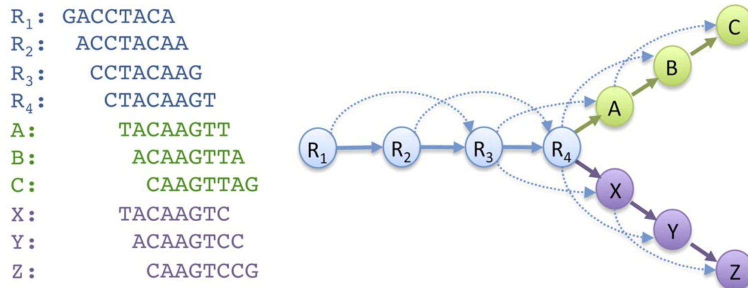
Overlap graph

Genome assembly is the computational process in which the actual sequence of nucleotides making up an organism's genome is determined. Researchers are presently unable to read the nucleotides of an entire chromosome one at a time. Instead, they rely on high-technological chemical methods to determine the order of bases along short strands of DNA (usually less than 1000 bp). These short strands are called **reads**.

To sequence a genome, researchers chemically "blow up" multiple copies of a genome (typically taken from multiple cells of the same organism) into reads and then determine the nature of the reads. To reassemble the genome, they must use overlaps in reads to determine which read pairs are adjacent in the genome. For example, if the short reads G**ACCTACA** and **ACCTACA**A are produced, then we might surmise from the fact that they overlap that they both belong to a substring G**ACCTACA**A. An efficient algorithm for assembly that can handle all possible wrinkles and complications arising from practical concerns (such as errors in reads) still does not exist, so that research in genome assembly remains a highly competitive field.

Assignment

As a first step in genome assembly, most algorithms try to get insight in the overlap between reads by constructing an **overlap graph**. This is a data structure where overlapping reads (represented as labeled circles in the picture below) are connected using arrows. Note that it is possible that one read overlaps multiple other reads, usually due to repeats in the genome or sequencing errors. This causes so-called "forks" in the overlap graph.



Example of an overlap graph (right) constructed from a set of 10 reads of 8 base pairs (left). Reads having an overlap of at least 6 base pairs are indicated by drawing an arrow between both reads. Transitive overlaps — which are implied by other longer overlaps — are indicated as dotted arrows. The fork in the graph is the result of repeat sequences or read errors, and makes it extremely difficult for assembly algorithms to figure out which route has to be followed. Note that we have only considered the forward orientation of each sequence to simplify the figure.

The goal of this assignment is to construct the overlap graph for a given set of reads. In doing so, you proceed as follows:

- Write a function `overlap` that takes three arguments. The first two arguments are reads, represented as strings that only contain the letters `A`, `C`, `G` and `T` (representing the four possible nucleotide bases). The third argument is an integer $k \in \mathbb{N}_0$. The function must return a Boolean value that indicates whether or not the first read has an overlap of length k with the second read. This is the case if the first read ends in the same k bases that are found at the start of the second read.
- Use the function `overlap` to write a function `maximalOverlap` that takes two reads as its arguments. The function must return the length of the maximal overlap between the first and the second read. If the given reads don't overlap at all, the function must return the integer value zero.
- Use the function `maximalOverlap` to write a function `overlapGraph` that takes two arguments: a collection (a list, tuple, set, …) of reads and an integer $k \in \mathbb{N}_0$. The function must return a dictionary that maps each read in the given collection onto the set of all **other** reads in the collection to which it overlaps with of a minimal length k (in case this set isn't empty). One key-value pair of the dictionary thus represents the outgoing arrows of the overlap graph that start at a particular read (used as the key in the dictionary) and lead to all **other** reads that overlap with it. Note that by definition, an arrow never points from a read to the read itself, even though the read might overlap itself.

