INSTRUCTIONS:

1. Submit your solution to Gradescope by the due time; no late submissions will be accepted.
2. Type your solution (except figures; figures can be hand-drawn and then scanned); no hand-written solutions will be accepted.

Problem 1 (10 points).

Construct the (implicit) suffix tree for string $s = aababab\$$ using the Ukkonen's algorithm. Show the intermediate steps (at least the implicit suffix tree after each phase and where each phase ends).

Problem 2 (10 points).

Prove that following a suffix link in an implicit suffix tree or a suffix tree the depth of a node (defined as the number of nodes from the root to this node) will decrease by at most 1.

Problem 3 (10 points).

You are given the suffix array SA of an (unknown) string S and an array C, of length $|\Sigma|$, where $C[c]$ says how many occurrences of letter $c \in \Sigma$ there are in S. Design an algorithm to reconstruct S from SA and C in $O(|S|)$ time. Your answer should be fewer than 3 sentences.

Problem 4 (10 points).

Consider two variants of the skew algorithm (to construct the suffix array).

1. In step 6 of sorting G_1, use the procedure of steps 3 and 4 (i.e., let $T = S$, sort all its tokens and then use a recursive call). Is this variant correct? If so, analyze its running time.
2. Partition all suffixes into 4 groups, sort the last 3 groups, followed by sorting the first group and merge the two sorted lists, all using the same procedure with the skew algorithm. Is this variant correct? If so, analyze its running time.

Solution of Part 1. *There is misunderstanding about the statement. Here is exactly how this variant of sorting all suffixes in G_1 works, with detailed explanation. Let $|S| = n$. W.o.l.g. assume $n \mod 3 = 0$; otherwise add necessary \$ to the end of S. Partition S into $n/3$ non-overlapping tokens, each of which has a length of 3. Sort all $n/3$ tokens using bucket-sorting in $O(n)$ time. The (only) purpose of sorting them is to transform a token into a single letter so that a token x is lexcographically smaller than token y if and only if the letter for x is alphabetically smaller than the letter for y. Now, S is transformed into a new string X, by replacing each token in S with the letter above. Clearly, the length of X is $n/3$. Recall that G_1 contains these suffixes of S starting from $S[3i-2]$, where $i = 1, 2, \cdots, n/3$. Suffix in G_1 starting from $S[3i-2]$ corresponds to suffix of X starting from $X[i]$. Note that a suffix s in G_1 is smaller than suffix t in G_1 if and only if the corresponding suffix of s in X is smaller than the corresponding suffix of t in X, again because how tokens of S are coded into letters of X. Finally, we use a recurvie call skew(X) to build the suffix array of X, which gives the sorted list of suffixes of X, and hence sorted list of suffixes in G_1.*

Back to the question. This variant is correct, since it correctly sorts all suffixes in G_1. The running time can be written as a recurrence: $T(n) = T(2n/3) + T(n/3) + O(n)$, in which $T(2n/3)$ corresponds to sorting suffixes in G_2 and G_3, $T(n/3)$ corresponds to sorting suffixes in G_1 with the variant above, and everything else runs in $O(n)$ time. Solving this recurrence gives $T(n) = O(n \log n)$.

Solution of Part 2. This variant is corrent as well. Note that now the length of each token will be 4. In steps

3 and 4, the string X will be of length $3n/4$. So the recurrence will be $T(n) = T(3n/4) + O(n)$, which gives $T(n) = O(n)$.

Problem 5 (16 points).

(We did not cover the construction of the LCP array in class; here is one.) You are given the suffix array *SA* of a string S and the *inverse suffix array* R where $SA[i] = j$ if and only if $R[j] = i$. (This inverse suffix array was named as "rank" array in the step 5 of the skew algorithm, as $R[j]$ gives the "ranking" of suffix-j.)

1. Prove that $LCP[R[i+1]-1] \geq LCP[R[i]-1]-1$, for any $1 \leq i \leq |S|$.
2. Use above to design an algorithm, running in $O(|S|)$ time, to construct the LCP array given *SA* and R.

