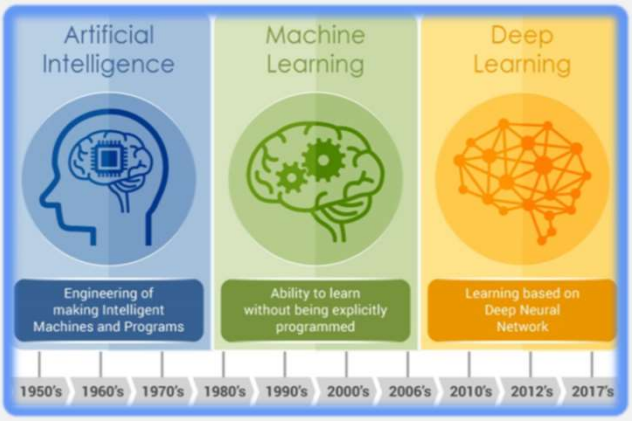
CENG 3511 Artificial Intelligence

Week 2

Intelligent Agents & Problem Solving

Instructor
Bekir Taner Dinçer

Teaching Assistant
Selahattin Aksoy



MUĞLA SITKI KOÇMAN UNIVERSITY
COMPUTER ENGINEERING
2024 – FALL

1

Intelligent Agents & Problem Solving

2

Outline

- Intelligent Agents
- Rationality and Environment Types
- Problem Solving by Search

Reading:

Chapter 3-4 of *Artificial Intelligence: A Modern Approach*

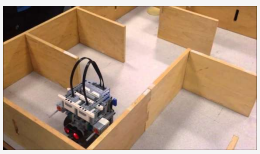
- ❑ **Rationality**: the quality of being based on or in accordance with reason or logic.
- ❑ TR: rasyonellik, mantık
- ✓ A rational person is someone who is sensible and is able to make decisions based on intelligent thinking rather than on emotion.

3

Intelligent Agents

An **intelligent agent** is an entity that **perceives** **its environment** through **sensors** and **acts** upon it through **actuators**

Robot Vacuum Cleaner
- Perceives
  - Clean/Dirty
- Acts
  - Move/Clean



Robot Car in Maze
- Perceives
  - Open/Wall
- Acts
  - Move/Turn

4

Types of Agents

Simple Reflex Agents

Act based on the current perception (state) without considering past history (no memory)

Automatic Doors

Model-Based Reflex Agents

Maintain an internal model of the world and act based on both current perception and past history.

Self-driving Cars

Robot Vacuum Cleaners

Goal-Based Agents

Have a goal and choose actions to achieve it.

Chess Play

Navigation

Utility-Based Agents

Consider both goals and the potential outcomes of actions to maximize their utility (happiness or satisfaction).

Recommendation Systems

Learning Agents

Can improve their performance through experience.

Image Recognition

Game Playing AI

Reflex → Learning

Weaker → Stronger

MSKU CENG | CENG-3511 Artificial Intelligence

5

Simple Reflex Agent Example

Agent:
A vacuum cleaner that moves and cleans the squares

Environment:
A 2x2 grid representing a room with two squares that can either be clean or dirty.

Sensors:
Detect whether a square is **clean** or **dirty**.

Actuators:
The agent can **move left**, **right**, **up**, or **down**, and can clean the current square.

Actions:

- If the square is dirty => clean
- If the square is clean => move

States = { (0,0), (0,1), (1,0), (1,1) }

Example Environments

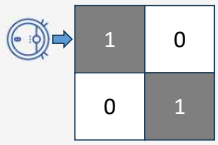
Environment Representations

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

MSKU CENG | CENG-3511 Artificial Intelligence

6

Simple Reflex Agent Example



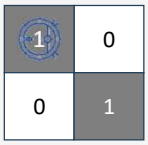
Environment: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Initial State: (0,0)

7

Simple Reflex Agent Example

Environment: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



Current State: (0,0)

Sensor reading: 1 (Dirty)

Decision Making

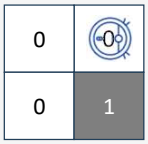
If the square is dirty:
Action: Clean

Move Next State:

8

Simple Reflex Agent Example

Environment: $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$



Current State: (0,1)

Sensor reading: 0 (Clean)

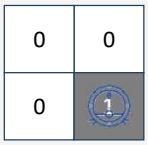
If the square is dirty:
Action: Clean

Move Next State:

9

Simple Reflex Agent Example

Environment: $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$



Current State: (1,1)

Sensor reading: 1 (Dirty)

If the square is dirty:
Action: Clean

Move Next State:

10

Simple Reflex Agent Example

Environment: $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Current State: (1,0)

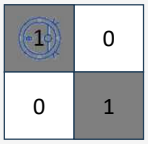
Sensor reading: 0 (Dirty)

If the square is dirty:
Action: Clean

Move Next State:

11

Simple Reflex Agent Example



Current State: (0,0)

Sensor reading: 1 (Dirty)

If the square is dirty:
Action: Clean

Move Next State:

```
# Initialize environment
environment = [[1, 0], [0, 1]]
print(f"Initial Environment: {environment}\n")

# Create Vacuum Agent
agent = VacuumAgent(environment)

# Run agent
agent.act()

# Show resulting environment
print(f"\nResulting Environment: {environment}")
```

```
class VacuumAgent:

    def __init__(self, environment):
        self.env = environment
        self.state = (0,0) # Initial State

    def sense(self):
        return self.env[self.state[0]][self.state[1]]

    def isDirty(self):
        return self.sense() == 1

    def clean(self):
        self.env[self.state[0]][self.state[1]] = 0
        return self.env

    def move(self,state):
        # Move agent to next state
        return True

    def act(self):
        while True:
          print(f"Agent at: {self.state}")
          if self.isDirty():
              self.clean()
          if self.move(self.state):
             print(f"Agent moved to: {self.state}")
          else:
            break
        print(f"Agent stoped at: {self.state}")
        return self.env
```

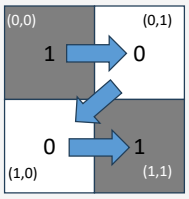
12

Simple Reflex Agent Example

CENG3511-AI-Lab1-SimpleReflexAgents.ipynb

```
def move(self,state):
    # Move agent to next state
    return True
```

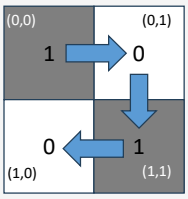
Option 1



Agent Path

(0,0)->(0,1)->(1,0)->(1,1)

Option 2



Agent Path

(0,0)->(0,1)->(1,1)->(1,0)

Rationality and Environment Types

Rationality

Rationality in AI refers to
an agent's ability to
make decisions that maximize its expected outcome
based on its knowledge, goals, and available information.

