MATH578A Assignment 1

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

February 23, 2020

Q1.

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 ends 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 we 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].

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

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:

$$\{\text{i - size(S) for i if D[i] == size(S)}\}$$

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
A ← Unique(S)
initialize Count[0...length(A)] to all 0
initialize RunningCount[0...length(A)] to all 0
for element e in S do
  Count[A.index[e]]++
end for
intiailize D[0...m-1]
D[-1] ← 0    //For convenience, In actual implementation we would
hardcode D[0] and start the following loop at 1.
for {i=0; i < m; i++} do
  if T[i] not in A then                                        //Case 1
    D[i] ← 0
    {RunningCount[j] ← 0 for j in interval [0,length(a))}
  else if RunningCount[A.index[T[i]] < Count[A.index[T[i]]]    //Case 2a
  then
    D[i] ← D[i-1] + 1
    RunningCount[A.index[T[i]]++
  else                                                         //Case 2b
    j ← i - D[i-1] + 1 //Case 2b
    while j≤i and T[j] != T[i] do
      RunningCount[A.index[T[j]]]−−
    end while
    D[i] ← i − j
  end if
end for
Initialize B[0..m-1] //length of substrings longest substrings formed
by elements of S from starting positions.
i ← 0
Initialize R              //List of starting positions of occurences
while {i < m } do                                              //O(m)
  B[i - D[i]] ← D[i]
  if D[i] == size(S) then
    R.add(i - D[i])
  end if
end while
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:

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:

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:
