Spring 2019: CSE/BMMB 566

Problem 1

Counting DNA Nucleotides

A Rapid Introduction to Molecular Biology

Making up all living material, the **cell** is considered to be the building block of life. The **nucleus**, a component of most **eukaryotic** cells, was identified as the hub of cellular activity 150 years ago. Viewed under a light microscope, the nucleus appears only as a darker region of the cell, but as we increase magnification, we find that the nucleus is densely filled with a stew of macromolecules called **chromatin**. During **mitosis** (eukaryotic cell division), most of the chromatin condenses into long, thin strings called **chromosomes**. See **Figure 1** for a figure of cells in different stages of mitosis.

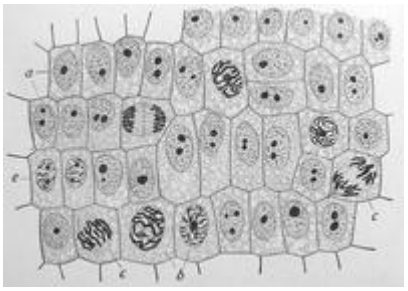

Figure 1. A 1900 drawing by Edmund Wilson of onion cells at different stages of mitosis. The sample has been dyed, causing chromatin in the cells (which soaks up the dye) to appear in greater contrast to the rest of the cell.

One class of the macromolecules contained in chromatin are called **nucleic acids**. Early 20th century research into the chemical identity of nucleic acids culminated with the conclusion that nucleic acids are **polymers**, or repeating chains of smaller, similarly structured molecules known as **monomers**. Because of their tendency to be long and thin, nucleic acid polymers are commonly called **strands**.

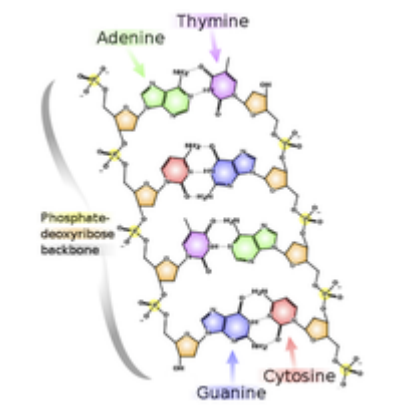

Figure 2. A sketch of DNA's primary structure.

The nucleic acid monomer is called a **nucleotide** and is used as a unit of strand length (abbreviated to nt). Each nucleotide is formed of three parts: a **sugar** molecule, a negatively charged **ion** called a **phosphate**, and a compound called a **nucleobase** ("base" for short). Polymerization is achieved as the sugar of one nucleotide bonds to the phosphate of the next nucleotide in the chain, which forms a **sugar-phosphate backbone** for the nucleic acid strand. A key point is that the nucleotides of a specific type of nucleic acid always contain the same sugar and phosphate molecules, and they differ only in their choice of base. Thus, one strand of a nucleic acid can be differentiated from another based solely on the *order* of its bases; this ordering of bases defines a nucleic acid's **primary structure**.

For example, **Figure 2** shows a strand of **deoxyribose nucleic acid** (DNA), in which the sugar is called **deoxyribose**, and the only four choices for nucleobases are molecules called **adenine** (A), **cytosine** (C), **guanine** (G), and **thymine** (T).

For reasons we will soon see, DNA is found in all living organisms on Earth, including bacteria; it is even found in many viruses (which are often considered to be nonliving). Because of its importance, we reserve the term **genome** to refer to the sum total of the DNA contained in an organism's chromosomes.

Problem

A **string** is simply an ordered collection of symbols selected from some **alphabet** and formed into a word; the **length** of a string is the number of symbols that it contains.

An example of a length 21 **DNA string** (whose alphabet contains the symbols 'A', 'C', 'G', and 'T') is "ATGCTTCAGAAAGGTCTTACG."

Given: A DNA string s of length at most 1000 nt.

Return: Four integers (separated by spaces) counting the respective number of times that the symbols 'A', 'C', 'G', and 'T' occur in s.

Sample Dataset

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
```

Sample Output

```
20 12 17 21
```

Problem 2

Transcribing DNA into RNA

The Second Nucleic Acid

In "Counting DNA Nucleotides", we described the primary structure of a nucleic acid as a polymer of nucleotide units, and we mentioned that the omnipresent nucleic acid DNA is composed of a varied sequence of four bases.

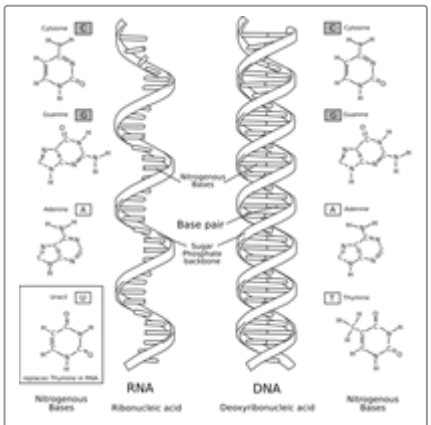

Figure 1. Structural differences between RNA and DNA

Yet a second nucleic acid exists alongside DNA in the chromatin; this molecule, which possesses a different sugar called **ribose**, came to be known as **ribose nucleic acid**, or RNA. RNA differs further from DNA in that it contains a base called **uracil** in place of thymine; structural differences between DNA and RNA are shown in **Figure 1**. Biologists initially believed that RNA was only contained in plant cells, whereas DNA was restricted to animal cells. However, this hypothesis dissipated as improved chemical methods discovered both nucleic acids in the cells of all life forms on Earth.

The primary structure of DNA and RNA is so similar because the former serves as a blueprint for the creation of a special kind of RNA molecule called **messenger RNA**, or mRNA. mRNA is created during **RNA transcription**, during which a strand of DNA is used as a template for constructing a strand of RNA by copying nucleotides one at a time, where uracil is used in place of thymine.

In eukaryotes, DNA remains in the nucleus, while RNA can enter the far reaches of the cell to carry out DNA's instructions. In future problems, we will examine the process and ramifications of RNA transcription in more

detail.

Problem

An **RNA string** is a string formed from the alphabet containing 'A', 'C', 'G', and 'U'.

Given a DNA string t corresponding to a coding strand, its transcribed RNA string u is formed by replacing all occurrences of 'T' in t with 'U' in u.

Given: A DNA string t having length at most 1000 nt.

Return: The transcribed RNA string of t.

Sample Dataset

