **Completeness:**
  Is the algorithm guaranteed to find a goal node, if one exists?
- **Optimality:**
  Is the algorithm guaranteed to find the best goal node, i.e. the one with the cheapest path cost?
- **Time complexity:**
  How many nodes are generated?

Search Algorithms

Properties of search algorithms

- **Completeness:**
  Is the algorithm guaranteed to find a goal node, if one exists?
- **Optimality:**
  Is the algorithm guaranteed to find the best goal node, i.e. the one with the cheapest path cost?
- **Time complexity:**
  How many nodes are generated?
- **Space complexity:**
  What is the maximum number of nodes stored in memory?

Search Algorithms

Properties of search algorithms

Time and space complexities are measured in terms of:

- Branching factor, ***b***,

Search Algorithms

Properties of search algorithms

Time and space complexities are measured in terms of:

- Branching factor, ***b***,
- Depth of least cost solution , ***d***,

Search Algorithms

Properties of search algorithms

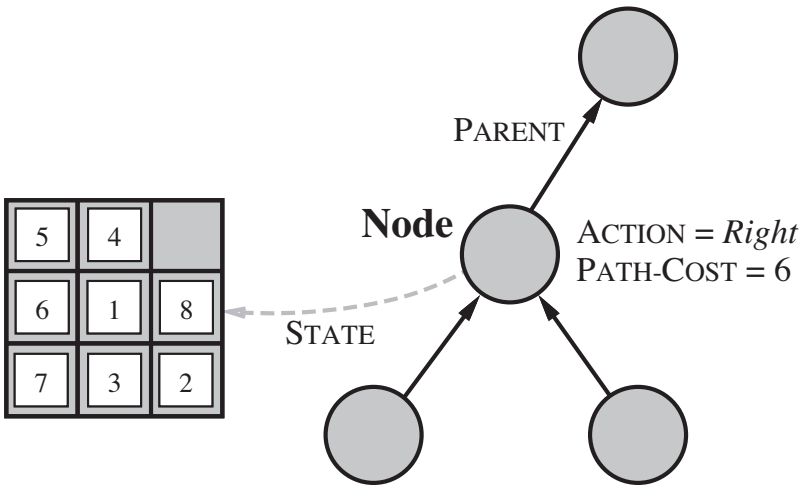
Time and space complexities are measured in terms of:

- Branching factor, ***b***,
- Depth of least cost solution , ***d***,
- Maximum depth of the search tree, ***m***,

Search Algorithms

Infrastructure for search algorithms

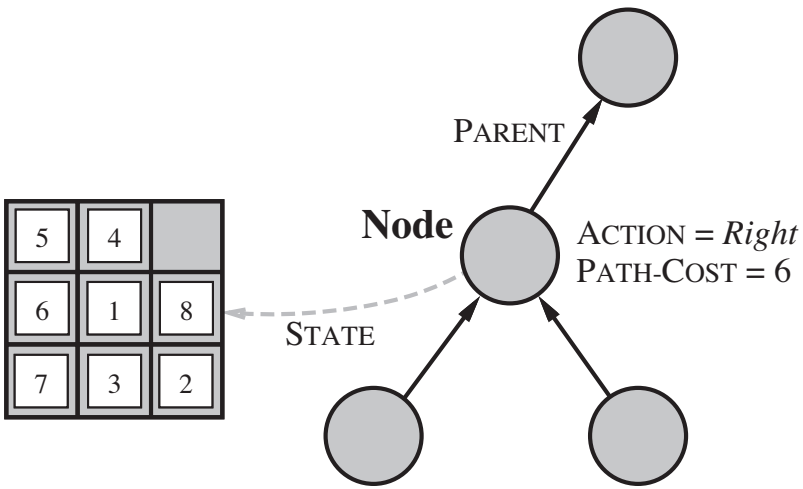
- Special data structures are needed to represent: the problem, the nodes, the frontier, and the explored states.



Search Algorithms

Infrastructure for search algorithms

- Special data structures are needed to represent: the problem, the nodes, the frontier, and the explored states.
- Each node *n* has the following components:
  - *n*.STATE, *n*.PARENT, *n*.ACTION, *n*.PATH-COST, *n*.CHILD-NODE(*problem*, *action*), and *n*.SOLUTION()



Search Algorithms

Infrastructure for search algorithms

- Each problem *p* has the following components:
  - *p*.INITIAL-STATE, *p*.ACTIONS(*state*), *p*.RESULT(*state*, *action*), *p*.GOAL-TEST(*state*), *p*.STEP-COST(*state*, *action*).

