MATH-578A: Homework # 1

Due on Tuesday, March 10, 2015

Saket Choudhary
2170058637

Contents

Question # 1

Definition: $SP(i) = \max k < i$ such that $P[1..k] = P[i-k+1..i]$

String: CACGCAACGA

NOTE: Iteration indexed at 0. So SP[0] = 0(by definition) and hence the loop iterations start from 1 and goes till $n-1=9$

Iteration	$SP[i]$	All other SP values examined	# of times inner while loop executed
1	0	-	0
2	1	-	0
3	0	$SP[0]$	1
4	1	-	0
5	2	-	0
6	1	$SP[0]$	1
7	1	-	0
8	1	$SP[0]$	1
9	0	-	0

continued on next page. . .

Question # 2

$S = CACGGCACGG$
NOTE: Indexing starts from 0. By definition $Z[0] = |S| = 10$

The 'cases' are choosen out of:

Case 1: $k > r$. The index for which Z value is being calculated is greater than the right most ending of all the previous(till $k-1$) Z boxes calculated. Since this is as good as having no pre-calculated Z scores, this case leads to explicit character comparison(starting at k) till a mismatch occurs.

Case 2: $k \leq r$ The current position k is inside one of the previoulsy calculated Z boxes. Hence there exists a correpsonding position $k' = k - l + 1$ where l is the left ending of the Z box with its right ending at r, such that $S[k`] = S[k]$. In short, there is a corresponding Z box that occurs in the prefix of S, by definition. There is a corresponding one to one match for $S[k'..r-l+1]$ with $S[k..r]$ and we define this to be another box β with $|\beta| = r-k+1$. $Z[k]$ can be caculated utilising the previoulsy calculated Z scores.

The following three cases arise(we list down explicit comparisons invloved for each case):
Case 2a: $Z'_k < |\beta|$ Starting at k' the length of largest substring that matches the prefix of S is less than size of that β box starting at k'. Since this β box appears starting from k too and $Z'_k < |\beta|$ implies $Z_k = Z'_k$. This is easy to see, since the character appearing at position $k'+1$ does not have a matching character in the prefix of S impllying this is also the case with the character appearing at k. Total comparisons:

1. Comparison: $k \leq r$
2. Assignment/Calculation: $k' = k - l + 1$
3. Lookup: Z'_k
4. Assignment/Caculation: $|\beta| = r - k + 1$
5. Comparison: $Z'_k < |\beta|$
6. Assigment: $Z_k = Z'_k$

No character comparisons are involved. All the above 'comparisons' are constant time.

Case 2b: $Z'_k > |\beta|$ The ubstring starting at k' matches a prefix of S and has length equal to the β box. If we call the box with it's leftmost end=l and rightmost end=r as α, then we know that $S[r+1] \neq S[|\alpha|+1]$ otherwise α would not have been the largest such box. Thus, $Z_k = \beta$ Thus no character comparisons involved in this case too. Comparisons involved:

1. Comparison: $k \leq r$
2. Assignment/Calculation: $k' = k - l + 1$
3. Lookup: Z'_k
4. Assignment/Caculation: $|\beta| = r - k + 1$
5. Comparison: $Z'_k > |\beta|$
6. Assigment: $Z_k = Z'_k$

Again, all the operations are constant time.

[Question # 2] continued on next page...

Case 2c: $Z'_k = |\beta|$
The substring starting at k *might* have a matching prefix in S, and hence explicit character comparions are required from $r+1$ to $q \geq r+1$ till the first mismatch occurs. These iterations are bound by $O(|S|)$ since the maximum possible mismatches are $O(|S|)$.
Comparisons involved:

1. Comparison: $k \leq r$
2. Assignment/Calculation: $k' = k - l + 1$
3. Lookup: Z'_k
4. Assignment/Caculation: $|\beta| = r - k + 1$
5. Comparison: $Z'_k == |\beta|$
6. Iteration for explicit character comparison: $while(Z_[r+1] == Z[\alpha+1])$..., bounded by $O(|S|)$

Except character comparison step, rest all steps are constant time.

