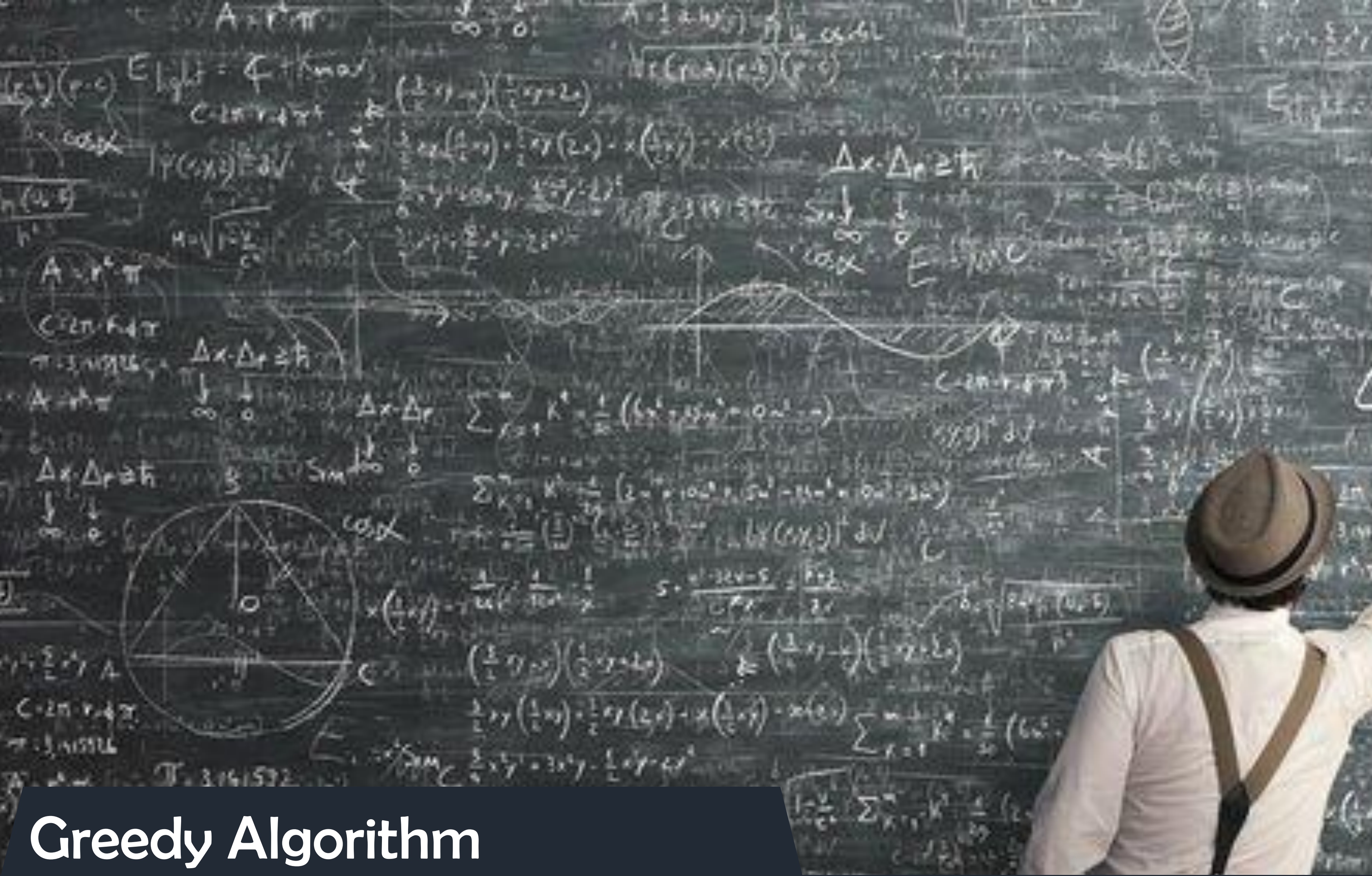


Greedy Algorithm

Ja-Hee Kim

Agenda

Introduction

Techniques

Examples

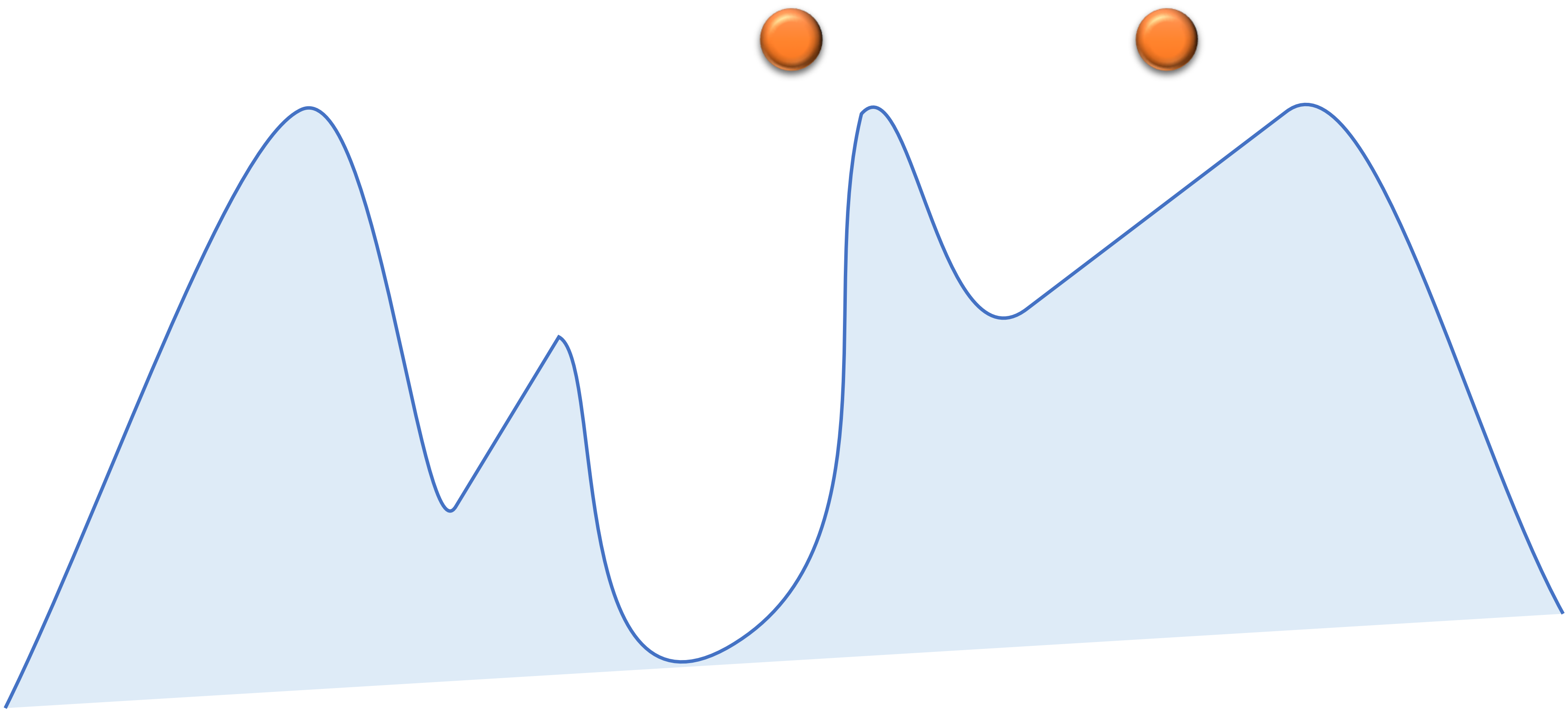
Introduction

Greedy algorithm

- "**take what you can get now**" strategy
- For most optimization problems you want to find, not just a solution, but the best solution.
- A **greedy algorithm** sometimes works well for optimization problems. It works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences.
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum.

Weakness

- **Local optimum**
  - In many problems, a greedy strategy does not usually produce an optimal solution.
  - Example: Finding the deepest valleys



Techniques

Components of greedy algorithms

- We repeat step 1 & step 2, until a solution function indicate that we find the solution.

  Step 1: A greedy algorithm selects a solution from a **candidate set** using a **selection function** to optimize **objective function**.

  Step 2: After that, it checks whether the solution is feasible using a **feasibility function**.

  - Where
    - A solution function, which will indicate when we have discovered a complete solution.
    - A **candidate set**, from which a solution is created
    - A **selection function**, which chooses the best candidate to be added to the solution
    - An **objective function**, which assigns a value to a solution, or a partial solution, and
    - A **feasibility function**, that is used to determine if a candidate can be used to contribute to a solution

Appropriate application

- Counting money: Suppose you want to count out a certain amount of money, using the fewest possible bills and coins
  - Solution function: making the amount of money
  - Objective function: minimize the number of bills and coins
  - Candidate set: bills and coins
  - Selection function: the largest possible bill or coin
  - Feasibility function: not overshoot

- For US money, the greedy algorithm always gives the optimum solution

Example of counting money

- Components
  - Solution function: making the amount of money
  - Objective function: minimize the number of bills and coins
  - Candidate set: \$5, \$1, 25¢, 10¢, 1¢
  - Selection function: the largest possible bill or coin
  - Feasibility function: not overshoot
- Example: To make \$6.39, you can choose:
  - a \$5 bill
  - a \$1 bill, to make \$6
  - a 25¢ coin, to make \$6.25
  - A 10¢ coin, to make \$6.35
  - four 1¢ coins, to make \$6.39
  - Answer: 1+1+1+1+4= 8

