CSE 411: Artificial Intelligence (Elective Course #6)

400 Level, Mechatronics Engineering
2nd Term 2016/2017, Lecture #3

Hazem Shehata

Dept. of Computer & Systems Engineering
Zagazig University

March 13th, 2017

Credits to Dr. Mohamed El Abd for the slides

Adminstrivia

Notes

- Assignment #1:
  - Released today.
  - Due: Monday, March 20, 2017
  - Formulate two problems in Python.
  - Email Python files and hand in a printout!

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Adminstrivia

Notes

- Assignment #1:
  - Released today.
  - Due: Monday, March 20, 2017
  - Formulate two problems in Python.
  - Email Python files and hand in a printout!
- ”Requirements & Reading Material” section:
  - Page numbers are based on 3rd US edition of AIMA.
  - Full TOC: http://aima.cs.berkeley.edu/contents.html.

Course Info:

- Website: http://hshehata.github.io/courses/zu/cse411/
- Office hours: Sunday 11:30am - 12:30pm

Outline

Outline

Problem solving by searching

Introduction

- One approach to solve any problem is to formulate it in ***STATE-SPACE*** representation.

Problem solving by searching

Introduction

- One approach to solve any problem is to formulate it in ***STATE-SPACE*** representation.
- The problem is well-defined by defining:
  - An initial state.
  - A set of actions.
  - A transition model.
  - A goal test.
  - A concept of cost: step and path cost.

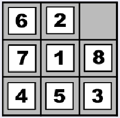
Problem solving by searching

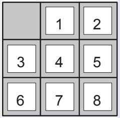
Introduction

- One approach to solve any problem is to formulate it in ***STATE-SPACE*** representation.
- The problem is well-defined by defining:
  - An initial state.
  - A set of actions.
  - A transition model.
  - A goal test.
  - A concept of cost: step and path cost.
- This is used to build a search tree to be traversed by a tree-search (or graph-search) algorithm.

Problem solving by searching

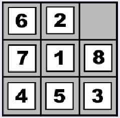
A puzzle example

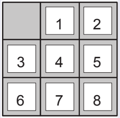




Problem solving by searching

A puzzle example

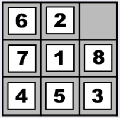


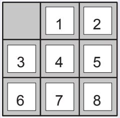


- **Initial state:** any arrangement of tiles.

Problem solving by searching

A puzzle example

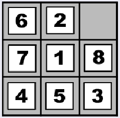


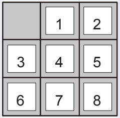


- **Initial state:** any arrangement of tiles.
- **Actions:** move blank left, right, up or down, provided it does not get out of the game.

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

6	2	
7	1	8
4	5	3

	1	2
3	4	5
6	7	8

- **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

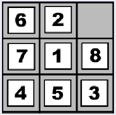
6	2	
7	1	8
4	5	3

	1	2
3	4	5
6	7	8

- **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.