```
GATGGAACTTGACTACGTAAATT
```

Sample Output

```
GAUGGAACUUGACUACGUAAAUU
```

Problem 3

Complementing a Strand of DNA

The Secondary and Tertiary Structures of DNA

In "Counting DNA Nucleotides", we introduced nucleic acids, and we saw that the primary structure of a nucleic acid is determined by the ordering of its nucleobases along the sugar-phosphate backbone that constitutes the bonds of the nucleic acid polymer. Yet primary structure tells us nothing about the larger, 3-dimensional shape of the molecule, which is vital for a complete understanding of nucleic acids.

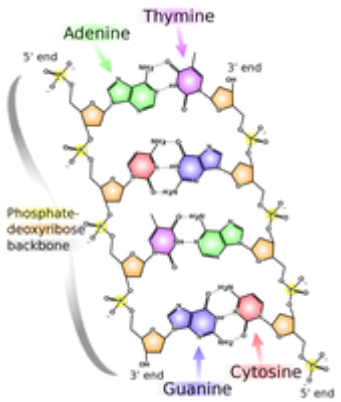

Figure 1. Base pairing across the two strands of DNA.

The search for a complete chemical structure of nucleic acids was central to molecular biology research in the mid-20th Century, culminating in 1953 with a publication in *Nature* of fewer than 800 words by James Watson and Francis Crick. Consolidating a high resolution X-ray image created by Rosalind Franklin and Raymond Gosling with a number of established chemical results, Watson and Crick proposed the following structure for DNA:

1. The DNA molecule is made up of two strands, running in opposite directions.

2. Each base bonds to a base in the opposite strand. Adenine always bonds with thymine, and cytosine always bonds with guanine; the **complement** of a base is the base to which it always bonds; see **Figure 1**.
3. The two strands are twisted together into a long spiral staircase structure called a **double helix**; see **Figure 2**.

Because they dictate how bases from different strands interact with each other, (1) and (2) above compose the **secondary structure** of DNA. (3) describes the 3-dimensional shape of the DNA molecule, or its **tertiary structure**.

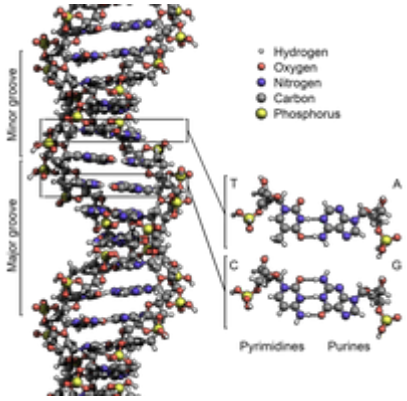

Figure 2. The double helix of DNA on the molecular scale.

In light of Watson and Crick's model, the bonding of two complementary bases is called a **base pair** (bp). Therefore, the length of a DNA molecule will commonly be given in bp instead of nt. By complementarity, once we know the order of bases on one strand, we can immediately deduce the sequence of bases in the complementary strand. These bases will run in the opposite order to match the fact that the two strands of DNA run in opposite directions.

Problem

In DNA strings, symbols 'A' and 'T' are complements of each other, as are 'C' and 'G'.

The **reverse complement** of a DNA string s is the string s^c formed by reversing the symbols of s, then taking the complement of each symbol (e.g., the reverse complement of "GTCA" is "TGAC").

Given: A DNA string s of length at most 1000 bp.

Return: The reverse complement s^c of s.

Sample Dataset

