**AA\_LAB\_12\_Assignment**

**CE\_054**

**Aim :-** Implementation the Simplex Tabular Method of Linear Programming.

1. Simplex Tabular Method :-

Code :-

#Author : Dhruv B Kakadiya

import itertools

import collections

import numpy as np

class simplex\_tabular\_solver:

    def \_\_init\_\_(self, eq1, eq2, max\_eq):

        self.eq1 = eq1

        self.eq1.insert(len(self.eq1) - 1, 0)

        self.eq1.insert(len(self.eq1) - 1, 1)

        self.eq2 = eq2

        self.eq2.insert(len(self.eq2) - 1, 1)

        self.eq2.insert(len(self.eq2) - 1, 0)

        self.max\_eq = max\_eq + [0, 0]

        self.Cb = [0, 0]

        self.Xb = [len(self.eq1) - 3, len(self.eq1) - 2]

        self.ratio = [1, 1]

        self.Z = [0] \* 5

        self.ci\_sub\_zi = [0] \* (len(self.eq1) - 1)

        self.ci\_zi()

    def ci\_zi(self):

        self.Z = [(self.eq1[i] \* self.Cb[0] + self.eq2[i] \* self.Cb[1]) for i in range(len(self.eq1))]

        self.ci\_sub\_zi = [(self.max\_eq[i] - self.Z[i]) for i in range(len(self.Z) - 1)]

    def next\_turn(self):

        if max(self.ci\_sub\_zi) > 0:

            index\_enter = self.ci\_sub\_zi.index(max(self.ci\_sub\_zi))

        else:

            return (True)

        self.ratio[0] = (self.eq1[len(self.eq1)-1]/self.eq1[index\_enter]) if self.eq1[index\_enter] > 0 else -1

        self.ratio[1] = (self.eq2[len(self.eq2)-1]/self.eq2[index\_enter]) if self.eq2[index\_enter] > 0 else -1

        min\_ratio = min(self.ratio)

        if(min\_ratio > 0):

            index\_exit = self.ratio.index(min\_ratio)

        elif(max(self.ratio) > 0):

            index\_exit = self.ratio.index(max(self.ratio))

        else:

            return True

        print("\nEntering var : x%d:= %d, Exiting var : x%d:= %d " % (index\_enter, self.max\_eq[index\_enter], self.Xb[index\_exit], self.Cb[index\_exit]))

        self.Xb[index\_exit] = index\_enter

        self.Cb[index\_exit] = self.max\_eq[index\_enter]

        if(index\_exit == 0):

            self.eq1 = [(self.eq1[i]/self.eq1[index\_enter]) for i in range(len(self.eq1))]

            row\_key = self.eq2[index\_enter]

            self.eq2 = [(self.eq2[i] - row\_key\*self.eq1[i]) for i in range(len(self.eq2))]

            self.ci\_zi()

        else:

            self.eq2 = [(self.eq2[i]/self.eq2[index\_enter])

                         for i in range(len(self.eq2))]

            row\_key = self.eq1[index\_enter]

            self.eq1 = [(self.eq1[i] - row\_key\*self.eq2[i])

                         for i in range(len(self.eq1))]

            self.ci\_zi()

        return False

    def table\_printing(self):

        print("Cb\t", "Xb\t", "previous ratios:\t\t", self.max\_eq, ", rhs]")

        print(self.Cb[0], "\t", self.Xb[0], "\t", self.ratio[0], "\t\t\t\t", self.eq1)

        print(self.Cb[1], "\t", self.Xb[1], "\t", self.ratio[1], "\t\t\t\t", self.eq2)

        print("-\t", "-\tcj - zj\t\t\t\t", self.ci\_sub\_zi)

        print("\n")

        return ""

    def fully\_solved(self):

        finish = False

        while not finish:

            finish = self.next\_turn()

            self.table\_printing()

        print("After Solving the Value of function is :- ", self.Z[len(self.Z) - 1])

        print("using x%d : = %.1f, x%d : = %.1f " %(self.Xb[0], self.eq1[-1], self.Xb[1], self.eq2[-1]))

print("\nEquation in format of x1 x2 c1\n")

expression1 = list(map(int, input("Enter the equation - 1 :- ").split()))

expression2 = list(map(int, input("Enter the equation - 2 :- ").split()))

maxEquation = list(map(int, input("Maximization the following function :- ").split()))

ans\_partially = simplex\_tabular\_solver(expression1, expression2, maxEquation)

ans\_partially.fully\_solved()

