Algorithms & Data Structures: Lab 09

week of 3rd December 2018

1 Setup

1.1 Saving your work from last week

As with all previous weeks, you will use `git` to download a bundle of lab code. You will probably have made modifications in your downloaded copy; if you have not already done so, you need to save those modifications. First examine the changes present in your downloaded copy by issuing the following commands from the labs directory:

```
git status
git diff
```

and if you are satisfied with the changes, store them in the git version control system by doing

```
git commit -a
```

and writing a suitable commit message

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `09/`) alongside the existing directories.

2 Module evaluation and feedback

At this point in the term, you have the opportunity to leave feedback on this module through the anonymous module evaluation process. A link to the survey is available on the main learn.gold page for this module. The feedback from this process is considered by Department management and by the Department's Learning & Teaching Committee, which try to improve our teaching practices across the board. Please complete your module evaluation surveys for all your modules, including this one, by the end of this term.

I am separately asking you for your feedback on some more specific aspects of this term; please note that this questionnaire is **not** anonymous, and that participation in it is entirely voluntary. If you do feel able to participate in this survey, please do so by **12:45** on **Monday 10th December**.

3 String matching

The skeleton files and tests provided give you a framework for implementing a number of different string matching algorithms. You should implement naïve string matching and at least one of the Rabin-Karp and Knuth-Morris-Pratt algorithms. You should also use the `OpCounter` class to count the number of character read operations; for example

```
if(T[i] = P[j])
```

should add 2 to the counter, and

```
h - T[i-1] + T[i+m-1]
```

should also add 2.

Note that `make test` in the `09/` lab bundle directory **only** tests naïve string matching. You can test your implementations of other algorithms by running the test driver and passing on the command-line an argument naming the algorithm you want to test.

3.1 Naïve string matching

First, implement naïve matching. Check your implementation for correctness against the provided tests. Are the provided tests sufficient to make you confident that your implementation is correct? Add test cases to the table of tests if not (and send me a merge request!)

Using the `StringMatchCount` program (or otherwise), measure the number of character reads to perform a match in the worst case (for example, matching a pattern of m characters `aaa...aab` against a text of n characters `aaa...aaa`). Copy and complete the following table:

m	n	char reads (worst case)
2	10	
5	10	
10	10	
2	100	
20	100	
50	100	
100	100	
2	1000	
20	1000	
200	1000	
500	1000	
1000	1000	

3.2 Rabin-Karp string matching

Implement Rabin-Karp matching using the rolling hash function

$$h(s_{i..i+m-1}) = \sum_{k=i}^{i+m-1} s_i \bmod 256$$

Use this implementation to investigate the number of character reads in the best case (when the value of the hash function applied to text substrings is always distinct from the hash value of the pattern) and

the worst case (when the value of the hash function applied to text substrings always collides with the hash of the pattern, without there being a match).

The best case is a pattern of m characters `aaa...aaab` against a text of n characters `aaa...aaa`; the worst case is a pattern of m characters `bbb...bbbca` against a text of n characters `bbb...bbb`

m	n	char reads (best case)	char reads (worst case)
2	10		
5	10		
10	10		
2	100		
20	100		
50	100		
100	100		
2	1000		
20	1000		
200	1000		
500	1000		
1000	1000		

Don't forget to include in your count of character reads the operations needed to compute the hash value of the pattern.

3.3 Knuth-Morris-Pratt string matching

Implement Knuth-Morris-Pratt matching following the pseudocode given in the slides.

Use this implementation to investigate the number of character reads in the worst case (which for this version of the algorithm is the same text and pattern as the worst case for naïve string matching)

m	n	char reads (worst case)
2	10	
5	10	
10	10	
2	100	
20	100	
50	100	
100	100	
2	1000	
20	1000	
200	1000	
500	1000	
1000	1000	

Don't forget to include in your count of operations the character reads needed to compute the prefix table.

4 List visualiser

4.1 Assessment

You must perform your assessment of **all five** of your assigned `ListVisualiser` submissions by **16:00** on **Friday 7th December**.

Please take the time to be as accurate as you can in your answers to the rubric marking questions, and to be helpful and constructive in your written comments. I will be looking at a sample of assessments to check that the comments are respectful and appropriate.

Failure to complete all five assessments, or an assessment that is unhelpful or shows no engagement, will lead to an **automatic mark of zero** for the assessment portion of this activity. There **no extensions** to the deadline of 16:00 on Friday 7th December.