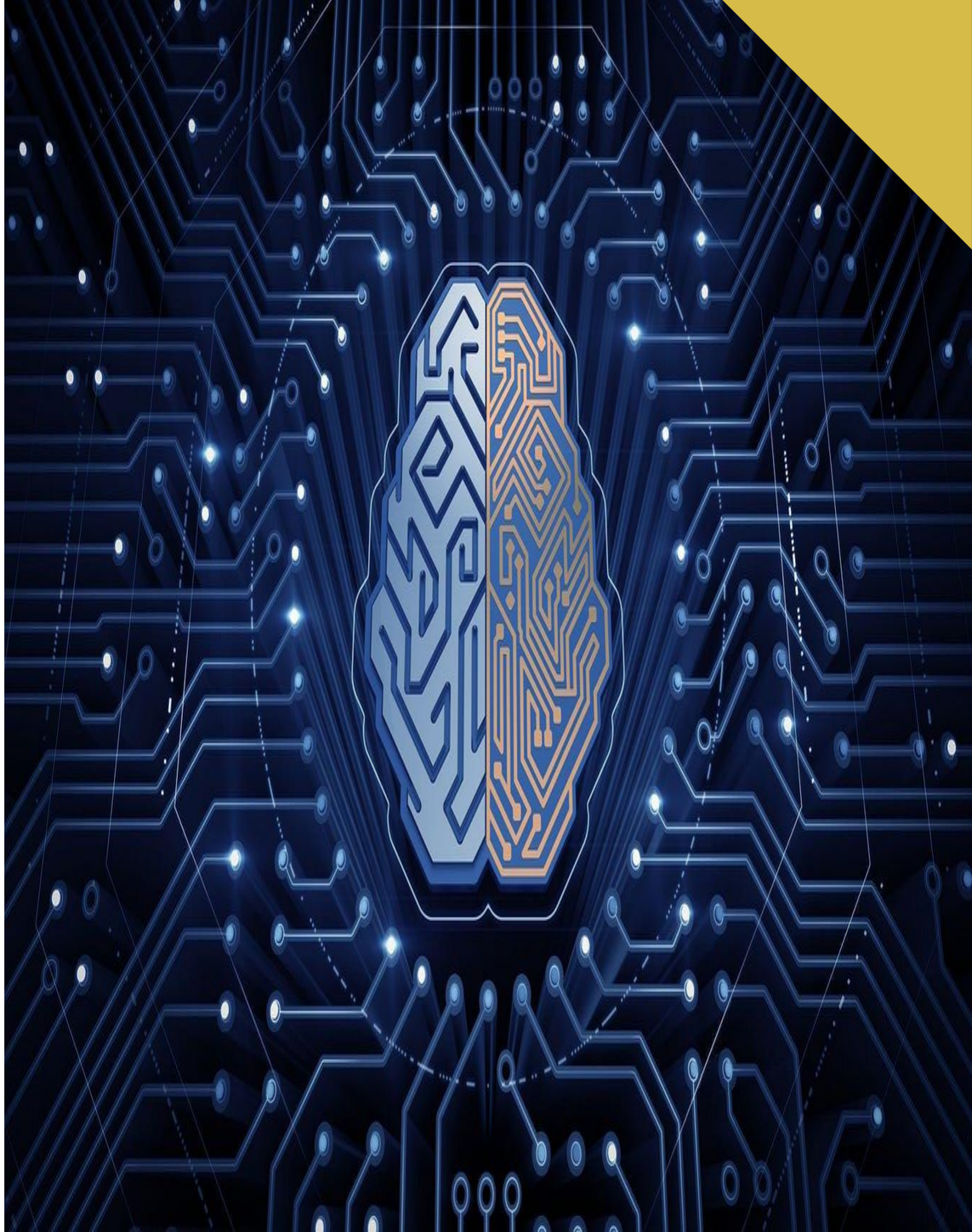
Greedy Algorithms

Dedan Kimathi University
of Technology

Introduction to Greedy Algorithms

Greedy algorithms are a fundamental concept in computer science. They provide a simple and efficient approach to solving optimization problems.



Definition of Greedy Algorithms

Algorithms

Definition: Greedy algorithms build up a solution piece by piece, always choosing the next piece that offers the most immediate benefit..

Principle :At each step, make a choice that seems the best now, aiming for a global optimum.

Characteristics of Greedy Algorithms

Algorithms

Greedy algorithms are a type of algorithm that make locally optimal choices at each step in the hope of finding a globally optimal solution. They are often used to solve optimization problems.

Greedy algorithms are characterized by their ability to make decisions without considering the future consequences. They are based on the principle of making the best choice at each step, without looking ahead to see if a better choice might be available later.

Greedy Choice Property

The greedy choice property states that a locally optimal choice will always lead to a globally optimal solution.

In other words, at each step, the algorithm makes the choice that appears to be the best at that moment, without considering the consequences of future choices.

Optimal Substructure

A key characteristic of problems solvable by greedy algorithms is the optimal substructure property. This means that an optimal solution to the overall problem can be constructed by combining optimal solutions to its subproblems.

Steps of Greedy Algorithm Design

1. Formulate the optimization problem in the form: we make a choice and we are left with one subproblem to solve.

2. Show that the greedy choice can lead to an optimal solution, so that the greedy choice is always safe.

3. Demonstrate that an optimal solution to original problem = greedy choice + an optimal solution to the subproblem

General Characteristics of Optimization Problem Problem

Ingredients

- Instances: The possible inputs to the problem
- Solutions for instances: Each instance has an exponentially large set large set of valid solutions
- Cost of solution: Each solution has an easy-to-compute cost or value cost or value

Specification

- Preconditions: The input is one instance
- Postconditions: A valid solution with optimal cost (minimum or (minimum or maximum)

Comparison: Dynamic Programming vs Greedy Algorithms

Dynamic Programming

- At each step, the choice is determined based on solutions of subproblems.
- Bottom-up approach
- Sub-problems are solved first before solving further sub-problems.
- Can be slower, more complex

Greedy Algorithms

- At each step, we quickly make a choice that currently looks best. --A local optimal (greedy) choice.
- Top-down approach
- Greedy choice can be made
- Usually faster, simpler

Common Problems Solved by Greedy Algorithms

Greedy algorithms find solutions by making locally optimal choices at each step. These algorithms are effective for a range of problems, including scheduling, finding optimal routes, and allocating resources.

Scheduling Problems

Problems

Scheduling problems involve optimizing the allocation of resources, such as time, people, or machines, to tasks or events.

Greedy algorithms are often used to solve scheduling problems, like finding the optimal schedule for a set of tasks with deadlines and durations.

Knapsack Problems Problems

Knapsack problems are classic optimization problems where you need to choose the most valuable items to fit in a knapsack with a limited weight capacity. These problems arise in various real-world scenarios like resource allocation, investment strategies, and package delivery.

Minimum Spanning Tree

A minimum spanning tree (MST) is a subset of edges from a graph that connects all vertices with the minimum total edge weight. MSTs are essential for network optimization, where minimizing the cost of connecting all nodes is critical.

Shortest Path Problems

Greedy algorithms can be used to find the shortest path between two points in a graph.

Dijkstra's algorithm is a classic example of a greedy algorithm for shortest path finding.

It works by iteratively finding the shortest path to each vertex in the graph, starting from the source vertex.

Advantages of Greedy Algorithms

Greedy algorithms are often used in computer science and other fields due to their simplicity and efficiency. They are relatively easy to implement and understand, even for complex problems.

Advantages of Greedy Algorithms

Greedy algorithms offer several advantages that make them attractive for solving certain types of problems.

They are often simple to understand and implement, making them efficient for coding and development.

Ease of Implementation

Greedy algorithms are known for their straightforward implementation. The simplicity of the greedy choice property makes it easy to translate the algorithm into code. This reduces development time and simplifies debugging.

Disadvantages of Greedy Algorithms

Algorithms

Greedy algorithms are a powerful tool for solving optimization problems, but they have limitations. They may not always produce the optimal solution. This is because they make locally optimal choices at each step, without considering the overall impact on the final solution.

Disadvantages of Greedy Algorithms

Algorithms

Greedy algorithms are often simple and efficient, but they can have limitations. A major disadvantage is that they may not always produce optimal solutions. The greedy approach focuses on making the best choice locally, but this may not lead to the overall best solution.

Local Optimum vs Global Optimum Optimum

Greedy algorithms often find a local optimum, meaning the best solution within a limited search space. However, this may not be the best overall solution, or global optimum, which considers the entire problem space.

Example: Huffman Coding

Huffman coding is a widely used algorithm for data compression.

It constructs a variable-length prefix code based on the frequency of characters in a message.

Characters with higher frequency are assigned shorter codes, while less frequent characters get longer codes.

Example: Activity Selection Problem

The Activity Selection Problem is a classic example of a greedy algorithm application.

A set of activities with start and finish times is given.

The goal is to select a maximum subset of non-overlapping activities.

Exercises

Here are some exercises to test your understanding of greedy algorithms:

1. **Fractional Knapsack Example**: Given weights and values of items, determine the maximum value that can be obtained with a knapsack of fixed capacity..

2. **Huffman Coding Example**: Given frequencies of characters, build the Huffman tree and determine the code for each character.

3. Analyze the time complexity of the greedy algorithm for the activity knapsack problem and Huffman coding. How does it compare to a brute-force approach?

THANK YOU

Dedan Kimathi University of Technology

Dr Jane Kuria

Private Bag- Dedan kimathi, Nyeri-Mweiga road

Telephone: ***+254-721709511***

Email: ***jane.kuria@dkut.ac.ke***

Website: ***www.dkut.ac.ke***