TA Class of Python and Algorithm

TA: Dongchen(Felix) HE

Contact: dc23.he@gmail.com

A Brief Review of Lecture One

Introduction to Greedy Algorithms

- Definition and characteristics of greedy algorithms
- Examples of real-life situations where greedy algorithms are used
- Comparison with other algorithmic paradigms (e.g., dynamic programming, divide and conquer)

Definition of Greedy Algorithms

- Greedy algorithms make locally optimal choices at each step to reach a global optimum.
- They make the best choice at each decision point without considering the complete solution.
- Greedy algorithms are characterized by their greedy property, which states that a locally optimal choice leads to a globally optimal solution.

Characteristics of Greedy Algorithms

1. Greedy Choice Property: A globally optimal solution can be reached by selecting the locally optimal choice at each step.
2. Optimal Substructure: An optimal solution to the problem contains optimal solutions to its subproblems.

Real-Life Examples of Greedy Algorithms

1. The Activity Selection Problem:
    - Select the maximum number of activities that can be performed, given their start and end times.
    - Greedy solution: Choose the activity with the earliest finish time and continue with the remaining compatible activities.
2. The Fractional Knapsack Problem:
    - Select items to maximize the total value in a knapsack with a limited capacity.
    - Greedy solution: Select items with the highest value-to-weight ratio until the knapsack is full.

Comparison with Other Algorithmic Paradigms

- Dynamic Programming:
  - Greedy algorithms make locally optimal choices at each step, while dynamic programming considers all possible choices and stores their solutions.
  - Greedy algorithms have a bottom-up approach, while dynamic programming uses a top-down approach.
  - Greedy algorithms are often faster but may not always yield the globally optimal solution.

- Divide and Conquer:
  - Greedy algorithms make locally optimal choices at each step, while divide and conquer divides the problem into subproblems and combines their solutions.
  - Greedy algorithms focus on the current step, while divide and conquer considers the entire problem recursively.

Summary

- Definition and characteristics of greedy algorithms
- Examples of real-life situations where greedy algorithms are used
- Comparison with other algorithmic paradigms (e.g., dynamic programming, divide and conquer)

Greedy Algorithm Design Paradigm

- Overview of the greedy algorithm design paradigm
- Basic steps involved in designing a greedy algorithm
- Step 1: Make a greedy choice
- Step 2: Prove that the greedy choice is always safe
- Step 3: Optimize the subproblem and solve recursively

Overview of the Greedy Algorithm Design Paradigm

- Greedy algorithm design paradigm:
    - Solves optimization problems by making locally optimal choices at each step.
    - Greedy algorithms aim to find a globally optimal solution through a series of locally optimal choices.
    - Does not always guarantee the globally optimal solution but often provides efficient solutions.

Basic Steps Involved in Designing a Greedy Algorithm

1. Step 1: Make a greedy choice:
    - Choose the best option available at the current step, considering the problem's constraints and objectives.
    - The choice should be locally optimal to increase the chances of reaching a global optimum.

2. Step 2: Prove that the greedy choice is always safe:
    - Analyze the problem to demonstrate that the locally optimal choice made in Step 1 is safe.
    - Ensure that choosing the locally optimal solution at each step does not lead to an incorrect or suboptimal final solution.

3. Step 3: Optimize the subproblem and solve recursively:
    - Formulate a subproblem by removing the selected choice from the original problem.
    - Solve the subproblem optimally using the same greedy algorithm design paradigm.
    - Combine the locally optimal solution of the subproblem with the greedy choice made in Step 1 to obtain the final solution.

some coding examples

Example 1: Minimum Coin Change

```python
def coin_change(coins, amount):
    coins.sort(reverse=True)
    num_coins = 0
    for coin in coins:
        num_coins += amount // coin
        amount %= coin
    return num_coins

coins = [1, 5, 10, 25]
amount = 63
minimum_coins = coin_change(coins, amount)
print(f"Minimum number of coins needed: {minimum_coins}")
```

- Overview of the greedy algorithm design paradigm
- Basic steps involved in designing a greedy algorithm
- Step 1: Make a greedy choice
- Step 2: Prove that the greedy choice is always safe
- Step 3: Optimize the subproblem and solve recursively

Greedy Algorithm Properties

- Greedy-choice property: Explanation and illustration
- Optimal substructure property: Explanation and illustration
- Examples showcasing the properties of greedy algorithms

Greedy-Choice Property

- Greedy-choice property states that a locally optimal choice leads to a globally optimal solution.
- At each step, a greedy algorithm selects the best available option without considering the future consequences.
- The choice made at each step is not reconsidered or modified later.