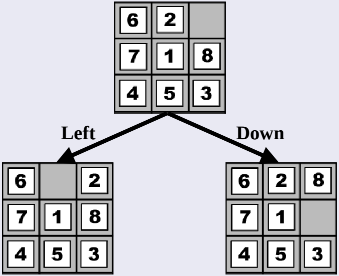
Problem solving by searching

A puzzle search tree



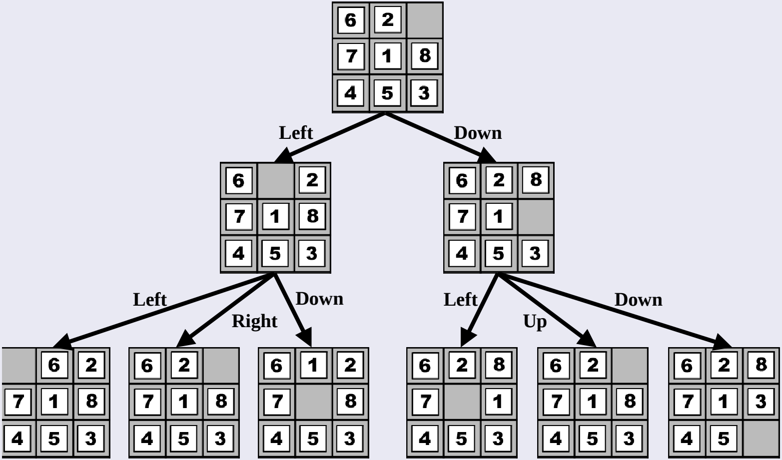
Problem solving by searching

A puzzle search tree



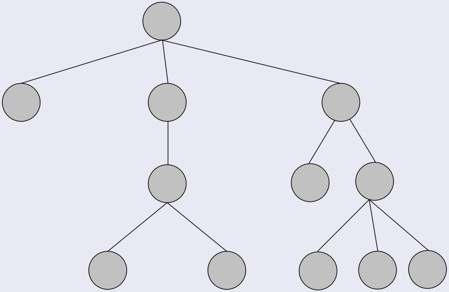
Problem solving by searching

A puzzle search tree



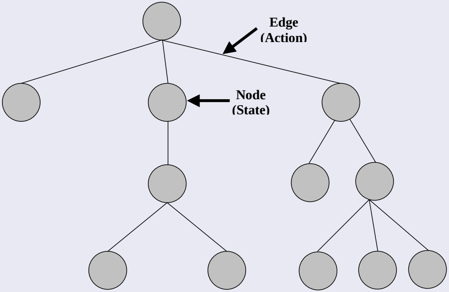
Problem solving by searching

A search tree



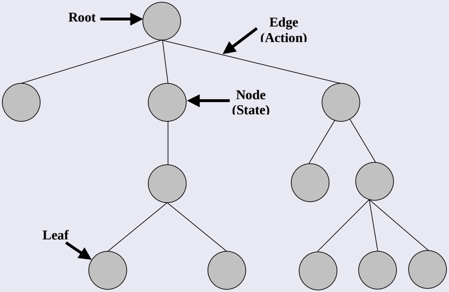
Problem solving by searching

A search tree



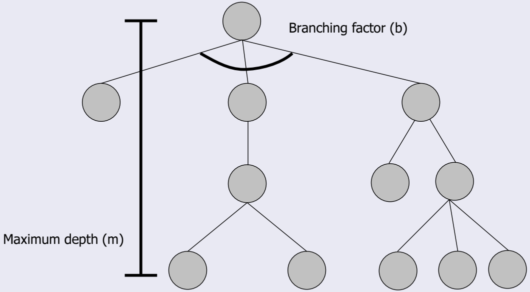
Problem solving by searching

A search tree



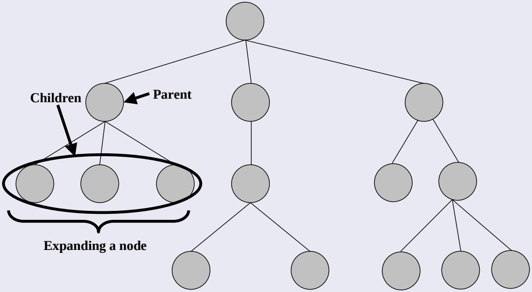
Problem solving by searching

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

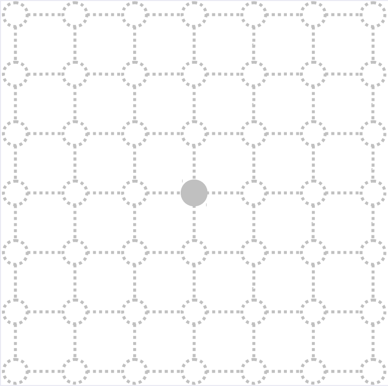
Search Algorithms

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