The Z and associated l, r values for different iterations are given by:

i	$Z[i]$	l_i	r_i	Case
1	2	1	0	1
2	3	3	1	1
3	4	3	0	1
4	5	4	0	1
5	6	10	5	1
6	6	10	0	2a
7	6	10	1	2a
8	6	10	0	2a
9	6	10	0	2a

Question # 3

In order to determine if α is a circular rotation of β, we make the following observations:

1. All possible $|\beta|+1$ length $|\beta|$ substrings of $\beta\beta$ represents all possible circular rotations of β. This is intuitive, since a cicular rotation would involve concatenating the start of string to its end.
2. The next step involves searching for P in TT. This is possible in linear time using either Z algorithm or any other linear time exact matching algorithm.
3. If P appears in TT, then P should appear atleast twice in $P\$TT$. The Z start indices of occurence of P in $P\$TT$ can be determined by querying all those points in the Z value array, which exceed $|\alpha|$.

The psuedocode is listed as Algorithm 1. It is bounded by $O(|\alpha| + 2|\beta|)$

Algorithm 1 Find circular rotation

```
Input: Two string $\alpha$, $\beta$ and a linear time algorithm say $Z$ algorithm to solve exact string matching problem in linear time
Output: Determine if $\alpha$ is a circular rotation of $\beta$
  $S \Leftarrow \alpha\$\beta\beta$
  $Z_{values} \Leftarrow getZValues(S)$
  $N \Leftarrow |S|$
  while $N \neq 3|S| + 1$ do
    if $Z_{values}[i] \geq |\alpha|$ then
      return $true$
    end if
    $N \Leftarrow N + 1$
  end while
  return $false$
```

Question # 4

Question 6:

Case 2b of Z algorithm can be split into following sub cases:
Case 2b $Z'_k > |\beta|$
Case 2c $Z'_k = |\beta|$
Let r denote the right most edge of the Z box(call it α) such that $k \leq r$. Let l denote the left most edge of this Z box. When $Z'_k > \beta$, let $S[r+1] = X$ and $k' = k - l + 1$ denote the cooresponding position(there is an α box that appears as the prefix of S by definition) in the prefix of S, such that $S[1...k']$ matches $S[l...k]$ and also $S[1...r-l+1]$ matches $S[l..r]$(The α box)
Consider $r' = r - l + 1$ let $S[r'+1] = Y$, then $X \neq Y$, else the Z box would have been longer than $|\alpha|$, contrary to the definition.
Now consider $Z'_k > |\beta| \implies$ there exists a matching prefix of S for substring starting at k' which also implies that $S[Z'_k + 1] = S[r'+1] = Y$ because Z'_k will be at least $|\beta| + 1$ in size.
Since $X \neq Y$, $Z_k = |\beta|$, because $|\beta|$ is the length of longest matching prefix given $S[|\beta|+1] = S[r'+1] \neq S[r+1]$

Question 7:

No. there is no extra speedup if we take into consideration all comparisons.
Case 2a, 2b approach: Comparison required: 1 character comparison(at max) on failure of conditional check $Z_k < |\beta|$

Case 2a,2b,2c approach: Comparison required: 1 integer comparison $Z_k == |\beta|$

continued on next page. . .

Question # 5

Observations:

1. The first occurence of parameters is very flexible, since they can be made to match to any other parameter.
2. Any parameter appearing more than is as good as a constraint

Approach:

1. In one pass, store the indexes where parameters appear and the total number of parameters appearing till each index
2. The first appearance of any parameter in the string is marked 0
3. All subsequent appearances of a parameter are changed to an integer denoting the number of parameters that appeared since it's last occurence(see Example). The tokens are left as is while the parameters get mapped to integers.
4. These steps are run on P and T individually and gives back P' and T' such that tokens remain the same while the parameters have been converted to an integer equivalent. This can be done in $O(|P| + |T|)$ time. This is done by $parameterEncoder()$ method in Algorithm 3.
5. In another $O(|P|+|T|)$ pass we run traditional Z algorithm, calculating the Z_i^p values on the modified string S'
6. A $p - match$ is equivalent to having Z_i^p values equal to $|P|$. This is done in Algorithm 2

Complexity:
$O(|P|+|T|)$ for converting the string P to P', T to T'. This is linear time and space since every parameter is accessed just once. And, $O(|P| + |T|)$ for running Z algorithm on $S = P\$T$. Thus in total bounded by $O(|P| + |T|)$
Correctness: The first occurence of parameters can be matched to any other parameter and hence this is reflected by assigning the same value of '0' to all such first occurences. Everytime a parameter appears again, Consider $S1 = abXab$ and $S2 = baXba$ Then $S1$ gets mapped to $00X22$ and $S2$ gets mapped to $00X22$. NOTE: The '2' appears counting the total number of parameters **including** the one at i itself. It is easy to see that $S1$ and $S2$ get converted to the same string by $parameterEncoder()$. The main property being exploited in this algorithm is that the first appearance of parameters can be mapped to any parameter, but all subsequent occurences have a fixed mapping and it is fixed by the number of parameters between two consequent occurences of a parameter.
Another example:
$S1 = XYabCaCXZddbW$ and
$S2 = XYdxCdCXZccxW$ so
$parameterEncoder(S1) = XY00C2CXZ014W$ and
$parameterEncoder(S2) = XY00C2CXZ014W$

Algorithm 2 Find p-matches of P in T

Input: String P, T

Output: Find all p-matches of P in T in $O(P+T)$

```
  $m \Leftarrow |P|$
  $n \Leftarrow |T|$
  $P' \Leftarrow parameterEncoder(P)$
  $T' \Leftarrow parameterEncoder(T)$
  $S \Leftarrow P'\$T'$
  $z_{values} \Leftarrow getZValues(S')$
  return all positions where $z_{values} == m$
```

Algorithm 3 parameterEncoder(P, lastPa)

```
  $m \Leftarrow |P|$
  $P' \Leftarrow null$
  $lastParameterMap \Leftarrow \{\}$
  $parametersTotal \Leftarrow []$
  for $i \Leftarrow 1$ to $m$ do
    if $isParameter(P[i])$ then
      if $P[i]$ in $lastParameterMap$ then
        $parametersTotal[i] \Leftarrow parametersTotal[i-1]+1$
        $lastOccurenceAt \Leftarrow lastParameterMap[P[i]]$
        $numParamsFromLastOccurnce \Leftarrow parametersTotal[i] - parametersTotal[lastOccuredAt]$
        $P' \Leftarrow concat(P', numFromLastOccurence)$
      else
        $lastParameterMap[P[i]] = i$
      end if
    else
      $P' \Leftarrow concat(P', P[i])$
    end if
  end for
  return P'
```

continued on next page...

Question # 6

Observations:

1. Any substring in T that can be formed from characters in S must have the same number of characters of each type.

Complexity:
$O(|S|)$ to create frequency table of characters in S. $O(|S|)$ to create the frequency table of first $|S|$ characters in T and then another $O(|S|-|T|)$ to loop through the rest of characters in T(This is done to maintain an aray of maximum possible length of substring that can be formed out of $T[i]$).
Correctness:
The only criteria for a substring to be made of characters in S is that the substring should have an exact match with the frequency map of characters in S, or the total sum of these frequencies matches(ensuring the sum is over the same keys for both S and substring in T). We iterate through first S characters in T to create an initial map, while maintaining a sum(initially set to 0). If the incoming character belongs to the frequency map of S and the current frequency of this incoming character is less than what it is in S, say f_x, we increment the count. On other hand, if it exceeds f_x we penalise the sum by subtracting one. Also, as the loop iterates through i, the $i+|S|$ element is being added to the frequency map, while the $i-|S|$ is being removed. Depending on whether the elements being added and deleted are present/absent in the frequency map of S, the sum is increased/decreased and is maintained in an array. This array stores the length of maximum possible substrings possible(such that its characters belong to S) at each i. The *sum* stored at each iteration is correctly updated, and hence returns the maximum length of substring possible at each position i.
The algorithm correctly finds occurence of all substrings of T that can be formed from bag of characters in S.