Search Algorithms

Infrastructure for search algorithms

- Each problem *p* has the following components:
  - *p*.INITIAL-STATE, *p*.ACTIONS(*state*), *p*.RESULT(*state*, *action*), *p*.GOAL-TEST(*state*), *p*.STEP-COST(*state*, *action*).
- Frontier *f* is represented as a (FIFO, LIFO, or priority) queue that has the following components:
  - *f*.EMPTY?(), *f*.INSERT(*element*), and *f*.POP()

Search Algorithms

Infrastructure for search algorithms

- Each problem *p* has the following components:
  - *p*.INITIAL-STATE, *p*.ACTIONS(*state*), *p*.RESULT(*state*, *action*), *p*.GOAL-TEST(*state*), *p*.STEP-COST(*state*, *action*).
- Frontier *f* is represented as a (FIFO, LIFO, or priority) queue that has the following components:
  - *f*.EMPTY?(), *f*.INSERT(*element*), and *f*.POP()
- Explored states are kept track of using a data structure that acts as a set.

Search Algorithms

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost),

Search Algorithms

Types of search algorithms

- **Uninformed Search:**
  Only has the information provided by the problem formulation (initial state, available actions, transition model, goal test, and step/path cost),

- **Informed Search:**
  Has additional information that allows it to judge the promise of an action, i.e. the estimated cost from a state to a goal.

Outline

Uninformed Search

Uninformed Search algorithms

- Breadth-first search (BFS),
- Uniform-cost search,
- Depth-first search (DFS),
- Depth-limited search,
- Iterative deepening search (IDS),

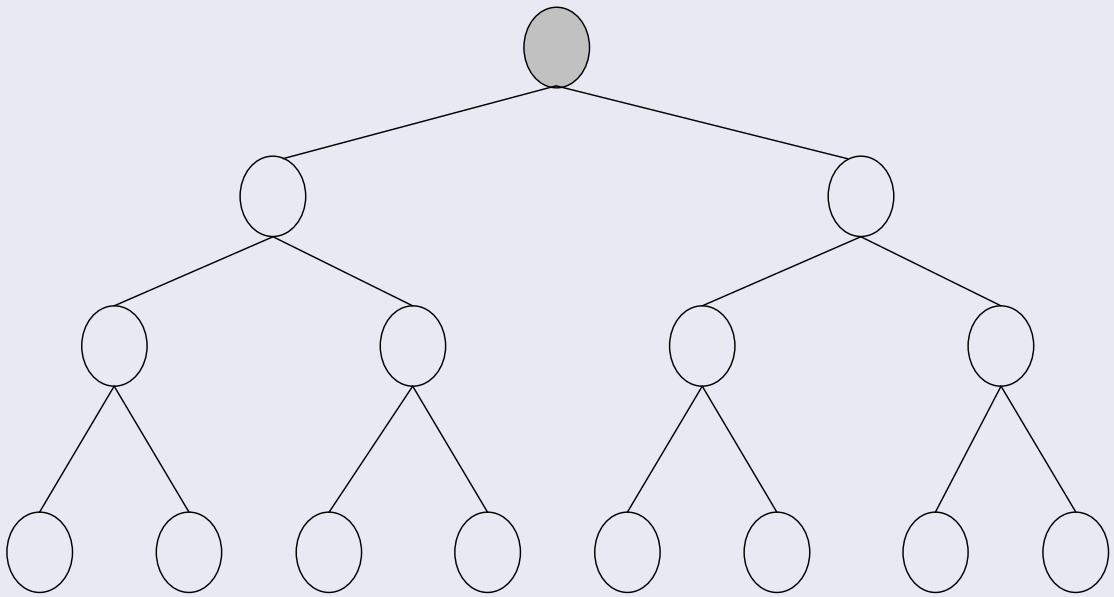
Uninformed Search

Breadth-first search

- The frontier is implemented as a FIFO queue,
- The tree is traversed on a level-by-level basis.

