test_1_notebook

July 14, 2021

1 Test 1

1.1 D1

You are given a rectangular field T with R rows and C columns. Each cell of T contains one of two possible characters: ’.’ or ’#’ (’.’ marks a passable cell, and ’#’ marks an impassable cell). Determine if there a passable cell (r , c) , such that one can visit all passable cells starting from (r , c) and moving each time only to one of the neighbouring passable cells (a neighbouring cell of (x , y) is one of four cells sharing a common side with (x , y)).

```
1. for r, c in [1..R]x[1..C]:

2.  if T[r][c] == '#':
3.      continue
4.  q - empty queue
5.  visited - empty RxC boolean 2D array, initialized with False
6.  visited[r][c] = True
7.  q.push((r,c)) // push starting cell into the queue
8.
9.  while q is not empty:
10.     cur_r, cur_c = q.pop() // extract the first element from the queue
11.     for (dr, dc) in [(+1, 0), (-1, 0), (0, +1), (0, -1)]:
12.         next_r = cur_r + dr
13.         next_c = cur_c + dc
14.         if (next_r, next_c) in [1..R]x[1..C] and T[next_r][next_c] == '.':
15.             visited[next_r][next_c] = True
16.             q.push((next_r, next_c))
17.
18. all_passed = True
19. for (check_r, check_c) in [1..R]x[1..C]:
```

```
20.     if T[check_r][check_c] == '.' and visited[check_r][check_c] == False:
21.         all_passed = False
22.
23. if all_passed:
24.     return True // (r, c) is a good starting cell
25.
26. return False // there is no good starting cell
```

In this question you are required to write code in the window below implementing a solution to the problem presented in the previous question (D1). #### Input format The first input line contains two integers R and C (1 R 2 0 , 1 C 2 0), the number of rows and columns. Each of the R following lines contains exactly C symbols '.' and '#'. It is guaranteed that T contains at least one passible cell. #### Output format Print a single word Yes if there is an appropriate cell or No otherwise. Examples are provided below.

[1]:
```python
import numpy as np
def allcellspassed(maze, rows, cols):
    for r in range(1, rows+1):      #This loop changes the starting node.
        for c in range(1, cols+1):
            if maze[r][c] == "#":
                continue

            q = []  #We make a Queue to keep track of the current node, as well␣
↪as all neighboring nodes.
            visited = np.zeros((rows +2, cols +2))  #We will also make a␣
↪boolean numpy array, to track each node and if it equals True or False␣
↪(visited or not).
                                                    #need double brackets to␣
↪keep within first arg.
            visited[r,c] = 1
            q.append((r,c))

            while len(q) !=0: #While there are more neighboring nodes..
                cur_r, cur_c = q.pop() #make the next node in q the current␣
↪node, by removing it from the q.
                for dr, dc in [(1, 0), (-1, 0), (0, 1), (0, -1)]: #check all␣
↪possible directions
                    next_r = cur_r +dr
                    next_c = cur_c + dc

                    if maze[next_r][next_c] == '.' and not visited[next_r,␣
↪next_c]: #Validate the next move, only move if the node is '.', and has␣
↪already been visited.
```

```
                        visited[next_r, next_c] = 1  #Mark the node as␣
 ↪"visited"
                        q.append((next_r, next_c))  #add the visited node to␣
 ↪the q. This will then be checked for it's neighbords once it's turn in the q␣
 ↪comes up.
            #All possible paths have been checked and validated. Now, we must␣
 ↪check if all of the '.'s have been passed through.

    #This will check every 0 and 1 in visited. if there is any '.' in maze that␣
 ↪doesn't = 1 (visited), then we will return false.

            true_count = np.sum(visited)
            period_count = sum(1 for x in maze for y in x if y == '.')
            if true_count == period_count:
                return "Yes"

    return "No"
file = open(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt1\d_input.txt').
 ↪read().strip().split('\n')
print(file)
rows, cols = map(int, file.pop(0).split())
#The following makes a border wall around our array. This will be used to check␣
 ↪if th neighboring nodes of each node are valid.
maze = []
maze.append("#" * (cols + 2))
for row in file:
    maze.append("#" + row + "#") #readline will iterate to the next unread line␣
 ↪of file.
maze.append("#" * (cols + 2))
print(allcellspassed(maze, rows, cols))
```

```
['3 3', '..#', '.#.', '#..']
No
```

1.2 D3: Optimize

The pseudocode given in D1 has sub-optimal complexity. Optimize your solution for the previous problem (D2) if necessary, allowing it to run efficiently. Your solution will run on grids with larger size constraints: 1 R 1 0 0 0 , 1 C 1 0 0 0 .