Illustration:
Consider a scenario where you need to select the maximum number of activities to schedule in a day. A greedy algorithm would choose the activity with the earliest finish time at each step. This locally optimal choice ensures that you can accommodate the maximum number of activities, leading to a globally optimal solution.

Optimal Substructure Property

- Optimal substructure property states that an optimal solution to the problem contains optimal solutions to its subproblems.
- A problem can be divided into smaller subproblems, and the optimal solution can be obtained by solving these subproblems independently.

Illustration:
In the Fractional Knapsack Problem, a greedy algorithm selects items with the highest value-to-weight ratio. The optimal solution for the entire knapsack problem contains optimal solutions for subproblems involving smaller knapsacks or fractions of items. By selecting items greedily, the algorithm ensures the optimality of the overall solution.

Examples Showcasing Greedy Algorithm Properties

1. Dijkstra's Algorithm for Shortest Path:
    - Greedy-choice property: At each step, select the vertex with the minimum distance from the source vertex.
    - Optimal substructure property: The shortest path from the source to any vertex contains the shortest paths from the source to its preceding vertices.
2. Huffman Coding for Data Compression:
    - Greedy-choice property: Assign shorter codes to more frequently occurring characters.
    - Optimal substructure property: The optimal prefix code for the entire set of characters is built from optimal prefix codes of smaller subsets of characters.

Summary

- Greedy-choice property: Explanation and illustration
- Optimal substructure property: Explanation and illustration
- Examples showcasing the properties of greedy algorithms