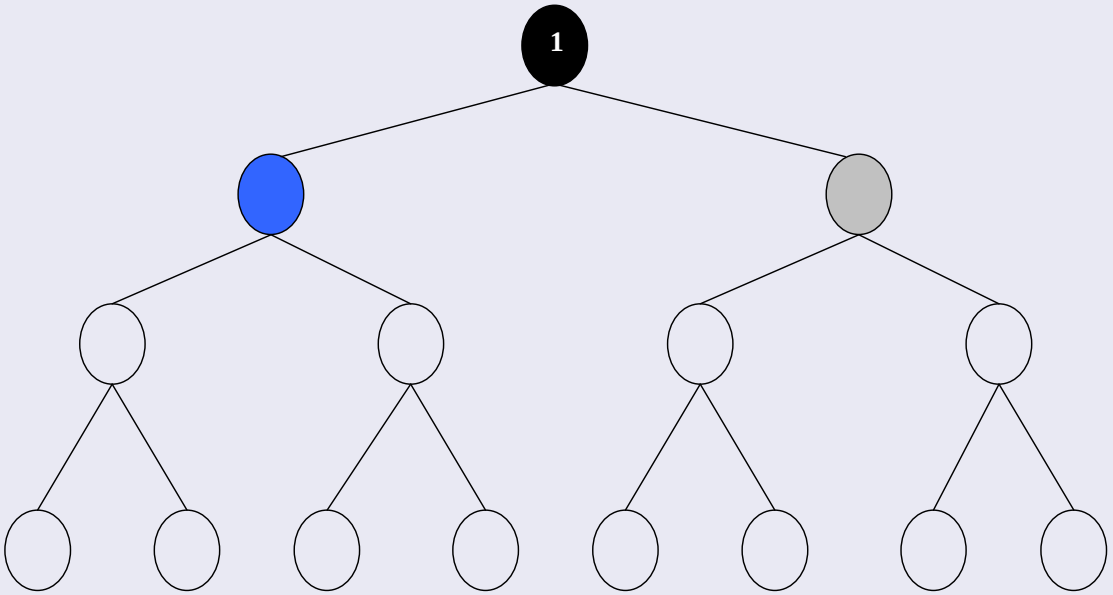
Uninformed Search

Breadth-first search



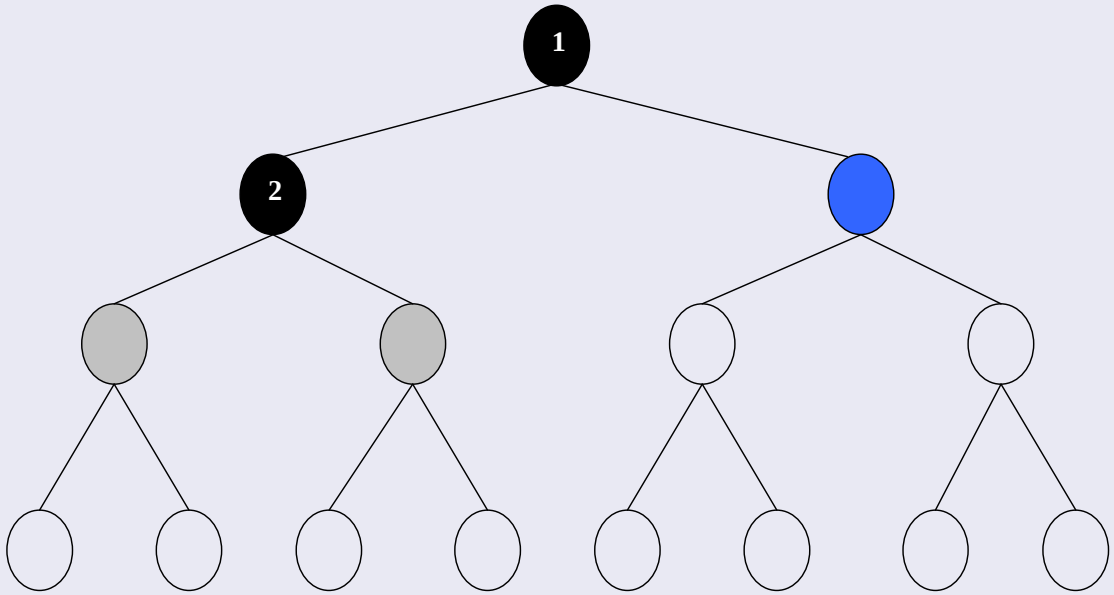
Uninformed Search

Breadth-first search



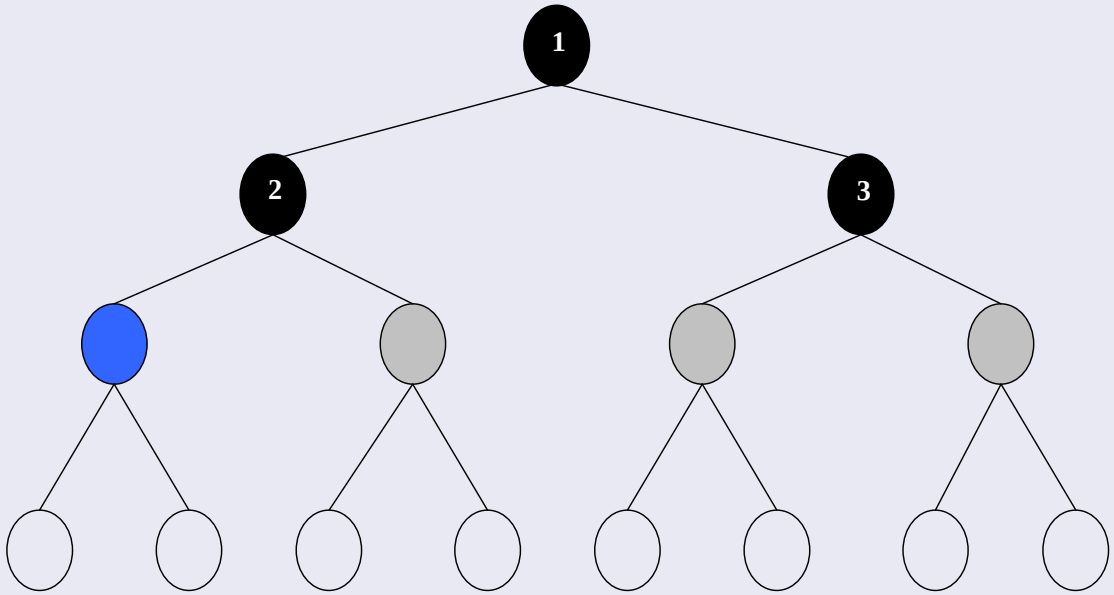
Uninformed Search

Breadth-first search



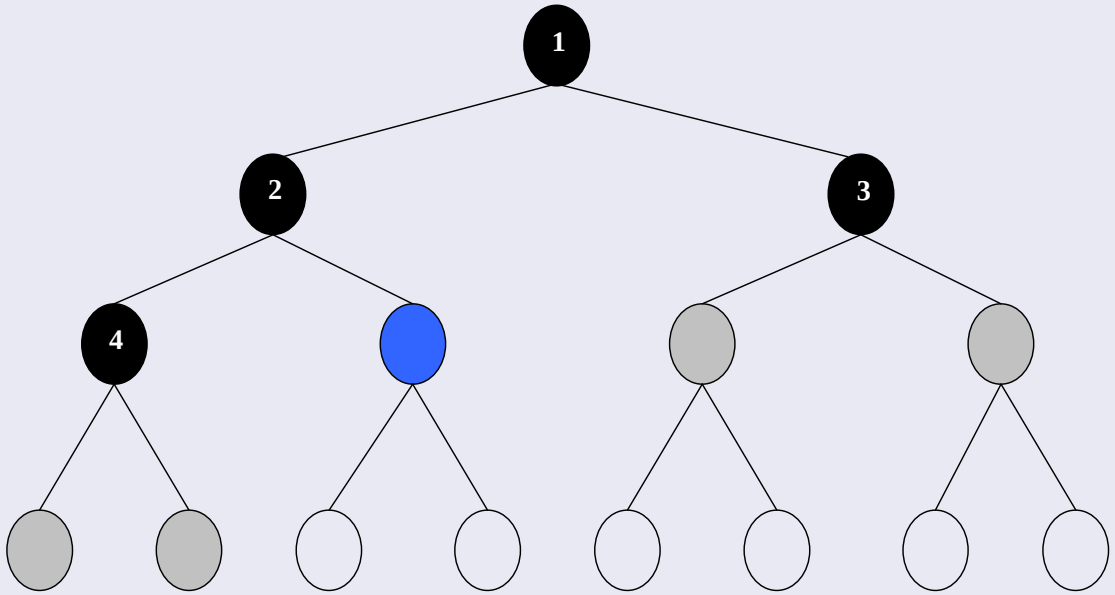
Uninformed Search

Breadth-first search



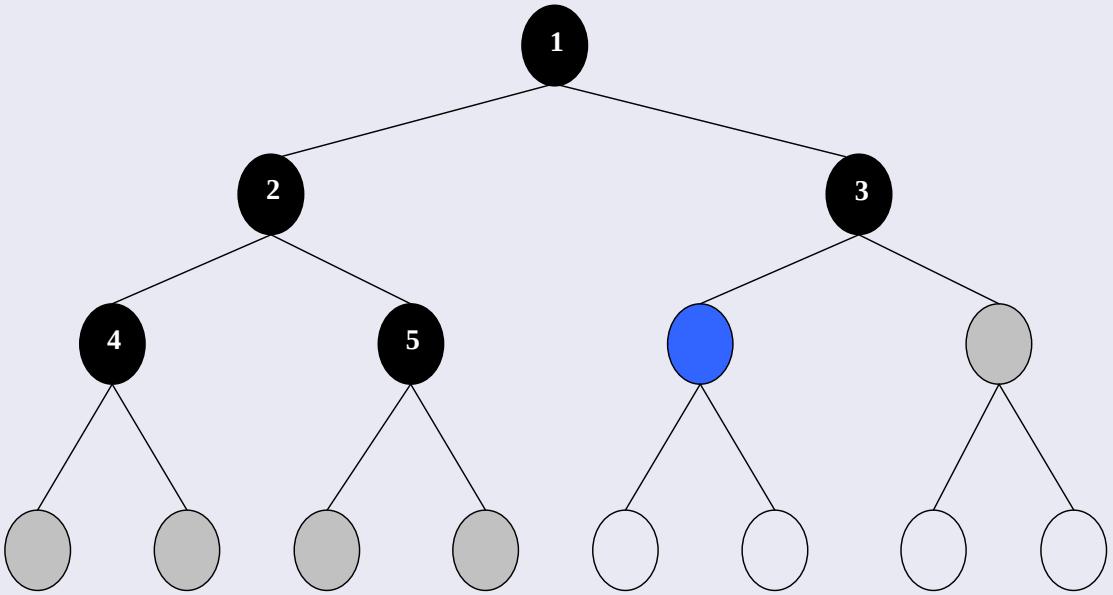
Uninformed Search

Breadth-first search



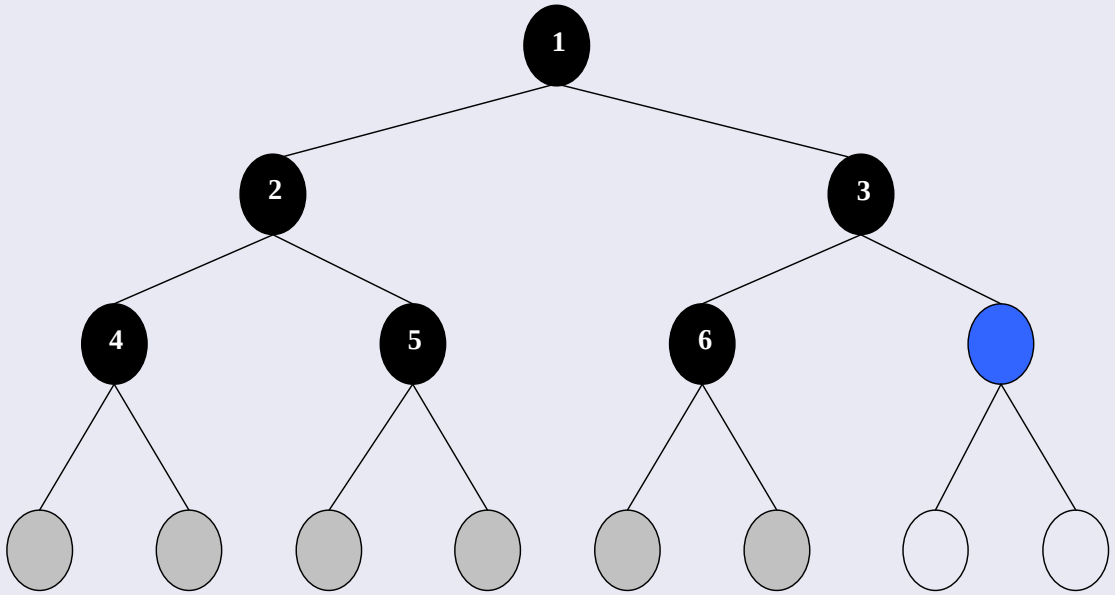
Uninformed Search

Breadth-first search



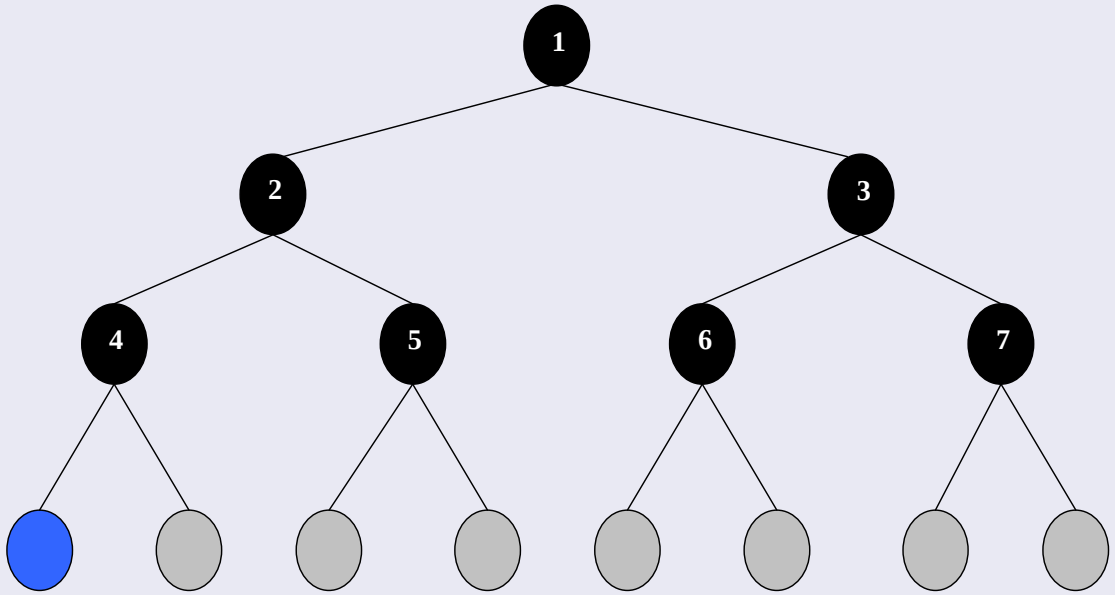
Uninformed Search

Breadth-first search



Uninformed Search

Breadth-first search



Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure

node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0

Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()

Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
  frontier ← a **FIFO queue** with *node* as the only element

Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
  frontier ← a **FIFO queue** with *node* as the only element
  loop do

Search Algorithms

Breadth-first search (tree version)

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE=*problem*.INITIAL-STATE, PATH-COST=0
  if *problem*.GOAL-TEST(*node*.STATE) **then return** *node*.SOLUTION()
  frontier ← a **FIFO queue** with *node* as the only element
  loop do
    if *frontier*.EMPTY?() **then return** failure

Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()          /* choose shallowest node in frontier */
```

Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
    if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
    frontier ← a FIFO queue with node as the only element
    loop do
      if frontier.EMPTY?() then return failure
      node ← frontier.POP()              /* choose shallowest node in frontier */
      for each action in problem.ACTIONS(node.STATE) do
```

Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()              /* choose shallowest node in frontier */
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem,varaction)
```

Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()              /* choose shallowest node in frontier */
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if problem.GOAL-TEST(child.STATE) then return child.SOLUTION()
```

Search Algorithms

Breadth-first search (tree version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()          /* choose shallowest node in frontier */
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if problem.GOAL-TEST(child.STATE) then return child.SOLUTION()
      frontier.INSERT(child)
```

Search Algorithms

Breadth-first search (graph version)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE=problem.INITIAL-STATE, PATH-COST=0
  if problem.GOAL-TEST(node.STATE) then return node.SOLUTION()
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if frontier.EMPTY?() then return failure
    node ← frontier.POP()              /* choose shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← node.CHILD-NODE(problem, action)
      if child.STATE is not in explored and not in frontier then
        if problem.GOAL-TEST(child.STATE) then return child.SOLUTION()
        frontier.INSERT(child)
```

Uninformed Search

Breadth-first search

BFS properties:

- Complete (if b is finite).

Uninformed Search

Breadth-first search

BFS properties:

- Complete (if b is finite).
- Optimal, if path cost is equal to depth:
  - Guaranteed to return the shallowest goal (depth d).

Uninformed Search

Breadth-first search

BFS properties:

- Complete (if b is finite).
- Optimal, if path cost is equal to depth:
  - Guaranteed to return the shallowest goal (depth d).
- Time complexity = $O(b^d)$.

