FIT2004: Assessment questions for week 12

THIS PRAC IS ASSESSED! (5 Marks)

DEADLINE: Sunday, 20-May-2018 23:55:00
LATE PENALTY: 25% per day
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 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 program (named cameradetector.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.

1 Assessed Task: Red-light Camera Detector

Alice bursts into your room and is visibly upset. She waves a paper and screams, "I cannot believe this. They sent me a ticket for violating the traffic lights. I am sure the light was amber when I crossed the intersection. This is unfair."

She then takes a long deep breath and firmly says, "Okay, I have decided that I will never drive through an intersection with red-light cameras again. Never! Can you please help me to write an app to detect the red-light cameras?".

By now, you know that you cannot say NO to Alice. So you respond: "Of course, Alice! But give me some more information."

"Aaaww! You have always been such a good friend!" Alice happily says. "I also have a small gift for you which I will give you later[1]. But let me give you details of what I need for my camera detector app."

"Basically, I will provide you two text files named `vertices.txt` and `edges.txt`. The file `vertices.txt` contains the IDs of the vertices/intersections that have red-light cameras. The file `edges.txt` contains the information of edges corresponding to the road segments connecting the intersections. Some of the edges (road segments) are toll roads. I want the application to take as input a source vertex v and a value k; it must return me the k closest cameras from the vertex v and the shortest paths to each of these cameras. I do not want to travel on a toll road or drive through an intersection that has a red-light camera. Therefore, none of the paths must contain a toll road or *pass through* a red-light camera. "

"hmmmm, okay I think I get it. But could you give me a more formal definition", you ask.

Alice: "Sure! A path from a vertex s to a vertex t is **NOT** safe if the source vertex s contains a camera. **Otherwise**, the path is called a *safe* path if 1) none of the edges on the path is a toll road; and 2) none of the vertices on the path (except the target vertex t) contains a camera (the target vertex t may or may not have a camera). *Safe distance* from vertex s to vertex t is the total length of the shortest *safe* path from s to t. Given an input vertex v, the goal is to find k-closest cameras considering only the safe paths.".

1.1 Input

The input files `vertices.txt` and `edges.txt` can be downloaded from Moodle[2]. The total number of vertices are 6105 and you can assume that their IDs are in the range 0 to 6104. Below are the first 5 lines from `vertices.txt` denoting that the vertices $1, 5, 12, 16$ and 20 have red-light cameras.

```
1
5
12
16
20
```

Below are some lines from `edges.txt`. The first two values in each row correspond to the vertices that the edge connects and the third value corresponds to the distance between the vertices (i.e., the length of the edge). Some rows in `edges.txt` have four values where the fourth value is `TOLL` denoting that this edge is a toll road. For example, the third line below

[1] Alice's reward is on page 5!

[2] This is a real data set corresponding to the road network in a German city Oldenburg.

shows that there is an edge between vertices with IDs 124 and 138. The length of this road segment is 544.010193 and this is a toll road.

```
122 127 526.950012
123 125 228.396591
124 138 544.010193 TOLL
125 126 259.437286
```

1.2 Output

The program must ask the user to enter their location (vertex ID) and the value of k. It must then display k-closest cameras considering only the safe paths. It must also display the safe path to each of these cameras. Below is a sample output.

```
Enter your location: 1609
Enter k: 3

Camera 1 : 1595   Distance from your location: 89.02904600000001
Shortest path:  1609 --> 1600 --> 1593 --> 1595

Camera 2 : 1624   Distance from your location: 143.87629700000002
Shortest path:  1609 --> 1600 --> 1607 --> 1624

Camera 3 : 1648   Distance from your location: 170.821331
Shortest path:  1609 --> 1622 --> 1616 --> 1625 --> 1627 --> 1636 --> 1648
```

Note that there is a camera at vertex 1580 with distance 166.19017100000002 but it is not returned in the above output because the shortest path to this camera passes through another camera (at intersection 1595), i.e., the path is not a safe path. Below are the details of the camera (and the shortest path to it) that must be ignored by your algorithm.

```
Camera : 1580   Distance from your location: 166.19017100000002
Shortest path:  1609 --> 1600 --> 1593 --> 1595 --> 1578 --> 1580
```

Below is another sample output.

```
Enter your location: 2011
Enter k: 2

Camera 1 : 1999   Distance from your location: 21.692297
Shortest path:  2011 --> 1999

Camera 2 : 1991   Distance from your location: 302.152092
Shortest path:  2011 --> 2016 --> 4563 --> 2003 --> 1985 --> 1976 --> 1991
```