```

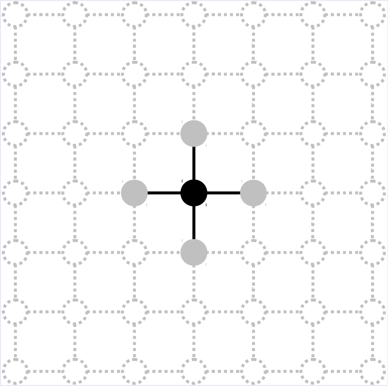
Search Algorithms

Tree-Search on a rectangular grid problem



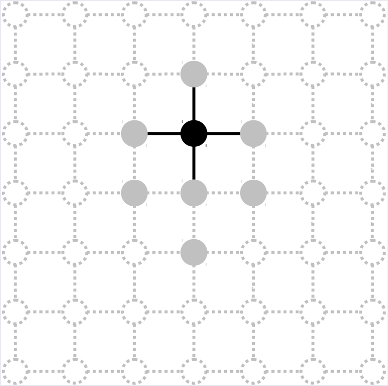
Search Algorithms

Tree-Search on a rectangular grid problem



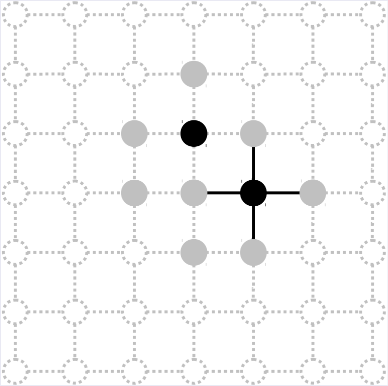
Search Algorithms

Tree-Search on a rectangular grid problem



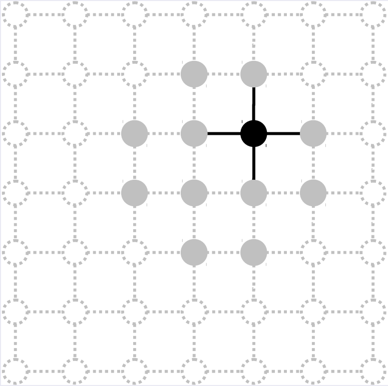
Search Algorithms

Tree-Search on a rectangular grid problem



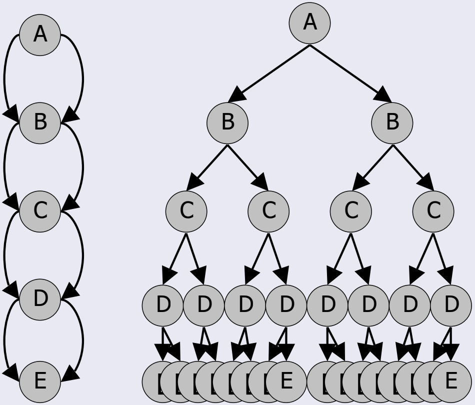
Search Algorithms

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

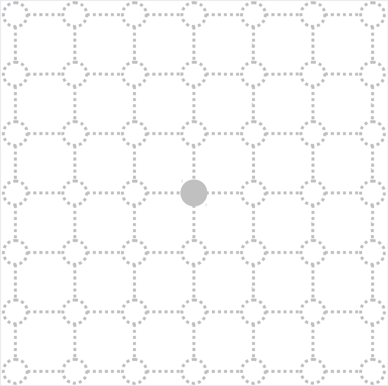
Search Algorithms

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

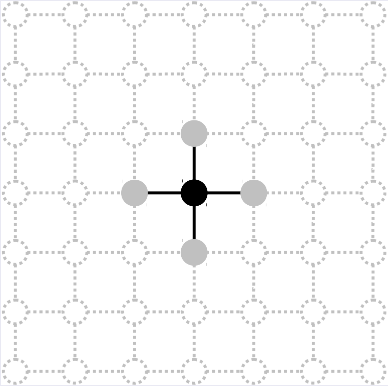
Search Algorithms

Graph-search on a rectangular grid problem



