CS251 - Homework 4: Strings I

Out: April 08, 2016 @ 9:00 pm
Due: April 15, 2016 @ 9:00 pm

Name: Joel Van Auken

Read this before start: Not all questions are multiple choice, some of them may require an explanation or simulating an algorithm. In those questions give as much information you consider is required for the solution. Read carefully each question in order to identify what is being asked.

**** For all questions, any answer without an explanation (even though it is correct) will be graded with 0 points. ****

1 (5 pts). A palindrome is a sequence of characters which reads the same backward or forward. The strings “taco cat”, “6847486” and “^O_O^” are examples of palindromes using alphanumeric and symbolic characters. Usually, the space character is not considered while checking whether a string is palindrome or not.

Given a string S of length n and using the **brute force algorithm** as described in class, what is the number of comparisons required for determining whether S is a palindrome or not?

a) $log(n)$
b) n
c) $nlog(n)$
d) n^2

The number of comparisons would be n because you need to compare every character to the character in the opposite position to confirm that it is a palindrome.

2 (5 pts). Describe an algorithm (in about three lines) for finding palindromes with at most $n/2$ comparisons.

Starting with index 0, compare this index with index of n-1. Continue until the index reaches n/2.

3 (5 pts). According to Wikipedia, a semordnilap[1] (palindromes spelled backwards) is the name coined for words that spell a different word in reverse. Examples of this are the words repaid ⇄

[1] https://en.wikipedia.org/wiki/Palindrome#Semordnilaps

diaper, swap ⇄ paws, stressed ⇄ desserts, deliver ⇄ reviled and rewarder ⇄ redrawer, to mention a few.

Let S_1 and S_2 be strings with $m = |S_1|$ and $n = |S_2|$ their respective lengths. We would like to know if they are semordnilaps. Which one or ones of the following conditions are true when S_1 and S_2 are semordnilaps, and why?

a) $m = n$

b) $S_1[i] = S_2[j] \quad \forall i, j$ such that $0 \le i \le m-1,\ 0 \le j \le n-1$ and $i \neq j$

c) $S_1[i] = S_2[m-i-1] \quad \forall i$ such that $0 \le i \le m-1$

d) $S_1[i] = S_2[j] \quad \forall i, j$ such that $0 \le i \le m-1,\ 0 \le j \le n-1$ and $j = m-i+1$

The lengths must be the same, or they cannot be palindromes. And the second condition states the fundamental condition for Palindromes

4 (5 pts). Alice and Bob found semordnilaps quite interesting. However, they cannot agree about a minor detail for defining an algorithm that checks if two strings are semordnilaps. Bob says the algorithm only has to traverse the first $m/2$ characters of S_1 and compare them with their respective character of the last $n/2$ characters of S_2. Alice is not convinced at all, she thinks the algorithm must traverse all the characters of both strings. Who is right in this argument? Provide an example (or counterexample) to justify your reasoning.

Alice is correct because you need to guarantee that the word is not the same reversed, as in NOON.

5 (10 pts). Let's have some practice with the Boyer-Moore algorithm. Given the text "testing a tested test text", finding the pattern "tested" should not be a problem. First, we need to calculate the last occurrence function. For the alphabet assume it is the letters that compose the text. Discard blank spaces.

A:-1 D:5 E:4 G:-1 I:-1 N:-1 S: 2 T:3 X:-1

6 (10 pts). Following the Boyer-Moore algorithm, what is the number of comparisons required for finding the the previous pattern in the given text? Help yourself with a drawing similar to the examples showed in class and paste it with your answer.

14 total comparisons. Graph at bottom of page

7 (10 pts). Alice and Bob strike back. After reading about the Boyer-Moore algorithm they claim checking if two words are semordnilaps is a piece of cake. They then proposed the following algorithm:

```
1   Algorithm Semordnilaps (T, P, S)
2      L ← lastOccurenceFunction(P, S)
3      i ← 0
4      j ← m - 1
5      repeat
6         if T[i] = P[j]
```

```
7           if  j = 0
8              return (i = n - 1) ? true : false
9           else
10             i ← i + 1
11             j ← j - 1
12       else
13          l ← L[T[i]]
14          i ← i + m – min(j, 1 + l)
15          j ← m - 1
16    until  i > n - 1
17 return false
```

a) What is the Big-Oh running time of this algorithm? This is O(log n)

b) Is there some part of this algorithm that can be removed in order to improve its performance and to what Big-Oh would it improve (if any)?

