Assignment-8

ELP-780 Software Lab

Sakshi Agrawal
2017EET2291
2017-19

A report presented for the assignment on
Python.

Department of Electrical Engineering
IIT DELHI
India

September 27, 2018

Contents

1 Problem Statement-1

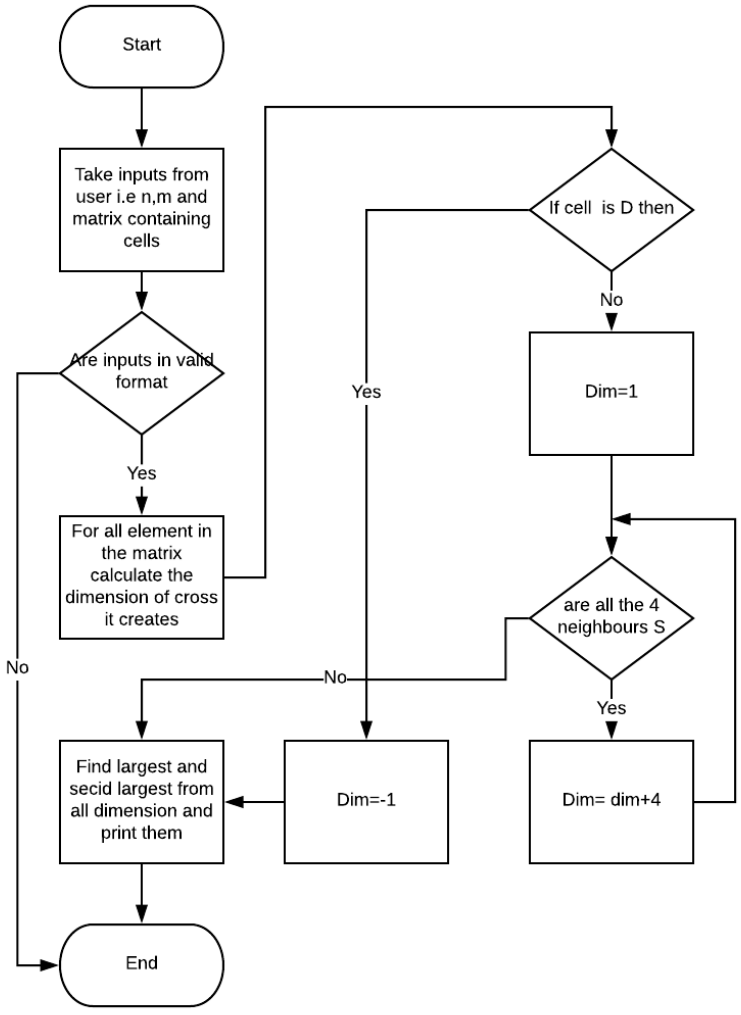
1.1 Problem statement

- Find the two largest valid crosses that can be drawn on smart cells in the grid, and return two integers denoting the dimension of the each of the two largest valid crosses.
- The two crosses cannot overlap, and the dimensions of each of the valid crosses should be maximal.

1.2 Assumptions

- 2 <= n <= 105
- 2 <= m <= 105

1.3 Program structure



1.4 Algorithm and Implementation

- Take input from user and check for valid inputs
- For each cell in the input find its dimension
- If the cell is 'D', its dimension is -1
- else if cell is 'S' its dimension is 1
- If all four neighbouring cell are also 'S' then increase dimension by 4
- Check for next 4 pair and continue until you get a 'D' or end of matrix
- From all dimensions find largest and second largest and print them.

1.5 Input and output format

- **Input Format**

  The first line contains two space-separated integers, n and m. Each of the next lines n contains a string of m characters where each character is either S (Smart) or D (Dull). These strings represent the rows of the grid. If the jth character in the ith line is S, then (i,j) is a cell smart. Otherwise it's a dull cell.

- **Output Format**

  Find two valid crosses that can be drawn on smart cell of the grid, and return the dimension of both the crosses in the reverse sorted order(i.e. First Dimension should be the larger one and other should be smaller one).

