Instructions: (Please read carefully and follow them!)

Try to solve all problems on your own. If you have difficulties, ask the instructor or TAs.

The implementation of the optimization algorithms in this lab will involve extensive use of the `numpy` Python package. It would be useful for you to get to know some of the functionalities of `numpy` package. For details on `numpy` Python package, please consult https://numpy.org/doc/stable/index.html

For plotting purposes, please use `matplotlib.pyplot` package. You can find examples in the site https://matplotlib.org/examples/.

Please follow the instructions given below to prepare your solution notebooks:

- Please use different notebooks for solving different Exercise problems.
- The notebook name for Exercise 1 should be `YOURROLLNUMBER_IE684_Lab1_Ex1.ipynb`.
- Similarly, the notebook name for Exercise 2 should be `YOURROLLNUMBER_IE684_Lab1_Ex2.ipynb`, etc.

There are only 2 exercises in this lab. Try to solve all problems on your own. If you have difficulties, ask the Instructors or TAs.

ALL QUESTIONS ARE COMPULSORY in this sheet. You can either print the answers using `print` command in your code or you can write the text in a separate text tab. To add text in your notebook, click `+Text`. Some questions require you to provide proper explanations; for such questions, write proper explanations in a text tab. Some questions require the answers to be written in LaTeX notation. Please see the demo video to know how to write LaTeX in Google notebooks. Some questions require plotting certain graphs. Please make sure that the plots are present in the submitted notebooks.

After completing this lab's exercises, click File → Download .ipynb and save your files to your local laptop/desktop. Create a folder with name `YOURROLLNUMBER_IE684_Lab1` and copy your `.ipynb` files to the folder. Then zip the folder to create `YOURROLLNUMBER_IE684_Lab1.zip`. Then upload only the `.zip` file to Moodle. There will be PENALTY for students who do not follow the proper naming conventions in their submissions.

Please check the **submission deadline announced in moodle.**

After the introduction to discrete optimization via - set cover problem and shortest path problem, we now discuss the Subset Sum Problem and solve a problem based on the previous exercise.

1 Subset Sum Problem

1.1 Description

The subset sum problem can be stated as follows: Given a set (or multiset) of positive integers $S = \{a_1, a_2, \ldots, a_n\}$ and a target sum T, the problem asks whether there exists a subset A of S such that the sum of the elements in A is EXACTLY equal to T.

Example:

Consider the multiset $S = \{3, 1, 7, 7, 5, 4\}$ and the target sum $T = 17$. We want to determine whether there exists a subset of S whose elements sum up to T.

Using the subset sum algorithm, we can find that there exists a subset $A = \{3, 7, 7\}$ such that the sum of its elements is equal to T. Therefore, the answer to this instance of the subset sum problem is Yes.

The subset sum problem has various practical applications, including:

- Cryptography: It is used in the study of knapsack problems, which have applications in cryptographic protocols.
- Resource Allocation: In scheduling and resource allocation problems where one needs to determine if a combination of resources can meet a specific constraint or target.
- Decision-Making: In decision-making processes where one needs to determine if a combination of elements can satisfy a given condition or requirement.

1.2 Problem Statement:

Given the following (multi)sets of integers and targets

Set1: {2, 4, 6, 8, 9, 9, 9, 10, 12, 14, 16, 18, 20, 22, 24}

Set2: {3, 3, 5, 7, 9, 11, 12, 12, 12, 13, 15, 17, 19, 21, 23}

Set3: {3, 4, 6, 6, 9, 11, 11, 12, 15, 17, 17, 17, 18, 21, 21, 24}

Set4: {2, 3, 4, 7, 11, 11, 12, 13, 13, 14, 14, 16, 19, 21, 22, 24}

Target1: 140

Target2: 165

Target3: 182

Listing 1: Algorithm 1 for Subset Sum

```
def is_subset_sum_recursive(arr, n, target, subset=[]):
    # Base Cases
    if target == 0:
        print("Subset with the given sum exists:", subset)
        return True
```

```
if n == 0 and target != 0:

    return False

# If last element is greater than target, ignore it
if arr[n - 1] > target:
    return ?

# Check if sum can be obtained by including the last element or excluding it
return is_subset_sum_recursive(arr, n - 1, ?, subset) or \
       is_subset_sum_recursive(arr, ? , ? , subset + [arr[n - 1]])
```

Listing 2: Algorithm 2 for Subset Sum

```
def subset_sum(numbers, target_sum):
n = len(numbers)
# Create a 2D table to store the results of subproblems
dp = [[False] * (target_sum + 1) for _ in range(n + 1)]

# Base case initialization (when sum is 0)
for i in range(n + 1):
    dp[i][0] = ?

# Fill the dp table
for i in range(1, n + 1):
    for j in range(1, target_sum + 1):
        # If the current number is greater than the target sum,
        # then the current number cannot be included in the subset
        if numbers[i - 1] > j:
            dp[i][j] = ?
        else:
            # Check if it's possible to achieve the target sum
            # by either including or excluding the current number
            dp[i][j] = dp[?][?] or dp[i - 1][j - numbers[i - 1]]

# Trace back to find the elements included in the subset sum
subset = []
i, j = n, target_sum
while i > 0 and j > 0:
    if dp[i][j] ? dp[i - 1][j]:
        subset.append(numbers[i - 1])
        j -= numbers[i - 1]
    i -= 1

return dp[n][target_sum], subset[::-1]
```