Example

```
>>> overlap('AAATTTT', 'TTTTCCC', 3)
True
>>> overlap('AAATTTT', 'TTTTCCC', 5)
False
>>> overlap('ATATATATAT', 'TATATATATA', 4)
False
>>> overlap('ATATATATAT', 'TATATATATA', 5)
True

>>> maximalOverlap('AAATTTT', 'TTTTCCC')
4
>>> maximalOverlap('ATATATATAT', 'TATATATATA')
9

>>> reads = ['AAATAAA', 'AAATTTT', 'TTTTCCC', 'AAATCCC', 'GGGTGGG']
>>> overlapGraph(reads, 3)
{'AAATTTT': {'TTTTCCC'}, 'AAATAAA': {'AAATTTT', 'AAATCCC'}}
>>> overlapGraph(reads, 4)
{'AAATTTT': {'TTTTCCC'}}

>>> reads = ['GACCTACA', 'ACCTACAA', 'CCTACAAG', 'CTACAAGT', 'TACAAGTT', 'ACAAGTTA', 'CAAGTTAG', 'TACAAGTC', 'ACAAGTCC', 'CAAGTCCG']
>>> overlapGraph(reads, 6)
{'CTACAAGT': {'ACAAGTCC', 'ACAAGTTA', 'TACAAGTC', 'TACAAGTT'}, 'TACAAGTT': {'CAAGTTAG', 'ACAAGTTA'}, 'ACCTACAA': {'CTACAAGT', 'CCTACAAG'}, 'ACAAGTCC': {'CAAGTCCG'}}
```

Note: the last example represents the overlap graph depicted in the introduction of this assignment.

Sources

- **Miller JR, Koren S, Sutton G (2010)**. Assembly algorithms for next-generation sequencing data. *Genomics* **95(6)**, 315-327.
- **Schatz MC, Delcher AL, Salzberg SL (2010)**. Assembly of large genomes using second-generation sequencing. *Genome Research* **20**, 1165-1173.

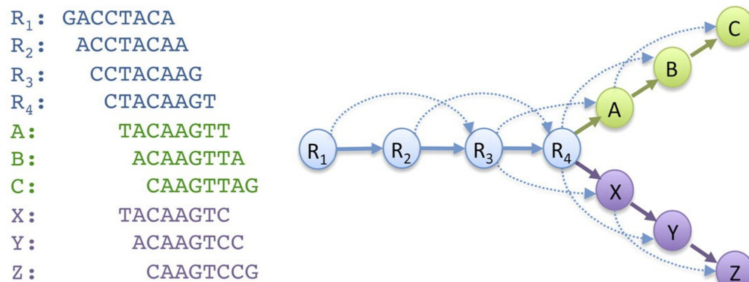
Genoomassemblage is het computationele proces waarbij de genoomsequentie van een organisme bepaald wordt. Onderzoekers zijn momenteel nog niet

in staat om de nucleotiden van een volledig genoom in één keer te bepalen. In plaats daarvan doen ze een beroep op hoogtechnologische chemische methoden om de basenvolgorde te bepalen van korte stukjes DNA (doorgaans minder dan 1000 bp lang). Deze korte stukjes worden de **reads** genoemd.

Om een volledig genoom te sequeneren, gaan onderzoekers verschillende kopieën van het genoom als het ware chemisch opblazen, waardoor elk van de bekomen reads door een sequenator kan uitgelezen worden. Om het genoom terug te reconstrueren, wordt gebruik gemaakt van de overlap tussen de reads om te bepalen welke reads naast elkaar op het genoom liggen. Als bijvoorbeeld de korte reads G**ACCTACA** en **ACCTACA**A uitgelezen worden, dan kunnen we uit het feit dat ze overlappen, afleiden dat beide afkomstig zijn van het DNA fragment G**ACCTACA**A.

Opgave

Als eerste stap bij genoomassemblage brengen de meeste algoritmen de overlappende reads in kaart door een **overlapgraaf** op te bouwen. Dit is een datastructuur waarbij overlappende reads (in onderstaande afbeelding voorgesteld door gelabelde cirkels) met elkaar worden verbonden door een pijl. Merk op dat het soms kan voorvallen dat een read overlapt met meerdere andere reads, vaak als gevolg van herhalingen in het genoom of fouten bij het uitlezen van de reads. Hierdoor ontstaan zogenaamde "vorken" in de overlapgraaf.



Voorbeeld van een overlapgraaf (rechts) die is opgebouwd op basis van 10 gegeven reads van 8 baseparen (links). Reads met een overlap van minstens 6 baseparen worden aangegeven door een pijl te trekken tussen beide reads. Transitieve overlaps — die kunnen afgeleid worden uit andere langere overlaps — zijn aangegeven als gestippelde pijlen. De vork in de graaf is het gevolg van herhalingen in het genoom of fouten in de reads en maakt het extreem moeilijk voor assemblage-algoritmen om te achterhalen welke route op welk ogenblik moet gekozen worden.

