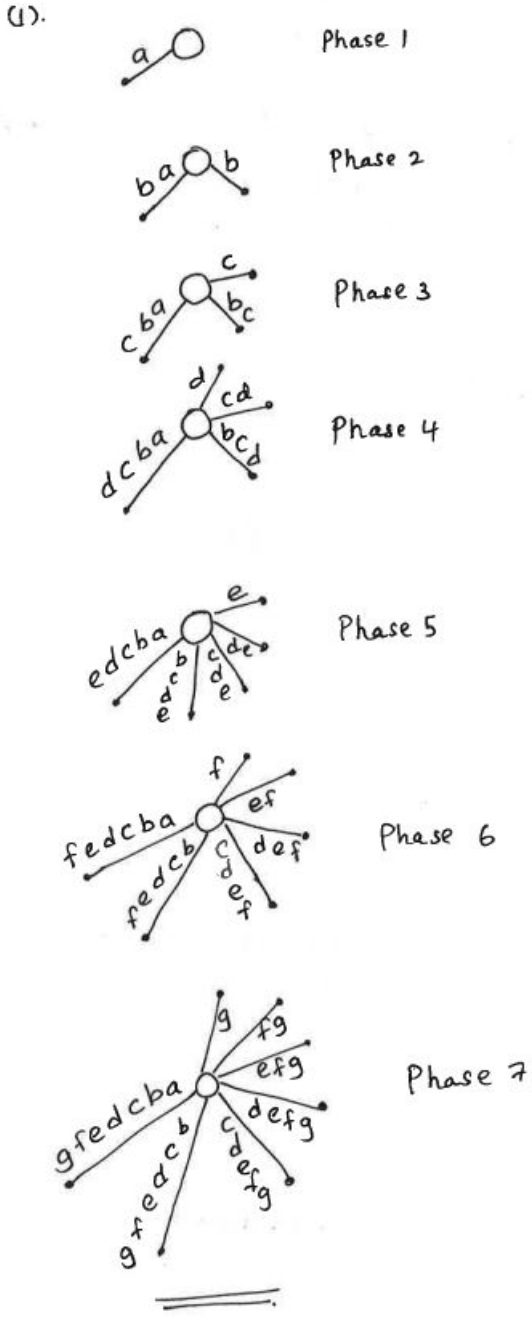
FIT3155: Week 4 Tutorial - Answer Sheet

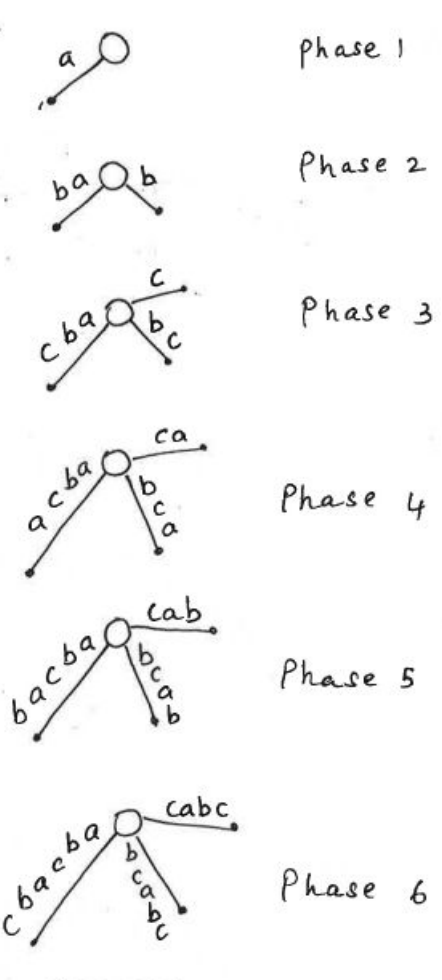
(Scribe: Dinithi Sumanaweera)

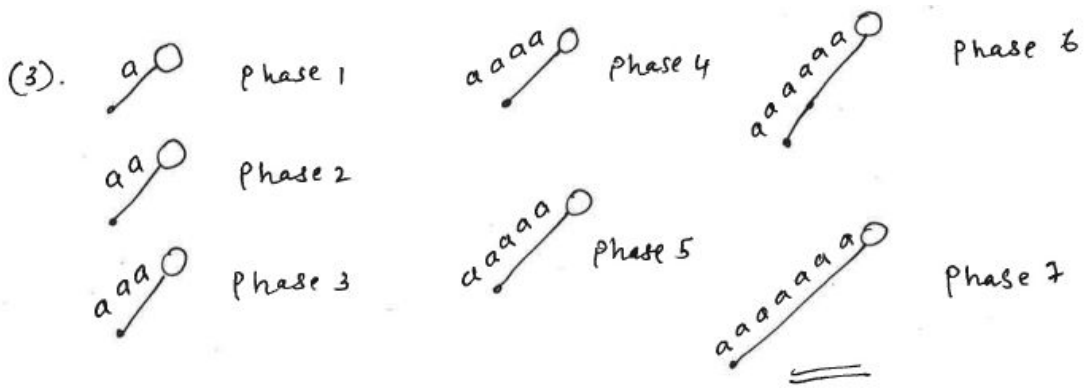
Question 01

(1).