```
AAAACCCGGT
```

Sample Output

```
ACCGGGTTTT
```

Problem 4

Computing GC Content

Identifying Unknown DNA Quickly

A quick method used by early computer software to determine the language of a given piece of text was to analyze the frequency with which each letter appeared in the text. This strategy was used because each language tends to exhibit its own letter frequencies, and as long as the text under consideration is long

enough, software will correctly recognize the language quickly and with a very low error rate. See **Figure 1** for a table compiling English letter frequencies.

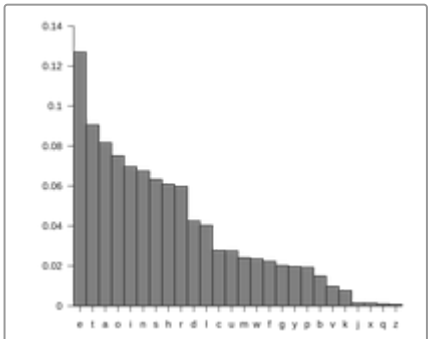

Figure 1. The table above was computed from a large number of English words and shows for any letter the frequency with which it appears in those words. These frequencies can be used to reliably identify a piece of English text and differentiate it from that of another language. Taken from http://en.wikipedia.org/wiki/File:English_lette

You may ask: what in the world does this linguistic problem have to do with biology? Although two members of the same species will have different genomes, they still share the vast percentage of their DNA; notably, 99.9% of the 3.2 billion base pairs in a human genome are common to almost all humans (i.e., excluding people having major genetic defects). For this reason, biologists will speak of *the* human genome, meaning an average-case genome derived from a collection of individuals. Such an average case genome can be assembled for any species, a challenge that we will soon discuss.

The biological analog of identifying unknown text arises when researchers encounter a molecule of DNA from an unknown species. Because of the base pairing relations of the two DNA strands, cytosine and guanine will always appear in equal amounts in a double-stranded DNA molecule. Thus, to analyze the symbol frequencies of DNA for comparison against a database, we compute the molecule's **GC-content**, or the percentage of its bases that are *either* cytosine or guanine.

In practice, the GC-content of most eukaryotic genomes hovers around 50%. However, because genomes are so long, we may be able to distinguish species based on very small discrepancies in GC-content; furthermore, most prokaryotes have a GC-content significantly higher than 50%, so that GC-content can be used to quickly differentiate many prokaryotes and eukaryotes by using relatively small DNA samples.

Problem

The GC-content of a DNA string is given by the percentage of symbols in the string that are 'C' or 'G'. For example, the GC-content of "AGCTATAG" is 37.5%. Note that the reverse complement of any DNA string has the same GC-content.

DNA strings must be labeled when they are consolidated into a database. A commonly used method of string labeling is called **FASTA format**. In this format, the string is introduced by a line that begins with '>', followed by some labeling information. Subsequent lines contain the string itself; the first line to begin with '>' indicates the label of the next string.

In Rosalind's implementation, a string in FASTA format will be labeled by the ID "Rosalind_xxxx", where "xxxx" denotes a four-digit code between 0000 and 9999.

Given: At most 10 DNA strings in FASTA format (of length at most 1 kbp each).

Return: The ID of the string having the highest GC-content, followed by the GC-content of that string. Rosalind allows for a default error of 0.001 in all decimal answers unless otherwise stated; please see the note on **absolute error** below.

Sample Dataset

```
>Rosalind_6404
CCTGCGGAAGATCGGCACTAGAATAGCCAGAACCGTTTCTCTGAGGCTTCCGGCCTTCCC
TCCCACTAATAATTCTGAGG
>Rosalind_5959
CCATCGGTAGCGCATCCTTAGTCCAATTAAGTCCCTATCCAGGCGCTCCGCCGAAGGTCT
ATATCCATTTGTCAGCAGACACGC
```

```
>Rosalind_0808
CCACCCTCGTGGTATGGCTAGGCATTCAGGAACCGGAGAACGCTTCAGACCAGCCCGGAC
TGGGAACCTGCGGGCAGTAGGTGGAAT
```

Sample Output

```
Rosalind_0808
60.919540
```

Note on Absolute Error

We say that a number x is within an absolute error of y to a correct solution if x is within y of the correct solution. For example, if an exact solution is 6.157892, then for x to be within an absolute error of 0.001, we must have that $|x - 6.157892| < 0.001$, or $6.156892 < x < 6.158892$.

Error bounding is a vital practical tool because of the inherent round-off error in representing decimals in a computer, where only a finite number of decimal places are allotted to any number. After being compounded over a number of operations, this round-off error can become evident. As a result, rather than testing whether two numbers are equal with $x = z$, you may wish to simply verify that $|x - z|$ is very small.

The mathematical field of **numerical analysis** is devoted to rigorously studying the nature of computational approximation.

Problem 5

Counting Point Mutations

Evolution as a Sequence of Mistakes

A **mutation** is simply a mistake that occurs during the creation or copying of a nucleic acid, in particular DNA. Because nucleic acids are vital to cellular functions, mutations tend to cause a ripple effect throughout the cell. Although mutations are technically mistakes, a very rare mutation may equip the cell with a beneficial attribute. In fact, the macro effects of evolution are attributable by the accumulated result of beneficial microscopic mutations over many generations.

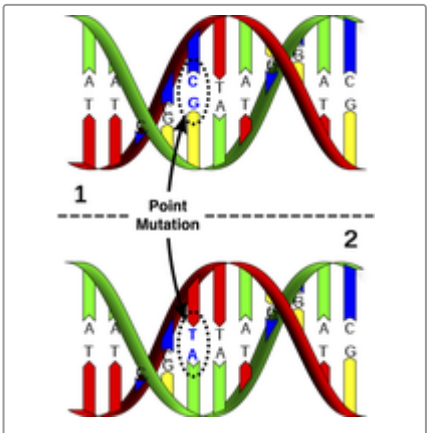

Figure 1. A point mutation in DNA changing a C-G pair to an A-T pair.

The simplest and most common type of nucleic acid mutation is a **point mutation**, which replaces one base with another at a single nucleotide. In the case of DNA, a point mutation must change the complementary base accordingly; see **Figure 1**.

Two DNA strands taken from different organism or species genomes are **homologous** if they share a recent ancestor; thus, counting the number of bases at which homologous

strands differ provides us with the minimum number of point mutations that could have occurred on the evolutionary path between the two strands.

We are interested in minimizing the number of (point) mutations separating two species because of the biological principle of **parsimony**, which demands that evolutionary histories should be as simply explained as possible.

Problem

Given two strings s and t of equal length, the **Hamming distance** between s and t, denoted $d_{\mathrm{H}}(s, t)$, is the number of corresponding symbols that differ in s and t. See **Figure 2**.

Given: Two DNA strings s and t of equal length (not exceeding 1 kbp).

Return: The Hamming distance $d_{\mathrm{H}}(s, t)$.

GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT

Figure 2. The Hamming distance between these two strings is 7. Mismatched symbols are colored red.

Sample Dataset