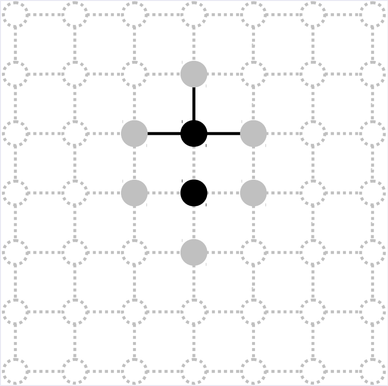
Search Algorithms

Graph-search on a rectangular grid problem



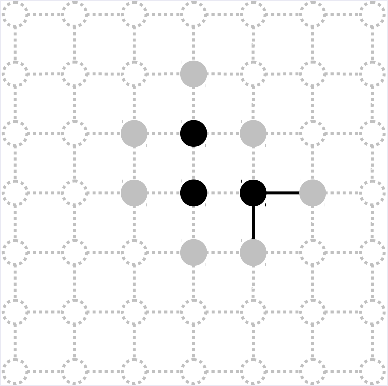
Search Algorithms

Graph-search on a rectangular grid problem



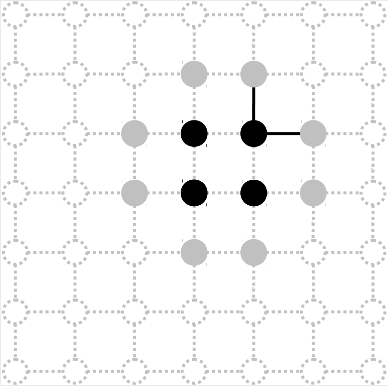
Search Algorithms

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

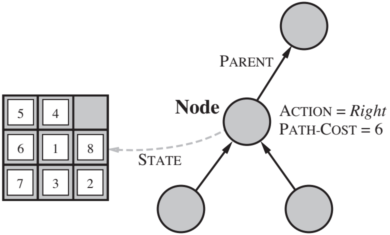
Time and space complexities are measured in terms of:

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

Search Algorithms

Infrastructure for search algorithms

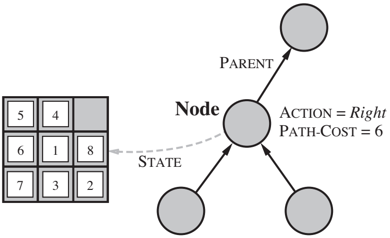
- Special data structures are needed to represent: the problem, the nodes, the frontier, and the explored states.



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