15

Rationality

Rationality in AI refers to
an agent's ability to
make decisions that maximize its expected outcome
based on its knowledge, goals, and available information.

Rational agents
choose the best possible action
by evaluating possible outcomes and selecting the one that maximizes utility.

This involves
considering both the available data and the potential consequences of actions.

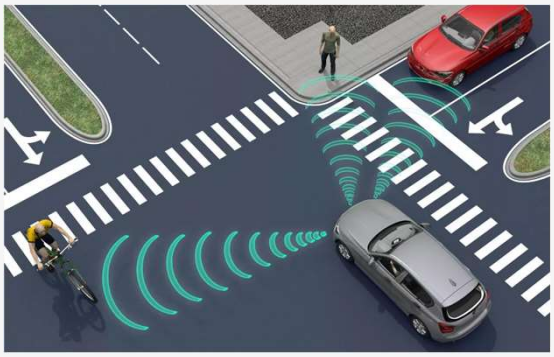
16

Examples of Rational Behavior in AI Agents

Self-Driving Cars

A self-driving car approaches a pedestrian crossing.
Based on its sensors and data,
the rational action would be to slow down or stop,
ensuring the pedestrian's safety.

It calculates this as the best action
because it aligns with its goal to avoid accidents,
which would incur legal and safety consequences.



17

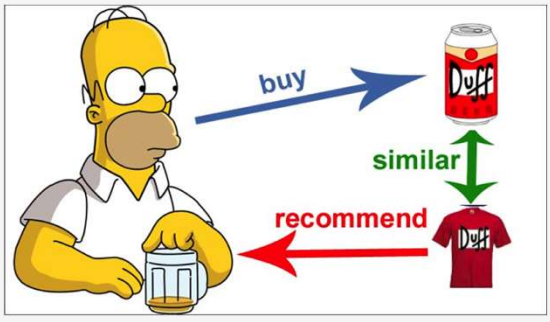
Examples of Rational Behavior in AI Agents

Recommendation Systems
(Netflix, Youtube, Spotify, X, TikToc,etc)

These systems suggest content
based on a user's past preferences and interactions.

The rational action here is to recommend items that
- **maximize user engagement**,
- **driving longer session times** and
- **satisfaction**.

It calculates this as the best action
because it aligns with its goal to
make the **best recommendations** that would
maximize the profit.



18

Environment Types

1. **Observability**: Fully Observable **vs** Partially Observable
2. **Determinism**: Deterministic **vs** Stochastic
3. **Temporal Dimension**: Episodic **vs** Sequential
4. **Environment Changes**: Static **vs** Dynamic
5. **State/Action Spaces**: Discrete **vs** Continuous

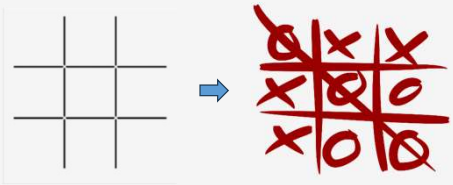
Observability

- **Fully Observable**: The agent has complete access to the relevant environment information at each point in time.
  - **Example**: Chess– the positions of all pieces are fully visible to both players at all times.

- **Partially Observable**: The agent only has access to partial information about the environment.
  - **Example**: Poker – players do not know the cards held by their opponents.

Determinism

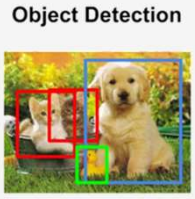
- **Deterministic**: The next state of the environment is entirely determined by the current state and the action performed by the agent.
  - **Example**: Tic-Tac-Toe – the results of placing a symbol on the board are deterministic.



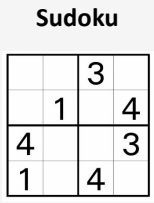
- **Stochastic**: The next state is not fully predictable, and there is some randomness or uncertainty involved.
  - **Example**: Rolling dice in board games like Monopoly or Backgammon (Tavla) – outcomes depend on probability.

Temporal Dimension

- **Episodic**: Each agent's action is an independent episode, with no influence on future actions or states.
  - **Example**: Image classification – each image is classified independently without affecting other classifications.

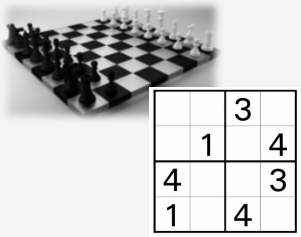


- **Sequential**: Current actions influence future states and decisions.
  - **Example**: Sudoku – Each move (filling in a number) affects the future state of the puzzle.



Environment Changes

- **Static**: The environment remains unchanged while the agent makes decisions.
  - **Example**: Crossword puzzle – the puzzle doesn't change while you solve it.



- **Dynamic**: The environment can change while the agent is deciding or acting.
  - **Example**: Real-time strategy games (StarCraft) – the game continues and evolves as the player acts.

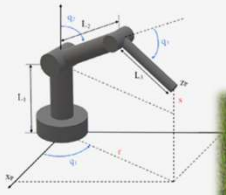
MSKU CENG | CENG-3511 Artificial Intelligence

23

State/Action Spaces

- **Discrete**: The environment has a finite set of states or actions.
  - **Example**: Board games like Chess or Go – there are a limited number of moves at any point.

- **Continuous**: The environment has a potentially infinite number of states or actions.
  - **Example**: Robot motion planning – the robot can move in continuous space with infinite possible positions.



MSKU CENG | CENG-3511 Artificial Intelligence

24

Environment Types (Examples)

Category	Types	Examples
Observability	Fully Observable	Chess, Checkers
	Partially Observable	Poker, Hide-and-Seek
Determinism	Deterministic	Tic-Tac-Toe, Sudoku
	Stochastic	Monopoly (dice rolls), Weather Forecast
Temporal Dimension	Episodic	Image Classification, Object Detection
	Sequential	Self-driving Cars, Robotics
Environment Changes	Static	Crossword Puzzle, Turn-based Strategy
	Dynamic	Real-time Strategy Games, Stock Markets
State/Action Spaces	Discrete	Chess, Go
	Continuous	Autonomous Vehicles, Drone Navigation

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	?				