```
GAGCCTACTAACGGGAT
CATCGTAATGACGGCCT
```

Sample Output

```
7
```

Problem 6

Translating RNA into Protein

The Genetic Code

Just as nucleic acids are polymers of nucleotides, **proteins** are chains of smaller molecules called **amino acids**; 20 amino acids commonly appear in every species. Just as the primary structure of a nucleic acid is given by the order of its nucleotides, the **primary structure** of a protein is the order of its amino acids. Some proteins are composed of several subchains called **polypeptides**, while others are formed of a single polypeptide; see **Figure 1**.

Proteins power every practical function carried out by the cell, and so presumably, the key to understanding life lies in interpreting the relationship between a chain of amino acids and the function of the protein that this chain of amino acids eventually constructs. **Proteomics** is the field devoted to the study of proteins.

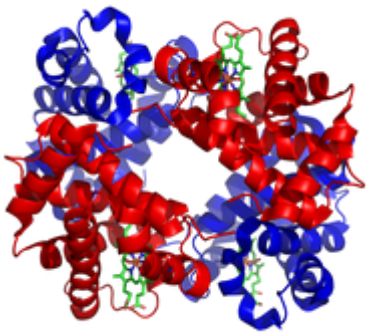

Figure 1. The human hemoglobin molecule consists of 4 polypeptide chains; α subunits

are shown in red and β subunits are shown in blue

How are proteins created? The **genetic code**, discovered throughout the course of a number of ingenious experiments in the late 1950s, details the **translation** of an RNA molecule called **messenger RNA** (mRNA) into amino acids for protein creation. The apparent difficulty in translation is that somehow 4 RNA bases must be translated into a language of 20 amino acids; in order for every possible amino acid to be created, we must translate 3-nucleobase strings (called **codons**) into amino acids. Note that there are $4^3 = 64$ possible codons, so that multiple codons may encode the same amino acid. Two special types of codons are the **start codon** (AUG), which codes for the amino acid methionine always indicates the start of translation, and the three **stop codons** (UAA, UAG, UGA), which do not code for an amino acid and cause translation to end.

The notion that protein is always created from RNA, which in turn is always created from DNA, forms the **central dogma of molecular biology**. Like all dogmas, it does not always hold; however, it offers an excellent approximation of the truth.

An organelle called a **ribosome** creates peptides by using a helper molecule called **transfer RNA** (tRNA). A single tRNA molecule possesses a string of three RNA nucleotides on one end (called an **anticodon**) and an amino acid at the other end. The ribosome takes an RNA molecule transcribed from DNA (see "Transcribing DNA into RNA"), called **messenger RNA** (mRNA), and examines it one codon at a time. At each step, the tRNA possessing the complementary anticodon bonds to the mRNA at this location, and the amino acid found on the opposite end of the tRNA is added to the growing peptide chain before the remaining part of the tRNA is ejected into the cell, and the ribosome looks for the next tRNA molecule.

Not every RNA base eventually becomes translated into a protein, and so an interval of RNA (or an interval of DNA translated into RNA) that does code for a protein is of great biological interest; such an interval of DNA or RNA is called a **gene**. Because protein creation drives cellular processes, genes differentiate organisms and serve as a basis for **heredity**, or the process by which traits are inherited.

Problem

The 20 commonly occurring amino acids are abbreviated by using 20 letters from the English alphabet (all letters except for B, J, O, U, X, and Z). **Protein strings** are constructed from these 20 symbols. Henceforth, the term **genetic string** will incorporate protein strings along with DNA strings and RNA strings.

The **RNA codon table** dictates the details regarding the encoding of specific codons into the amino acid alphabet.

Given: An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp).

Return: The protein string encoded by s.

Sample Dataset