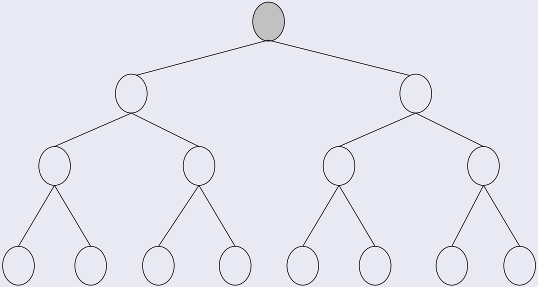
Uninformed Search

Breadth-first search

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

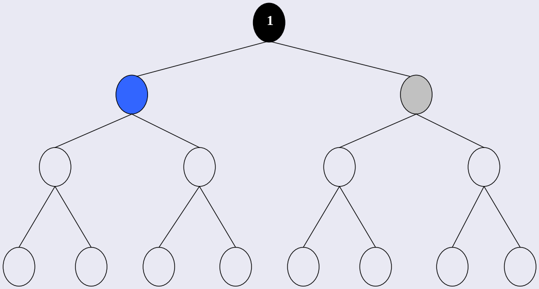
Uninformed Search

Breadth-first search



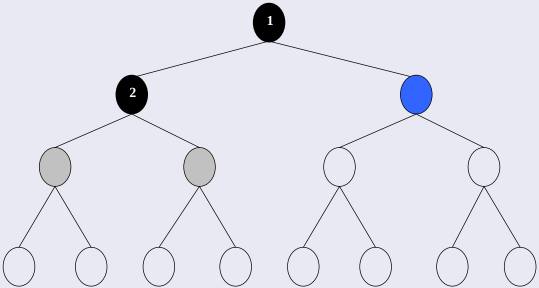
Uninformed Search

Breadth-first search



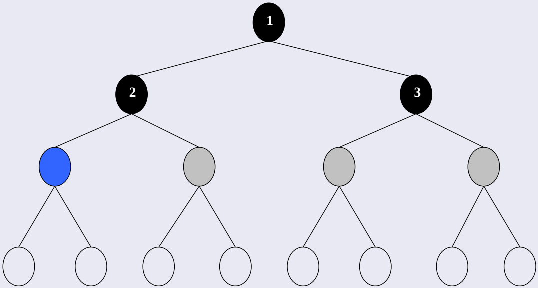
Uninformed Search

Breadth-first search



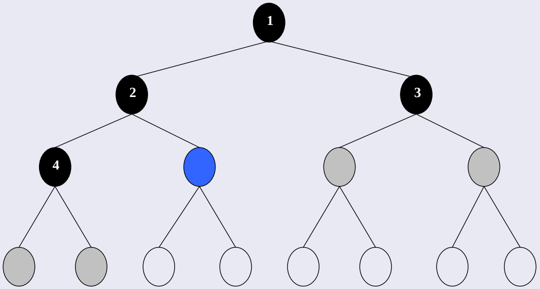
Uninformed Search

Breadth-first search



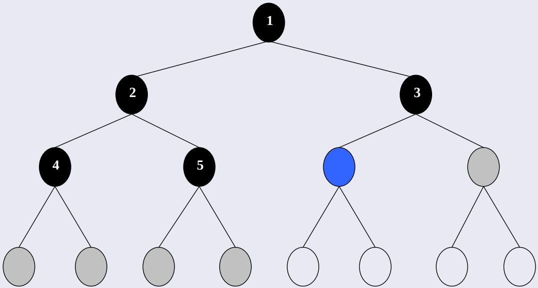
Uninformed Search

Breadth-first search



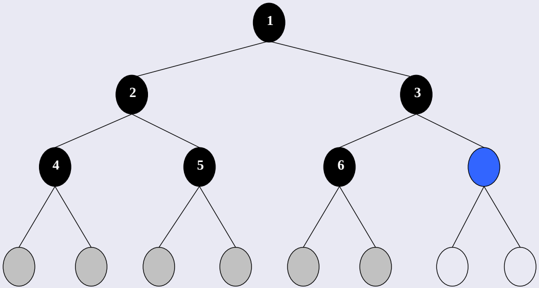
Uninformed Search

Breadth-first search



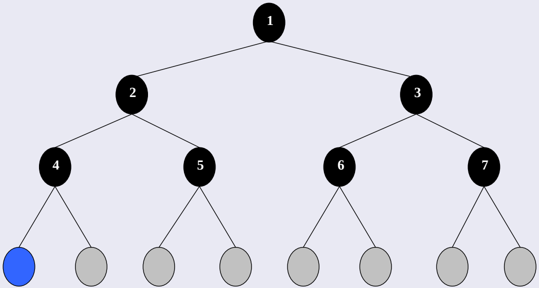
Uninformed Search

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, action)
```