[2]:
```
import numpy as np
def allcellspassed(maze, rows, cols):
    for r in range(1, rows+1):
        for c in range(1, cols+1):
            if maze[r][c] == "#":
                continue
            q = []
            visited = np.zeros((rows +2, cols +2)) #need double brackets to␣
↪keep within first arg.
            q.append((r,c))

            while len(q) !=0: #While there are more neighboring nodes..
                cur_r, cur_c = q.pop()
                for dr, dc in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                    next_r = cur_r +dr
                    next_c = cur_c + dc
                    if maze[next_r][next_c] == '.' and not visited[next_r,␣
↪next_c]:
                        visited[next_r, next_c] = 1
                        q.append((next_r, next_c))
###This is the optimized section. The code would unnecessarily keep switching␣
↪all_passed from true to false on every iteration.
###Here, we don't have to keep saving a variable.
            for check_r in range(1, rows + 1):
                for check_c in range(1, cols + 1):
                    if maze[check_r][check_c] == '.' and visited[check_r,␣
↪check_c] == 0:
                        return 'No'
            return 'Yes'

    ##Alternate method to check for untraversed '.'s
    #         true_count = np.sum(visited) #This numpy function counts trues in␣
↪the array.
    #         period_count = sum(1 for x in maze for y in x if y == '.')
    #         if true_count == period_count:
    #             return True
    # return False
file = open(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt1\d_input.txt').
↪read().strip().split('\n')
print('input: ', file)
rows, cols = map(int, file.pop(0).split())
maze = []
maze.append("#" * (cols + 2))
for row in file:
    maze.append("#" + row + "#")
maze.append("#" * (cols + 2))

print(allcellspassed(maze, rows, cols))
```

```
input:  ['3 3', '..#', '.#.', '#..']
No
```

1.3 E1

To solve this problem you should submit a result file after processing an input file.

You are given a file article.txt. Your task is to parse text:

split the text into sentences by the full stop symbol '.'. split the sentences into words (only latin alphabet letters), please replace all symbols other than latin alphabet letters with space. change all uppercase letters to lowercase. #### Input format article.txt #### Output format The output file should contain the parsed sentences, each sentence on a separate line. The words in each sentence should be separated by single spaces. The should be no full stops at the ends of sentences.

Sample Text (this is all one long line on the first line of the text document.) Nigel Reuben Rook Williams (15 July 1944 21 April 1992) was an English conservator and expert on the restoration of ceramics and glass. From 1961 until his death he worked at the British Museum, where he became the Chief Conservator of Ceramics and Glass in 1983. There his work included the successful restorations of the Sutton Hoo helmet and the Portland Vase.

[3]:
```python
import re
import pandas as pd
file = open(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt1\e1_article.txt')
file = file.read().lower().split('.')

for line in file:
    new = ' '.join(re.findall("[a-z]+", line))
    #Output is too long to print on paper
```

sample answer: nigel reuben rook williams july april was an english conservator and expert on the restoration of ceramics and glass from until his death he worked at the british museum where he became the chief conservator of ceramics and glass in there his work included the successful restorations of the sutton hoo helmet and the portland vase joining as an assistant at age williams spent his entire career and most of his life at the british museum he was one of the first people to study conservation not yet recognised as a profession and from an early age was given responsibility over high profile objects in the s he assisted with the re excavation of the sutton hoo ship burial and in his early to mid twenties he conserved many of the objects found therein most notably the sutton hoo helmet which occupied a year of his time he likewise reconstructed other objects from the find including the shield drinking horns and maplewood bottles

1.4 E2

You should delete from each sentence all stop words 'a', 'the', 'of', 'at', 'in', 'and', 'to' (for this problem), and find the most frequent word, 2-gram (two consecutive words) and 3-gram (three consecutive words).

Input format Sentences one per line, all words are separated by single spaces and lowercased. #### Output format Print three lines: The most frequent word in the article. The most frequent 2-gram in the article. The most frequent 3-gram in the article. It is guaranteed that there is only one possible answer.

