MATH578A Assignment 1

Raktim Mitra
USC ID: 1487079265 email: raktimmi@usc.edu

February 29, 2020

Q1.

I chose Rabbit (Oryctolagus cuniculus) genome downloaded from NCBI genome page for rabbit. (`https://www.ncbi.nlm.nih.gov/genome/?term=Oryctolagus+cuniculus`).

Results on the test pattern searches are as follows:

Pattern	Occurence	comparisons	runtime(seconds)
ACACACACACACACACACACACACACACACACACACACAC	614	539841559	12.111216
GAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGA	15195	473455211	9.997119
CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGC	10	524563641	12.610945
GACGACGACGACGACGACGACGACGACGACGACGACGACG	0	523252832	13.063918
ACAGACAGACAGACAGACAGACAGACAGACAGACAGACAG	1	507595359	11.662198
AACGAACGAACGAACGAACGAACGAACGAACGAACGAACG	0	523598924	12.964050
AAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGC	1	525172676	12.699295
CCACCAGGGG	4037	1206400435	31.577522
GGAGGACCCC	2648	1188278808	30.708362

Results on he test sequences on HG38 (human genome) taken from (`https://www.ncbi.nlm.nih.gov/genome/?term=Homo+sapiens`):

Pattern	Occurence	comparisons	runtime(seconds)
ACACACACACACACACACACACACACACACACACACACAC	27877	667532560	15.362110
GAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGAGA	4365	544768912	11.428859
CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGC	103	643608461	15.230797
GACGACGACGACGACGACGACGACGACGACGACGACGACG	7	640093906	15.678619
ACAGACAGACAGACAGACAGACAGACAGACAGACAGACAG	31	628499838	14.257126
AACGAACGAACGAACGAACGAACGAACGAACGAACGAACG	0	640427323	15.587231
AAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGCAAGC	45	645121107	15.222942
CCACCAGGGG	3391	1449562317	36.682713
GGAGGACCCC	2437	1423537154	36.751045

Chosen ALU sequence: "GGCGGGCAGATCATGAGGTCAGGAGATCGAGACCATCCTGGCTAACACGG".
In Hg38, Total Matches found: 117
Char comparisons: 1014339834
Time taken in seconds: 30.342192

Q2. (Chapter 1, exercise 11)

Q: Let T be a text string of length m and let S be a multiset of n characters. The problem is to find all substrings in T of length n that are formed by the characters of S. For example, let S = {a, a, b, c} and T = 'abahgcabah'. Then 'caba' is a substring of T formed from the characters of 5. Give a solution to this problem that runs in O(m) time. The method should also be able to state, for each position i, the length of the longest substring in T starting at i that can be formed from S.

Ans: Let, A[ ] contains unique elements of S. Count[ ] array contains number of occurences of each element of A in S.

Now, we shall construct the array D[ ] where D[i] contains length of the longest substring of T that <u>ends</u> at position i which can be formed by elements from S.
i ranges from 0 to m-1.

We also need a RunningCount[ ] array similar to Count[ ], which will use to keep track of counts of elements of S faced for D[i]. A, Count, RunningCount all have the length count(unique(S)). Note: A.index[c] gives position of char c in A, constant time since A is of constant length assuming alphabet size is constant. Therefore, we can write the following recurrence for D[i].

$$D[i] = \begin{cases} \text{0 if T[i] is not in A (case 1)} \\ \text{if T[i] is in A} \begin{cases} D[i-1]+1 & \text{if RunningCount[A.index[T[i]]]} < \text{Count[A.index[T[i]]]} \\ & (\uparrow \text{ case 2a}) \\ i-j & \text{if RunningCount[A.index[T[i]]]} == \text{Count[A.index[T[i]]]} \\ & \text{where j is the first occurence of T[i] after i - 1 - D[i-1]} \\ & (\uparrow \text{ case 2b}) \end{cases} \end{cases}$$

For the above recurrence to work properly , we need to update RunningCount in the following manner:

$$\begin{cases} \text{(case 1) RunningCount[i]} = 0 \text{ for all i} \\ \begin{cases} \text{(case 2a) RunningCount[A.index[T[i]]]} \mathrel{+}= 1 \\ \text{(case2b) RunningCount[A.index[T[k]]]} \mathrel{-}= 1\text{, for k in interval (i,j)} \end{cases} \end{cases}$$

After we fill up the array D we can report all starting occurence position as: {i - size(S) for i if D[i] == size(S)}.

P.T.O.

We can also construct length of longest substrings starting at i and formed by elements of S from D. Let, this array be B[ ]. Shown in full pseudocode below:

Algorithm 1: Multiset Matching

```
Input: Text T and Multiset S
 1: A ← Unique(S)
 2: initialize Count[0...length(A)] to all 0
 3: initialize RunningCount[0...length(A)] to all 0
 4: for element e in S do
 5:   Count[A.index[e]]++
 6: end for
 7: intiailize D[0...m-1]
 8: D[-1] ← 0     //For convenience, In actual implementation we would
    hardcode D[0] and start the following loop at 1.
 9: for {i=0; i < m; i++} do
10:   if T[i] not in A then                                    //Case 1
11:     D[i] ← 0
12:     {RunningCount[j] ← 0 for j in interval [0,length(a))}
13:   else if RunningCount[A.index[T[i]] < Count[A.index[T[i]]]     //Case 2a
      then
14:     D[i] ← D[i-1] + 1
15:     RunningCount[A.index[T[i]]++
16:   else                                                     //Case 2b
17:     j ← i - D[i-1] + 1 //Case 2b
18:     while j≤i and T[j] != T[i] do
19:       RunningCount[A.index[T[j]]]−−
20:     end while
21:     D[i] ← i − j
22:   end if
23: end for
24: Initialize B[0..m-1] //length of substrings longest substrings formed
    by elements of S from starting positions.
25: i ← 0
26: Initialize R                //List of starting positions of occurences
27: while {i < m} do   //O(m), only correct when i goes from 0 to m-1,
    the descending order of traversal won't work.
28:   B[i - D[i]] ← D[i]
29:   if D[i] == size(S) then
30:     R.add(i - D[i])
31:     i++
32:   end if
33: end while
34: return R, B
```