No, this code looks pretty optimized to me.

8 (10 pts). Finding patterns in DNA sequences is a common task in bioinformatics. A DNA sequence is composed of characters A, C, G and T representing adenine, cytosine, guanine and thymine respectively. The KMP algorithm is used when the text and the pattern are not too long. Before running KMP we must calculate the failure function of the pattern. We need to find the pattern "GACAGATGA" in a DNA sequence. Calculate the failure function for the given pattern. 000012012

9 (10 pts). Following the KMP algorithm, what is the number of comparisons required for finding the pattern "GACAGATGA" in "GGTACCCGACAGATGACAGA"? Help yourself with a drawing similar to the examples showed in class and paste it with your answer.

18

10 (10 pts). The explained pattern matching algorithms find the first occurrence of the pattern in the text. However, sometimes real problems don't require the first occurrence but all of them. Below is a list of sentences we can propound considering this situation. For each one of them say whether it is true or false. Remember to give a brief explanation.

a) Once a match is found, the algorithm must continue on the next location after the initial location of the match. True. If the string repeats, such as GTCGTCGA for GTCG

b) The running time $O(nm+s)$ increases for the Boyer-Moore and KMP algorithms while finding all the possible matches. True. You have to run the algorithm many times

c) It is possible to have nested matches (a match inside another previously found match). True, see A

d) If we consider the text to be circular (after the last characters comes the first one), then checking matches will require to traverse the text as much twice.

False, would only have to check until the first char of the pattern was compared against the first letter again.

11 (10 pts). Use the LZ78 algorithm for encoding this wonderful tongue twister: “she sells sea shells”. Consider blank spaces as characters. Show the output diagram as described in class.

Attached

(0s)(0h)(0e)(0)(1e)(0l)(6l)(4s)(3a)(8h)(3l)(6l)

12 (10 pts). Use the LZ78 algorithm for decoding this quote from a recent movie: (0, 'f'), (0, 'l'), (0, 'a'), (0, 's'), (0, 'h'), (0, ','), (0, ' '), (1, 'l'), (3, 's'), (5, ' '), (0, '-'), (7, 'h'), (0, 'u'), (0, 'n'), (0, 'd'), (0, 'r'), (0, 'e'), (15, ' '), (0, 'y'), (3, 'r'), (18, 'd'), (9, 'h'), (0, ''). Show the output diagram as described in class.

Attached. "flash, flash - hundred yard dash"

Submit Instructions:
The homework must be turned in by the due date and time using the turnin command. Follow the next steps:

1. Please make sure your submission is legible! (No cellphone pictures please! All ITAP labs have scanners for you to use)
2. Login to data.cs.purdue.edu (you can use the labs or a ssh remote connection).
3. Make a directory named with your username and copy your solution (in pdf format) there. (**Important:** Such pdf file should be **the only one** contained within the folder).
4. Go to the upper level directory and execute the following command:
   turnin -c cs251 -p hw4 your_username
   (Important: previous submissions are overwritten with the new ones. Your last submission will be the official and therefore graded).
5. Verify what you have turned in by typing **turnin -v -c cs251 -p hw4** (**Important:** Do not forget the -v flag, otherwise your submission would be replaced with an empty one). If you submit the wrong file you will not receive credit.