1.6 Test cases

- **Sample Input 1**

  5 6

  SSSSSS

  SDDDSD

  SSSSSS

  SSDDSD

  SSSSSS

  Sample Output 1

  5 1

- **Sample Input 2**

  6 6

  DSDDSD

  SSSSSS

DSDDSD

SSSSSS

DSDDSD

DSDDSD

Sample Output 2

5 5

- **Sample Input 3**

  5 9

  SSSSDSDDD

  DDSDDDDDD

  SSSSSDDDD

  DDSDDSDDD

  DSSSDDDDD

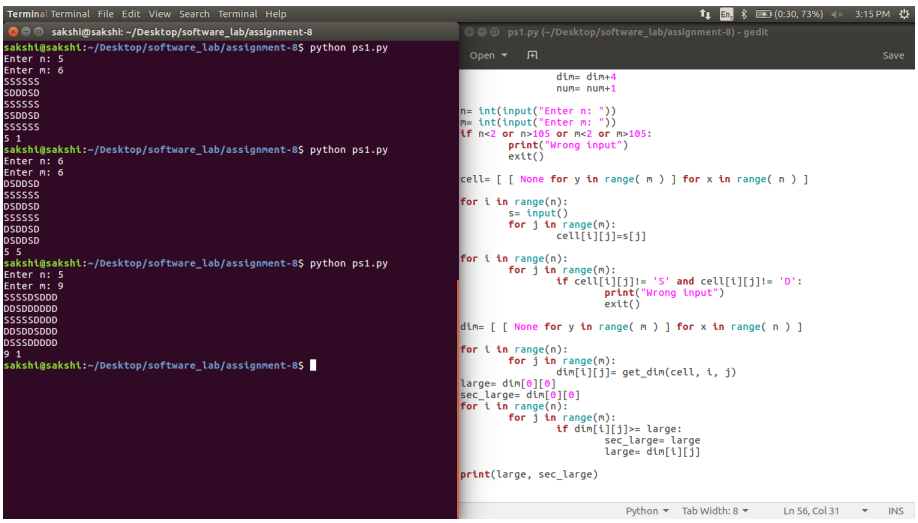
  Sample Output 3

  9 1

1.7 Difficulties/Issues faced

- In getting non overlapping crosses.

1.8 Screenshots



2 Problem Statement-2

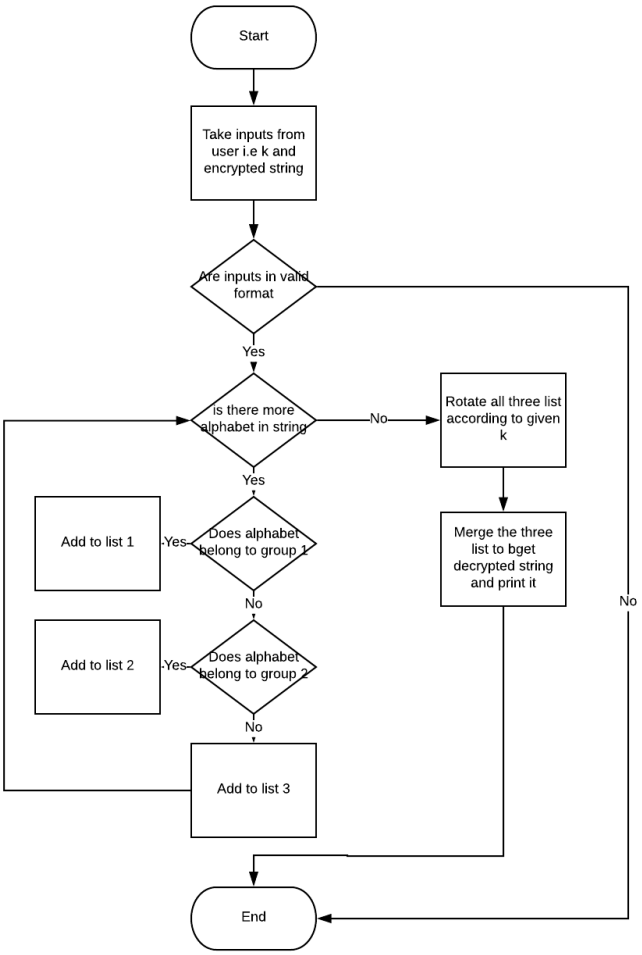
2.1 Problem statement

- Given an Encrypted data we have to decrypt it.
- Encryption of a message requires three keys, k1, k2, and k3.
- The 26 letters of English and underscore are divided in three groups, [a-i] form one group, [j-r] a second group, and everything else ([s-z] and underscore) the third group.
- Within each group the letters are rotated left by ki positions in the message. Each group is rotated independently of the other two.
- Decrypting the message means doing a right rotation by ki positions within each group.

2.2 Assumptions

- 1 <= Length of the string <= 150
- 1 <= ki <= 150 (i=1,2,3)

2.3 Program structure



2.4 Algorithm and Implementation

1. Take input from users and check for proper format
2. Break the entire string into three lists, each list containing elements of particular group.
3. Rotate all three list independently according to the ki
4. Merge the three list in the same order in which they were created.
5. Display the decrypted message.

2.5 Input and output format

- **Input Format**
  - All input strings comprises of only lowercase English alphabets and underscores(_).

- **Output Format**
  - For each encrypted message, the output is a single line containing the decrypted string.

2.6 Test cases

- **Sample Input 1:**

  2 3 4

  dikhtkor_ey_tec_ocsusrsw_ehas_

  Sample Output 1 :

  hardwork_is_the_key_to_success

- **Sample Input 2:**

  1 1 1 bktcluajs

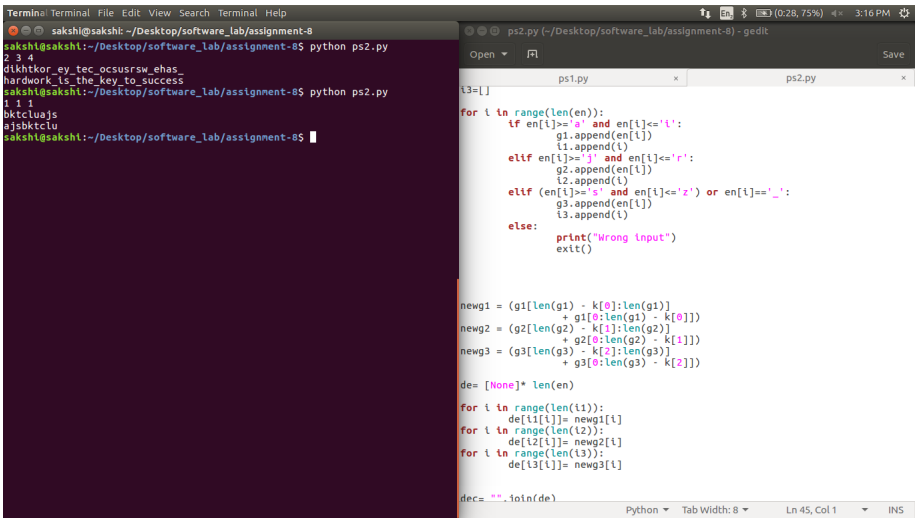
  Sample Output 2:

  ajsbktclu

2.7 Difficulties/Issues faced

- In case insensitive search.