Uninformed Search

Breadth-first search

BFS properties:

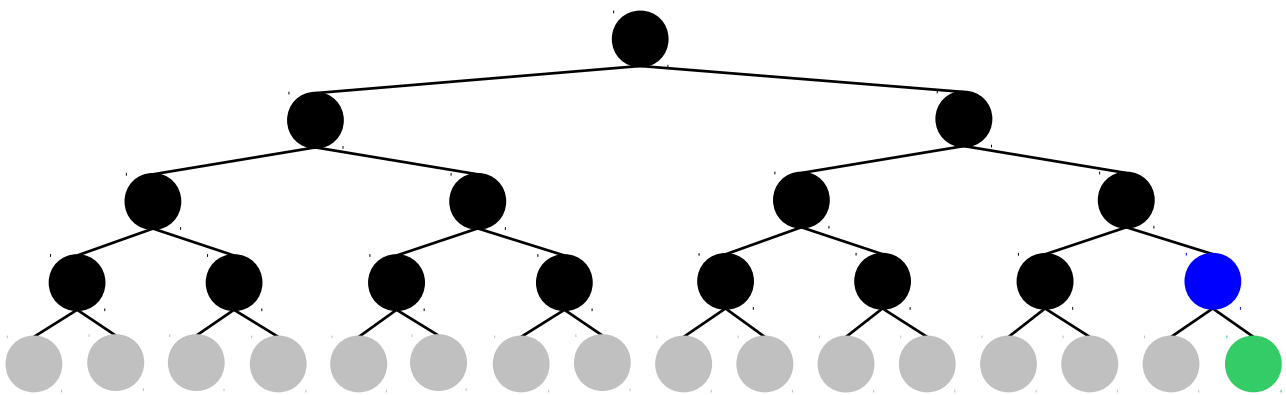
- Complete (if b is finite).
- Optimal, if path cost is equal to depth:
  - Guaranteed to return the shallowest goal (depth d).
- Time complexity = $O(b^d)$.
- Space complexity = $O(b^d)$.

Uninformed Search

Breadth-first search

BFS properties:

- Upper-bound case: when the goal node is the last node at depth d:

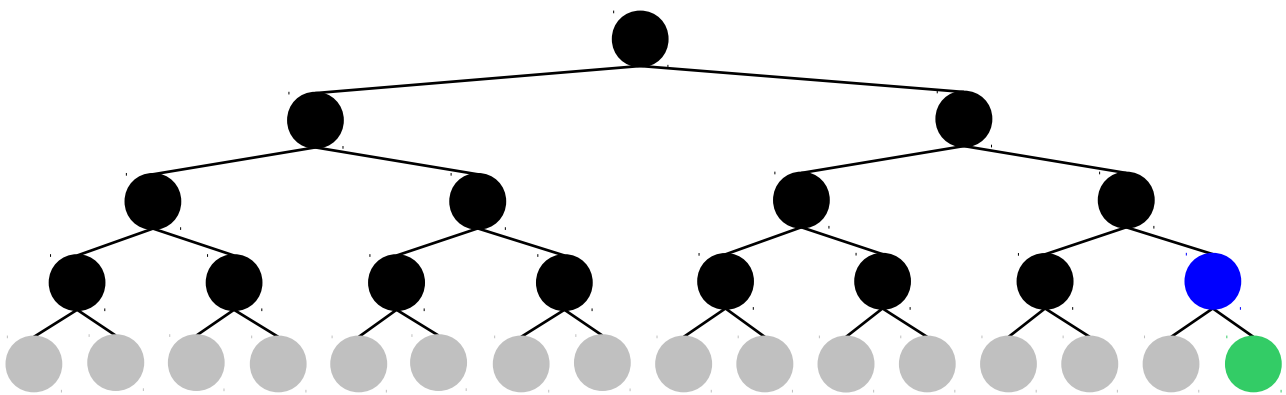


Uninformed Search

Breadth-first search

BFS properties:

- Upper-bound case: when the goal node is the last node at depth d:
  - Goal is detected once goal node is generated.

