1 Circular string matching

This problem comes from Gusfield’s book: Exercise 2 on p.11. You are given two strings $\alpha$ and $\beta$, with length $n$ and $m$ respectively. We assume $n \leq m$. Here, $\beta$ is treated as a circular string. A circular string $\beta$ of length $n$ is a string in which character at position $n$ is considered to precede character at position 1. That is, you can create a substring with a suffix of $\beta$ followed by a prefix of $\beta$. For example, let $\beta = accatggc$. Then $\alpha = gcac$ matches the circular string $\beta$.

Now give a linear time algorithm for determining whether $\alpha$ matches a substring of the circular string $\beta$.

2 Match counts

This problem is the Exercise 9 on p.14 from Gusfield’s book. You are given two strings $S_1$ and $S_2$ of $n$ characters each, and an additional parameter $k$ (where $k \leq n$). Each string has $n-k+1$ substrings of length $k$, and so there are $\Theta(n^2)$ pairs of substrings where one substring is from $S_1$ and the other from $S_2$. For a pair of substrings, we define the match-counts as the number of opposing characters that match when the two substrings of length $k$ are aligned (without gaps). The problem is to compute the match-count for each pair of $\Theta(n^2)$ substrings. For example, let $S_1 = aabb$ and $S_2 = abab$ and $k = 3$. Then we have four pairs of substrings: $(aab, aba), (aab, bab), (abb, aba), (abb, bab)$. The match-counts are 1, 2, 2, 1 respectively. Since we need to compute for each pair, the time is $\Omega(n^2)$. The naive algorithm runs in $O(kn^2)$ time.

Now give a simple algorithm for computing the match-counts for all pairs of substrings in $O(n^2)$ time.

3 On unique tandem repeats

This problem is about tandem repeat (TR), which may have biological meaning in DNA sequence comparison. Tandem repeat is, given a string $S$, the substring $S[a..a+2d]$ where $S[a..a+d] = S[a+d+1..a+2d]$. Here $S[a..b]$ is a substring of $S$ starting from position $a$ and ending at position $b$.

For example, $S = caabaabcd$. A tandem repeat is aaabaab. In the assignment, you are to study some combinatorial properties of tandem repeat. We are interested in the number of unique tandem repeats. For example, $S = aaaab$. Here, aa appears three times, but in terms of unique tandem repeat, we consider aa as one occurrence. There is a nice result that says given a string with $n$ letters, the number of unique tandem repeats is bounded by $2 \cdot n$. Note, if we drop the uniqueness requirement, the number of tandem repeats can be as large as $O(n^2)$. Now you are going to prove (part of) the claim.
Consider string $S$, where three TR start at position 0 of $S$, with length $2a$, $2b$ and $2c$ respectively (and $a < b < c$). I claim that at least one of three TR must also occur somewhere inside $S$.

1. Given a brief argument why the above claim leads to the $2n$ bound.
2. Now show if $c \geq 2a$, then the claim holds.
3. The case $c < 2a$ (and so $2a > b$). I claim that the TR of length $2a$ also appears in $S$, starting from $b - a$ and ending at $a + b - 1$. We needs some case analysis and will just do one here. Show that for $0 \leq x < 2a - b$, $S[x] = S[x + b - a]$. Can you see how to finish the proof from here?

4 Cyclic rotations

You are given two (both of length $n$) strings $\alpha$ and $\beta$. The task is to check whether these two strings are cyclic rotation of each other. We say $\alpha$ and $\beta$ are cyclic rotation of each other if they are both linearizations of some circular string. For example, $abcab$ and $cabab$ are cyclic rotation of each other.

Someone proposes the following algorithm for this problem.

```
1: $x \leftarrow \alpha\alpha$
2: $y \leftarrow \beta\beta$
3: $i \leftarrow 1, j \leftarrow 1$
4: while $i \leq n$ and $j \leq n$ do
5:     $k \leftarrow 1$
6:     while $k \leq n$ and $x[i + k] = y[j + k]$ do
7:         $k \leftarrow k + 1$
8:     end while
9:     if $k > n$ then
10:        print “$\alpha$ is a circular rotation of $\beta$”
11:        break
12:    else
13:        if $x[i + k] > y[j + k]$ then
14:            $i \leftarrow i + k$
15:        else
16:            $j \leftarrow j + k$
17:        end if
18:    end if
19: end while
20: if $i > n$ or $j > n$ then
21:    print “$\alpha$ is not a circular rotation of $\beta$”
22: end if
```

Now, here are your tasks.

1. Is this algorithm correct? Justify your answer. Note that I am asking whether the main algorithmic idea works and not about implementation details. You need to give a proof if you believe the algorithm is correct. Or give a counter-example if you think the algorithm is wrong.

2. What is the running time of the algorithm? You should base your answer on the number of character comparisons.