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:

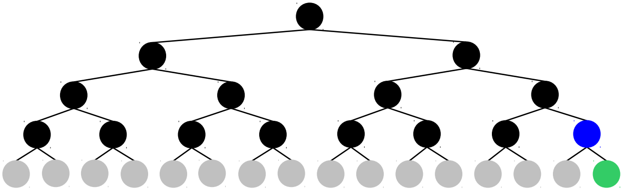
- Complete (if b is finite).
- Optimal, if path cost is equal to depth:
  - Guaranteed to return the shallowest goal (depth d).
- Time complexity = $O(b^d)$.
- Space complexity = $O(b^d)$.

Uninformed Search

Breadth-first search

BFS properties:

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

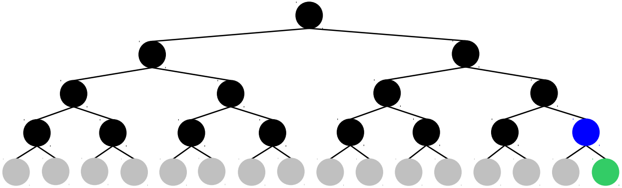


Uninformed Search

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.

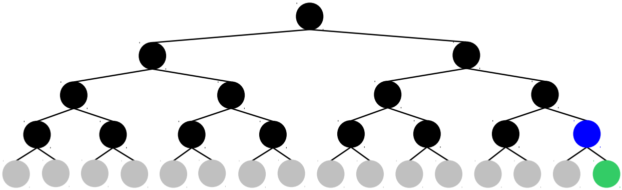


Uninformed Search

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: $b + b^2 + b^3 + ... + b^d = O(b^d)$.

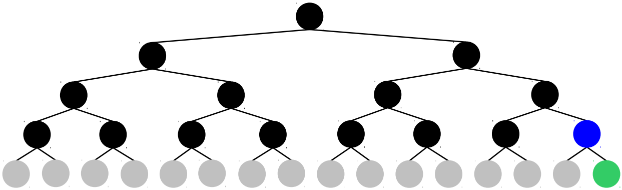


Uninformed Search

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.



Uninformed Search

Depth-first search

- Algorithm is similar to BFS, except that the frontier is implemented as a LIFO queue (*i.e.*, stack).

Uninformed Search

Depth-first search

- Algorithm is similar to BFS, except that the frontier is implemented as a LIFO queue (*i.e.*, stack).
- Algorithm always expands deepest unexpanded node in current frontier.

Uninformed Search

Depth-first search

- Algorithm is similar to BFS, except that the frontier is implemented as a LIFO queue (*i.e.*, stack).
- Algorithm always expands deepest unexpanded node in current frontier.
- Once a leaf is reached, the search backs up (*i.e.*, backtracks) to the next deepest node that still has unexplored successors.

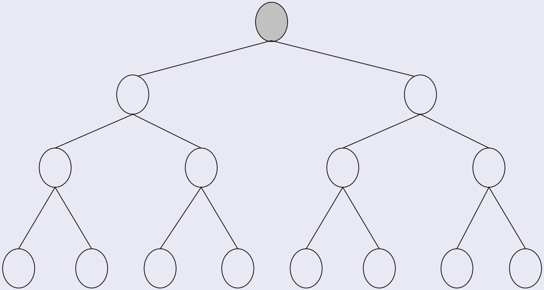
Uninformed Search

Depth-first search

- Algorithm is similar to BFS, except that the frontier is implemented as a LIFO queue (*i.e.*, stack).
- Algorithm always expands deepest unexpanded node in current frontier.
- Once a leaf is reached, the search backs up (*i.e.*, backtracks) to the next deepest node that still has unexplored successors.
- Search continues until reaching the goal or no frontier nodes are left for expansion.