[4]:

```
#Upload article file, and delete the stop words.
article = open(r'C:
↪\Users\sfrie\Ydata\ydata_test_1\test1_attempt1\E1_Sample_Test1.txt')
working_file = article

stop_words = ['a', 'the', 'of', 'at', 'in', 'and', 'to']

working_file = working_file.read().split()
#print(working_file)
for x in working_file:
    if x in stop_words:
        working_file.remove(x)
for x in working_file:
    if x == 'the':
        working_file.remove(x)

#Count the most frequent word
max_word = max(working_file, key=working_file.count)
print(max_word)
# print(working_file)
# print(max_word)

#Create a dictionary to keep track of every gram, with its counts
gram_2 = {}
gram_3 = {}
#create three variables, as the current three words
word_1 = 0
word_2 = 1
word_3 = 2
length = len(working_file)

#Count the most frequent 2gram
for x in working_file:
    while word_3 < len(working_file):
        two_gram = ''.join([working_file[word_1], ' ', working_file[word_2]])
        if two_gram not in gram_2:
            gram_2[two_gram] = 1
        else:
            gram_2[two_gram] +=1
#Most frequent 3gram
        three_gram = ''.join([working_file[word_1], ' ', working_file[word_2],␣
↪' ', working_file[word_3]])
        if three_gram not in gram_3:
            gram_3[three_gram] = 1
        else:
            gram_3[three_gram] += 1
        word_1 +=1
        word_2 +=1
        word_3 +=1
print(max(gram_2, key = gram_2.get))
print(max(gram_3, key = gram_3.get))
```

```
williams
british museum
sutton hoo helmet
```

1.5 E3

You should delete from each sentence all stop words 'a', 'the', 'of', 'at', 'in', 'and', 'to' (for this problem), and find two statistics for each word w that occurs at least once in the file:

number of sentences which contains word w ; an average length of sentence containing word w (if some sentence contains w more than once it should be counted once). #### Input format Sentences one per line, all words are separeted by single spaces and lowercased. #### Output format Print several lines (one line per word, separate items with single space): Word from the input file. Number of sentences. Average length of the sentence. Words can be listed in arbitrary order.

```
[ ]: article = open(r'C:
      ↪\Users\sfrie\Ydata\ydata_test_1\test1_attempt1\E1_Sample_Test1.txt')

     stop_words = ['a', 'the', 'of', 'at', 'in', 'and', 'to']
     working_file = []
     for sentence in article:
         working_sentence = []
         sentence = sentence.split()
         for word in sentence:
             if word not in stop_words:
                 working_sentence.append(word)
         working_file.append(working_sentence)
     #Make a list of unique words
     unique_words = []
     for sentence in working_file:
         for y in sentence:
             if y not in unique_words:
                 unique_words.append(y)

     #Count number of sentences which contain the word
     #dict = {}
     for word in unique_words:
         count = 0
         average_list = []
         for sentence in working_file :
             if word in sentence:
```

```
            count+=1
            length = len(sentence)
            average_list.append(len(sentence))
    average_length = sum(average_list)/ len(average_list)
    #dict[word] = [count, average_length]
    print(word, count, average_length)
    #Output is too long to print
```

1.6 F1

[5]:
```
import pandas as pd
import numpy as np
import os

df = pd.read_csv(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt2\titanic (2).
 ↪csv')
df.head()
```

[5]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

[6]:
```
df = df.drop(['Ticket', 'Cabin'], axis = 1)
df['Age'] = df['Age'].fillna(df['Age'].mean())
df.head(4)
```

[6]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
```

```
3            4          1       1

                                              Name     Sex   Age  SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th…  female  38.0      1
2                               Heikkinen, Miss. Laina  female  26.0      0
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1

   Parch     Fare Embarked
0      0   7.2500        S
1      0  71.2833        C
2      0   7.9250        S
3      0  53.1000        S
```

[7]:
```
df.to_csv('titanic_new.csv', index = False)
```

1.7 F2

In this problem you need to upload a source file solving the following problem: In the working directory there is a file titanic.csv as a result described in F1. Implement function similarity from the pseudocode and print the result of this function for some passenger pairs.

function similarity(passenger_a, passenger_b):
result = 0
if passenger_a.pclass = passenger_b.pclass:
result += 1
if passenger_a.sex = passenger_b.sex:
result += 3
if passenger_a.sibsp = passenger_b.sibsp:
result += 1
if passenger_a.parch = passenger_b.parch:
result += 1
result += max(0, 2 - abs(passenger_a.age - passenger_b.age) / 10.0)
result += max(0, 2 - abs(passenger_a.fare - passenger_b.fare) / 5.0)
return result / 10.0 #### Input format The first line of the input contains one integer q (1 q 1 0 0), a number of queries. It is followed by q lines containing each two integers i d 1 and i d 2 (1 i d 1 8 9 1 , 1 i d 2 8 9 1), corresponding to the i d of two passengers to be compared. #### Output format Output exactly q lines. In the i -th one print the result of i -th query. Round the result value to 2 digits after the decimal point.

[8]:
```
import pandas as pd
import numpy as np


