Name: Prashant Kumar
Roll No: 222CS2097
Semester: 1st (MTech)

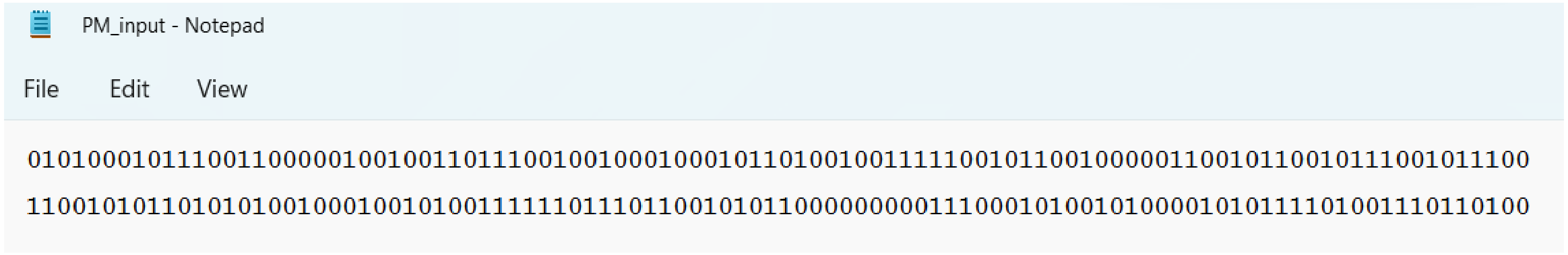
Q.18. String Matching

//stringMatching.py

```
# String Pattern Matching KMP Algorithm including regex pattern
import argparse
def match_pattern(text, pattern):
    positions = []
    text_length = len(text)
    pattern_length = len(pattern)
    position = 0
    for i, c in enumerate(text):
        if position < text_length - pattern_length and c == pattern[position]:
            offset = 0
            while(offset < pattern_length):
                if text[i + offset] == pattern[position + offset]:
                    offset = offset + 1
                else:
                    break
            if offset == pattern_length:
                positions.append(i)
            position = 0

    return positions
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Brute Force Pattern Matching')
    parser.add_argument('--case_sensitive', default = False, type=bool, help='Allow
case sensitive checking')
    args = parser.parse_args()

    case_sensitive = args.case_sensitive
    with open('PM_input.txt') as file:
        lines = file.readlines()
    text = ""
    text = text.join(lines)
    pattern = input('Enter pattern for checking\n')
    print("Text Length : {}".format(len(pattern)))
    print("Text Text : {}".format(len(text)))
    if not case_sensitive:
        text = text.lower()
        pattern = pattern.lower()
    res = pattern.find("*")
    if res > 0 :
        print("Initiating Regex Search")
        parts = pattern.split("*")
        first_position = match_pattern(text, parts[0])
        second_position = match_pattern(text, parts[1])
        if len(first_position)>0 or len(second_position)>0:
            for u in first_position:
                for v in second_position:
                    if v >= u + len(parts[0]):
                        print("Matched found at {} {}".format(u,v))
        else:
            print("Pattern not found")
    else:
        print("Initiating Normal Search")
        position = match_pattern(text, pattern)
        for p in position:
            print("Matched found at {}".format(p))
```

PM_input.txt



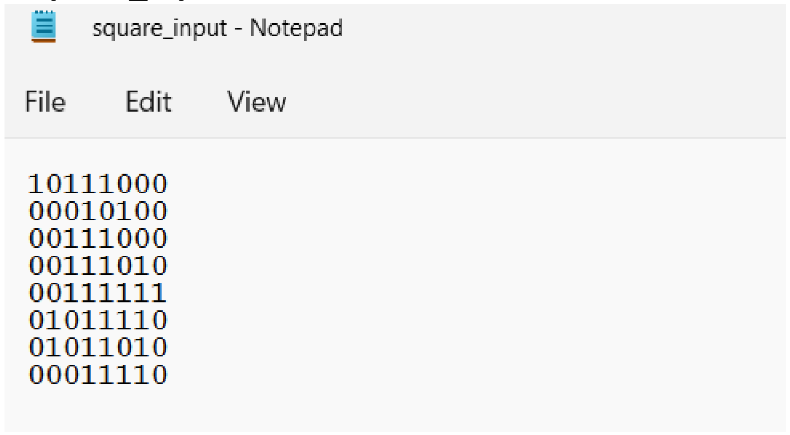
```
0101000101110011000001001001101110010010001000101101001001111100101100100000110010110010111001011100
1100101011010101001000100101001111110111011001010110000000001110001010010100001010111101001110110100
```