Het doel van deze opgave is dat je voor een gegeven verzameling reads zelf een overlapgraaf opbouwt. Hiervoor ga je als volgt te werk:

- Schrijf een functie `overlap` waaraan drie argumenten moeten doorgegeven worden. De eerste twee argumenten zijn reads die voorgesteld worden als strings die enkel de letters `A`, `C`, `G` en `T` bevatten (deze stellen de vier mogelijke basen voor). Het derde argument is een getal $k \in \mathbb{N}_0$. De functie moet een Booleaanse waarde teruggeven, die aangeeft of de eerste read een overlap van lengte k heeft met de tweede read. Dat is het geval als de laatste k basen van de eerste read gelijk zijn aan de eerste k basen van de tweede read.
- Gebruik de functie `overlap` om een functie `maximaleOverlap` te schrijven. Aan deze functie moeten twee reads doorgegeven worden. De functie moet de lengte van de maximale overlap tussen de eerste en de tweede read teruggeven. Indien beide reads totaal niet overlappen, dan moet de waarde nul teruggegeven worden.
- Gebruik de functie `maximaleOverlap` om een functie `overlapgraaf` te schrijven. Aan deze functie moeten twee argumenten doorgegeven worden: een collectie (een lijst, tuple, verzameling, …) van reads en een getal $k \in \mathbb{N}_0$. De functie moet een dictionary teruggeven, waarin elke read uit de gegeven collectie wordt afgebeeld op de verzameling van alle **andere** reads uit de collectie die ermee overlappen met lengte minstens k (indien deze verzameling niet leeg is). Een dergelijke afbeelding stelt dus de pijlen uit de overlapgraaf voor die vertrekken vanuit een read (die als sleutel gebruikt in de dictionary) naar alle **andere** reads die ermee overlappen. Merk dus op dat er per definitie nooit een pijl getrokken wordt tussen een read en zichzelf, ook al zou die read met zichzelf overlappen.

Voorbeeld

```
>>> overlap('AAATTTT', 'TTTTCCC', 3)
True
>>> overlap('AAATTTT', 'TTTTCCC', 5)
False
>>> overlap('ATATATATAT', 'TATATATATA', 4)
False
>>> overlap('ATATATATAT', 'TATATATATA', 5)
True

>>> maximaleOverlap('AAATTTT', 'TTTTCCC')
4
>>> maximaleOverlap('ATATATATAT', 'TATATATATA')
9

>>> reads = ['AAATAAA', 'AAATTTT', 'TTTTCCC', 'AAATCCC', 'GGGTGGG']
>>> overlapgraaf(reads, 3)
{'AAATTTT': {'TTTTCCC'}, 'AAATAAA': {'AAATTTT', 'AAATCCC'}}
>>> overlapgraaf(reads, 4)
{'AAATTTT': {'TTTTCCC'}}

>>> reads = ['GACCTACA', 'ACCTACAA', 'CCTACAAG', 'CTACAAGT', 'TACAAGTT', 'ACAAGTTA', 'CAAGTTAG', 'TACAAGTC', 'ACAAGTCC', 'CAAGTCCG']
>>> overlapgraaf(reads, 6)
{'CTACAAGT': {'ACAAGTCC', 'ACAAGTTA', 'TACAAGTC', 'TACAAGTT'}, 'TACAAGTT': {'CAAGTTAG', 'ACAAGTTA'}, 'ACCTACAA': {'CTACAAGT', 'CCTACAAG'}, 'ACAAGTCC': {'CAAGTCCG'}}
```

Opmerking: het laatste voorbeeld bouwt de overlapgraaf op die hierboven grafisch wordt weergegeven.

Bronnen

- **Miller JR, Koren S, Sutton G (2010)**. Assembly algorithms for next-generation sequencing data. *Genomics* **95(6)**, 315-327.
- **Schatz MC, Delcher AL, Salzberg SL (2010)**. Assembly of large genomes using second-generation sequencing. *Genome Research* **20**, 1165-1173.