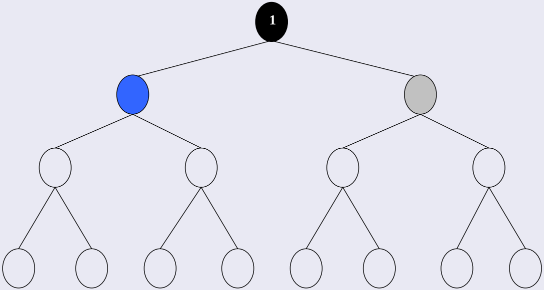
Uninformed Search

Depth-first search



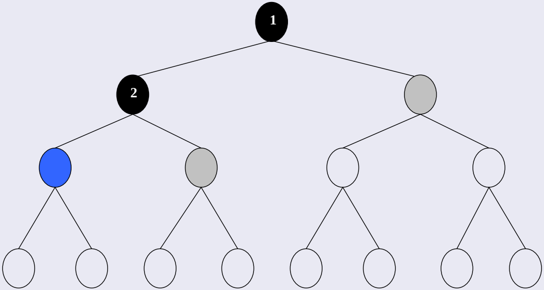
Uninformed Search

Depth-first search



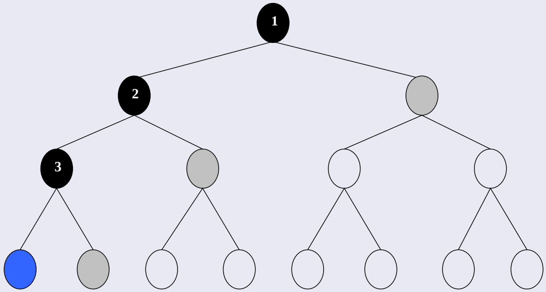
Uninformed Search

Depth-first search



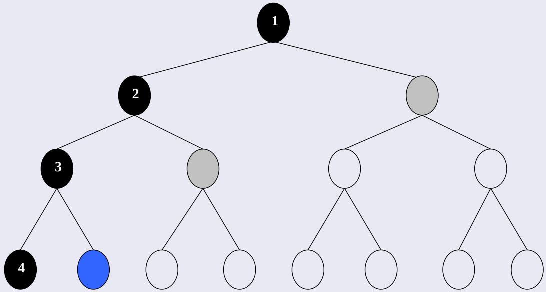
Uninformed Search

Depth-first search



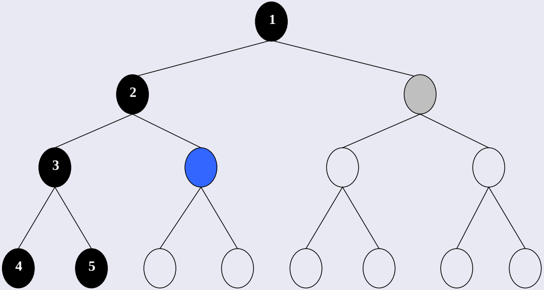
Uninformed Search

Depth-first search



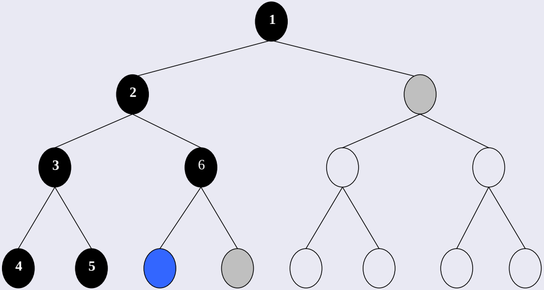
Uninformed Search

Depth-first search



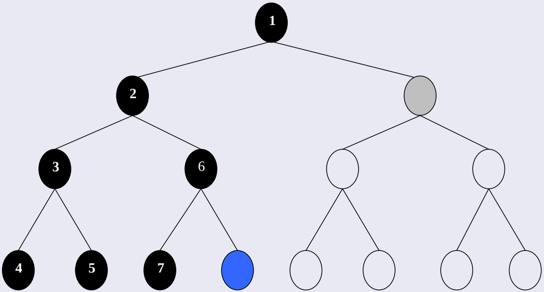
Uninformed Search

Depth-first search



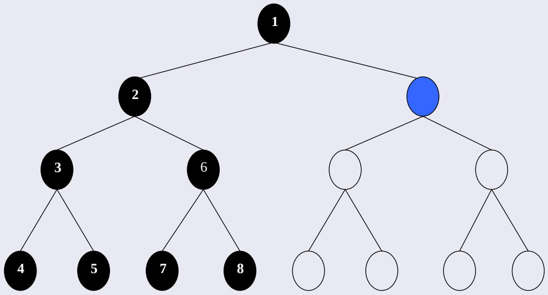
Uninformed Search

Depth-first search



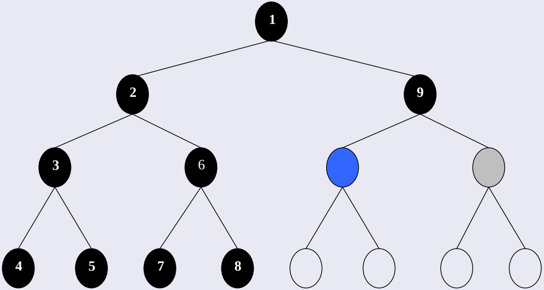
Uninformed Search

Depth-first search



Uninformed Search

Depth-first search



Uninformed Search

Depth-first search

DFS properties:

- Not complete (tree version).

Uninformed Search

Depth-first search

DFS properties:

- Not complete (tree version).
- Not Optimal.

Uninformed Search

Depth-first search

DFS properties:

- Not complete (tree version).
- Not Optimal.
- Time complexity = $O(b^m)$.

Uninformed Search

Depth-first search

DFS properties:

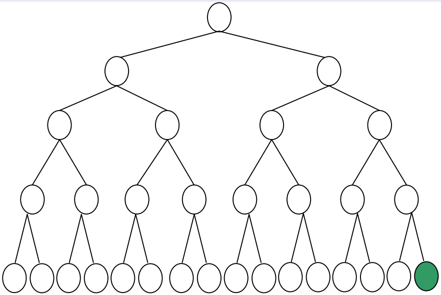
- Not complete (tree version).
- Not Optimal.
- Time complexity = $O(b^m)$.
- Space complexity = $O(bm)$.

Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for time: goal is last node of last branch:

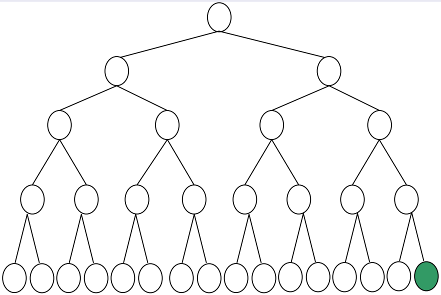


Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for time: goal is last node of last branch:
  - Number of nodes generated: b nodes for each node of m levels (entire tree).

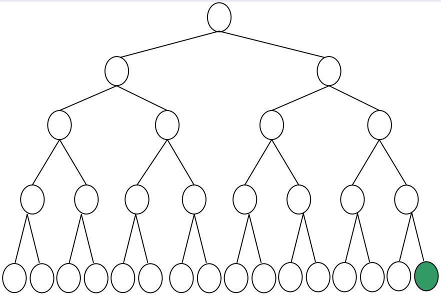


Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for time: goal is last node of last branch:
    - Number of nodes generated: b nodes for each node of m levels (entire tree).
    - Time complexity: all generated nodes $O(b^m)$.

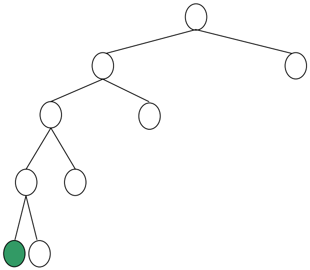


Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for space: goal is last node of first branch:

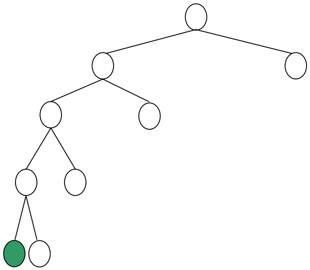


Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for space: goal is last node of first branch:
  - Number of generated nodes: b nodes at each of m levels.

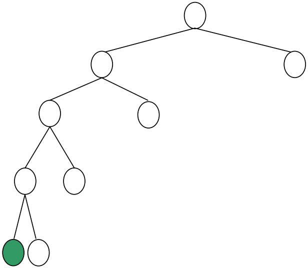


Uninformed Search

Depth-first search

DFS properties:

- Upper-bound case for space: goal is last node of first branch:
  - Number of generated nodes: b nodes at each of m levels.
  - Space complexity: all generated nodes = $O(bm)$.



Outline

Requirements

What do I need from you

- When given a certain problem you should be able to:

Requirements

What do I need from you

- When given a certain problem you should be able to:
  - 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:
  - 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:
  - 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 75 - 83.
  - Pages 85 - 87.