FIT2004 S2/2016: Assessment questions for week 12

THIS PRAC IS ASSESSED! (6 Marks)

DEADLINE: Monday, 17-Oct-2016 10:00:00 AM
CLASS: This programming exercise has to be completed before the deadline. Your demonstrator will check your submission during the lab. He 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 performance 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.
After/befor your demonstrators have interviewed you, you are expected to work towards the programming competition or other questions that will be provided to you on the day.
SUBMISSION REQUIREMENT: You will need to submit a zipped file containing your Python programs (named decompress.py) as well as a PDF file briefly describing your solutions and their space and time complexities. 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. The PDF file must also include your solution to the problems related to recurrence relation. The zipped file is to be submitted on Moodle before the deadline.
Important: You should carefully consider the border cases and make sure that you return correct results for all cases. Although you are not required to include the proof of correctness in your submissions, you are encouraged to formally prove the correctness of your programs. This will help you in identifying and fixing the bugs in your program if present.

1 Task 1: Decompression

Alice is a fan of open source softwares and she had been using an open source compression software. After she upgraded to Windows 10, her software got corrupted and crashes whenever Alice attempts to decompress any file. Alice had compressed some really important files using the software and needs to decompress the files. Other available softwares fail to decompress the files probably due to compatibility issues. Well, you got it right. She has come to ask for your help.

"Hello dear friend! I hope you can help me one last time. I have some really important compressed files that I am unable to decompress. Please help me!", Alice says.

"Sure, I will help you Alice. But I need to know the compression algorithm that the software uses.", you reply.

"Hmmmm... I had read the documentation file, I think the algorithm is called By The Way. I remember its abbreviated form was BTW or something like that.", Alice says.

© Muhammad Aamir Cheema, Monash University

You smile, "You mean BWT. Well, that's Burrows-Wheeler Transform. I can certainly help you with this Alice! But I would need a compressed file to test whether my decompression works fine.".

Alice is so happy to hear this, "Thank you so much! I have a very special compressed file for you my friend!". And she hands you over a USB containing a file named exam.bz2.

"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 handed over 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, the file exam.bz2 contains one randomly chosen page from your upcoming FIT2004 exam as well as a special message from your lecturer. But he asked me to tell you not to share the decompressed text with the others! Let them earn the reward!!!".

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

1.1 Input

The input files exam.bz2 is a result of converting the text using Burrows-Wheeler Transform and then applying run-length encoding as explained in the lecture week 07. The source file used to compress the text is provided in Moodle (see bwtGenerator.py). Below are the details.

Suppose the input text was the following.

```
A: (ten marks)

B: (ten marks)

C: (ten marks)

D: (ten marks)
```

To make things easier for you, firstly, the newline character `\n` is replaced with `-` and whitespace is replaced with `*`.

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

Then, bwtGenerator.py applies Burrows-Wheeler Transform to this text. Below is the BWT for the 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
4 )
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 file exam.bz2 follows the format above.

1.2 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)-
```

Once you invert the text, you will need to replace `-` with the newline character `\n` and `*` with whitespace to obtain the original text.

```
A: (ten marks)

B: (ten marks)

C: (ten marks)

D: (ten marks)
```

Once you correctly reproduce the original text, apply your algorithm to exam.bz2 file to decompress it.

Important: You only need to output the original text. The intermediate steps are displayed just to illustrate the steps you will need in your implementation.

Note that, as stated in the lecture, BWT is effective for compression only when the data is large and has patterns. After you finish the implementation, create a text file by pasting the same text several times and compress it using bwtGenerator.py. Compare the length of the compressed text with the length of the original text. You do not need to submit this with the assignment. It is just for you to test the effectiveness of BWT in compressing the text with repeated patterns.

1.3 Implementation Requirements

Let N be the length of the original (decompressed) text and M be the alphabet size (i.e., M is the total number of possible unique characters in the text). Your algorithm must invert BWT in $O(M + N)$ time using $O(M + N)$ space. You can assume that the unicode values of the characters range between 0 to 255 (i.e., the number of unique characters M is 256).

In your algorithm, to compute the next row to be visited, you will need to create the `Rank` array (see lecture week 07) and number each character in the last column. Creating the `Rank` array requires sorting the N characters. This can be done in $O(N \log N)$ by using Python's built-in `sort()` but you would need to sort it within $O(M + N)$ time. Hint: The number of unique characters is at most 256 (also see direct-addressing in lecture week 05). Numbering each character in the last column requires scanning the last column top to bottom and record in a separate array how many times each character has been seen already. This also needs to be done in $O(M + N)$ space and time using direct-addressing.

I would suggest you to first use Python's built-in `sort()` function (that takes $O(N \log N)$) and make sure your inversion algorithm works fine. You should then think about writing your own sort function to sort the characters in $O(M + N)$. Also, you may first want to handle the text that only has English alphabets (e.g., A to Z) and then extend your idea for the unicode characters.

2 Task 2: Recurrence Relations

In the merge sort, we divide an array into two equal parts and then sort recursively. Suppose that we modify the merge sort such that the array is divided into three equal parts and then sort each part recursively. Assuming that merging the three partitions requires aN operations, we have the following recurrence relation.

$$T(N) = \begin{cases} 3T(\frac{N}{3}) + aN, & \text{if } N > 1 \\ b & \text{if } N = 1. \end{cases}$$

where a and b are two **constants**.

Task 2a. Solve the above recurrence relation. Use the solution of the recurrence relation and report the complexity of the algorithm in big-O notation.

Task 2b. Using the induction, prove that your solution to the recurrence relation is correct.