Exercise 1 *(5+10+5+10+10)*

1. *The algorithm in Listing 1 exhaustively tries all possible solutions using recursion. Fill up the "?" s of the given algorithm.*
2. *Enumerate all the solutions you get by implementing the algorithm: for each set and each target. Your implementation should print "No Subset Found with given sum" if there doesn't exist a given subset sum. Use any*

Python command to note down the time taken in execution of the algorithm for each case.

3. *The algorithm in Listing 2 uses dynamic programming to create a table to store the answers from previous cases. Fill up the "?" s of the given algorithm.*
4. *Solve for each set and each target using the algorithm in Listing 2. Your implementation should print "No Subset Found with given sum" if there doesn't exist a given subset sum. Use any Python command to note down the time taken in execution of the algorithm for each case.*
5. *Plot a graph of input size vs worst-case time taken for both the algorithms.*

2 Weighted Set Cover Problem

The Weighted Set Cover Problem is an extension of the Set Cover Problem, which we discussed in the last exercise. The Weighted Set Cover Problem introduces weights to the subsets, meaning each subset has an associated cost. The objective is to minimize the total cost of the selected subsets while ensuring that their union covers the entire universe.

Formally, the Weighted Set Cover Problem can be defined as follows:

- Given a universe U with elements $\{1, 2, \ldots, n\}$, a collection $S = \{S_1, S_2, \ldots, S_m\}$ of subsets of U, and weights $c_1, c_2, \ldots, c_m$ associated with each subset S_i, where c_i represents the cost of selecting subset S_i.
- Find a subset of S, denoted as S', such that the union of subsets in S' covers the entire universe U and the total cost of the selected subsets is minimized.

2.1 Problem Statement:

Eagles Airways needs to assign its crews to cover all its upcoming flights. We will focus on the problem of assigning its crews based in Mumbai to the flights listed in the first column of Table below. The other 9 columns show the 9 feasible sequences of flights for a crew. (The numbers in each column indicate the order of the flights.) Few of these sequences are needed to be chosen in such a way that every flight is covered. It is permissible to have more than one crew on a flight, where the extra crews would fly as passengers, but union contracts require that the extra crews would still need to be paid for their time as if they were working. The cost of assigning a crew to a particular sequence of flights is given (in ten thousands of INR) in the bottom row of the table. The objective is to minimize the total cost of the crew assignments that cover all the flights.

FLIGHTS	A	B	C	D	E	F	G	H	I
Mumbai to Chennai	1			1	1				
Mumbai to Guwahati		1				1		1	
Mumbai to Delhi			1				1		1
Delhi to Mumbai	4					5		5	5
Delhi to Chennai		3							2
Delhi to Kolkata			2				2		
Kolkata to Chennai			3				3		
Kolkata to Delhi	3					4		4	
Chennai to Mumbai		4	4	4					
Chennai to Kolkata	2					3		3	
Chennai to Guwahati				2	2		4		3
Guwahati to Delhi		2							4
Guwahati to Mumbai					3		5		
Guwahati to Chennai				3		2		2	
	45	45	45	45	35	55	55	60	60

Exercise 2 *(10+20+20+10)*

1. *Formulate this problem as an integer programming problem and solve it using a solver. Report the solution (the number of crews and assigned tours) and the cost.*

2. *Formulate it as a weighted set cover problem and solve it using a greedy algorithm. You may modify the code used in the last lab. Pseudocode for greedy algorithm is:*

Algorithm 1 Greedy algorithm for Weighted Set Cover

Require: Subsets $S_1.S_2...S_m$, costs $C_1, C_2, ...C_m$
1: Initialize $I = \{\}$
2: **while** $I \neq U$ **do**
3: : Find $S_i \in \{S_1, S_2....S_m\}$ such that $C_i/|(S_i - I)|$ is minimum.
4: : Add elements of above picked S_i to I, i.e., I = I $\cup S_i$
5: **end while**
6: **Output:** S_i which are picked.

3. *Suppose the flights from Chennai to Guwahati are cancelled and hence associated tours are no longer feasible. Find the new solution using:*

   (a) *IP formulation.*

   (b) *Greedy Algorithm.*

4. *Compare the solutions of 2.1, 2.2 and 2.3 given by the integer program and greedy algorithm in the following counts:*

   (a) *Running time.*

   (b) *Ratio of objective value of IP solution to Greedy solution.*

3 References

1. Subset Sum Problem :
   https://www.geeksforgeeks.org/subset-sum-problem-dp-25/
2. Weighted Set Cover Problem:
   https://www.geeksforgeeks.org/greedy-approximate-algorithm-for-set-cover-problem/