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	Fully	?			

27

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	Fully	Deterministic	?		

28

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	Fully	Deterministic	Sequential	?	

29

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	Fully	Deterministic	Sequential	Static	?

30

Exercise

Example	Observability	Determinism	Temporal	Environment	State/Action Space
Chess	Fully Observable	Deterministic	Sequential	Static	Discrete
Poker	Partially	Stochastic	Sequential	Static	Discrete
Tic-Tac-Toe	Fully	Deterministic	Sequential	Static	Discrete
Driving a Car	?	?	?	?	?
Image Classification	?	?	?	?	?
Pac-Man	?	?	?	?	?

31

Problem Solving by Search

32

What is a Search Problem?

A **search problem** is a problem where the goal is to **find a sequence of actions** that lead **from an initial state to a goal state**.

33

What is a Search Problem?

A **search problem** is a problem where the goal is to **find a sequence of actions** that lead **from an initial state to a goal state**.

Key Components:

State Space: The set of all possible states that can be reached from the initial state.

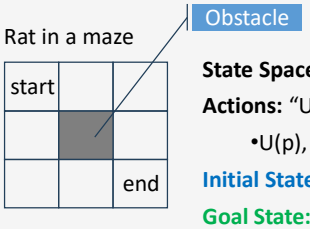
Actions: The possible operations that can be performed to transition from one state to another.

Initial State: The starting point of the search.

Goal State: The desired end state that signifies a solution.

Solution: A sequence of actions that leads from the initial state to the goal state.

Rat in a maze



State Space: { all cells except for (1,1) }

Actions: "UDLR"

- U(p), D(own), L(eft), R(ight)

Initial State: (0,0)

Goal State: (2,2)

Solution: "RRDD" or "DDRR"

34

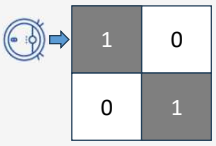
What is a Search Problem?

Simple Reflex Agents

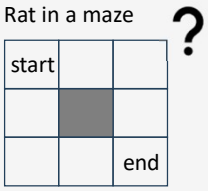
Act based on the current perception (state) without considering **past history** (no memory)

Solution

Vacuum Cleaner



Rat in a maze



What about the maze problem?

Can we solve it using a simple reflex agent?

To solve the maze problem, we need to achieve the goal.

Goal
find a path from start to end.

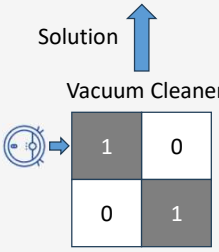
What is a Search Problem?

Simple Reflex Agents

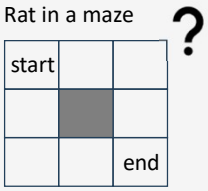
Act based on the current perception (state) without considering **past history** (no memory)

Solution

Vacuum Cleaner



Rat in a maze



Solution

Model-Based Reflex Agents

Maintain an internal model of the world and act based on both current perception and past history.

What about the maze problem?

Can we solve it using a simple reflex agent?

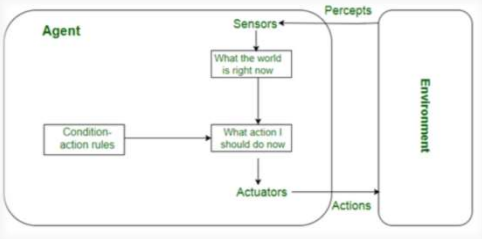
To solve the maze problem, we need to achieve the goal.

Goal
find a path from start to end.

What is a Search Problem?

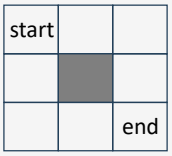
Simple Reflex Agents

Act based on the current perception (state) without considering **past history** (no memory)



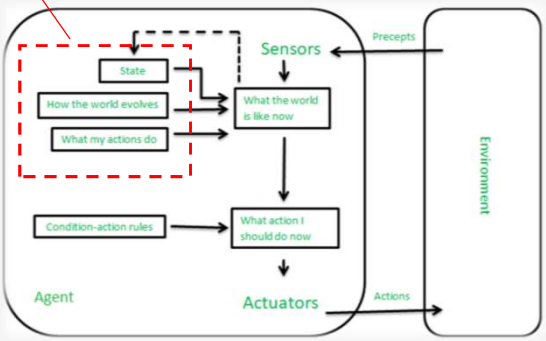
Internal Model of the world

Rat in a maze



Model-Based Reflex Agents

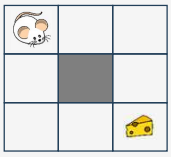
Maintain an internal model of the world and act based on both current perception and past history.



37

Maze Problem: Environment Types

Rat in a maze



Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Problem	Observability	Determinism	Temporal	Environment	State/Action Space
Maze	Partial	Deterministic	Sequential	Static	Discrete

Partially Observable: The rat (agent) cannot see the entire maze or the locations of all walls at once. It only perceives the walls or pathways in its immediate vicinity, meaning it must explore to learn about the environment.

Deterministic: The outcome of the rat's actions is predictable. For example, if the rat moves left, it either encounters a wall or finds an open path, and the result of each move is consistent every time the same action is taken in the same position.

Sequential: The rat's current action affects the future state of the maze and the rat's position. Each move builds on the previous one as it tries to find the exit, making the problem sequential rather than episodic.

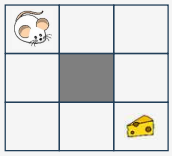
Static: The maze does not change while the rat is navigating it. Walls remain in the same place, and the environment doesn't evolve dynamically over time.

Discrete: The maze consists of discrete locations (grid cells), and the rat's actions (up, down, left, right) move it from one cell to another.