2.8 Screenshots



3 Appendix

3.1 Appendix-A Code for ps1

```
###### this is the first .py file ##########

####### write your code here ##########

####### Code for problem 1 ######

import numpy as np

#function to find dimension of each cell
def get_dim(cell, i, j):
  #vectors that search in all 4 direction
  x= [-1, 0, 1, 0]
  y= [0, -1, 0, 1]
  dim=-1
  #getting shape of the matrix
  n, m = np.array(cell).shape
  #if the cell is dull cell then its dimension is -1
  if cell[i][j]== 'D':
    return dim
  #else dimension is 1
  else:
    dim=1
  num=1
  #searching for largest cross around the cell
  while True:
    #checking for all 4 neighbour
    for search in range(4):
      newi= i+num*x[search]
      newj= j+num*y[search]
      #if any of the for neighbour is missing or any one is a dull cell then return with the previous dimension
      if newi<0 or newi >=n or newj<0 or newj >=m or cell[newi][newj]== 'D':
        return dim

    #else dimension is increased by 4
    dim= dim+4
    num= num+1

#getting input from user
n= int(input("Enter n: "))
m= int(input("Enter m: "))
#checking for boundry condition
if n<2 or n>105 or m<2 or m>105:
  print("Wrong input")
  exit()
#getting the cells from user
cell= [ [ None for y in range( m ) ] for x in range( n ) ]

for i in range(n):
  s= input()
  for j in range(m):
    cell[i][j]=s[j]
```

```

#checking if the cells are valid
for i in range(n):
  for j in range(m):
    if cell[i][j]!= 'S' and cell[i][j]!= 'D':
      print("Wrong input")
      exit()
#list to store dimension of all the cells
dim= [ [ None for y in range( m ) ] for x in range( n ) ]


#getting the dimension of all cell one by one
for i in range(n):
  for j in range(m):
    dim[i][j]= get_dim(cell, i, j)

#to get largest and second largest cross
large= dim[0][0]
sec_large= dim[0][0]
#searching in the entire matrix, largest and second largest
for i in range(n):
  for j in range(m):
    if dim[i][j]>= large:
      sec_large= large
      large= dim[i][j]

#displaying the result
print(large, sec_large)
```

3.2 Appendix-B Code for ps2

```
###### this is the second .py file ##########

####### write your code here ##########

######Code for problem 2######

import numpy as np

#Getting input from user
k= (input().split(" "))

#checking for boundry conditions
if len(k)!=3:
  print("Wrong number of arguments")
  exit()
for i in range(3):
  k[i]= int(k[i])
  if k[i]<1 or k[i]>150:
    print("Wrong input")
    exit()

#getting encrypted message from user
en= input()
if len(en) <1 or len(en)>150:
  andprint("Wrong input")
  exit()
```

```
27
28
29 #list to store alphabet of all three groups and their index
30 g1=[]
31 g2=[]
32 g3=[]
33 i1=[]
34 i2=[]
35 i3=[]
36
37 #for entire message
38 for i in range(len(en)):
39   #if it lies in 1st group the add to list one
40   if en[i]>='a' and en[i]<='i':
41     g1.append(en[i])
42     i1.append(i)
43   #if it lies in 2nd group the add to list two
44   elif en[i]>='j' and en[i]<='r':
45     g2.append(en[i])
46     i2.append(i)
47   #if it lies in 3rd group the add to list three
48   elif (en[i]>='s' and en[i]<='z') or en[i]=='_':
49     g3.append(en[i])
50     i3.append(i)
51   #else the message contains some wrong character
52   else:
53     print("Wrong input")
54     exit()
55
56
57
58 #rotating list g1 by k1
59 newg1 = (g1[len(g1) - k[0]:len(g1)]
60                  + g1[0:len(g1) - k[0]])
61 #rotating list g2 by k2
62 newg2 = (g2[len(g2) - k[1]:len(g2)]
63                  + g2[0:len(g2) - k[1]])
64 #rotating list g3 by k3
65 newg3 = (g3[len(g3) - k[2]:len(g3)]
66                  + g3[0:len(g3) - k[2]])
67
68 #to store decrypted message
69 de= [None]* len(en)
70
71 #merging all the three list
72 #merging list one
73 for i in range(len(i1)):
74   de[i1[i]]= newg1[i]
75 #merging list two
76 for i in range(len(i2)):
77   de[i2[i]]= newg2[i]
78 #merging list three
79 for i in range(len(i3)):
80   de[i3[i]]= newg3[i]
81
82
```

```
#joining the list to form a string
dec= "".join(de)
#dsiplaying the decrypted message
print(dec)
```

References

[1] 'Python 3.7.1rc1 documentation'. Available: https://docs.python.org/3/.

[2] 'Git tutorial'. Available: https://www.atlassian.com/git/tutorials.

[3] 'Python List/Array Methods'. Available: https://www.w3schools.com/python/python_ref_list.asp.

[4] 'Python Dictionaries'. Available: https://www.w3schools.com/python/python_dictionaries.asp.

[5] 'Python Strings'. Available: https://www.w3schools.com/python/python_strings.asp.