Sample output:

```
Enter pattern for checking

00110

Text Length : 5

Text Text : 100

Initiating Normal Search

Matched found at 12
Matched found at 25

Matched found at 74

Enter pattern for checking

1010*01101

Text Length : 10

Text Text : 100

Initiating Regex Search

Matched found at 40 54

Matched found at 46 54
```

Q.19. Pattern matching

#square_input.txt



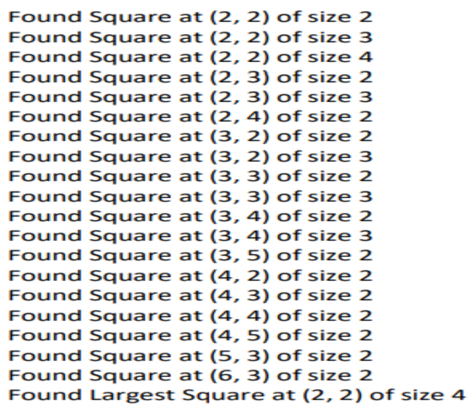
```
10111000
00010100
00111000
00111010
00111111
01011110
01011010
00011110
```

#SquarePatternMatchingAlgorithm.py
Code:

```
import numpy as np
class Square_Pattern:
    def __init__(self) -> None:
        pass

    def get_dimension(self, array):
        row_length = len(array)
        col_length = len(array[0])
        return row_length, col_length

    def display_position(self, positions):
            largest = 0
            for position in positions:
                print("Found Square at {} ofsize {}".format(position['position'], position['size']))
                if int(position['size']) > largest:
                    largest = position['size']
                    # print("Largest Square size is : {}".format(largest))
            return largest

    def show_largest(self, positions, largest):
        for position in positions:
            if position['size'] == largest:
                print("Found Largest Square at {} ofsize {}".format(position['position'],
position['size']))

    def check_square(self, array):
        # print(array)
        all_one = True
        length = len(array)
        for m in range(length):
            for n in range(length):
                if array[m][n] != "1":
                    all_one = False
                    break
            if not all_one:
                break
        return all_one

    def find_square(self, array, rows, cols):
        position = []
        for i in range(rows):
            for j in range(cols):
                e = 2
                if array[i][j] == "1":
                    while (i+e <= rows) and (j+e <= cols):
                        # print("{} <= {} and {} <= {}".format(i+e,rows,j+e,cols))
                        if self.check_square(array[i:i+e,j:j+e]):
                            position.append({'position' : "({}, {})".format(i, j), 'size' : e})
                            e = e + 1
                        else:
                            break
                    e = 2
                else:
                     continue

                return position
```

```python
if __name__ == ' __main__':
    array = []
    with open('square_input.txt','rb') as file:
        lines = file.readlines()
        print(lines)
        for line in lines:
            col = []
            for c in line.strip():
                col.append(c)
                array.append(col)
                array=np.array(array)
                matcher = Square_Pattern()

                rows, cols = matcher.get_dimension(array)
                if rows == cols:
                    positions = matcher.find_square(array, rows, cols)
                    largest = matcher.display_position(positions)
                    matcher.show_largest(positions, largest)
```

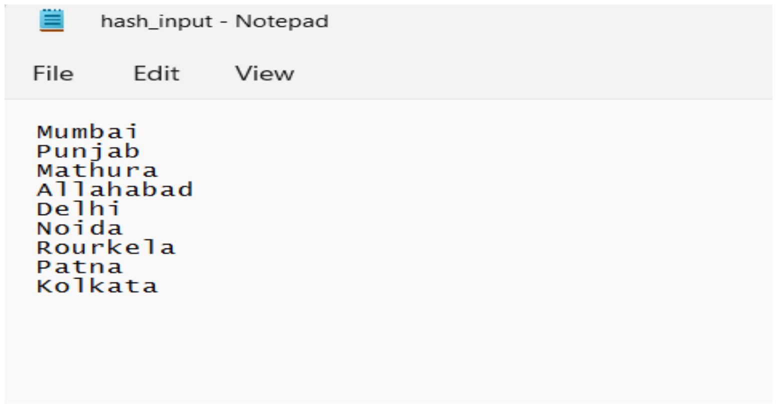
Output:



```
Found Square at (2, 2) of size 2
Found Square at (2, 2) of size 3
Found Square at (2, 2) of size 4
Found Square at (2, 3) of size 2
Found Square at (2, 3) of size 3
Found Square at (2, 4) of size 2
Found Square at (3, 2) of size 2
Found Square at (3, 2) of size 3
Found Square at (3, 3) of size 2
Found Square at (3, 3) of size 3
Found Square at (3, 4) of size 2
Found Square at (3, 4) of size 3
Found Square at (3, 5) of size 2
Found Square at (4, 2) of size 2
Found Square at (4, 3) of size 2
Found Square at (4, 4) of size 2
Found Square at (4, 5) of size 2
Found Square at (5, 3) of size 2
Found Square at (6, 3) of size 2
Found Largest Square at (2, 2) of size 4
```