Testing a tested test text

TESTED

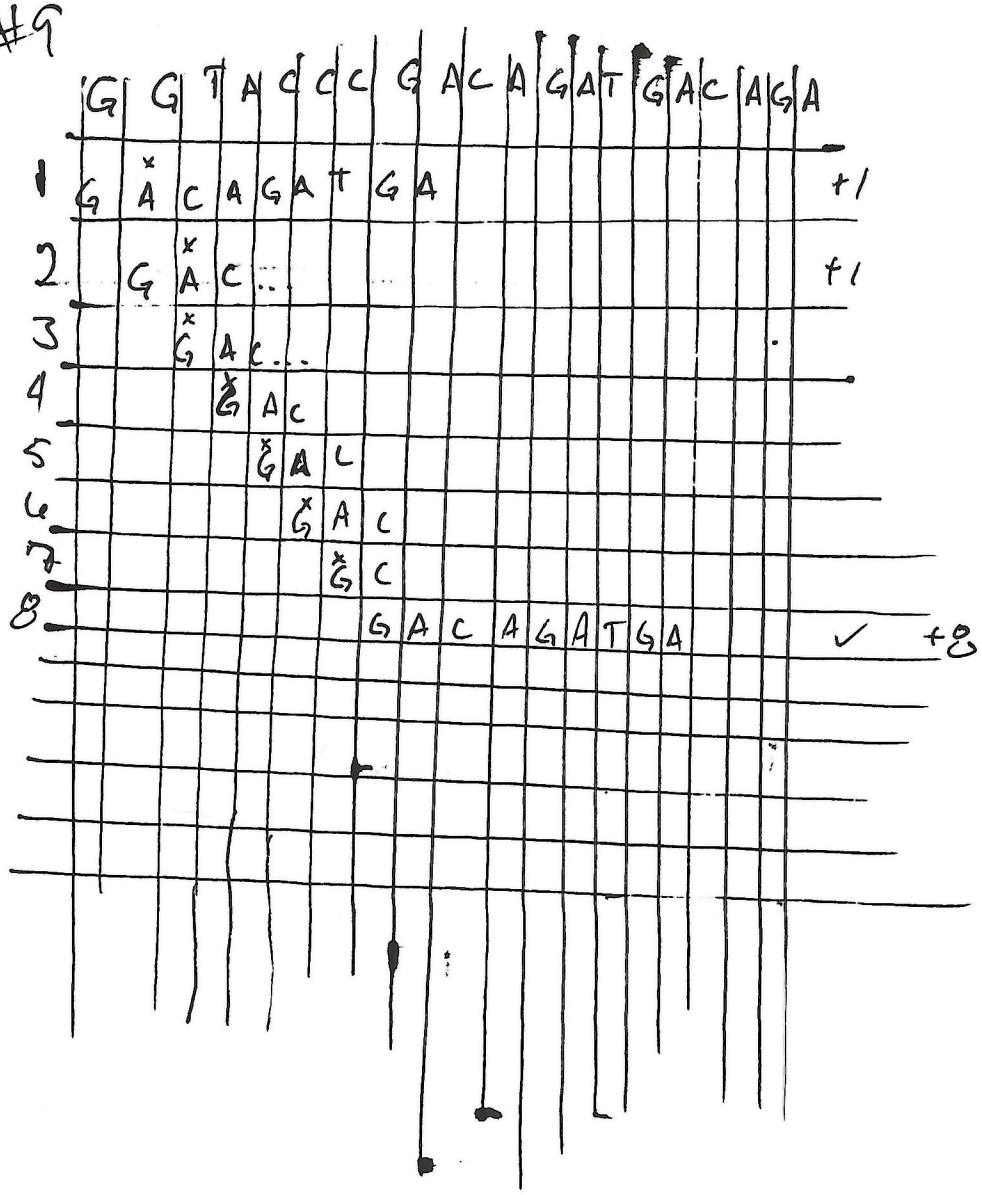
TESTED

TESTED

TESTED

TESTED

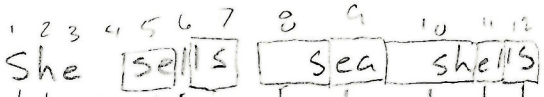
#9



Failure:

G	A	C	A	G	A	T	G	A
0	0	0	0	1	2	0	1	2

18 total



Dictionary

Output	Index	String
(0, S)	1	S
(0, h)	2	h
(0, e)	3	e
(0, _)	4	_
(1, e)	5	Se
(0, l)	6	l
(6, l)	7	ll
(4, s)	8	_s
(3, a)	9	ea
(8, h)	10	_sh
(3, l)	11	el
(6, l)	12	ls

Compressed message:

0S0h0e0_1e0l6l4s3a8h3l6l

#11

#12

(0,f) (0,l) (0,a) (0,s) (0,h) (0,_) (0,) (1,d) (3,s) (5,) (0,-) (7,h) (0,u)

(0,n) (0,d) (0,r) (0,e) (15,) (0,y) (3,r) (15,d) (9,h) (0,)

Dictionary

Output	Index	String
f	1	f
l	2	l
a	3	a
s	4	s
h	5	h
_	6	_
fl	8	fl
as_	9	as
h_	10	h_
-	11	-
_h	12	_h
u	13	u
n	14	n
d	15	d
r	16	r
e	17	e
d_	18	d_
y	19	y
ar	20	ar
d_d	21	d_d
ash	22	ash
.	23	.

decompressed message:

flash, flash - hundred yard dash.

#p