There is a path to camera at intersection 4529 with distance 238.061092 but it is ignored because the path contains a toll road. Below is a path to this camera which is invalid (i.e., not safe) because it contains a toll road.

```
Camera : 4529   Distance from your location: 238.061092
Shortest path:  2011 --> 2016 --> 4563 --> 4556 --> 4549 --> 4543 --> 4541
 --> 4529
```

For the above camera, the road segment `4543 --> 4541` is a toll road so this path must NOT be considered by your algorithm. The shortest path to 4529 that does not pass through any toll road or any other camera is shown below.

```
Camera : 4529   Distance from your location: 474.407198
Shortest path:  2011 --> 2016 --> 4563 --> 4556 --> 3433 --> 3446 --> 4557
 --> 4545 --> 4542 --> 4529
```

This camera is not among the 2-closest cameras because its safe distance is 474.407198 which is greater than the safe distances of the top-2 cameras (1999 and 1991).

If the user is already on an intersection with red-light camera (i.e., the entered location is a vertex with camera), then the algorithm prints a message stating that it is not possible to avoid the red-light cameras. Below is a sample output.

```
Enter your location: 2010
Enter k: 5
Oops! Cannot help, Alice!!! Smile for the camera!
```

In some cases, it is possible that the k-closest cameras cannot be found (e.g., there may be less than k cameras that can be reached without passing through any toll road or any other camera). In this case, your algorithm must return only the cameras that can be reached. For example, assume that you are on a vertex v that has only two adjacent vertices u and x and both u and x have red-light cameras. If $k > 2$, the algorithm will return only u and x because no other cameras can be reached without passing through an intersection with a camera.

In a case, where no camera can be reached (e.g., Alice is on a vertex for which all adjacent edges are toll roads), your program must print `Oops! You're stuck here Alice!`

Important: Your program must read data from the input files named `vertices.txt` and `edges.txt` (provided in the zipped folder on Moodle). You will lose marks if the program does not correctly read from the provided files or reads from different files. Also, your program must display the output in the format as shown in above sample outputs. Your program will be tested on various test cases using an automated script. Therefore, it is critical that you follow the output format shown above. You can assume that your program will be tested on the same set of vertices and edges but with different toll roads and different locations of cameras.

1.3 Implementation Requirements

Let V and E denote the number of vertices and edges, respectively. Your solution must have $O(V \log V + E \log V)$ time complexity (in the worst-case) to **determine** the k closest cameras. The total time complexity required to print the **shortest paths** to these k cameras must be $O(V \log V + E \log V + P)$ where P is the total number of edges printed in the output, i.e., the total number of edges on the shortest paths to these k cameras. To achieve this time complexity, you may need to modify min-heap as described in the lecture slides.

The task listed below is **NOT** assessed and will **NOT** be marked. Completion of this task is **OPTIONAL. DO NOT SUBMIT THE SOLUTION OF THIS PART WITH YOUR ASSIGNMENT 4.** Tutors are not going to mark it.

Alice's reward

It is a beautiful afternoon and you are taking a nap when you hear a knock on the door. It is Alice. With a wide smile, she hands over a USB.

"What's this Alice?", you curiously ask.

Alice smiles and says, "Well, I met your lecturer the other day and told him how hard you have been working in the past few months and that you have helped me in solving many challenging problems. I asked him if you could be rewarded. He was proofreading the final exam for FIT2004 that he had just finished writing. He nodded his head and spread out all the papers upside down and asked me to choose one. I picked one paper and gave it to him. He copied the text from that page into a file, added a special message for you (containing some advice for you related to the final exam) and compressed the file. To cut the long story short, this USB has a file[3] named `FIT2004Exam.bz2` which contains one randomly chosen page from your upcoming FIT2004 exam as well as a special message from your lecturer. Decompress the file to earn your reward ;)".

You: "Thanks Alice!!! You are a true friend".

Alice: "No problem. Oh, and before I forget, your lecturer asked me to tell you not to share the decompressed content of the file with the other students. Let them earn the reward. If they insist, just tell them something random that seems believable, e.g., tell them it was a prank, or even worse, invent an exam style question and give it to them"

"Sure thing! I will start working on it right away...", you reply and turn towards your computer.