```
AUGGCCAUGGCGCCCAGAACUGAGAUCAAUAGUACCCGUAUUAACGGGUGA
```

Sample Output

```
MAMAPRTEINSTRING
```

Problem 7

Locating Restriction Sites

The Billion-Year War

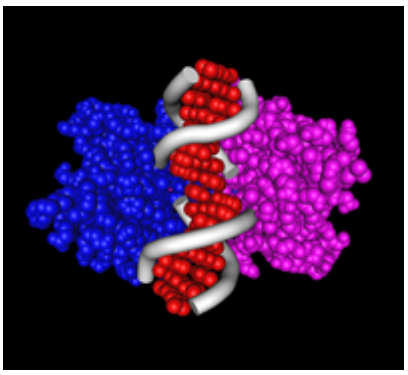

Figure 1. DNA cleaved by EcoRV restriction enzyme

The war between viruses and bacteria has been waged for over a billion years. Viruses called **bacteriophages** (or simply phages) require a bacterial host to propagate, and so they must somehow infiltrate the bacterium; such deception can only be achieved if the phage understands the genetic framework underlying the bacterium's cellular functions. The phage's goal is to insert DNA that will be replicated within the bacterium and lead to the reproduction of as many copies of the phage as possible, which sometimes also involves the bacterium's demise.

To defend itself, the bacterium must either obfuscate its cellular functions so that the phage cannot infiltrate it, or better yet, go on the counterattack by calling in the air force. Specifically, the bacterium employs aerial scouts called **restriction enzymes**, which operate by cutting through viral DNA to cripple the phage. But what kind of DNA are restriction enzymes looking for?

The restriction enzyme is a **homodimer**, which means that it is composed of two identical substructures. Each of these structures separates from the restriction enzyme in order to bind to and cut one strand of the phage DNA molecule; both substructures are pre-programmed with the same target string containing 4 to 12 nucleotides to search for within the phage DNA (see **Figure 1**.). The chance that both strands of phage DNA will be cut (thus crippling the phage) is greater if the target is located on both strands of phage DNA, as close to each other as possible. By extension, the best chance of disarming the phage occurs when the two target copies appear directly across from each other along the phage DNA, a phenomenon that occurs precisely when the target is equal to its own reverse complement. Eons of evolution have made sure that most restriction enzyme targets now have this form.

Problem

A DNA string is a **reverse palindrome** if it is equal to its reverse complement. For instance, GCATGC is a reverse palindrome because its reverse complement is GCATGC. See **Figure 2**.

```
5'...GAT|ATC...3'
3'...CTA|TAG...5'
```

Figure 2. Palindromic recognition site

Given: A DNA string of length at most 1 kbp in FASTA format.

Return: The position and length of every reverse palindrome in the string having length between 4 and 12. You may return these pairs in any order.

Sample Dataset

```
>Rosalind_24
TCAATGCATGCGGGTCTATATGCAT
```

Sample Output

```
4 6
5 4
6 6
7 4
17 4
18 4
20 6
21 4
```

Extra Information

You may be curious how the bacterium prevents its own DNA from being cut by restriction enzymes. The short answer is that it locks itself from being cut through a chemical process called **DNA methylation**.

Problem 8

Identifying Maximal Repeats

Spies in the War Against Phages

In "Locating Restriction Sites", we saw how one weapon used by bacteria in their age-old fight with phages is the use of restriction enzymes. Another defense mechanism found in the genomes of most bacteria and archaea centers on intervals of DNA called **CRISPRs** (Clustered Regularly Interspaced Short Palindromic Repeats), which allow the cell to distinguish its own DNA from that of phages or plasmids.

Figure 1. A genomic region containing a CRISPR. Red substrings correspond to CRISPR repeats, and blue substrings correspond to unique spacers. Repeats are highly palindromic and fold into a hairpin loop when transcribed.

Specifically, a CRISPR is an interval of DNA consisting of identical repeats (approximately 23 to 47 bp long), alternating with unique intervals (approximately 21 to 72 bp long) called **spacers**; see **Figure 1**. Spacers correspond to fragments of foreign DNA that were integrated into the genome between repeats and serve as a memory bank for genetic material captured from invading phages. As a result, spacers can be used to recognize and silence invasive elements.

Specifically, CRISPRs are transcribed into RNA molecules, each consisting of a spacer flanked by partial repeats. The small CRISPR RNAs, together with associated proteins translated from this RNA, target foreign DNA that matches the CRISPR spacer. In eukaryotes, a similar process is achieved by a process called **RNA interference** (RNAi).

To locate a CRISPR in a genome, we need to search for its repeats. We have already located long repeats in "Finding the Longest Multiple Repeat", but the case here is different because of the repeats appearing in CRISPRS are relatively short. Instead, we are looking for repeated intervals that cannot be lengthened in either direction (otherwise, we would intersect with a spacer).

Problem

A **maximal repeat** of a string s is a repeated substring t of s having two occurrences t_1 and t_2 such that t_1 and t_2 cannot be extended by one symbol in either direction in s and still agree.

For example, "**AG**" is a maximal repeat in "T**AG**TT**AG**CG**AG**A" because even though the first two occurrences of "AG" can be extended left into "TAG", the first and third occurrences differ on both sides of the repeat; thus, we conclude that "AG" is a maximal repeat. Note that "**TAG**" is also a maximal repeat of "**TAG**T**TAG**CGAGA", since its only two occurrences do not still match if we extend them in either direction.

Given: A DNA string s of length at most 1 kbp.

Return: A list containing all maximal repeats of s having length at least 20.

Sample Dataset

```
TAGAGATAGAATGGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTATTATATAGAGATAGAAT
GGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTAT
```

Sample Output