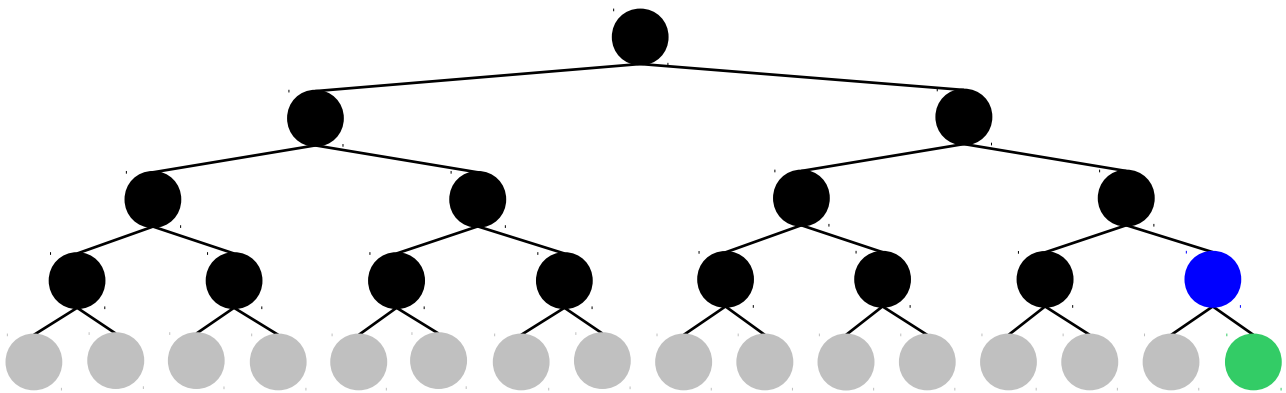


Uninformed Search

Breadth-first search

BFS properties:

- Upper-bound case: when the goal node is the last node at depth d:
  - Goal is detected once goal node is generated.
  - Number of nodes generated:
    $b + b^2 + b^3 + \ldots + b^d = O(b^d)$.

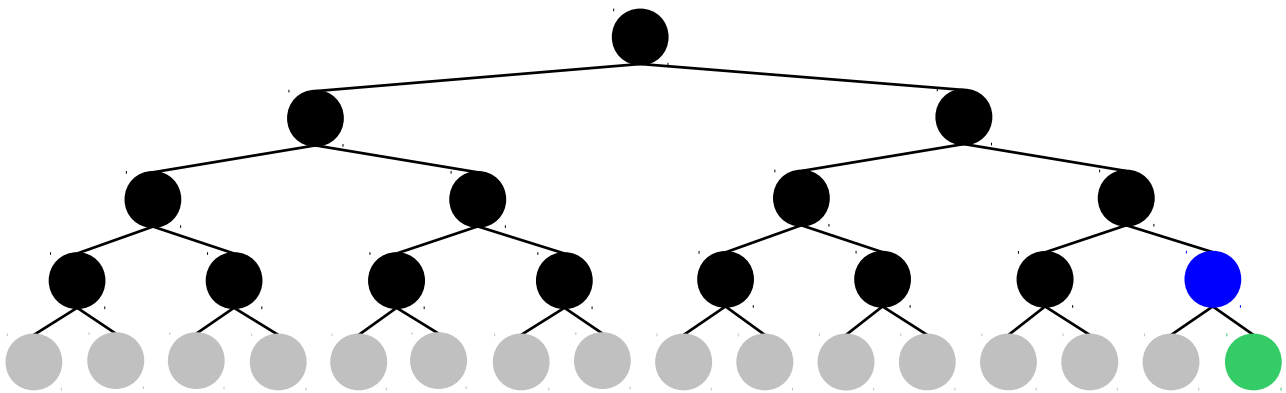


Uninformed Search

Breadth-first search

BFS properties:

- Upper-bound case: when the goal node is the last node at depth d:
  - Goal is detected once goal node is generated.
  - Number of nodes generated:
    $b + b^2 + b^3 + ... + b^d = O(b^d)$.
  - Space and time complexity: all generated nodes.



Outline

Requirements

What do I need from you

When given a certain problem you should be able to:

- Formulate the problem:

Requirements

What do I need from you

When given a certain problem you should be able to:

- Formulate the problem:
  - What is a state?
  - What is the initial state?
  - What actions could be applied to a state?
  - What is the transition model?
  - What is the goal test?
  - What is the step cost as well as the path cost?

Requirements

What do I need from you

When given a certain problem you should be able to:

- Formulate the problem:
  - What is a state?
  - What is the initial state?
  - What actions could be applied to a state?
  - What is the transition model?
  - What is the goal test?
  - What is the step cost as well as the path cost?
- Build the search tree up to a given depth.

Requirements

What do I need from you

When given a certain problem you should be able to:

- Formulate the problem:
  - What is a state?
  - What is the initial state?
  - What actions could be applied to a state?
  - What is the transition model?
  - What is the goal test?
  - What is the step cost as well as the path cost?
- Build the search tree up to a given depth.
- Traverse the search tree according to a given strategy.

Requirements

What do I need from you

When given a certain problem you should be able to:

- Formulate the problem:
  - What is a state?
  - What is the initial state?
  - What actions could be applied to a state?
  - What is the transition model?
  - What is the goal test?
  - What is the step cost as well as the path cost?
- Build the search tree up to a given depth.
- Traverse the search tree according to a given strategy.
- Answer descriptive questions.

Reading Material

Which parts of the textbook are covered

- Russell-Norvig, Chapters 3:
  - Pages 64 - 83.