Solution of Part 1. Consider suffix-i, i.e., the suffix starting from $S[i]$ to the end of S. Recall that $R[i]$ gives the index (i.e.,"ranking") of suffix-i in the suffix array. Assume that suffix-k be the suffix *right* before suffix-i in the suffix array, i.e., $R[k] = R[i]-1$. By the definition of LCP-array, $LCP[R[i]-1]$ gives the length of the longest common prefix (LCP) of suffix-i and suffix-k. If $LCP[R[i]-1] \leq 1$, then $LCP[R[i]-1]-1 \leq 0$, so the desired inequality must be correct as $LCP[\cdot] \geq 0$. Now we can assume $LCP[R[i]-1] \geq 2$. We further assume suffix-i is in the form of aAB and suffix-k is in the form of aAC and their LCP is aA, where a represents a single letter while A, B, and C can be strings. Hence $LCP[R[i]-1] = |aA| = 1+|A|$.

Now consider suffix-$(i+1)$, which is in the form of AB, and suffix-$(k+1)$, which is in the form of AC. Note that suffix-$(k+1)$ must be ranked smaller than suffix-$(i+1)$ in the suffix array, i.e., $R[k+1] < R[i+1]$. This is because string AC is smaller than AB as aAC is smaller than aAB. Note also that suffix-$(k+1)$ could be *right* before suffix-$(i+1)$, i.e., $R[k+1] = R[i+1]-1$, but this is not necessarily the case. Now consider $LCP[R[i+1]-1]$, i.e., the length of the LCP between suffix-$(i+1)$ and the suffix *right* before it in the suffix array. If suffix-$(k+1)$ is the one, i.e., $R[k+1] = R[i+1]-1$, then we must have that A is their LCP and hence $LCP[R[i+1]-1] = |A| = LCP[R[i]-1]-1$. Otherwise, if suffix-$(k+1)$ is not the one right before suffix-$(i+1)$ in the suffix array, i.e., $R[k+1] < R[i+1]-1$, then we must have that *every* suffix between suffix-$(k+1)$ and suffix-$(i+1)$ in the suffix array, of course including the one right before suffix-$(i+1)$, must have the same prefix of A, simply because the suffix array is sorted. Hence, in this case we must have $LCP[R[i+1]-1] \geq |A| = LCP[R[i]-1]-1$.

Note that, if $R[i] = 1$ or $R[i+1] = 1$, i.e., either of them is the smallest suffix, then the desired inequality is not defined, as $LCP[0]$ is not defined.

Common issues found in grading: (a), assuming that suffix-$(k+1)$ is always the one right before suffix-$(i+1)$ in the suffix array; deducting 40% points.

Solution of Part 2. We build the LCP-array by sequentially considering suffix-i, $i = 1, 2, \cdots, |S|$. For each i, we will calculate $LCP[R[i]-1]$, i.e., the length of the LCP between suffix-i and the suffix right before suffix-i in the suffix array. Note that the one that is right before suffix-i is suffix-$(SA[R[i]-1])$.

Consider the case of $i = 1$. To compute $LCP[R[1]-1]$, i.e., the length of the LCP between suffix-1 and suffix-$(SA[R[1]-1])$, we compare each pair of letters of the two suffixes until a mismatch happens. The total number of comparions is therefore $LCP[R[1]-1]+1$ where the last "1" accounts for the mismatch. Now consider the general case of processing suffix-$(i+1)$, i.e., to compute $LCP[R[i+1]-1]$, assuming that $LCP[R[i]-1]$ is already available. Let $k := SA[R[i+1]-1]$, i.e., suffix-k is right before suffix-$(i+1)$ in the suffix array. Note that we do not need to calculate the LCP of suffix-$(i+1)$ and suffix-k from scratch as in the base case. In fact, we can use the statement proved previously to skip unnecessary comparisons.

Specifically, since we know $LCP[R[i+1]-1] \geq LCP[R[i]-1]-1$, i.e., the first $LCP[R[i]-1]-1$ letters of suffix-$(i+1)$ and suffix-k are identical, we can safely compare them starting from position $LCP[R[i]-1]-1+1 = LCP[R[i]-1]$. We initialize $LCP[R[i+1]-1] = LCP[R[i]-1]-1$, and increase it by 1 when a match happens and terminate when a mismatch occurs. Hence, the total number of comparisons needed is exactly $LCP[R[i+1]-1] - (LCP[R[i]-1]-1)+1 = LCP[R[i+1]-1] - LCP[R[i]-1]+2$. The total number of comparisons, by summing up for $i = 1,2,\cdots,|S|$, is $(LCP[R[1]-1]+1)+(LCP[R[2]-1]-LCP[R[1]-1]+2)+\cdots+(LCP[R[|S|]-1]-LCP[R[|S|-1]-1]+2) = LCP[R[|S|]-1]+2|S| = O(|S|)$.

