COMP7107 Assignment3 README

3035021742 Zhang Zhuan

General Info:

- The programs is to design a top-k algorithm to calculate the total score of restaurants in round-robin way and find the top-k highest scores without reading all the data from seq1.txt and seq2.txt. The algorithm is carried out with random access
- Written in Jupyter Notebook/ Python 3.6.
- To run the data please prepare data files including: three score files, one is sorted by ID and the other two are sorted by scores.
- Python Library applied: NumPy, linecache and math

1. Receive the input of top-k; read the score_rnd.txt file ; function of read the seq1 and seq2 files by line.

```
# command line input top-k
k = int(input('please input the number of results required:'))
```

```
please input the number of results required:5
```

```
# read file rnd.txt and build an array for further index
score_rnd = []
with open('rnd.txt','r') as file:
    for line in file:
        new_line = line.split(' ')
        new_line_float = list(map(float, new_line))
        score_rnd.append(new_line_float[1])

score_rnd = np.array(score_rnd)
```

```
# function read files seq1.txt and seq2.txt by line
def readtxtline (filename, a):
    new_line = lc.getline(filename, a).strip().split(' ')
    seq_id = int(new_line[0])
    score = float(new_line[1])
    return [seq_id, score]
```

2. Set empty list for further loop to enter and update total scores from the three files. At the meantime, collect the IDs of the objects have been seen currently.

```
# set info of filename for readtxtline function
fn1 = 'seq1.txt'
fn2 = 'seq2.txt'

# set empty list for the further loop to enter and update total
total_score = []

# empty list to collect all the objects' IDs have been seen
IDs = []

# the top-k results
Wk = []
```

LARA algorithm:

- The objects in seq1 and seq2 are scanned in a sequential way. Tsum works as a general upper bound score, whose calculation method in the program is 5.0 (the highest score in score_rnd.txt) plus the latest values reading from seq1.txt and seq2.txt.
- When the T <= t (which is the score of the current k-th object), the growing phase of the scan is over. There is no unseen objects could surpass the current top-k objects.
- Then the scan enters the shrinking phase, when we need to check if there are any upper bound scores of existing objects in the list can surpass the current top-k objects.

3. The scan loop to add objects into the total_score[] list.

```
# the scan loop to add objects into the total_score[] list

for j in range(len(score_rnd)):
    o1 = readtxtline(fn1,j+1) # read seq1.txt by line
    o2 = readtxtline(fn2,j+1) # read seq2.txt by line
    id_o1 = o1[0]
    id_o2 = o2[0] # IDs of the objects to be added into IDs[]
```

Check if the ID of the current object is already read, then calculate the total score. If the ID is new, then calculate the lower bound score and add the ID into list, for further check

[id, lb score or total score, score type (1,2,3)]
1 stands for object firstly read from seq1, 2 stands for object firstly read from seq2, 3 stands for object with total score

```
#check if the ID of the current object is already read, then ca
# if the ID is new, then calculate the lower bound score
# and add the ID into list, for further check
# [id, lb score or total score, score type]
## 1 stands for object firstly read from seq1, 2 stands for obj
    if id_o1 in IDs:
        for i , sub_o in enumerate(total_score):
            if sub_o[0] == id_o1:
                O1 = [o1[0],o1[1]+total_score[i][1],3]
                total_score[i] = O1
    else:
        O1 = [o1[0],o1[1]+score_rnd[o1[0]],1]
        total_score.append(O1)
        IDs.append(id_o1)

    if id_o2 in IDs:
        for i , sub_o in enumerate(total_score):
            if sub_o[0] == id_o2:
                O2 = [o2[0],o2[1]+total_score[i][1],3]
                total_score[i] = O2
    else:
        O2 = [o2[0],o2[1]+score_rnd[o2[0]],2]
        total_score.append(O2)
        IDs.append(id_o2)
```

Sort the total_score[] in ascending order base on the lb score or total score of the objects

```
    # sorted the total_score[] in ascending order base on the l
    total_score = sorted(total_score, key = lambda x:x[1])
```

4. Calculate the T sum value to judge if the scan entering shrink phase.

```
    # calculate the T sum value to judge if the scan entering s
    Tsum = o1[1]+o2[1]+5.0
    if len(total_score) >= k:
        # TA is the lowest lb or total score of current top-k o
        TA = total_score[-k][1]
        if Tsum <= TA :
            # then we start to calculate the upper bound score
            # objects from seq1 and seq2 both have the potentia
            # ub1 stands for the upper bound score of seq1 obje
            ub1 = None
            ub2 = None

            # check the objects from seq1.txt whose lb is now t
            for i in range(len(total_score)-1, -1, -1):
                if total_score[i][-1] == 1:
                    ub1 = total_score[i][1]+o2[1]

                if ub1 is not None:
                    break

            # check the objects from seq2.txt whose lb is now t
            for i in range(len(total_score)-1, -1, -1):
                if total_score[i][-1] == 2:
                    ub2 = total_score[i][1]+o1[1]

                if ub2 is not None:
                    break

            # the scan loop can stop when both ub1 and ub2 <= l
            if ub1 <= TA and ub2 <= TA:
                break
```

Then we start to calculate the upper bound score for the most potential objects already in the list.
Objects from seq1 and seq2 both have the potential to surpass the current top-k objects.

ub1 stands for the upper bound score of seq1 object, ub2 stands for that of seq2 object

The scan loop can stop when both ub1 and ub2 <= lowest total score of current top-k objects.

5. Number:2(j+1) of the above loop indicates the number of objects read before reaching the result.

```
# take out the top-k objects from the sorted total_score[] as r
Wk = list(reversed(total_score[-k:]))

# j means the loop's number-1
print('Number of sequential accesses = ' + str(j*2+2))
print('Top ' + str(k) + ' objects:')
for i in Wk:
    print(str(i[0]) + ': ' + str(round(i[1],2)))
```

The results of top-k queries (can match the brutal force queries' results)

```
Number of sequential accesses = 1380
Top 1 objects:
50905: 14.84
```

```
Number of sequential accesses = 2666
Top 2 objects:
50905: 14.84
85861: 14.76
```

```
Number of sequential accesses = 3018
Top 5 objects:
50905: 14.84
85861: 14.76
22652: 14.74
75232: 14.74
20132: 14.74
```

```
Number of sequential accesses = 5848
Top 10 objects:
50905: 14.84
85861: 14.76
22652: 14.74
75232: 14.74
20132: 14.74
21824: 14.7
9041: 14.66
97866: 14.65
83759: 14.64
96407: 14.58
```

```
Number of sequential accesses = 9046
Top 20 objects:
50905: 14.84
85861: 14.76
22652: 14.74
75232: 14.74
20132: 14.74
21824: 14.7
9041: 14.66
97866: 14.65
83759: 14.64
96407: 14.58
35055: 14.57
594: 14.54
16564: 14.54
78315: 14.54
33288: 14.52
79330: 14.51
11283: 14.49
4885: 14.47
9745: 14.45
55165: 14.44
```