```
TAGAGATAGAATGGGTCCAGAGTTTTGTAATTTCCATGGGTCCAGAGTTTTGTAATTTAT
ATGGGTCCAGAGTTTTGTAATTT
```

> ### Hint
>
> How can we use the suffix tree of s to find maximal repeats?

Problem 9

Encoding Suffix Trees

Creating a Suffix Tree

In "Finding the Longest Multiple Repeat", we introduced the suffix tree. This data structure has a wide array of applications, one of which was to help us identify long repeats in a genome. In that problem, we provided the tree as part of the dataset, but a vital computational exercise is to create the suffix tree solely from a string.

Problem

Given a string s having length n, recall that its suffix tree $T(s)$ is defined by the following properties:

- $T(s)$ is a rooted tree having exactly n leaves.

- Every edge of $T(s)$ is labeled with a substring of s^*, where s^* is the string formed by adding a placeholder symbol `$` to the end of s.
- Every internal node of $T(s)$ other than the root has at least two children; i.e., it has degree at least 3.
- The substring labels for the edges leading down from a node to its children must begin with different symbols.
- By concatenating the substrings along edges, each path from the root to a leaf corresponds to a unique suffix of s^*.

Figure 1 contains an example of a suffix tree.

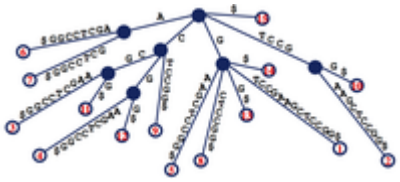

Figure 1. The suffix tree for s = GTCCGAAGCTCCGG. Note that the dollar sign has been appended to a substring of the tree to mark the end of s. Every path from the root to a leaf corresponds to a unique suffix of GTCCGAAGCTCCGG, and each leaf is labeled with the location in s of the suffix ending at that leaf.

Given: A DNA string s of length at most 1kbp.

Return: The substrings of s^* encoding the edges of the suffix tree for s. You may list these substrings in any order.

Sample Dataset

```
ATAAATG$
```

Sample Output

```
AAATG$
G$
T
ATG$
TG$
A
A
AAATG$
G$
T
G$
$
```

Problem 10

Global Alignment with Scoring Matrix

Generalizing the Alignment Score

The edit alignment score in "Edit Distance Alignment" counted the total number of edit operations implied by an alignment; we could equivalently think of this scoring function as assigning a cost of 1 to each such operation. Another common scoring function awards matched symbols with 1 and penalizes substituted/inserted/deleted symbols equally by assigning each one a score of 0, so that the maximum score of an alignment becomes the length of a longest

common subsequence of s and t (see "Finding a Shared Spliced Motif"). In general, the **alignment score** is simply a scoring function that assigns costs to edit operations encoded by the alignment.

One natural way of adding complexity to alignment scoring functions is by changing the alignment score based on which symbols are substituted; many methods have been proposed for doing this. Another way to do so is to vary the penalty assigned to the insertion or deletion of symbols.

In general, alignment scores can be either maximized or minimized depending on how scores are established. The general problem of optimizing a particular alignment score is called **global alignment**.

Problem

To penalize symbol substitutions differently depending on which two symbols are involved in the substitution, we obtain a **scoring matrix** S in which $S_{i,j}$ represents the (negative) score assigned to a substitution of the ith symbol of our alphabet $\mathcal{A}$ with the jth symbol of $\mathcal{A}$.

A **gap penalty** is the component deducted from alignment score due to the presence of a gap. A gap penalty may be a function of the length of the gap; for example, a **linear gap penalty** is a constant g such that each inserted or deleted symbol is charged g; as a result, the cost of a gap of length L is equal to gL.

Given: Two protein strings s and t in FASTA format (each of length at most 1000 aa).

Return: The maximum alignment score between s and t. Use:

- The BLOSUM62 scoring matrix.
- Linear gap penalty equal to 5 (i.e., a cost of -5 is assessed for each gap symbol).

Sample Dataset

```
>Rosalind_67
PLEASANTLY
>Rosalind_17
MEANLY
```

Sample Output

```
8
```

Problem 11

Local Alignment with Affine Gap Penalty

Building Upon Local Alignments

We have thus far worked with local alignments with a linear gap penalty and global alignments with affine gap penalties (see "Local Alignment with Scoring Matrix" and "Global Alignment with Scoring Matrix and Affine Gap Penalty").

> It is only natural to take the intersection of these two problems and find an optimal local alignment given an affine gap penalty.

Problem

Given: Two protein strings s and t in FASTA format (each having length at most 10,000 aa).

Return: The maximum local alignment score of s and t, followed by substrings r and u of s and t, respectively, that correspond to the optimal local alignment of s and t. Use:

- The BLOSUM62 scoring matrix.
- Gap opening penalty equal to 11.
- Gap extension penalty equal to 1.

If multiple solutions exist, then you may output any one.

Sample Dataset

```
>Rosalind_8
PLEASANTLY
>Rosalind_18
MEANLY
```

Sample Output

```
12
LEAS
MEAN
```

Problem 12

Align Two Strings Using Linear Space

The pseudocode below for **LinearSpaceAlignment** describes how to recursively find a longest path in the alignment graph constructed for a substring v_{top+1} ... v_{bottom} of v and a substring w_{left+1} ... w_{right} of w. **LinearSpaceAlignment** calls the function *MiddleNode*(*top*, *bottom*, *left*, *right*), which returns the coordinate *i* of the middle node (*i*, *j*) defined by the sequences v_{top+1} ... v_{bottom} and w_{left+1} ... w_{right}. **LinearSpaceAlignment** also calls *MiddleEdge*(*top*, *bottom*, *left*, *right*), which returns → , ↓, or ↘ depending on whether the middle edge is horizontal, vertical, or diagonal. The linear-space alignment of strings *v* and *w* is constructed by calling **LinearSpaceAlignment**(0, *n*, 0, *m*). The case *left* = *right* describes the alignment of an empty string against the string v_{top+1} ... v_{bottom}, which is trivially computed as the score of a gap formed by *bottom* − *top* vertical edges.