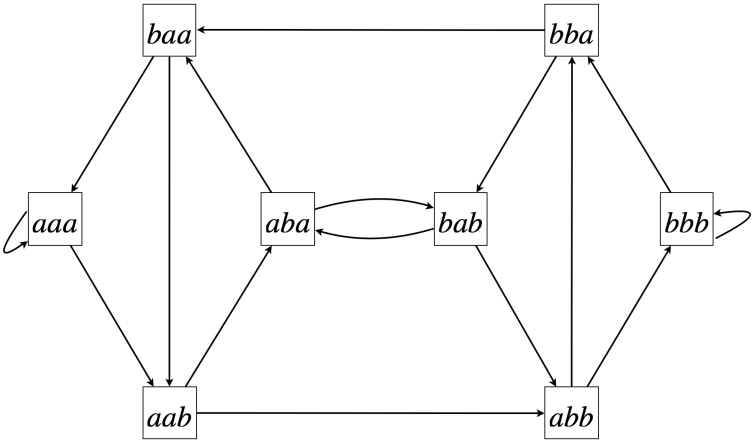
Again be careful about the case of $R[i]=1$. When process suffix-i where $R[i]=1$ we simply skip this case as $LCP[0]$ is not defined and hence no need to compute it. For suffix-$(i+1)$, since the inequality does not apply, we simply compare suffix-$(i+1)$ with the suffix right before it starting from the first position. This twist may increase the running time, but up to $O(|S|)$, so the total running time is still in $O(|S|)$ time.

Problem 6 (10 points).

Give two distinct 9-letter words over the alphabet $\Sigma = \{a,b\}$ that are not cyclic rotations of each other but that have the same first 3 columns of the BWT matrix. *Hint:* consider paths in directed graph $G = (V,E)$ where $V = \Sigma^3$ and there is an edge from xyz to yzw for any $x,y,z,w \in \Sigma$.

Solution. Let s be a length-9 circular string, i.e., assuming that the last letter of s is followed by its first letter. Recall that the BWT matrix of s is the sorted list of all its 9 (circular) rotations. What are the first 3 columns of the BWT matrix? It is the sorted list of all the length-3 substrings of s (again there are 9 of them since s is circular). Therefore, if two strings s and t have the same set of length-3 substrings, then their first 3 columns of the BWT matrices will be the same. To be precise, each string can have duplicated length-3 substrings, and s and t have the same first 3 BWT columns if and only if they have the same *multi-set* of length-3 substrings, i.e., the duplicate of each length-3 substring should also be the same.

Above analysis leads to this approach. Construct the graph $G = (V,E)$ according to the hint; see below. Clearly, a circular string of length-9 corresponds to a cycle in G with 9 vertices. Such a cycle may traverse one vertex more than once, corresponding to duplicate length-3 substrings. Hence, two circular length-9 strings with identical multi-set of length-3 substrings correspond to two cycles in G that go through the same multi-set of vertices. If the two cycles in G are not cyclic rotations of each other (cycles $v_1 \to v_2 \to v_3 \to v_4 \to v_1$ and $v_3 \to v_4 \to v_1 \to v_2 \to v_3$ are cyclic rotations of each other), then the resulting circular strings are not cyclic rotations of each other.



It is not hard to find such two cycles in G. There are several of them. Here are a few pairs:

$s = aaababbaa$ and $t = aaabbabaa$;
$s = bbbabaabb$ and $t = bbbaababb$;
$s = aaababbba$ and $t = aaabbbaba$;
$s = bbbabaaab$ and $t = bbbaaabab$;
$s = aaabbbabb$ and $t = aaabbabbb$;
$s = aaabbbaab$ and $t = aaabaabbb$.

Common issues found in grading: (a), just gave an answer without any intermediate procedure; deducting 50% points. (b), built the graph according to the hint but failed to mention the key point (paths/cycles share the same multi-set vertices); deducting 30% points. (c), two strings are circular rotations of each other; deducting 50% points.