FIT2004: Assessment questions for week 4

THIS PRAC IS ASSESSED! (5 Marks)

DEADLINE: Sunday, 18-Mar-2018 23:55:00 AEDT
MARKING: You will be interviewed during your lab or at another time as arranged by your demonstrator who will ask you a series of questions to assess your understanding of this exercise, and gauge how you implemented it. It is required that you implement this exercise strictly using **Python programming language**. Practical work is marked on the time and space complexity of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. Demonstrators are not obliged to mark programs that do not run or that crash.
You will lose all marks for a given task if your algorithm does not produce correct results for the task. Also, heavy penalties are applied if your complexity is worse than the required complexity and you may lose all marks if the complexity is similar to that of the naive algorithm. This is because our focus is on developing efficient algorithms and data structures. Also, do not assume that each task carries equal marks.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python programs (named topk.py and buddies.py) as well as a PDF file briefly describing your solution and its space and time complexity. The PDF file must give an outline of your solution (e.g., a high level idea of how did you solve it) and the **worst-case** space and time complexity of your solution. Penalties will be applied if you fail to submit the PDF file. The zipped file is to be submitted on Moodle before the deadline.
Important: The assignments will be checked for plagiarism using an advanced plagiarism detector and the students will be interviewed by tutors to demonstrate the understanding of their code. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT okay. Please do not share your solutions with others. If someone asks you for help, ask them to visit us during consultation hours for help.

Movie Buddies

Your imaginary friend Alice has built an app called `MovieBuddies` which is a social network for movie lovers. The app allows users to rate movies, write reviews, discuss plots and make movie fan pages etc. She has been thinking about adding two new features. However, her implementation is too inefficient and slows down the app. She knows you are taking FIT2004 which focuses on developing efficient algorithms and data structures. She hopes that you would be able to help her and has come to seek your help.

Task 1: Finding the top-k users

The first feature that she wants to add is rewards for top-k users of the app each month. Specifically, she has written a script that generates, at the end of each month, a file containing the time spent (in milli seconds) by each user on the app. She wants to find the top-k most active users (based on the time they spend on the app) and give them rewards each month. If the two users have spent exactly the same amount of time, the user with smaller ID (the user who had registered at the app earlier) should be be given preference. She initially implemented merge sort algorithm that sorts all the users based on the time they spend on the app and retrieved the first k users. Let U be the total number of users. This algorithm clearly takes $O(U \log U)$ time to retrieve the top-k users assuming that comparing the time spent by two users takes $O(1)$. She is concerned that this algorithm would be too slow when her app becomes a major hit (and the number of users becomes huge).

Your task is to implement an algorithm that reports the top-k users in $O(U \log k)$ and uses $O(U)$ space assuming that comparing the time spent by two users takes $O(1)$. Note that you cannot assume that the maximum possible time spent by any user on the app is a constant, i.e., counting/radix sort does not meet the time and space complexity requirements.

Input

The input file named `timeSpent.txt` records the time spent (in milli seconds) by each user. Each line in the file has two values separated by a colon `:`. The first value is the user ID and the second value is the time spent by the user. Below are the first 5 lines from the file.

```
1:9695819
2:5004755
3:611496
4:2006676
5:423596
```

The user with ID 1 spent 9695819 ms on the app, the user with ID 2 spent 5004755 ms and so on.

Output

Your solution for this task must be written in a file named `topk.py`. The program must read from the file `timeSpent.txt` and ask the user to enter a value of k. Then, it must print the top-k users (in sorted order) in $O(U \log k)$ time and using $O(U)$ space. Below is a sample output that your program must generate.

```
Enter the value of k: 10
#1: User ID: 61 Time spent: 9999446
#2: User ID: 7199 Time spent: 9998848
#3: User ID: 5666 Time spent: 9997642
#4: User ID: 4655 Time spent: 9996306
#5: User ID: 5922 Time spent: 9996306
#6: User ID: 7392 Time spent: 9994884
#7: User ID: 653 Time spent: 9994178
#8: User ID: 3117 Time spent: 9993810
```

```
#9: User ID: 4245 Time spent: 9993528
#10: User ID: 6043 Time spent: 9992581
```

The above output shows the top-10 users in descending order of the time they spent on the app. Note that users with IDs 4655 and 5922 spent exactly the same amount of time on the app. However, the user 4655 is ranked higher because its ID is smaller. Your program will be tested on a different version of `timeSpent.txt`, e.g., the number of users in `timeSpent.txt` (and/or the time they spend on the app) may be different.

Task 2: Movie buddy recommendation

Alice has written a script to obtain a file called `favoriteMovies.txt` which contains favorite movies for each user. The favorite movies for each user are not listed in any particular order. She wants to introduce a movie buddy recommendation feature that sends people notification about other people who like exactly the same set of movies. She wants your help in writing an algorithm that generates groups of people who like exactly the same set of movies. Below are the details.