Q. 20. Hash Table

#hash_input.txt

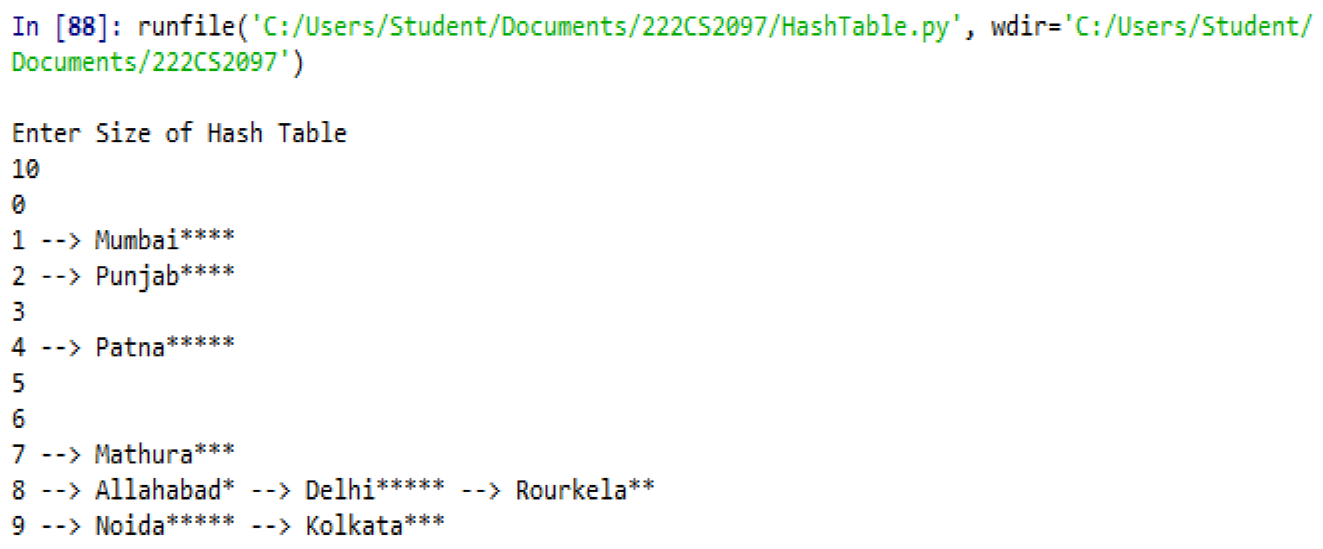


```
Mumbai
Punjab
Mathura
Allahabad
Delhi
Noida
Rourkela
Patna
Kolkata
```

#HashTable.py

```
import random, math
class SymblTable:
    def __init__(self, table_size):
        self.table_size = table_size
        self.HashTable = [[] for _ in range(table_size)]

    def display_hash(self):
        for i in range(len(self.HashTable)):
            print(i, end = " ")

            for j in self.HashTable[i]:
                print("-->", end = " ")
                print(j, end = " ")
            print()


    def Hashing(self, keyvalue):
        k = (math.sqrt(5)-1)/2
        fraction, _ = math.modf(k*keyvalue)
        hashvalue = math.floor(self.table_size * fraction)
        return hashvalue

    def insert(self, keyvalue, value):
        hash_key = self.Hashing(keyvalue)
        self.HashTable[hash_key].append(value)


with open('hash_input.txt','r') as file:
    lines = file.readlines()
    #print(lines)
    text = ''.join(lines)
    #print(text)
    tokens = text.split()


size = (input('Enter Size of Hash Table\n'))
hash = SymblTable(int(size))

modified = []
for token in tokens:
    if len(token)>10:
        modified.append(token[:10])
    elif len(token)<10:
        extra = ''.join(random.choices('*', k = 10-len(token)))
        modified.append(token + extra)

for token in modified:
    ascii_sum = sum([ord(c) - 96 for c in token])
    hash.insert(ascii_sum, token)
hash.display_hash()
```

Output:



```
In [88]: runfile('C:/Users/Student/Documents/222CS2097/HashTable.py', wdir='C:/Users/Student/
Documents/222CS2097')

Enter Size of Hash Table
10
0
1 --> Mumbai****
2 --> Punjab****
3
4 --> Patna*****
5
6
7 --> Mathura***
8 --> Allahabad* --> Delhi***** --> Rourkela**
9 --> Noida***** --> Kolkata***
```

==========================End==========================