```
LinearSpaceAlignment(top, bottom, left, right)
  if left = right
    return alignment formed by bottom − top vertical edges
```

```
if top = bottom
    return alignment formed by right – left horizontal edges
middle ← ⌊ (left + right)/2 ⌋
midNode ← MiddleNode(top, bottom, left, right)
midEdge ← MiddleEdge(top, bottom, left, right)
LinearSpaceAlignment(top, midNode, left, middle)
output midEdge
if midEdge = "→" or midEdge = "↘"
    middle ← middle + 1
if midEdge = "↓" or midEdge ="↘"
    midNode ← midNode + 1
LinearSpaceAlignment(midNode, bottom, middle, right)
```

Global Alignment in Linear Space Problem

Find the highest-scoring alignment between two strings using a scoring matrix in linear space.

Given: Two **long** amino acid strings (of length approximately 10,000).

Return: The maximum alignment score of these strings, followed by an alignment achieving this maximum score. Use the BLOSUM62 scoring matrix and indel penalty σ = 5.

Sample Dataset

```
PLEASANTLY
MEANLY
```

Sample Output

```
8
PLEASANTLY
-MEA--N-LY
```

Extra Dataset

Click for an extra dataset

Problem 13

Constructing a De Bruijn Graph

> **Wading Through the Reads**

Because we use multiple copies of the genome to generate and identify reads for the purposes of fragment assembly, the total length of all reads will be much longer than the genome itself. This begs the definition of **read coverage** as the average number of times that each nucleotide from the genome appears in the reads. In other words, if the total length of our reads is 30 billion bp for a 3 billion bp genome, then we have 10x read coverage.

To handle such a large number of k-mers for the purposes of sequencing the genome, we need an efficient and simple structure.

Problem

Consider a set S of $(k+1)$-mers of some unknown DNA string. Let S^{rc} denote the set containing all reverse complements of the elements of S. (recall from "Counting Subsets" that sets are not allowed to contain duplicate elements).

The **de Bruijn graph** B_k of order k corresponding to $S \cup S^{\text{rc}}$ is a digraph defined in the following way:

- Nodes of B_k correspond to all k-mers that are present as a substring of a $(k+1)$-mer from $S \cup S^{\text{rc}}$.
- Edges of B_k are encoded by the $(k+1)$-mers of $S \cup S^{\text{rc}}$ in the following way: for each $(k+1)$-mer r in $S \cup S^{\text{rc}}$, form a directed edge $(r[1:k], r[2:k+1])$.

Given: A collection of up to 1000 (possibly repeating) DNA strings of equal length (not exceeding 50 bp) corresponding to a set S of $(k+1)$-mers.

Return: The adjacency list corresponding to the de Bruijn graph corresponding to $S \cup S^{\text{rc}}$.

Sample Dataset

```
TGAT
CATG
TCAT
ATGC
CATC
CATC
```

Sample Output

```
(ATC, TCA)
(ATG, TGA)
(ATG, TGC)
(CAT, ATC)
(CAT, ATG)
(GAT, ATG)
(GCA, CAT)
(TCA, CAT)
(TGA, GAT)
```

Problem 14

Error Correction in Reads

Genome Sequencing Isn't Perfect

In "Genome Assembly as Shortest Superstring", we introduce the problem of assembling a genome from a collection of reads. Even though genome sequencing is a multi-billion dollar enterprise, sequencing machines that identify reads still produce errors a substantial percentage of the time. To make matters worse, these errors are unpredictable; it is difficult to determine if the machine has made an error, let alone where in the read the error has occurred. For this reason, error correction in reads is typically a vital first step in genome assembly.

Problem

As is the case with point mutations, the most common type of sequencing error occurs when a single nucleotide from a read is interpreted incorrectly.

Given: A collection of up to 1000 reads of equal length (at most 50 bp) in FASTA format. Some of these reads were generated with a single-nucleotide error. For each read s in the dataset, one of the following applies:

- s was correctly sequenced and appears in the dataset at least twice (possibly as a reverse complement);
- s is incorrect, it appears in the dataset exactly once, and its Hamming distance is 1 with respect to exactly one correct read in the dataset (or its reverse complement).

Return: A list of all corrections in the form "[old read]->[new read]". (Each correction must be a single symbol substitution, and you may return the corrections in any order.)

Sample Dataset

```
>Rosalind_52
TCATC
>Rosalind_44
TTCAT
>Rosalind_68
TCATC
>Rosalind_28
TGAAA
>Rosalind_95
GAGGA
>Rosalind_66
TTTCA
>Rosalind_33
ATCAA
>Rosalind_21
TTGAT
>Rosalind_18
TTTCC
```

Sample Output

