Longest repeated substring

One of the most remarkable qualities of genome sequences is the large repetitions of certain fragments within the genome. This is mainly the case with eukaryotes. The human chromosome Y, for example, consists mainly of repeated subsequences. When screening 3.6 million DNA base pairs extracted from *C. elegans* more than 7000 families with repeated sequences were identified. Prokaryotes**,** on the other hand are hardly ever a part of repeated sequences in the genome, except for some minor and well structured repetitions. Aside from the incredible amount of repetitions, DNA also stands out because of the large variety of repeated structures, the diversity of mechanisms which are represented to explain the origin and maintenance of these repetitions, and the biological functions that some of these repetitions can play.

Assignment

It can be concluded from the introduction that genome research requires efficient algorithms to trace different types of repeated structures in genome sequences. Especially required is the implementation of a algorithm to find the *longest repeated substring* (LRS) within a given DNA sequence. To do this, follow these steps:

1. Write a function LCP to which two strings must be passed as argument. This function should return the *longest common prefix* (LCP) of the two given strings. If the given strings have no common prefix, then an emtpy string should be returned as a result. The longest common prefix of the strings banaan and banana, for example, is bana. The function should differentiate between uppercase and lowercase letter when comparing the prefixes.

   Note(terminology): A prefix consisting of one or more letters in front of a string is called the *prefix* of that string. A suffix consisting of one or more letters at the end of a string is called the *suffix* of that string. For example, bio is the prefix of the string biotechnology, and technology is the suffix of the same string.
2. Write a function LRS to which a string must be passed as argument. This function should return the *longest repeated substring* (LRS) of a given string. Note that we are mainly interested in using this function to determine the longest repeated subsequence of a given DNA sequence (a string consisting only of the letters A, C, G and T), but that this function should also be able to extract the longest repeated substring out of any string. The required algorithm to determine the LRS of a given string, works as follows:
   1. First make a list of all suffixes of a given string. For the string banana, for example, a list with the suffixes a, na, ana, nana, anana and banana should be drawn up. The order of the suffixes in this list is initially of no importance (see next step).
   2. Sort the suffix list alphabetically. This causes the suffixes with a common prefix to be listed underneath each other. The sorted suffix list of the previous example would then be: a, ana, anana, banana, na en nana.
   3. Calculate the LCP for each pair of successive suffixes (use the function LCP). The longest of these common prefixes should then be the LRS we are looking for. If there are multiple common prefixes that are the longest, then the first one which was found when going through the pairs (from left to right) should be returned. From the sorted suffix list a, ana, anana, banana, na and nana it is immediately clear that ana is the LRS of the string banana. This example also shows that repetitions of a LRS can partially overlap.

Example

```
>>> LCP('banaan', 'banana')
'bana'
>>> LCP('mango', 'mandarin')
'man'
>>> LCP('peach', 'peach')
'peach'
>>> LCP('apple', 'appletree')
'apple'
>>> LCP('appletree', 'apple')
'apple'
>>> LCP('apple', 'pear')
''
>>> LCP('potential', 'Pomelo')
''

>>> LRS('CAATCCGCCTGTATTAAGGTACGGCCTTTTGACTGAAGTT')
'GCCT'
>>> LRS('AGTGTGGGGCATGCCTATTGGCCACCTTGTAGGAGCCTTG')
'CCTTG'
>>> LRS('GACTAGCCGTCTACTTATTGAATCGCAGCATCATTTCGAG')
'ACT'
>>> LRS('GGCCCGTCGCGACATTTGAGCTAGTCAGCGATCTTTACCT')
'GCGA'
>>> LRS('TGCCCCGTCAACCTCTACTCATCTCGCTCTAAACTGTAAG')
'CTCTA'
```

Eén van de meest opvallende eigenschappen van genoomsequenties is de grote mate waarin bepaalde fragmenten binnen het genoom herhaald worden. Dit is voornamelijk het geval bij eukaryoten. Zo bestaat het Y chromosoom van de mens grotendeels uit herhaalde deelsequenties. Bij een screening van de 3,6 miljoen DNA baseparen geëxtraheerd uit *C. elegans* werden meer dan 7000 families van herhaalde sequenties geïdentificeerd. Daar staat tegenover dat in het genoom van prokaryoten nauwelijks herhaalde sequenties voorkomen, behalve enkele kleinschalige en duidelijk gestructureerde herhalingen. Naast de ongelofelijke hoeveelheid aan herhalingen valt DNA ook op door de grote variëteit aan herhaalde structuren, de verscheidenheid aan mechanismen die worden voorgesteld om de oorsprong en het behoud van deze herhalingen te verklaren, en de biologische functies die sommige van deze herhalingen kunnen spelen.