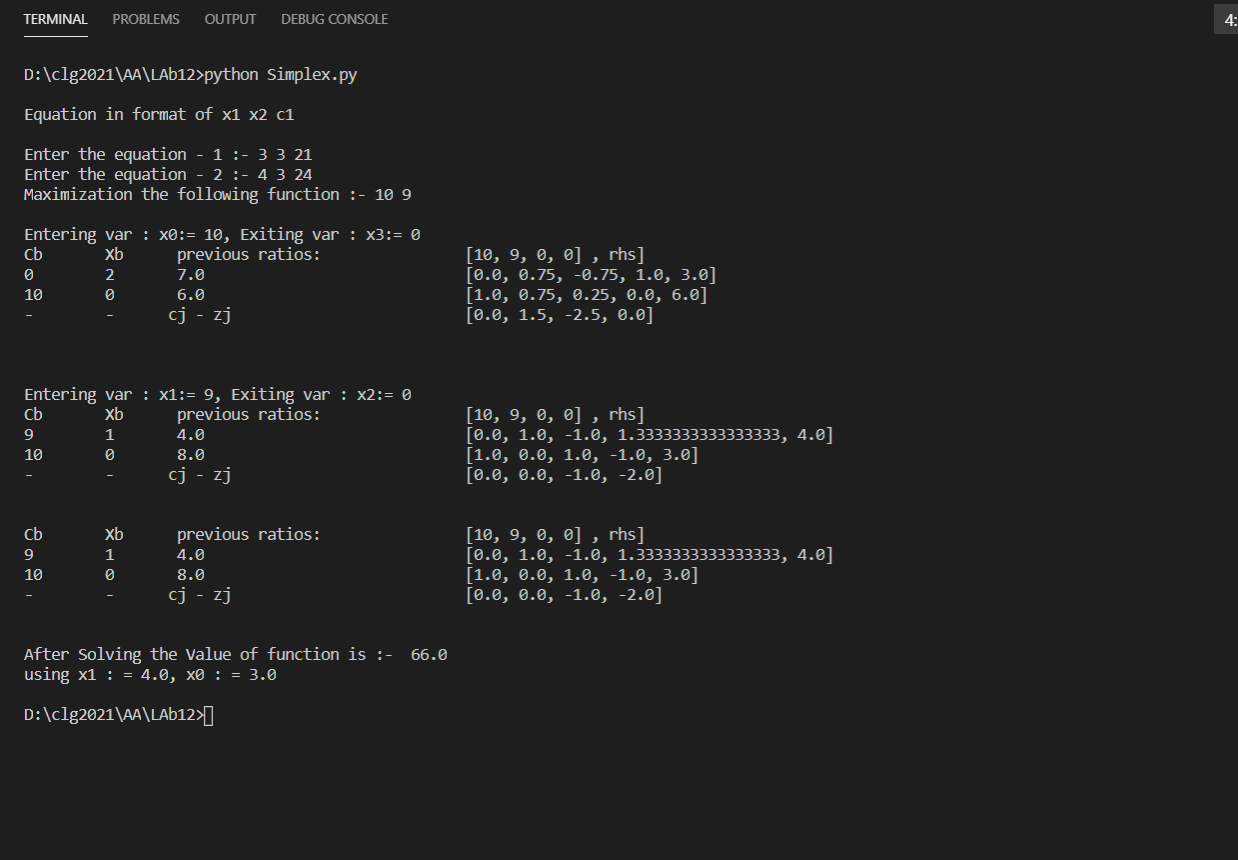
Output :-

Maximize => 10x1 + 9x2

Eq1 => 3x1 + 3x2 <= 21

Eq2 => 4x1 + 3x2 <= 24

Where x1, x2 >= 0



Maximize => 6x1 + 5x2

Eq1 => x1 + x2 <= 5

Eq2 => 3x1 + 3x2 <= 12

Where x1, x2 >= 0