38

Search Trees: All Possible Solutions

Rat in a maze

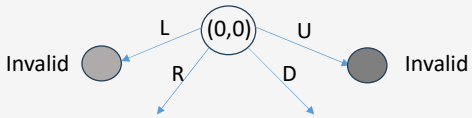


Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

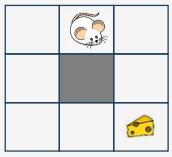
Search Tree



39

Search Trees: All Possible Solutions

Rat in a maze



Environment:

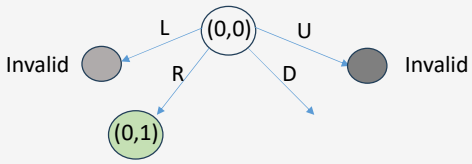
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: R

Current State: (0,1)

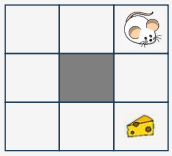
Search Tree



40

Search Trees: All Possible Solutions

Rat in a maze



Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

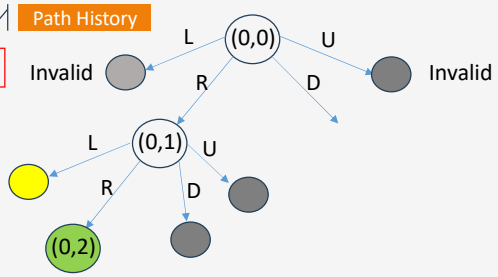
Initial State: (0,0)

Action: RR

Current State: (0,2)

Path History

Search Tree

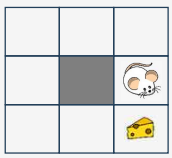


We need to keep track of the visited the nodes

41

Search Trees: All Possible Solutions

Rat in a maze



Environment:

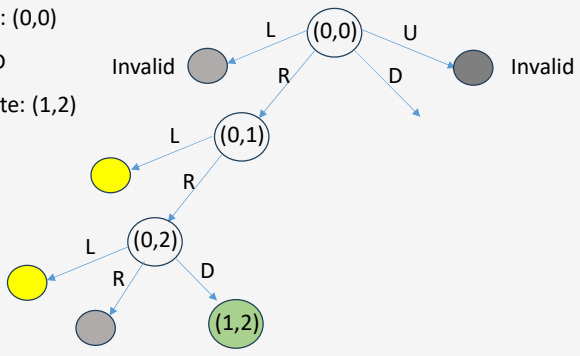
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: RRD

Current State: (1,2)

Search Tree



42

Search Trees: All Possible Solutions

Rat in a maze

Environment:

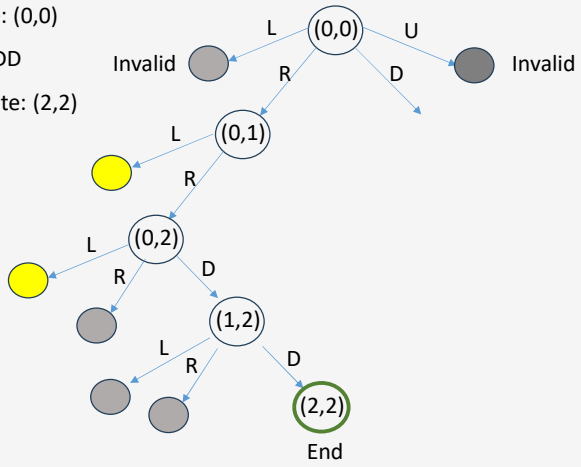
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: RRDD

Current State: (2,2)

Search Tree



43

Search Trees: All Possible Solutions

Rat in a maze

Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

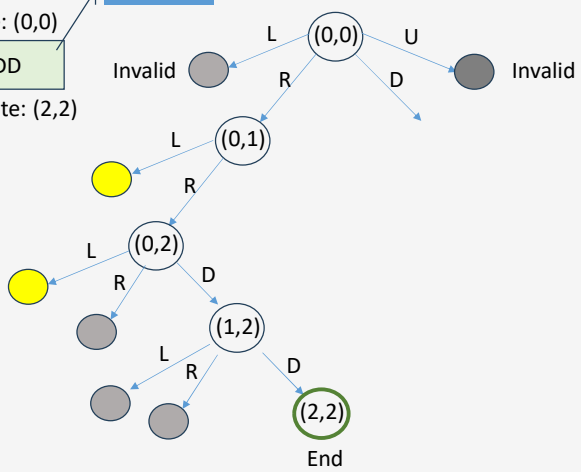
Initial State: (0,0)

Action: RRDD

Current State: (2,2)

Solution

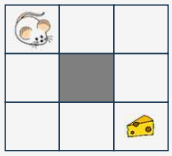
Search Tree



44

Search Trees: All Possible Solutions

Rat in a maze



Environment:

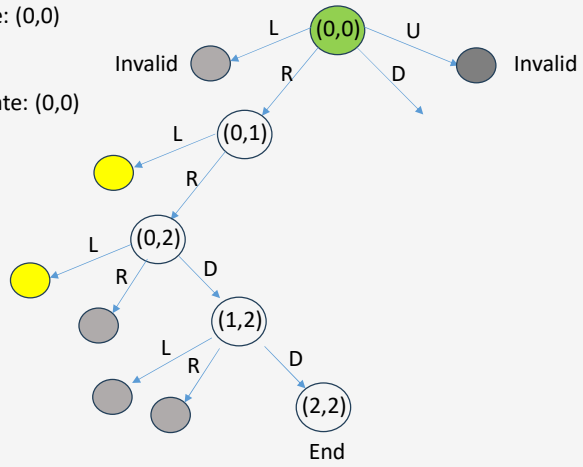
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action:

Current State: (0,0)

Search Tree



45

Search Trees: All Possible Solutions

Rat in a maze

Environment:

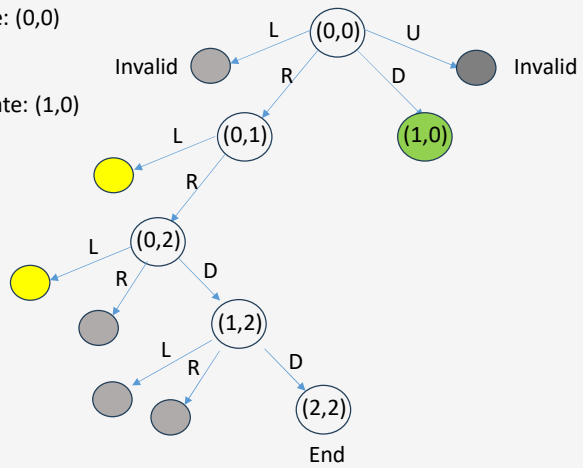
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: D

Current State: (1,0)

Search Tree



46

Search Trees: All Possible Solutions

Rat in a maze

Environment:

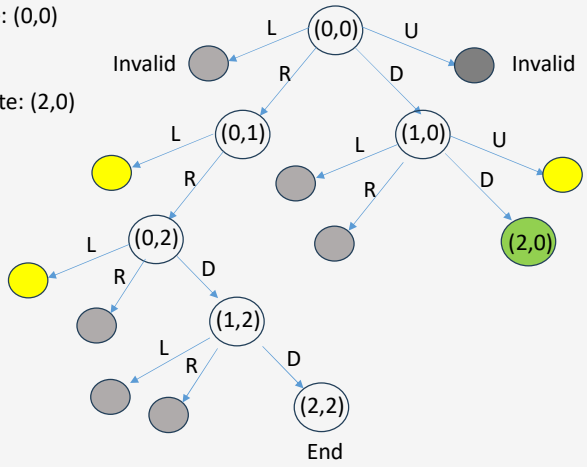
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: DD

Current State: (2,0)

Search Tree



47

Search Trees: All Possible Solutions

Rat in a maze

Environment:

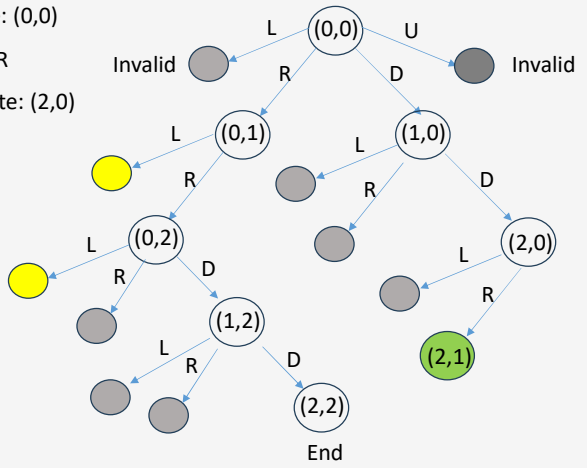
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: DDR

Current State: (2,0)

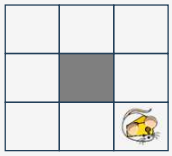
Search Tree



48

Search Trees: All Possible Solutions

Rat in a maze



Environment:

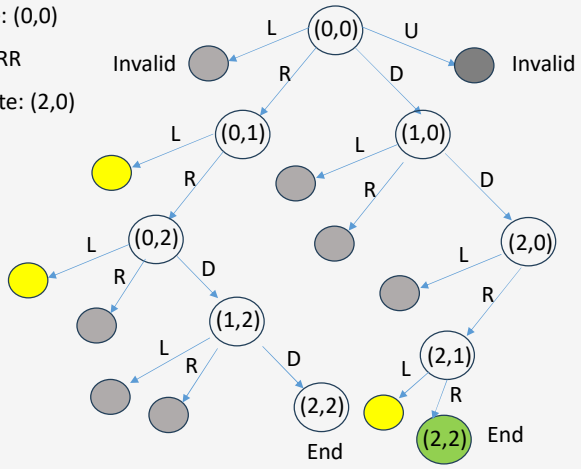
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Initial State: (0,0)

Action: DDRR

Current State: (2,0)

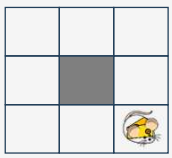
Search Tree



49

Search Trees: All Possible Solutions

Rat in a maze



Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

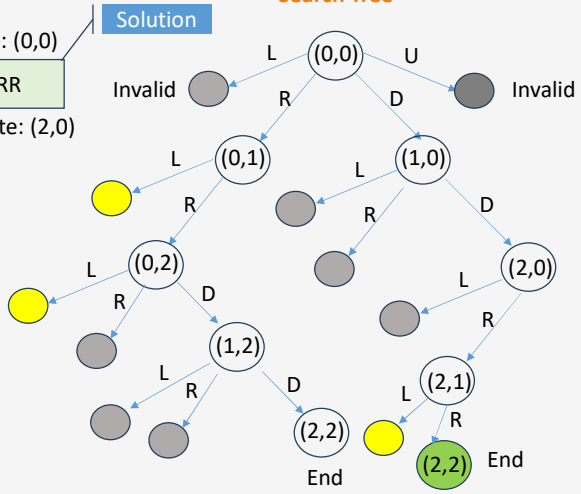
Initial State: (0,0)

Action: DDRR

Current State: (2,0)

Solution

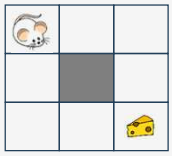
Search Tree



50

Search Trees: All Possible Solutions

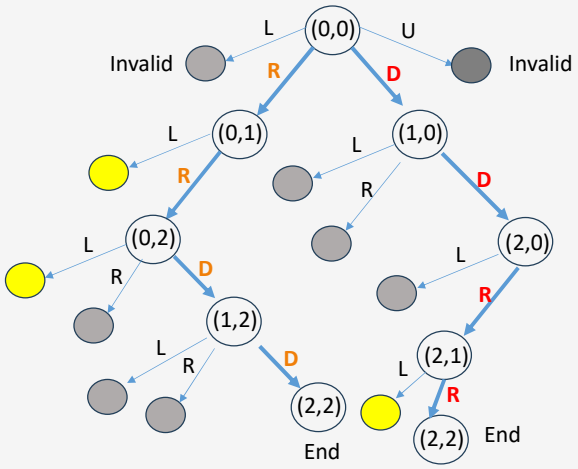
Rat in a maze



Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Search Tree

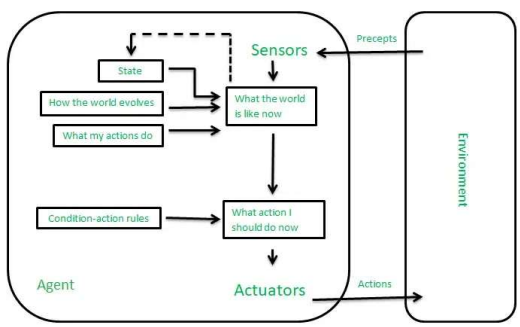


Model-Based Reflex Agent Implementation

A **model-based agent** is one that maintains **an internal representation (or model) of the world**, allowing it to act in environments where it cannot perceive everything directly.

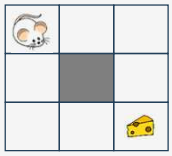
Characteristics:

These agents use **memory** to store knowledge about the world (e.g., **previously seen locations**, **obstacles**, or **hidden objects**) and update this memory as new information is perceived.



Model-based Reflex Agent Implementation

Rat in a maze



Environment:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

CENG3511-AI-Lab1-ModelBasedReflexAgent.ipynb

```
# Define the maze parameters
environment = [ [1,1,1], [1,0,1], [1,1,1] ]
initial_state = (0, 0)
goal_state = (2, 2)

# Create the agent
agent = RatRobot(environment, initial_state, goal_state)

# Run the agent and print the actions
actions = agent.run()
print("Actions taken:", actions)
```

```
Actions taken: ['down', 'down', 'right', 'right']
```

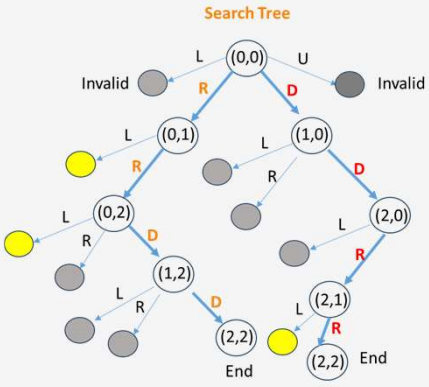
In Class Exercise
Implement RatRobot

```
class RatRobot:

    def __init__(self, environment, initial_state, goal_state):
        pass

    def get_next_state(self, state, action):
        pass

    def get_possible_actions(self, state):
        pass

    def choose_action(self, visited_states):
        pass

    def run(self):
        pass
```


Search Model/Strategies

Breadth First Search and Depth First Search

What is a Search Strategy/Model



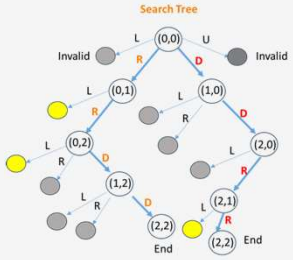
Problem Definition:

- s_{start} : starting state
- s_{goal} : goal state
- **Actions(s)**: possible actions given state *s*
- **Cost**(s,a): cost of action *a*, given state *s*
- **Next**(s,a): next state to *s* given action *a*
- **IsGoal**(s): is the state *s* the target state

Example Definition for Maze Problem:

- s_{start} : (0,0)
- s_{goal} : (2,2)
- **Actions**(s_{start}): {Right, Down}
- **Cost**(s_{start} , {Right}): 0 (no cost for each action)
- **Next**(s_{start} , Right): (0,1) ; **Next**(s_{start} , Down): (1,0)
- **IsGoal**(s_{start}): False

What is a Search Strategy/Model



Given a Search Tree and a Problem Definition,

a **Search Problem** turns into
a problem of
finding the best **Search Strategy**

Problem Definition:

- s_{start} : starting state
- s_{goal} : goal state
- **Actions(s)**: possible actions given state *s*
- **Cost**(s,a): cost of action *a*, given state *s*
- **Next**(s,a): next state to *s* given action *a*
- **IsGoal**(s): is the state *s* the target state

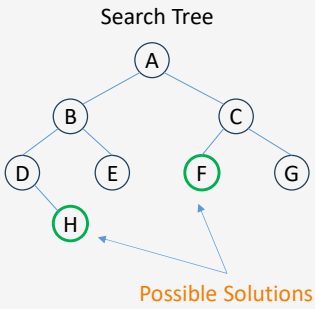
An Example Search Strategy: Breadth First Search – BFS

Key idea:

- Exploring all the neighbor nodes of a search tree at the present depth before moving on to nodes at the next depth level.

Best suited for

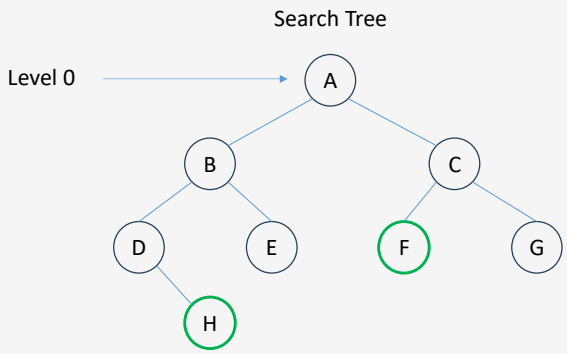
- Finding the **shortest path** in *unweighted graphs*, such as in social networks or web crawlers.