Input

The input file named `favoriteMovies.txt` records, for each user, a list of its favorite movies. Below are the first five lines from the file.

```
1:FANTASTIC FOUR,THE SECOND BEST EXOTIC MARIGOLD HOTEL
2:THE PEANUTS MOVIE,GET HARD,THE SECOND BEST EXOTIC MARIGOLD HOTEL,PIXELS,THE
 WEDDING RINGER,UNFRIENDED,SICARIO
3:UNFRIENDED,THE SECOND BEST EXOTIC MARIGOLD HOTEL,THE WEDDING RINGER,GET
 HARD,ALVIN AND THE CHIPMUNKS THE ROAD CHIP
4:MINIONS,UNFRIENDED,CRIMSON PEAK,SICARIO,SPY
5:PAPER TOWNS,THE PERFECT GUY,ENTOURAGE,CREED,INSIDE OUT,A WALK IN THE WOODS,
 THE AGE OF ADALINE
```

Each user ID and its list of movies is separated by a colon `:`. In each list, movie names are separated by a comma (`,`). You can safely assume that the movie names consist only of letters from English alphabet in upper case. In the above example, user 1 likes two movies: Fantastic Four; The Second Best Exotic Marigold Hotel. The user 4 likes movies named Minions, Unfriended, Crimson Peak, Sicario and Spy.

Output

You must write a program name `buddies.py` which prints groups of users (of size at least 2) such that all users in a group like exactly the same set of movies. Below is a sample output for the above file.

```
GROUP 1
Movies: JURASSIC WORLD,A WALK IN THE WOODS
Buddies: 5921,8894

GROUP 2
```

```
Movies: THE SECOND BEST EXOTIC MARIGOLD HOTEL,THE VISIT,NO ESCAPE,SISTERS,
MINIONS,PADDINGTON,BLACK MASS
Buddies: 2819,6836,8294,8573

GROUP 3
Movies: WAR ROOM,TRAINWRECK,CHAPPIE,STRAIGHT OUTTA COMPTON
Buddies: 1672,2007,6710

GROUP 4
Movies: THE AGE OF ADALINE,FANTASTIC FOUR,SICARIO,TOMORROWLAND
Buddies: 299,3871,6738,9758,9993

GROUP 5
Movies: INSIDE OUT,HOT PURSUIT,UNFRIENDED,KRAMPUS
Buddies: 6582,9711

GROUP 6
Movies: WAR ROOM,MINIONS,JURASSIC WORLD,INSIDE OUT
Buddies: 1205,5890

GROUP 7
Movies: SICARIO,MAGIC MIKE XXL,SISTERS,TOMORROWLAND,SPY
Buddies: 3921,4607,5161,6931,7433,7452

GROUP 8
Movies: THE PEANUTS MOVIE,MINIONS,STRAIGHT OUTTA COMPTON,THE INTERN
Buddies: 3250,8428
```

In the above output, the users in each group like exactly the same movies. For example, in the first group the users 5921 and 8894 both like JURASSIC WORLD and A WALK IN THE WOODS. This can be confirmed by looking at the input file `favoriteMovies.txt`. Similarly, in group 7, each of the users (3921,4607,5161,6931,7433,7452) likes the same set of movies: SICARIO,MAGIC MIKE XXL,SISTERS,TOMORROWLAND,SPY.

In your output, the order of groups does not matter as long as you list all groups with their movies and users. Similarly, the order of movies and users in a group also does not matter.

Complexity Requirements

Let U be the total number of users, C be the maximum number of characters in any movie and K be the maximum number of movies liked by a user. The total space taken by the input file `favoriteMovies.txt` is $O(UCK)$ because the total number of characters is at most $U \times C \times K$. Your algorithm must report all groups in $O(UCK)$ time using $O(UCK)$ space. Note that this time complexity is optimal because reading the input itself takes $O(UCK)$. In other words, such an algorithm would be optimal. You may need to use a linear sorting algorithm (i.e., a sorting algorithm that runs in $O(N)$ where N is the input size) which we will learn in the week 2 lecture.

Recall that, as (to be) discussed in lecture 2, string comparison (e.g., `str1<str2` or `str1 == str2`) takes $O(L)$ in the worst-case where L is the number of characters in the smaller string.

Things to note

If you decide to use in-built Python functions and structures, you must carefully consider their worst-case complexities. For example, inserting/retrieving an element in Python dictionary (which uses hashing) takes $O(N)$ in worst-case where N is the number of elements currently in the dictionary. This is because, as we will later see in week 5, although hashing is quite efficient in practice its worst-case complexity for insertion/retrieval is still $O(N)$. It may not always be easy to determine the worst-case complexities of all in-built functions and structures. Therefore, it is recommended that you use only the built in structures/functions for which you know the time/space complexities.

Your submitted code must be your own work. If you take code from the internet, it is considered plagiarism even if you have made some modifications.

```
-=o0o=-
  END
-=o0o=-
```