Time Complexity: Clearly all the preprocessing can be done in O(m) time and so is generating R and B from D.

That leaves constructing D. The for loop (line 9-23) runs m times. Case 1 and Case 2a are clearly O(1) [Assuming alphabet size is O(1)]. However, the while loop (Case 2b, line 18-20) can be O(size(S)) in worst case. But, we can use amortized analysis to show that amortized cost of each iteration of the for loop is O(1). The reason is, case2b can decrement RunningCount (line 19) size(S) times only when Case(2a) has been executed size(S) times. (Because only Case 2a increments RunningCount (line 15).)

Therefore, Case 2b can be O(size(S)) only after O(size(s)) iterations of case 2b which are all O(1) iterations. Therefore, on average each iteration takes O(1), making the whole for loop O(m).

Hence, time complexity of Multiset Matching is O(m).

Q3. (Chapter 6, exercise 1.)

Q: Construct an infinite family of strings over a fixed alphabet, where the total length of the edge-labels on their suffix trees grows faster than O(m) (m is the length of the string). That is, show that linear-time suffix tree algorithms would be impossible if edge-labels were written explicitly on the edges.

Ans: Let, $B^k = BBB...B$ (k times) Then the following family of strings over the alphabet $\Sigma = \{A, B\}$ is a required example.

$$S_n = AB^0AAB^1AAB^2A...AB^nA$$
$$S_0 = AA$$
$$S_1 = AAABA$$
$$S_2 = AAABAABBA$$

Note: S_n has length $(2+3+4+...+(n+2)) = \frac{(n+2)(n+3)}{2} - 1 = O(n^2)$.
Suffix trees for S_0, S_1, S_2:

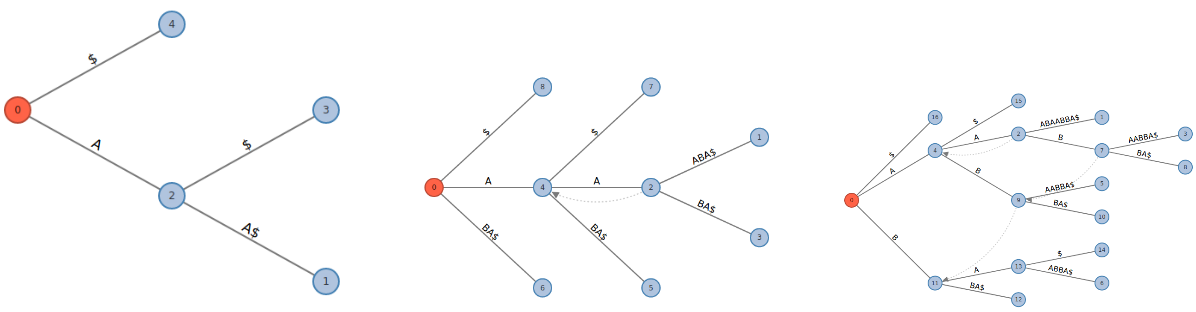

Figure 1: suffix trees for AA, AAABA and AAABAABBA

We can simply look at how total length of edge labels scales w.r.t lengths of S_n:

n	length(S_n)	total edge label length(excluding $s)
0	2	2
1	5	11
2	9	33

Clearly total length of edge labels grows much faster than length of the strings.

Q4. (Chapter 7, exercise 1.)

Q: Given a set S of k strings, we want to find every string in S that is a substring of some other string in S. Assuming that the total length of all the strings is n, give an O(n)-time algorithm to solve this problem.

Ans: We can build a suffix tree in linear time and check if there's an inner node that corresponds to a full string (constant time per node).

Assume we are given strings $S_1, S_2..., S_k$

Build a generalized suffix tree of $S_1\$_1S_2\$_2...S_k\$_k$ with k distinct terminal markers $\$_1, ..., \$_k \notin \Sigma$.

This can be done in linear time.

Assuming that we label leaves with (i,j) if they represent suffix $S_i[j..|S_i|]$ of S_i, traverse the tree and find those leaves labelled (i,0), i.e. the leaves that correspond to the full strings.

This takes time linear in the tree size, which itself is linear in the input size.

The descendant leaves of the parent of (i,0) (which is reached by an edge labelled $\$_i$) represent all matches from the set; this follows from the basic invariant of suffix trees. Find any one match by descending to any leaf (but (i,0)).

This again takes linear time.

Q5. (Chapter 7, exercise 2.)

Q: For a string S of length n, show how to compute the N(i), L(i), L'(i) and sp, values (discussed in Sections 2.2.4 and 2.3.2) in O(n) time directly from a suffix tree for S.

Ans:
