FIT3155: Week 2 Tutorial - Answer Sheet

(Scribe: Dinithi Sumanaweera)

Question 2

In the above Z-algorithm, for any given input string if we found that $Z_2 = q$ (where $q > 0$), then the values of $Z_3, Z_4, \ldots, Z_{q+1}, Z_{q+2}$ can be immediately obtained without additional character comparisons. Reason why?

If Z[2] = q, we know that the first q characters of the string match exactly the q characters starting from k=2. That is str[1..q] = str[2..q+1].

This implies:
str[2] = str[1],
str[3] = str[2] (which we know from above = str[1])
str[4] = str[3] = str[2] = str[1]
…
str[q+1] = str[q] = str[q-1] = ... = str[1].

This implies that the first q+1 characters of the string repeat the same character, with q+2 being a different character. For example: "aaaaaaaaab...".

Therefore, every Z[k] values for 2 < k <= q+1 can be inferred without explicit comparisons, as we know that it will simply be the **length** of the string str[k... q+1]. That is:
Z[3] = q-1
Z[4] = q-2
...
Z[q+1]=1

Finally, by the definition of Z[2]=q, we know that str[q+2] is different from (str[q+1]=str[1]). This implies Z[q+2] has to be 0.

Question 3

Stare at the lecture slide handling the preprocessing CASE 2b of the Z-algorithm. When $Z_{k-l+1} \geq r-k+1$, the algorithm does explicit comparisons until it finds a mismatch. This is a reasonable way to organize the algorithm, but in fact CASE 2b can be refined so as to eliminate an unneeded character comparison. Argue that when $Z_{k-l+1} > r-k+1$, then $Z_k = r-k+1$, and hence no character comparisons are necessary. Therefore, explicit character comparisons are needed only in the case when $Z_{k-l+1} = r-k+1$.

See worked out illustration attached at the end.

Question 4

Reason a potential linear-time solution for the following problem: Given two equal-length strings α and β from a fixed alphabet, determine if α is a circular (or cyclic) rotation of β. For example, $\alpha =$ defabc is a circular rotation of $\beta =$ abcdef.

Solution using Z-boxes:

If α is a cyclic rotation of β, then:
α is of the form someprefix + somesuffix
β is of the form somesuffix + someprefix

So, to check if α is a cyclic rotation of β, double up the string β to get $\beta\beta$ which will be of the form:
$\beta\beta$ = somesuffix + someprefix + somesuffix + someprefix

This allows us to convert the problem into an exact pattern matching problem since we are search for α of the form:
someprefix + somesuffix

Thus, perform an exact pattern matching using Z-algorithm over α$\$\beta\beta$ to find if α occurs in $\beta\beta$. If so, α is a circular rotation of β.

An "out of the (Z-)box" solution! (credited to Dijkstra, popularized by Gusfield) -- NOT EXAMINABLE BUT A VERY CLEVER TO NOTE HOW IT FUNCTIONS

There is another approach that works, related closely to the problem of finding the (lexicographically) smallest rotation of a given string.

Consider this logic (using 1-based indexes):

```
A = αα;
B = ββ;
i = j = -1;
n = len(α)    // also = len(β), note
while(i < n and j < n ) {
len = 1
while (len < n && (A[i+len] == B [j+len])) len++
if (len >= n) {
     print "YES" // smallest rotation at i+1, j+1
     break
}
else if (A[i+len] > B[j+len])  i = i+len
else j = j+len
}
if (i >=n or  j>=n) print "NO"
```

Key idea is that, this solution is answering a yes/no question by attempting to find the positions in A=αα and B=$\beta\beta$ respectively of the *lexicographically smallest* string of length n (assuming α is a rotation of β). Note: αα and $\beta\beta$ are doubled up here to avoid modular arithmetic, although the same logic can be implemented using modulo-n arithmetic on simply α and β strings.

Correctness: For a given (i,j), the above finds the value of len where A[i...i+len-1] is same as B[j...j+len-1], before a mismatch is found (or when n-length string is found). If mismatch is found, and A[i+len] > B[j+len], we know that, for a length n-string (circular rotation) starting at any i+delta (where delta=2...len-1), there is always a corresponding string (circular rotation) that starts at j+delta that is lexicographically less than the one starting at i+delta. This implies i+delta cannot be the starting point of the overall smallest rotation/string, assuming alpha is a circular rotation of beta. This allows us to apply this rule:

```
     else if (A[i+len] > B[j+l])  i = i+l
```

and by symmetry:

```
     else if (A[i+len] < B[j+l])  j = j+l
```

If the assumption (that alpha is a rotation of beta) doesn't hold, it would never find a n-length string.

Question 5

Similar to the above exercise, give a linear-time algorithm to determine whether a linear string α is a SUBstring of a circular string β. Note: a circular string **str[1..n]** is such that the character **str[n]** precedes character **str[1]**.

Can be addressed using the Z-box based solution to the previous question. For this problem, a circular string β can be of course represented in the form of $\beta\beta$. But in fact, you don't have to double up fully that string, and rather just append the prefix of β that is of the same size as α, and run the Z-algorithm as before.