57

Breadth First Search – BFS

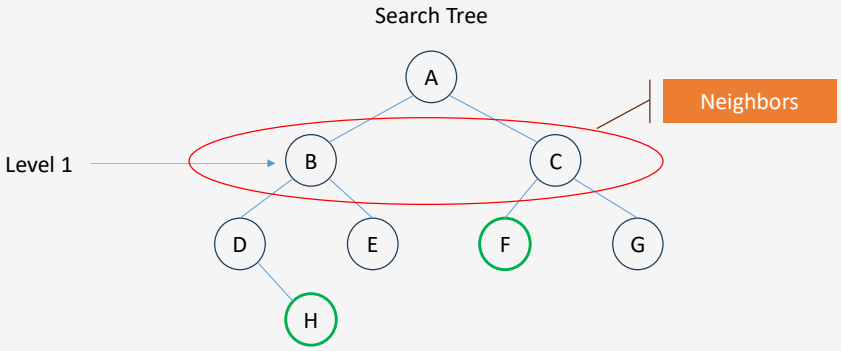
Trace: A



58

Breadth First Search – BFS

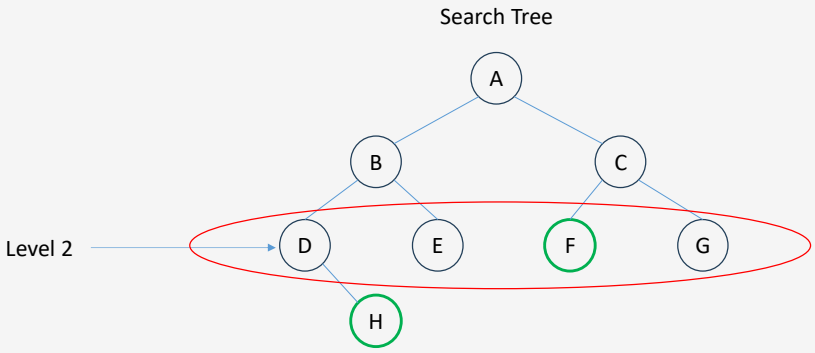
Trace: A B C



59

Breadth First Search – BFS

Trace: A B C D E F G

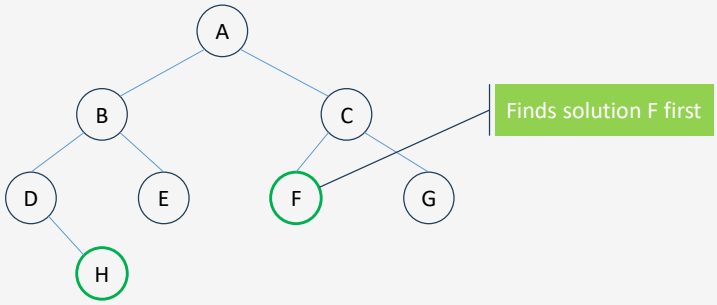


60

Breadth First Search – BFS

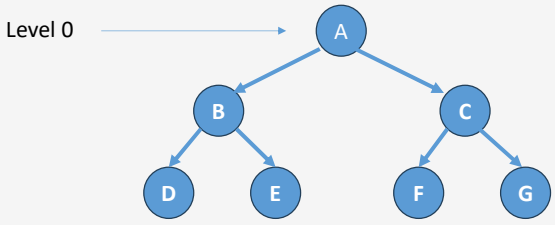
Trace: A B C D E F G

Search Tree



61

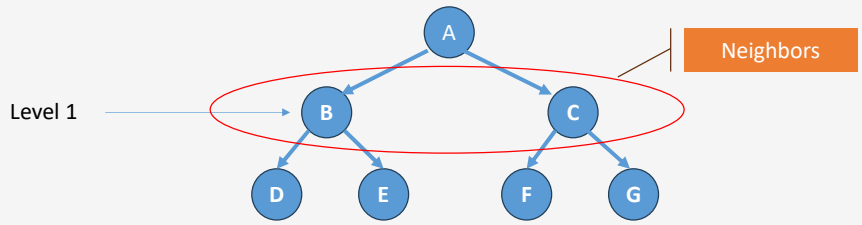
Breadth First Search - BFS



Trace: A

62

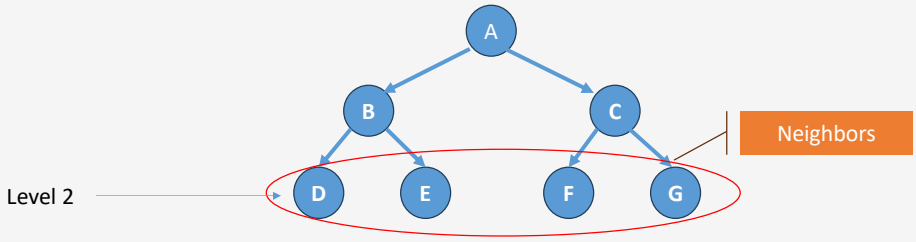
Breadth First Search - BFS



Trace: A B C

63

Breadth First Search - BFS



Trace: A B C D E F G

64

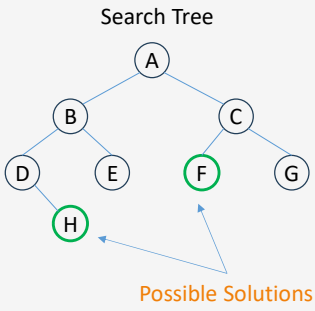
Depth First Search - DFS

Key Idea:

- exploring as far as possible along each branch before backtracking

Best suited for

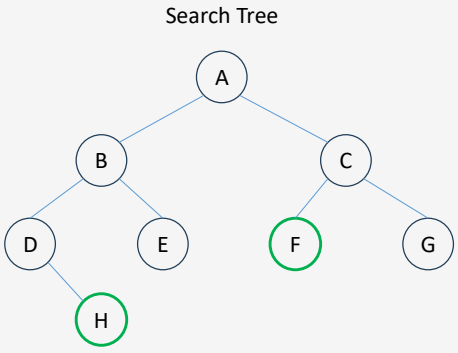
- Problems where we need to explore the depth of a tree/graph, such as solving mazes, or when searching in infinite spaces.



