CMPSCI 311: Introduction to Algorithms **Fall 2016**

Programming Assignment

Released 11/4/2016 Due 8:00pm 12/2/2016 in Moodle

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Problem. Design and implement an efficient algorithm for the following task:

Given a string S, find the longest substring of S that appears at least twice in S.

Specifically, you will be given a string x of n characters. Your task is to find indices $i \neq j$ and a number k such that $x[i, \ldots, i+k] = x[j, \ldots, j+k]$ where k is as large as possible. Then you should output $x[i, \ldots, i+k]$ which is the longest repeated substring.

Other issues:

1. You may resolves ties arbitrarily. If there is a tie (two repeated substrings with the same length, which is the maximum), you may output whichever you like.
2. Observe that while i is not allowed to equal j, it could be the case that $j = i + 1$ so the substrings overlap. For example if $x =$ "aaaa", then the longest repeated substring is "aaa."

Collaboration. You must work independently. You may not work in groups or discuss your solution with anyone. You may use the internet and external resources for programming help *only*, you may not use other resources for algorithmic ideas.

Submission instructions. This assignment is due by 8:00pm on 12/2/2016 in Moodle.

You may use whatever programming language you like, but your program should consume the input string from `stdin` and write the output to `stdout` followed by a single newline. Make sure that the only output of your program is your answer, followed by a newline. Any sequence of bytes on `stdin` should be interpreted as a string and only the end-of-file (`EOF`) character terminates the string.

You must also prepare a `README` file with the following information.

1. The intuition for your algorithm.
2. High-level pseudocode.
3. Run-time analysis.

This file should be in plain ASCII text.

You will submit a gzip-ed tarball (extension `.tar.gz`) to Moodle. The tarball should have at least two files `README` and `Makefile`, and potentially any source code files you want. If your gzip-ed tarball is called `submission.tar.gz` we will test your program using the following protocol:

```
$ gunzip submission.tar.gz
$ mkdir submission
$ mv submission.tar submission/
$ cd submission
$ tar -xf submission.tar
$ make -s compile
```

```
$ cat input1 | make -s run 1>output1 2>/dev/null
$ diff -q output1 answer1
```

If your program produces the correct answer, there should be no output. If your program is incorrect there will be some output. We will also time the execution of your algorithm and terminate it if it is too slow

Included with the assignment is a submission in python that meets the protocol but is an incorrect algorithm.

We will grade your submission on the edlab cluster, so you can check that your submission works on those systems by running the above protocol there (You'll have to create your own `input1` and `answer1` files). You can connect to edlab via `ssh` to the host `elnux@cs.umass.edu`. Your username is your UMass NetID and your password is ELxxxaaa where xxx is the last 3 digits of your ID number and aaa is the first three characters of your username.

Do not submit pre-compiled files. Specify in the `compile` directive of your makefile how to compile your source code.

Grading.

1. 45 points for a clear description of your algorithm and a creative correct solution in the `README`. The Brute force approach will get very few points here.
2. 35 points for correct implementation that passes our test cases.
3. 20 points for runtime analysis in the `README`.

Advice. Spend some time on algorithm design before diving into implementation. Looking at the grading scheme, your implementation counts for less than half of the points.

Make sure your submission follows the above protocol, and that it compiles and runs on the edlab machines.