Input

Text from a page of your upcoming FIT2004 final exam has been compressed using Burrows-Wheeler Transform and stored in a file named `FIT2004Exam.bz2` (see files in `bwt.zip` on Moodle under "Assignment 4"). `bwtGenerator.py` has been used to compress the text. You will need to decompress this file to retrieve the text.

Since `FIT2004Exam.bz2` is big, the compression approach used by `bwtGenerator.py` is explained below using a small text file `small_example.txt` which is also provided on Moodle. Below is the text in this file.

```
A: (ten marks)

B: (ten marks)

C: (ten marks)

D: (ten marks)
```

[3]Do not be mislead by the file name. It is NOT compressed using bzip2 format. The name is just a tribute to BWT (as it is used in bzip2). To see the content of this file, open it in a text editor such as notepad++.

`bwtGenerator.py` compresses this text and stores in a file name `small_example.bz2`. To make things easier for you, firstly, `bwtGenerator.py` replaces the newline character `\n` with `-` and whitespace with `*`.

```
A:*(ten*marks)--B:*(ten*marks)--C:*(ten*marks)--D:*(ten*marks)
```

Then, the Burrows-Wheeler Transform of this text is computed using `get_bwt` function. Below is the BWT for the above text.

```
)****ssss::::nnnn)))---DABC$---mmmmttttrrrr****eeeeaaaakkkk((((
```

The sorting of the characters is based on their unicode values, i.e., a character that has a smaller unicode is considered smaller. For example, the unicode value of `$` is 36 and the unicode value of `(` is 40. So, `$` appears before `(` in the sorted order. You can safely assume that `$` will be the smallest character if unicode based sorting is used (which is required for correctly decompressing the text). The `sort()` function in Python sorts the characters based on their unicode values.

The BWT of the text is then compressed. Specifically, the function `compress(bwt)` takes the BWT of the text as input and compresses it using run-length encoding. In run length encoding, a text `aaabbbbaa` is converted to `3a4b2a` (because the text has 3 occurrences of `a` followed by 4 occurrences of `b` followed by 2 occurrences of `a`). So, the above text is converted to `1-4*4s4:4n4)3-1D1A1B1C1$3-4m4t4r4*4e4a4k4(` (see the output produced by `print(encoded)` in `bwtGenerator.py`). To give you an easy to handle compressed format, the run-length encoding is split into lines as below.

```
1 )
4 *
4 s
4 :
4 n
3 )
3 -
1 D
1 A
1 B
1 C
1 $
3 -
4 m
4 t
4 r
4 *
4 e
4 a
4 k
4 (
```

The text above is stored in `small_example.bz2`. If you want to see the content of this file, you need to open this in a text editor such as notepad++. The text from the final exam is compressed in the same way using `bwtGenerator.py` and is stored in `FIT2004Exam.bz2`.

Output

You will first need to decompress the run-length encoding, i.e., the text `3a4b2a` needs to be converted to `aaabbbbaa`. For the above example, after you decompress, you will get the following.

```
)****ssss::::nnnn)))---DABC$---mmmmttttrrrr****eeeeaaaakkkk((((
```

The above is the BWT of the text. You will then apply the inversion algorithm on this text to invert the Burrows-Wheeler Transform. If your inversion algorithm is correct, you will get the following.

```
A:*(ten*marks)--B:*(ten*marks)--C:*(ten*marks)--D:*(ten*marks)
```

Assume you have inverted the text and it is stored in the variable `inverted_text`. You will need to copy `abra_cadabra` function provided in `bwtGenerator.py` to your code and call it by passing `inverted_text`. This function will print the original text by replacing each `-` back to `\n` and replacing each `*` back to a whitespace. E.g., for the above example, the function `abra_cadabra(inverted_text)` will print the following.

```
A: (ten marks)

B: (ten marks)

C: (ten marks)

D: (ten marks)
```

Once you correctly reproduce the original text shown in the small example above, apply your algorithm to `FIT2004Exam.bz2` file to decompress it.

Important: Before compressing the text using `bwtGenerator`, I added a lot of occurrences of `FIT2004ExamPage` in the original text. This is to ensure that the text size is big enough such that if you use naive algorithm for inversion, your program takes too long and possibly crashes before completely inverting it. Therefore, do NOT try to invert the text using the naive algorithm. It will most possibly crash. Furthermore, implementing the naive algorithm takes almost similar amount of efforts. So, it is better to implement the efficient inversion algorithm.

Have fun!