Opgave

Uit de inleiding zal het wel duidelijk zijn dat genoomonderzoek nood heeft aan efficiënte algoritmen voor het opsporen van verschillende types van herhaalde structuren in genoomsequenties. In het bijzonder wordt gevraagd om voor deze opgave een algoritme te implementeren waarmee de *langste herhaalde deelstring* (LHD) binnen een gegeven DNA sequentie kan gevonden worden. Hiervoor ga je als volgt te werk.

1. Schrijf een functie `LGP` waaraan twee strings als argument moeten doorgegeven worden. Deze functie moet als resultaat de *langste gemeenschappelijke prefix* (LGP) van de twee gegeven strings teruggeven. Indien de gegeven strings geen gemeenschappelijke prefix hebben, dan moet een lege string als resultaat teruggegeven worden. De langste gemeenschappelijke prefix van de strings `banaan` en `banana` is bijvoorbeeld `bana`. Deze

functie moet bij het vergelijken van de prefixen onderscheid maken tussen hoofdletters en kleine letters.

Opmerking (terminologie): Een voorvoegsel bestaande uit één of meer letters vooraan een string noemen we een *prefix* van die string. Een achtervoegsel bestaande uit één of meer letters achteraan een string noemen we een *suffix* van die string. Zo is bio bijvoorbeeld een prefix van de string biotechnologie, en is technologie een suffix van diezelfde string.

2. Schrijf een functie LHD waaraan een string als argument moet doorgegeven worden. Deze functie moet als resultaat de *langste herhaalde deelstring* (LHD) van de gegeven string teruggeven. Merk op dat we voornamelijk geïnteresseerd zijn om deze functie te gebruiken om de langste herhaalde deelsequentie van een gegeven DNA sequentie (een string die enkel bestaat uit de letters A, C, G en T) te bepalen, maar dat deze functie meer algemeen uit elke string de langste herhaalde deelstring kan extraheren. Het algoritme dat je moet gebruiken om de LHD van een gegeven string te bepalen, werkt als volgt:
    1. Maak eerst een lijst met alle suffixen van de gegeven string. Zo moet voor de string banana bijvoorbeeld een lijst met de suffixen a, na, ana, nana, anana en banana opgebouwd worden. De volgorde waarin de suffixen in deze lijst geplaatst worden, is initieel niet belangrijk (zie volgende stap).
    2. Sorteer de suffixlijst alfabetisch. Dit zorgt ervoor dat suffixen met een gemeenschappelijke prefix na elkaar komen te staan. De gesorteerde suffixlijst van bovenstaand voorbeeld wordt dan: a, ana, anana, banana, na en nana.
    3. Bereken voor elk paar opeenvolgende suffixen in de gesorteerde suffixlijst de LGP (gebruik hiervoor de functie LGP). De langste van deze gemeenschappelijke prefixen is dan de LHD waarnaar we op zoek zijn. Indien er meerdere gemeenschappelijke prefixen zijn die het langst zijn, dan moet de eerste die gevonden wordt bij het overlopen van de paren (van links naar rechts) teruggegeven worden. Uit de gesorteerde suffixlijst a, ana, anana, banana, na en nana volgt onmiddellijk dat ana de LHD is van de string banana. Dit voorbeeld geeft meteen ook aan dat de herhalingen van een LHD elkaar gedeeltelijk kunnen overlappen.

Voorbeeld

```
>>> LGP('banaan', 'banana')
'bana'
>>> LGP('mango', 'mandarijn')
'man'
>>> LGP('perzik', 'perzik')
'perzik'
>>> LGP('appel', 'appelboom')
'appel'
>>> LGP('appelboom', 'appel')
'appel'
>>> LGP('appel', 'peer')
''
>>> LGP('pompelmoes', 'Pomelo')
''

>>> LHD('CAATCCGCCTGTATTAAGGTACGGCCTTTTGACTGAAGTT')
'GCCT'
>>> LHD('AGTGTGGGGCATGCCTATTGGCCACCTTGTAGGAGCCTTG')
'CCTTG'
```

```
>>> LHD('GACTAGCCGTCTACTTATTGAATCGCAGCATCATTTCGAG')
'ACT'
>>> LHD('GGCCCGTCGCGACATTTGAGCTAGTCAGCGATCTTTACCT')
'GCGA'
>>> LHD('TGCCCCGTCAACCTCTACTCATCTCGCTCTAAACTGTAAG')
'CTCTA'
```