Experiment No. 4

Title: Implementation of problem using Greedy Programming Approach

Batch: B-4 **Roll No: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.: 4

Aim: To study Greedy Programming approach for implementation of problem statement to obtain optimal solution.

Resources needed: Text Editor, C/C++ IDE

Theory:

Greedy Programming is a problem-solving strategy that makes locally optimal decisions at each stage to achieve a globally optimum solution. This simple, intuitive algorithm can be applied to solve any optimization problem which requires the maximum or minimum optimum result

What is Greedy Programming?
A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage. In many problems, a greedy strategy does not produce an optimal solution, but a greedy heuristic can yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

Greedy algorithms work on problems for which it is true that, at every step, there is a choice that is optimal for the problem up to that step, and after the last step, the algorithm produces the optimal solution of the complete problem. To make a greedy algorithm, there is need to identify an optimal substructure or subproblem in the problem.

Steps for creating Greedy Algorithm-
By following the steps given below, you will be able to formulate a greedy solution for the given problem statement:
Step 1: In a given problem, find the best substructure or subproblem.
Step 2: Determine what the solution will include (e.g., largest sum, shortest path).
Step 3: Create an iterative process for going over all subproblems and creating an optimum solution.

Why to use Greedy Programming -
Here are some reasons for using Greedy Programming Approach –

- The greedy approach has a few tradeoffs, which may make it suitable for optimization.
- One prominent reason is to achieve the most feasible solution immediately. In the activity selection problem (Explained below), if more activities can be done before finishing the current activity, these activities can be performed within the same time.
- Another reason is to divide a problem recursively based on a condition, with no need to combine all the solutions.

Example – Greedy Programming approach to find minimum number of coins.

Given a value V, if we want to make change for V Rs, and we have infinite supply of each of the denominations in Indian currency, i.e., we have infinite supply of D = { 1, 2, 5, 10, 20, 50, 100, 500, 1000} valued coins/notes, what is the minimum number of coins and/or notes needed to make the change?

Input: V = 121
Output: 3
Here, we need a 100 Rs note, a 20 Rs note and a 1 Rs coin.

The idea to implement Greedy Algorithm is to start from the largest possible denomination and keep adding denominations while remaining value is greater than 0. Below is the complete algorithm.

1) Initialize result array S as empty.
2) Find the largest denomination that is smaller than V.
3) Add the chosen denomination to the result. Subtract value of chosen denomination from V.
4) If V becomes 0, then print the result.
Else repeat steps 2 and 3 for new value of V

Limitation –
The limitation of the greedy algorithm is that it may not provide an optimal solution for some denominations. For example, the above algorithm fails to obtain the optimal solution for D = {1, 6, 10} and V= 13. In particular, it would provide a solution with four coins, i.e., S = {10, 1, 1, 1} .

However, the optimal solution for the said problem is three coins, i.e. S = {6, 6, 1}

The reason is that the greedy algorithm builds the solution in a step-by-step manner. At each step, it picks a locally optimal choice in anticipation of finding a globally optimal solution. As a result, the greedy algorithm sometimes traps in the local optima and thus could not provide a globally optimal solution.

As an alternative, we can use a dynamic programming approach to ascertain an optimal solution for general input.

Activity:
You are given a number *N*.
You are required to form two numbers *X* and *Y* such that:

- The sum of frequency of each digit in *X* and *Y* is equal to the frequency of that digit in *N*.
- The sum of numbers *X* and *Y* must be minimum.

Your task is to determine the minimum possible sum of *X* and *Y*.

Input format
The first line contains an integer *T* that denotes the number of test cases.

For each test case:

The first line contains an integer *N*.

Output format

For each test case, you are required to print the minimum possible sum of X and Y in a new line.

Constraints

$1 \leq T \leq 10^5$ $10 \leq N \leq 2 \times 10^{18}$

Sample Input

2
1321
42255

Sample Output

25
270

Solution:

```
def sumXY(N, sum):
    sum = []
    for integer in N:
        sorted_digits = sorted(integer)
        X_digits = sorted_digits[::2]
        Y_digits = sorted_digits[1::2]
        X = int("".join(X_digits))
        Y = int("".join(Y_digits))
        sum.append(X + Y)
    return sum

T = int(input())
N = []
for _ in range(T):
    N.append(input())
sum = sumXY(N, sum)
for _ in sum:
    print(_)
```

```
2
1321
42255
25
270
```

RESULT: Accepted   Refer judge environment

Score	Time (sec)	Memory (KiB)	Language
20	0.49028	9992	Python 3.8

Input	Result	Time (sec)	Memory (KiB)	Score	Your output	Correct output	Diff
Input #1	Accepted	0.130108	8456	4			
Input #2	Accepted	0.178559	9992	20			
Input #3	Accepted	0.122694	7616	20			
Input #4	Accepted	0.017007	2	20			
Input #5	Accepted	0.016952	2	20			
Input #6	Accepted	0.024961	3268	16			

Outcomes: Understand the fundamental concepts for managing the data using different data structures such as lists, queues, trees etc.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

This experiment has deepened our understanding of Greedy Programming, including its applications, strengths, and limitations. We've learned that while greedy algorithms are powerful for a subset of optimization problems, their applicability must be carefully considered against the problem's requirements.

References:

1. https://tutorialspoint.dev/algorithm/greedy-algorithms/greedy-algorithm-to-find-minimum-number-of-coins
2. https://www.baeldung.com/cs/min-number-of-coins-algorithm