Implementation:

1. $createFrequencyOfCharacters$ method in Algorithm 4 takes a string and creates a hash table out of it where the keys represent unique characters and the associated value represents the frequency of occurence.

continued on next page...

Algorithm 4 Find multisets

Input: String S, T

Output: Find all substrings of T that are formed by characters of S. For each position in T, the maximum possible length substring that can be formed using character in S

```
$patternMap \Leftarrow CreateFrequencyOfCharacters(S)$
$longestSubstringPossible \Leftarrow []$
$m \Leftarrow |S|$
$n \Leftarrow |T|$
$i \Leftarrow 2$
$sum \Leftarrow 0$
while $i \leq m$ do
  if $S[i]$ in $sequenceMap.keys()$ then
    $sequenceMap[S[i]] \Leftarrow sequenceMap[S[i]] + 1$
  end if
  if $patternMap[S[i]] >= 1$ and $sequenceMap[S[i]] < sequenceMap[S[i]]$ then
    $sum \Leftarrow sum + 1$
  else
    $sum \Leftarrow sum - 1$
  end if
  $i \Leftarrow i + 1$
end while
$longestSubstringPossible[1] = sum$
$previous \Leftarrow S[1]$
for $i \Leftarrow 2$ to $n - m$ do
  $next \Leftarrow S[i + m]$
  if $sequenceMap[previous] > patternMap[previous]$ then
    $sum \Leftarrow sum + 1$
  else
    $sum \Leftarrow sum - 1$
  end if
  if $patternMap[next] >= 1$ and $sequenceMap[next]] < sequenceMap[S[i]]$ then
    $sum \Leftarrow sum + 1$
  else
    $sum \Leftarrow sum - 1$
  end if
  $longestSubstringPossible[i] = sum$
  $previous \Leftarrow S[i]$
end for
```

continued on next page. . .

Question # 7

Observations:

1. $Z = PT$ can have Z values greater than $|P|$ in case of multiple repeats.
2. If sp value of any index is greater than $|S|$ then, it for sure has a P occuring in the prefix, by definition.
3. If the start and end of a substring in PT have sp values $\geq |P|$ and they can be matched to $P[1]$ and $P[|P|]$, then this substring in T is an exact match of P

Code Attached. It is a modification of the $kmp_algorithm.cpp$ that was provided.
Complexity:
$O(|P|+|T|)$ for running the sp algorithm on PT and another pass over $|P|+|T|$ values to determine which i are greater than or equal to $|P|$ such that $S[i] == P[|P|]$ and $S[i-|P|] == P[1]$.
Correctness:
Anywhere along the string $S = PT$, if sp value at position i is greater than equal to $|P|$, it implies there exists a substring in $S[|P|...i]$ that is an exact match to P. If it is exactly equal to $|P|$, then P occurs in T(by definition). but if it is strictly greater, then we need at most 2 character comparisons to check if the substring is indeed P. sp values can be strictly greater, given that there is no separator between P and T.
Example:
$P = abc$,
$T = bcabcbabcabcbcabc$
$PT = abcabcbabcabcbcabc$
$sp\ = 00123012312345678$
In SP consider positions $14, 15, 16[bca]$ with sp values $4, 5, 6$ $sp_{16} > |P|$. But $S[16] = a \neq P[3] = c$ and hence this can be disregarded for occurence of P in T.

Algorithm 5 Find occurence of P in T in linear time using sp values
Input: Strings P and T
Output: Find all occurences of P in T in linear time using sp values

```
$S \Leftarrow PT$
$sp_{values} \Leftarrow SPCalculator(S)$
$N \Leftarrow |S|$
$P_{occurences} = []$
while $N \geq |P|+1$ do
  if $sp_{values}[i] \geq |P|$ then
    if $S[N] == P[|P|] and S[N-|P|] == P[1]$ then
      $P_{occurences}.push(i)$
      $N \Leftarrow N - |P|$
    else
      $N \Leftarrow N - 1$
    end if
  else
    $N \Leftarrow N - 1$
  end if
end while
return $P_{occurences}$
```

continued on next page...