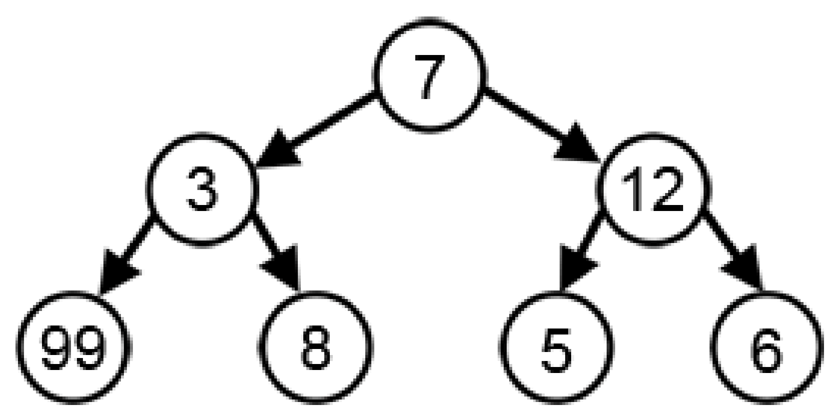
A failure of the greedy algorithm

- Fictional monetary system

- Make 15 krons
  - Result of Greedy algorithm
  - Result of Optimal algorithm

Misapplication

- Example: finding the root-to-leaf path of which sum is maximum.
  - Solution function: path from the root
  - Objective function: maximize the sum of values of the node in the path
  - Candidate set: children
  - Selection function: children that has the max value
  - Feasibility function: check whether a node is a leaf



Examples

Fractional Knapsack Problem

Knapsack Capacity: 5kg

ItemA: $60, 1kg

ItemB: $100, 2kg

ItemC: $120, 3kg

- **Solution function**: the sum of the value in the knapsack
- **Objective function**: maximize the value
- **Candidate set**: ItemA, ItemB, ItemC
- **Selection function**: the most valuable item per the unit weight
- **Feasibility function**: not exceed knapsack capacity

Fractional Knapsack Problem

Knapsack Capacity: 5kg

ItemA: $60, 1kg

ItemB: $100, 2kg

ItemC: $120, 3kg

	Value in knapsack	Remain capacity
Initial	$0	5 kg
Select		
Select		
Select		

Example of greedy algorithms

- **Optimal Solution**
  - **Dijkstra's** algorithm for finding the shortest path in a graph
  - **Kruskal's** algorithm for finding a minimum-cost spanning tree
  - **Prim's** algorithm for finding a minimum-cost spanning tree
  - **Huffman Coding** for a lossless data compression.
- Approximate Greedy Algorithm for NP Complete Problem
  - Graph Coloring
  - Travelling Sales Problem

Dijkstra's algorithm

- **Dijkstra's** algorithm for finding the shortest path in a graph
  - Always takes the *shortest* edge connecting a known node to an unknown node
- **Solution function**: the sum of the weight in the path
- **Objective function**: minimize the sum of the weight
- **Candidate set**: an unknown node connecting from a known node
- **Selection function**: the *shortest* edge connecting a known node to an unknown node
- **Feasibility function**: the queue is empty

Kruskal's algorithm

- **Kruskal's** algorithm for finding a minimum-cost spanning tree
  - Always tries the *lowest-cost* remaining edge
- **Solution function**: the sum of the weight of all edges in the tree
- **Objective function**: minimize the sum of the weight
- **Candidate set**: remaining edges
- **Selection function**: the *lowest-cost* remaining edge
- **Feasibility function**: the added edge does not make a cycle

Prim’s algorithm

- **Prim’s** algorithm for finding a minimum-cost spanning tree
  - Always takes the *lowest-cost* edge between nodes in the spanning tree and nodes not yet in the spanning tree
- **Solution function**: the sum of the weight of all edges in the tree
- **Objective function**: minimize the sum of the weight
- **Candidate set**: edge between nodes in the spanning tree and nodes not yet in the spanning tree
- **Selection function**: the *lowest-cost* edge in the candidate set
- **Feasibility function**: the added edge does not make a cycle

Huffman Coding

- **Huffman Encoding** for compressing data without loss
  - Always takes the two smallest percentages to combine
- **Solution function**: the length of compresses data
- **Objective function**: minimize the data length
- **Candidate set**: a node or tree
- **Selection function**: the two smallest percentages
- **Feasibility function**: the heap contains only one node

Thanks