# %%
input = open(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt2\input_f1.txt').
 ↪readlines()
input = [x.split() for x in input]
```

```python
input.pop(0)


# %%
df = pd.read_csv(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt2\titanic_new.
↪csv')


# %%
def similarity(id1, id2):
    result = 0
    passenger_a = df.loc[id1-1]
    passenger_b = df.loc[id2-1]
    if passenger_a.Pclass == passenger_b.Pclass:
        result += 1
    if passenger_a.Sex == passenger_b.Sex:
        result += 3
    if passenger_a.SibSp == passenger_b.SibSp:
        result += 1
    if passenger_a.Parch == passenger_b.Parch:
        result += 1
    result += max(0, 2 - abs(passenger_a.Age - passenger_b.Age) / 10.0)
    result += max(0, 2 - abs(passenger_a.Fare - passenger_b.Fare) / 5.0)
    return result / 10.0


for x in input:
    id1 = int(x[0])
    id2 = int(x[1])
    print(similarity(id1, id2))
```

```
1.0
0.24
```

1.8 F3

Given a passenger's ID you need to find another passenger such that the result of similarity function described in F2 reaches maximal value. Note! Please round up to 2 decimal digits before comparing results of similarity function.

Input format The first line of the input contains one integer q (1 q 1 0 0), a number of queries. It is followed by q lines containing each one integer i d (1 I D 8 9 1), corresponding to the i d of the target passenger. #### Output format Output exactly q lines. In the i -th one print the i d of the passenger most similar to the one from the i -th query. If there are several applicants of the same similarity, then choose one the minimal I D .

```
[9]: import pandas as pd
     import numpy as np

     # create list of lists of the input file.
     ## (could have done this with just one list)
     input = open(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt2\input_f1.txt').
      ↪readlines()
     input = [x.split() for x in input]
     input.pop(0)
     df = pd.read_csv(r'C:\Users\sfrie\Ydata\ydata_test_1\test1_attempt2\titanic_new.
      ↪csv')
     df.head()
     #The original pseudocode called for calling on the dataframe each time the␣
      ↪function was called.
     # This made the script run for too long. Therefore, I changed the function, and␣
      ↪the later code, to
     # read in the data frame as a list of lists. I changed the pseudocode to read␣
      ↪that list of listsm as opposed to
     # reading the data frame each time.
     def similarity(id1, id2):
         result = 0
         passenger_a = all_rows[id1-1]
         passenger_b = all_rows[id2-1]
         #Keep in mind, that this is calling on the new, edited excel file, not the␣
      ↪original.
         # In teh new one, we deleted columns from the old, so the column indexes␣
      ↪will be different.
         # for example, index of "Fare" is now 8, not 9.
         if passenger_a[2] == passenger_b[2]:
             result += 1
         if passenger_a[4] == passenger_b[4]:
             result += 3
         if passenger_a[6] == passenger_b[6]:
             result += 1
         if passenger_a[7] == passenger_b[7]:
             result += 1
         result += max(0, 2 - abs(float(passenger_a[5]) - float(passenger_b[5])) /␣
      ↪10.0)
         result += max(0, 2 - abs(float(passenger_a[8]) - float(passenger_b[8])) / 5.
      ↪0)
         return result / 10.0

     #Make list of lists of each row, to iterate over (to find id2).
     all_rows = []
     for x in df.index:
        all_rows.append(list(df.iloc[x]))
```

```
for x in input:
    id1 = int(x[0])
    best_score = 0
    score_id = 0
    for row in all_rows:
        id2 = row[0]
        if id1 != id2:
            score = round(similarity(id1, id2), 2)
            #If the similarity score is the current highest, make it the best␣
 ↪score.
            ## This also allows for, in case of a tie, to ensure that the␣
 ↪smallest index is called (assuming that the passengers are in numerical␣
 ↪order)
            if score > best_score:
                best_score = score
                score_id = id2
    print(score_id)
```

```
665
665
```