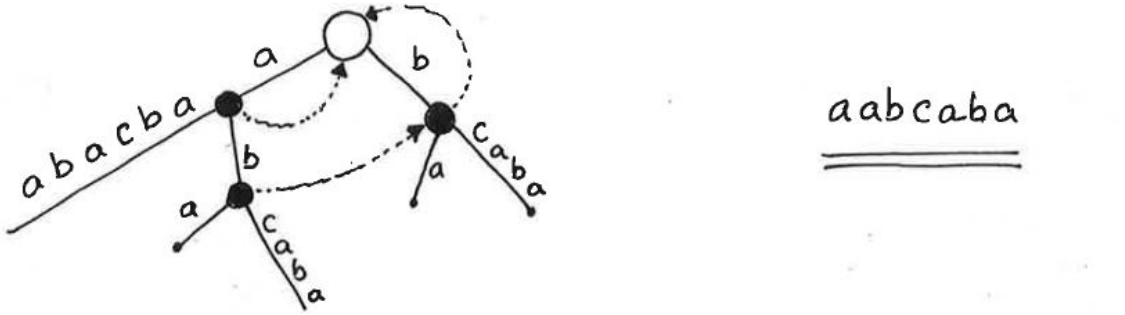
(2).





Question 02

`Identify the suffix links in the implicit suffix tree for string` aabcaba



Question 03

`Given a string t,  find if t is a substring of S`

Recall, every substring of a string S is a prefix of a suffix of S. Therefore, if t is a substring of S, we will have a path from the root of the explicit suffix tree of S (to some internal node) that denotes (contains) t. If such a path representing the substring t in the suffix tree of S exists, then t is a substring of S.

`Given a string t,  find if t is a suffix of S`

If there is a path denoting t$ terminating in a leaf node of the explicit suffix tree of S, then t is a suffix of S.

`Find the number of times string t occurs in S as a substring`

Starting from the root node of the suffix tree of S, traverse the path denoted by the substring t to reach the shallowest internal node (i.e., closest to the root) that contains the string t. The

number of leaves below that internal node gives the number of times the string t occurs in S as a substring.

```
Find the smallest substring of S occurring exactly k times
```

Do a full post-order traversal of the suffix tree of S, and while doing that, record in each internal node the number of leaves under it, while keeping track of the shallowest internal node (i.e., the internal node closest to the root) that has exactly k leaves under it.

```
Find the longest repeated substring within S
```

The longest repeated substring is given by the path to the deepest internal node (i.e., internal node that is farthest from the root) that has >=2 leaf nodes under it.

```
Find the lexicographically smallest suffix of S
```

Traverse the suffix tree of S starting from the root by following the outgoing edge associated with the lexicographically smallest letter at each node (ignore traversing $ only at the root node), until a leaf node is reached. This path gives the lexicographically smallest suffix of S.

```
Find the suffix array of string S
```

Traverse the full suffix tree lexicographically (descending below each node in the alphabetical order of the [first] characters associated with its outgoing edges), and record in an array all the leaf node indexes in the order they are encountered during this traversal. The resultant array upon the full lexicographical traversal of the suffix tree of S gives the suffix array of S.

Question 05

Reason the linear runtime of Ukkonen's algorithm.

- For a string S[1... n], there are n phases in the algorithm.
- Between 2 phases of Ukkonen, there is **at most one** suffix extension (starting at some index j) that is repeated. Thus, in the worst case, there are only 2n extensions overall in this algorithm.
- Therefore, the run time can be then quantified as:
    - 2n + Time taken to run all those 2n suffix extensions.
- Now let's quantify the time taken to run the 2n (worst case) suffix extensions overall. Specifically, we need to identify how many nodes are traverses (skipped) as a sum total over all the 2n suffix extension. To do this:
    - Imagine you have completed some explicit suffix extension j, and going to the next extension j+1.
    - In doing so, you will take a suffix link from some internal node (say) U whose path from the root yields (say) the substring S[j...p0]).
    - The node that receives the suffix link out of U is another internal node (say) V. By the definition of the suffix link from U to V, V's path from the root is defined by the substring S[j+1...p0].

- Once at V, you skip down (zero or more) internal nodes to pursue the j+1 suffix extension, and reach another internal node (say) W, defined by the string (say) S[j+1...p0...p1], on whose outgoing edge you performed the suffix extension to complete the extension j+1. (Note, we have: p0 <= p1.)
- Again, in the next EXPLICIT SUFFIX extension going from j+1 to j+2, you take the suffix link out of W to some internal node X whose path from the root will be S[j+2...p0...p1] (by the definition of suffix link), and then skip down (zero or more nodes) to some internal node Y defined by (say) S[j+2..p0...p1...p2]), **and so on**. (Note, we now have p0 <= p1 <= p2)
- Therefore, over all the 2n EXPLICIT extensions over all phases, we get a non-decreasing sequence of numbers for the form:

  1 <= p0 <= p1 <= p2 <= p3 <= …. <= n.
- The maximum number of suffix extensions is bounded by 2n, so this also bounds the maximum length of this sequence of numbers p0, p1, p2, p3…. by O(n).
- Thus the total effort during suffix extension is bounded by O(n).
- Therefore, Ukkonen's algorithm has the time complexity of O(n).