65

Depth First Search - DFS

Trace: A

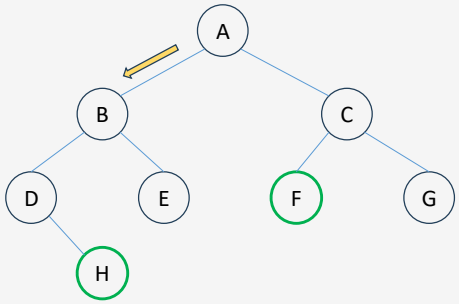


66

Depth First Search - DFS

Trace: A B

Search Tree

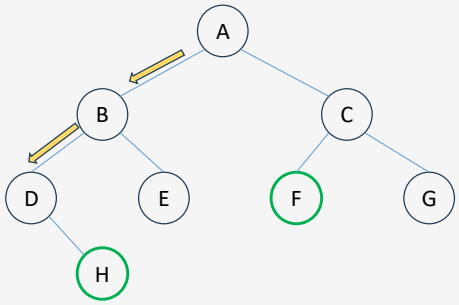


67

Depth First Search - DFS

Trace: A B D

Search Tree

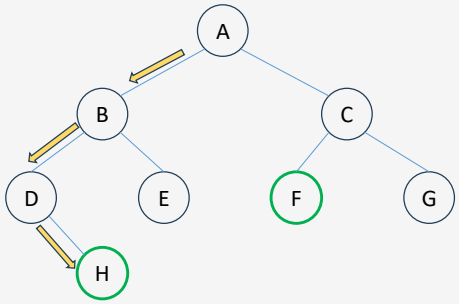


68

Depth First Search - DFS

Trace: A B D H

Search Tree

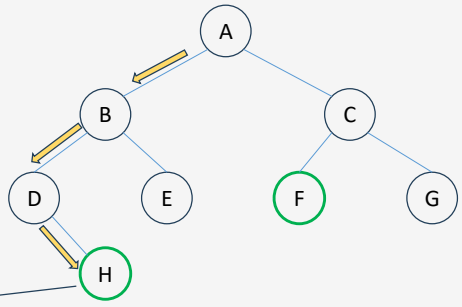


69

Depth First Search - DFS

Trace: A B D H

Search Tree

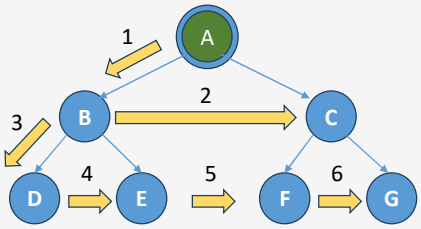


Finds solution H first

70

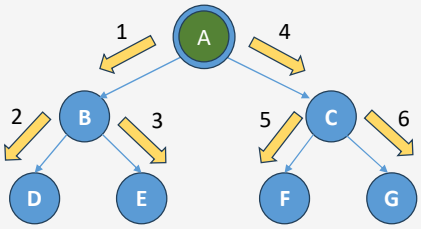
BFS vs DFS

Breadth First



Trace: A B **C D E** F G

Depth First

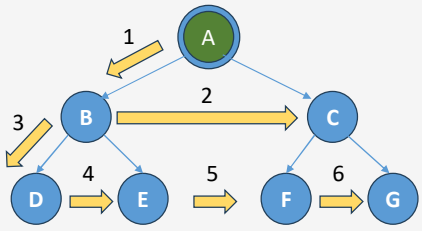


Trace: A B D E C F G

71

BFS vs DFS

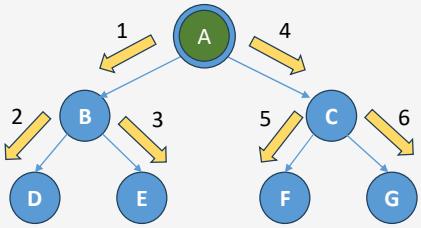
Breadth First



Applications:

- **Finding the shortest path** in an unweighted graph.
- Peer-to-peer networking, web crawlers, and social network analysis.

Depth First



Applications:

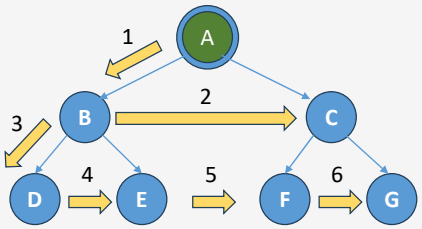
- Solving puzzles like mazes or Sudoku.
- Navigating decision trees in AI (e.g., game playing).

72

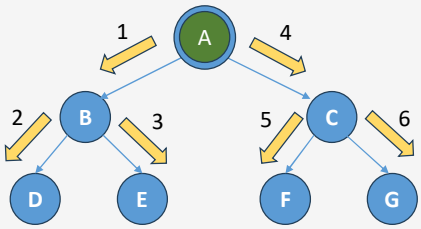
BFS vs DFS

Feature	DFS	BFS
Data Structure Used	Stack (or Recursion)	Queue
Best For	Deep traversal, exploring all possibilities	Finding the shortest path in unweighted graphs

Breadth First



Depth First



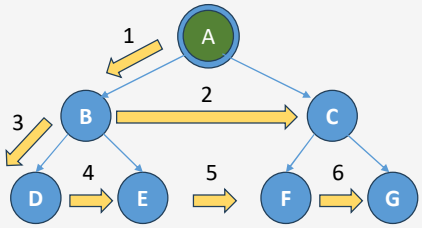
73

Exercise: BFS Implementation

Pseudo code for BFS

```
BFS(node):
    create a queue Q
    mark node as visited and enqueue it into Q
    while Q is not empty:
        current = dequeue(Q)
        for each neighbor of current:
            if neighbor is not visited:
                mark neighbor as visited
                enqueue neighbor into Q
```

Breadth First

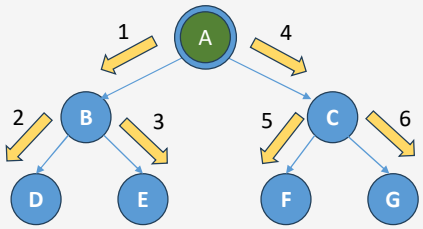


Trace: A B **C D E** F G

74

Exercise: DFS Implementation

Depth First



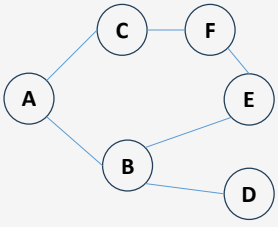
Trace: A B D E C F G

DFS using Stack (Implemented)

```
DFS(node):
    create an empty stack S
    push node onto S
    while S is not empty:
        current = pop from S
        if current is not visited:
            mark current as visited
            for each neighbor of current:
                if neighbor is not visited:
                    push neighbor onto S
```

DFS using Recursion (Exercise)

```
DFS(node):
    mark node as visited
    for each neighbor of node:
        if neighbor is not visited:
            DFS(neighbor)
```

Homework 1

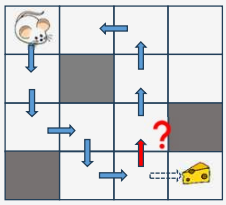
Due next week

77

New Maze Setup

CENG3511-AI-Lab1-ModelBasedReflexAgent.ipynb

4x4 Maze



Obstacles/Walls:
{(3,0), (1,1), (2,3)}

The agent implementation fails to find a solution

Why? **Fix it as a Homework**

```
# Define the maze parameters
environment = [ [1,1,1,1], [1,0,1,1], [1,1,1,0], [0,1,1,1] ]
initial_state = (0, 0)
goal_state = (3, 3)

# Create the agent
agent = RatRobot(environment, initial_state, goal_state)

# Run the agent and print the actions
actions = agent.run()
print("Actions taken:", actions)
```

```
No action is possible! !!! Robot Stucked !!!
Actions taken: ['down', 'down', 'right', 'down', 'right', 'up', 'up', 'up', 'left']
```


78

Next Week

Advanced Search Techniques

Reading:

Chapter 3 & 4 of *Artificial Intelligence: A Modern Approach*

Next Week's Topics

Uninformed Search Strategies

- *Breadth-First Search (BFS),*
- *Depth-First Search (DFS),*
- *Depth-Limited Search (DLS),*
- *Iterative Deepening Depth-First Search (IDDFS)*
- *Uniform Cost Searches (UCS)*

Informed Search Strategies

- *Greedy Search,*
- *A* Search*

Local Search Algorithms

- *Hill Climbing,*
- *Simulated Annealing,*
- *Genetic Algorithms*