Common Techniques and Strategies

- Huffman coding problem
- Dijkstra's algorithm

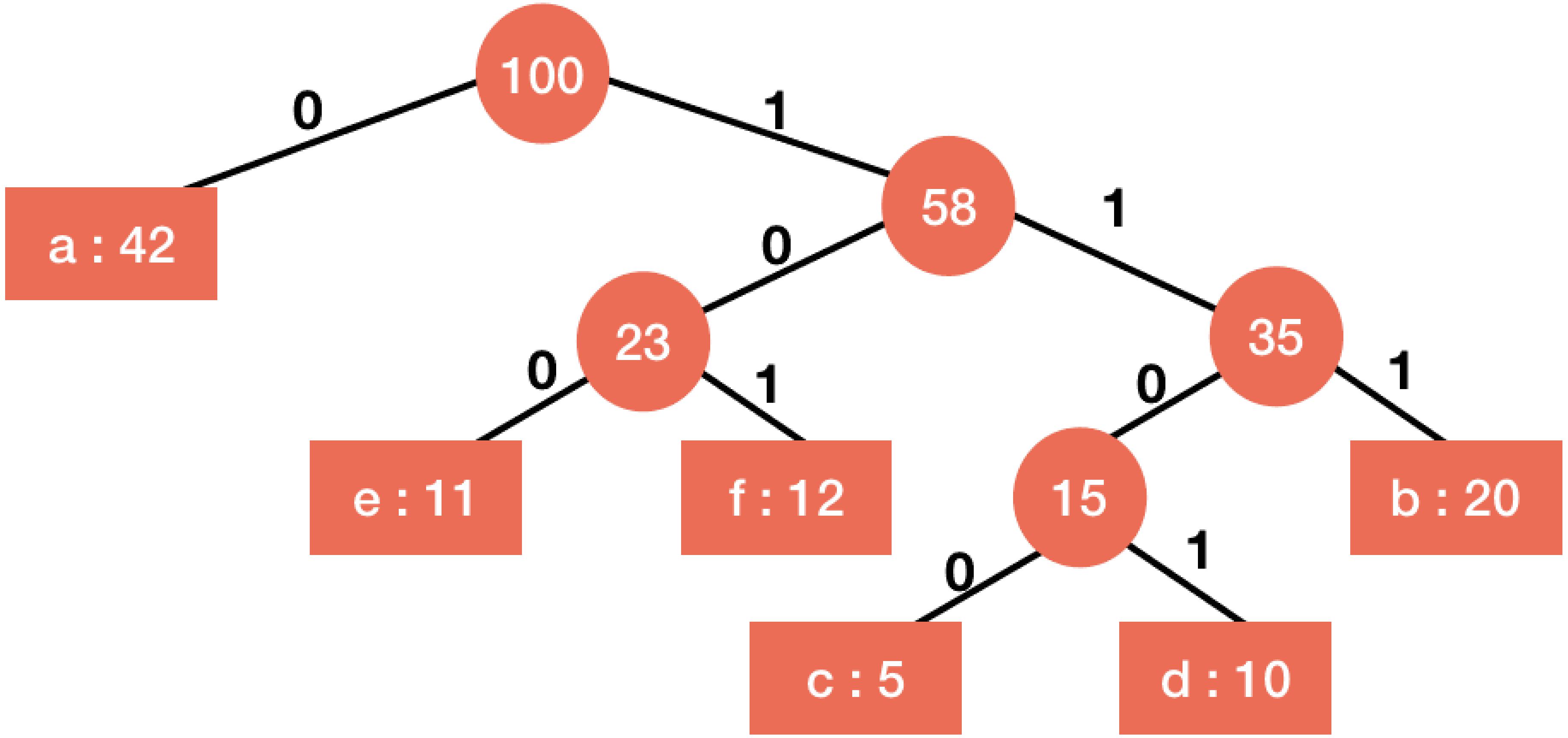
Huffman Coding Problem

- Given a set of characters and their frequencies, construct an optimal prefix code to minimize the total encoded length.
- Greedy strategy: Build a binary tree by repeatedly merging the two least frequent characters until all characters are merged.

Dijkstra's Algorithm

- Given a weighted graph, find the shortest path from a source vertex to all other vertices.
- Greedy strategy: At each step, select the vertex with the minimum distance from the source and update the distances of its adjacent vertices.

Let's focus on huffman coding problem first!


100
0
1
a : 42
58
0
1
23
35
0
1
0
1
e : 11
f : 12
15
b : 20
0
1
c : 5
d : 10

Problem:

a: 1/5

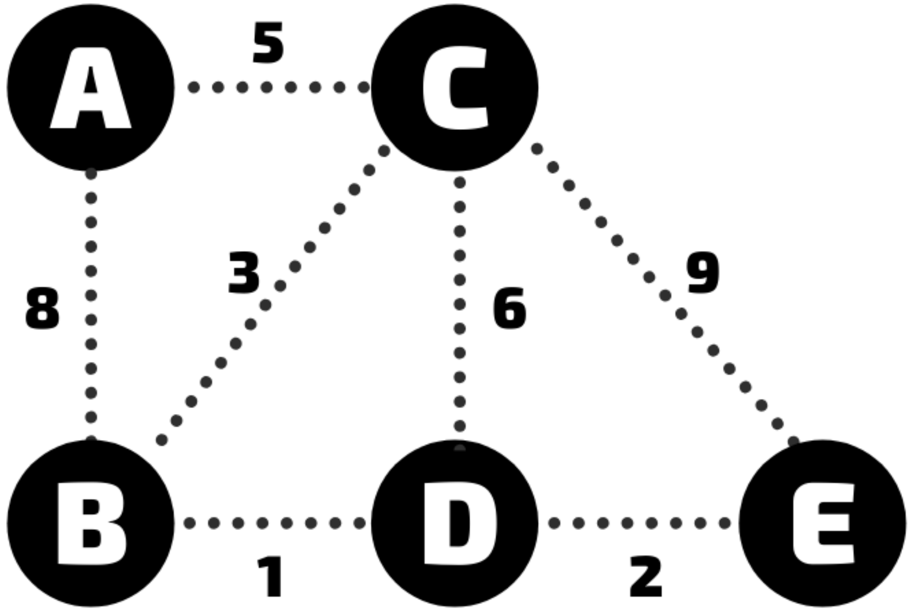
b: 2/5

c: 3/10

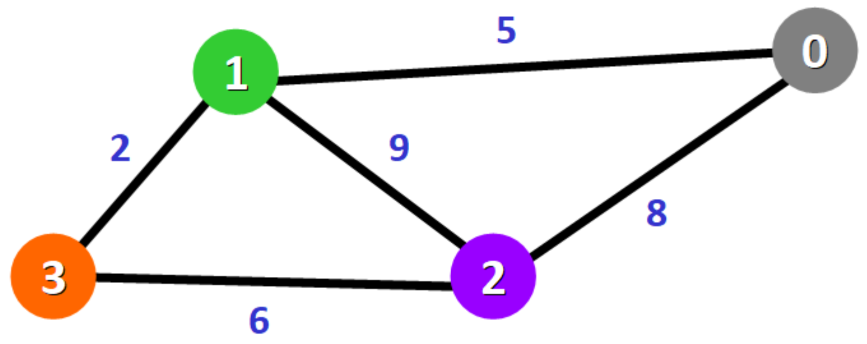
d: 1/10

Give me the huffman code of the following (a,b,c,d)

Then Dijkstra's Algorithm


A
C
B
D
E
5
8
3
6
9
1
2

Problem:



Summary

- Huffman coding problem
- Dijkstra's algorithm