```
TTCAT->TTGAT
GAGGA->GATGA
TTTCC->TTTCA
```

Problem 15

Genome Assembly with Perfect Coverage

Cyclic Chromosomes

Recall that although chromosomes taken from eukaryotes have a linear structure, many bacterial chromosomes are actually circular. We represented a linear chromosome with a DNA string, so we only need to modify the definition of string to model circular chromosomes.

> **Perfect coverage** is the phenomenon in fragment assembly of having a read (or k-mer) begin at every possible location in the genome. Unfortunately, perfect coverage is still difficult to achieve, but fragment assembly technology continues to improve by leaps and bounds, and perfect coverage is perhaps not the fantasy it once was.

Problem

A **circular string** is a string that does not have an initial or terminal element; instead, the string is viewed as a necklace of symbols. We can represent a circular string as a string enclosed in parentheses. For example, consider the circular DNA string (ACGTAC), and note that because the string "wraps around" at the end, this circular string can equally be represented by (CGTACA), (GTACAC), (TACACG), (ACACGT), and (CACGTA). The definitions of substrings and superstrings are easy to generalize to the case of circular strings (keeping in mind that substrings are allowed to wrap around).

Given: A collection of (error-free) DNA k-mers ($k \leq 50$) taken from the same strand of a circular chromosome. In this dataset, all k-mers from this strand of the chromosome are present, and their de Bruijn graph consists of exactly one **simple cycle**.

Return: A cyclic superstring of minimal length containing the reads (thus corresponding to a candidate cyclic chromosome).

Sample Dataset

```
ATTAC
TACAG
GATTA
ACAGA
CAGAT
TTACA
AGATT
```

Sample Output

```
GATTACA
```

> **Note**
>
> The assumption made above that all reads derive from the same strand is practically unrealistic; in reality, researchers will not know the strand of DNA from which a given read has been sequenced.

Problem 16

Genome Assembly Using Reads

Putting the Puzzle Together

In practical genome sequencing, even if we assume that reads have been sequenced without errors, we have no idea of knowing immediately the particular strand of DNA a read has come from.

> Also, our reads may not have the same length. In 1995, Idury and Waterman proposed a way to boost read coverage and achieve uniform read length by breaking long reads into overlapping k-mers for some fixed value of k. For example, a 100 bp read could be split into 51 overlapping 50-mers.

Problem

A **directed cycle** is simply a cycle in a directed graph in which the head of one edge is equal to the tail of the next (so that every edge in the cycle is traversed in the same direction).

For a set of DNA strings S and a positive integer k, let S_k denote the collection of all possible k-mers of the strings in S.

Given: A collection S of (error-free) reads of equal length (not exceeding 50 bp). In this dataset, for some positive integer k, the de Bruijn graph B_k on $S_{k+1} \cup S_{k+1}^{\text{rc}}$ consists of exactly two directed cycles.

Return: A cyclic superstring of minimal length containing every read or its reverse complement.

Sample Dataset

```
AATCT
TGTAA
GATTA
ACAGA
```

Sample Output

```
GATTACA
```

> **Note**
>
> The reads "AATCT" and "TGTAA" are not present in the answer, but their reverse complements "AGATT" and "TTACA" are present in the circular string (GATTACA).

Problem 17

Genome Assembly with Perfect Coverage and Repeats

Repeats: A Practical Assembly Difficulty

Genome assembly is straightforward if we know in advance that the de Bruijn graph has exactly one directed cycle (see "Genome Assembly with Perfect Coverage").

In practice, a genome contains repeats longer than the length of the k-mers that we wish to use to assemble the genome. Such repeats increase the number of cycles present in the de Bruijn graph for these k-mers, thus preventing us from assembling the genome uniquely.

For example, consider the circular string (ACCTCCGCC), along with a collection S of error-free reads of length 3, exhibiting perfect coverage and taken from the same strand of an interval of DNA. The corresponding de Bruijn graph B_2 (where edges correspond to 3-mers and nodes correspond to 2-mers) has at least two directed cycles: one giving the original circular string (ACCTCCGCC), and another corresponding to the misfit (ACCGCCTCC).

Also, note that these cycles are not simple cycles, as the node corresponding to "CC" is visited three times in each cycle.

To generalize the problem of genome assembly from a de Bruijn graph to the case of genomes containing repeats, we therefore must add a constraint: in a cycle corresponding to a valid assembly, every 3-mer must appear as many times in the cycle as it does in our collection of reads (which correspond to all 3-mers in the original string).

Problem

Recall that a directed cycle is a cycle in a directed graph in which the head of one edge is equal to the tail of the following edge.

In a de Bruijn graph of k-mers, a circular string s is constructed from a directed cycle $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_i \rightarrow s_1$ is given by $s_1 + s_2[k] + \ldots + s_{i-k}[k] + s_{i-k+1}[k]$. That is, because the final $k-1$ symbols of s_1 overlap with the first $k-1$ symbols of s_2, we simply tack on the k-th symbol of s_2 to s, then iterate the process.

For example, the circular string assembled from the cycle "AC" → "CT" → "TA" → "AC" is simply (ACT). Note that this string only has length three because the 2-mers "wrap around" in the string.

If every k-mer in a collection of reads occurs as an edge in a de Bruijn graph cycle the same number of times as it appears in the reads, then we say that the cycle is "complete."

Given: A list S_{k+1} of error-free DNA $(k+1)$-mers ($k \leq 5$) taken from the same strand of a circular chromosome (of length ≤ 50).

Return: All circular strings assembled by complete cycles in the de Bruijn graph B_k of S_{k+1}. The strings may be given in any order, but each one should begin with the first $(k+1)$-mer provided in the input.

Sample Dataset

```
CAG
AGT
GTT
TTT
TTG
TGG
GGC
GCG
CGT
GTT
TTC
TCA
CAA
AAT
ATT
TTC
TCA
```

Sample Output

```
CAGTTCAATTTGGCGTT
CAGTTCAATTGGCGTTT
CAGTTTCAATTGGCGTT
CAGTTTGGCGTTCAATT
CAGTTGGCGTTCAATTT
CAGTTGGCGTTTCAATT
```