Question 6

Give an algorithm that takes in two string α and β of lengths m and n, and finds the longest suffix of α that exactly matches a prefix of β. Reason the run-time of your algorithm.

If you concatenate $\beta\$\alpha$ and run the Z-algorithm, the largest Z[k], for values of k>|β|+1, such that Z[k]+k-1 = |α|+|β| gives your the longest suffix of α that exactly matches the prefix of β.

Z-box computation takes O(|α|+|β|) time. The subsequent scanning starting from position |β|+2 to find the largest Z values that satisfies the above condition takes O(|α|)-time.

Question 7

Given a string $\mathtt{str}[1..n]$, let $\mathtt{len}(\mathbf{i})$ denote the length of the largest suffix of $\mathtt{str}[\mathbf{i}..n]$ that is also a prefix of $\mathtt{str}$. Give an algorithm that computes $\mathtt{len}(\mathbf{i})$ values. Reason the run-time of your algorithm.

Recall that Z[i] gives the length of the longest substring that start from i, and matches with a prefix.

Let us first initialize len array of size n.

len = [0 0 0 … 0]

and compute Z-array over str[1..n].

Now, we are going to scan Z-array from right to left (from n to 1), checking each Z-value to see if the z-box at i reaches the end, we can find each len[i] value (as follows):

```
If Z[i]+i-1 = n :
     len[i] = Z[i] //found a new largest suffix matches a prefix
Else :
     len[i] = len[i+1] //otherwise, previously found value is copied.
```

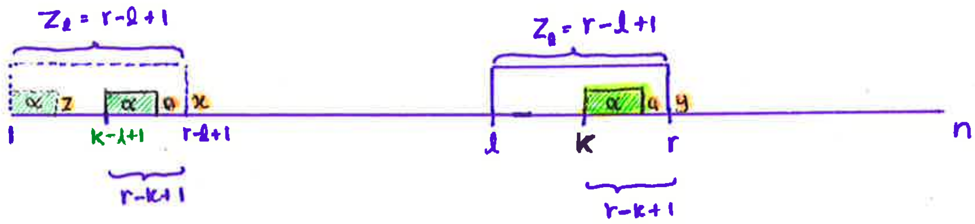
Time complexity:

Computation of Z-values takes O(n). Filling up len array in the reverse direction is also O(n). Total = O(n) time.

Question ③ – Tutorial 1

Z-box
prefix
Newly determined Z box = Z[k] box
Next character at end of a z-box or prefix

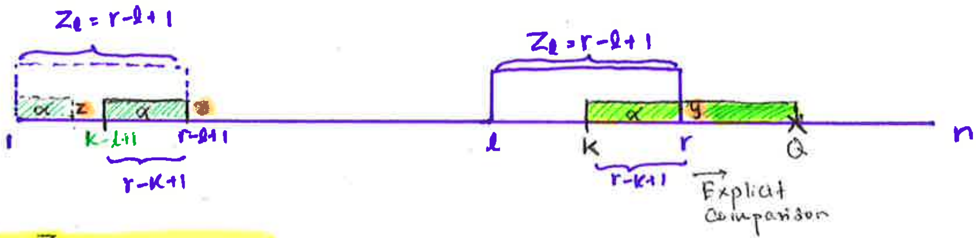
①



If $Z_{k-l+1} < r-k+1$: $Z_k = Z_{k-l+1}$

l and r left unchanged

②



If $Z_{k-l+1} == r-k+1$:

$\boxed{\begin{matrix} x \neq y \\ z \neq x \end{matrix}}$ ← Z-box property always ensures

But we cannot say if $z == y$ or $z \neq y$ until we explicitly compare.

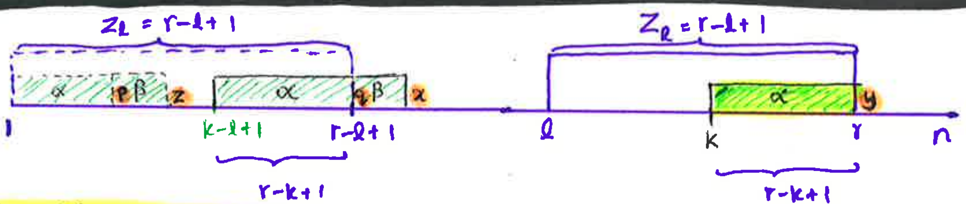
∴ Start naive comparison until a mismatch.

If mismatch occurs at Q,

$Z_k = \alpha + (Q-1)-(r+1)+1$

r = Q-1
l = k

③



If $Z_{k-l+1} > r-k+1$

$\boxed{\begin{matrix} x \neq z \\ p = q \\ q \neq y \end{matrix}}$ ← Z-box property ensures

Since $Str[r-l+2] \neq Str[r+1]$ (i.e $q \neq y$)

We cannot extend Z_k-box beyond r

∴ $Z_k = r-k+1$

l and r left unchanged.