Assume that the first state is $\alpha$.

We start with the HMM $M = (\Sigma, Q, A, E, I)$ in Figure 11.7, where

\[
\begin{align*}
Q &= \{ \alpha, \beta \}, \\
\Sigma &= \{ a, c, g, t \}, \\
A &= \begin{pmatrix} 
\frac{9}{10} & \frac{1}{10} \\
\frac{1}{10} & \frac{9}{10} 
\end{pmatrix}, \\
E &= \begin{pmatrix} 
\frac{2}{5} & \frac{1}{5} & \frac{2}{5} & \frac{1}{5} \\
\frac{1}{5} & \frac{3}{10} & \frac{1}{5} & \frac{3}{10} 
\end{pmatrix}, \\
I(\alpha) &= 1, \\
I(\beta) &= 0.
\end{align*}
\]
Now, we run Viterbi’s algorithm on \( x = gact \). Instead of building a table, I will just compute each of the \( S_{k,t} \) values in turn, with the help of Matlab.

\[
S_{\alpha,0} = \log_2 I(\alpha) \\
= 0
\]

\[
S_{\beta,0} = \log_2 I(\beta) \\
= -\infty
\]

\[
S_{\alpha,1} = \log_2 e_\alpha(g) + \max_{\ell \in Q} S_{\ell,0} + \log_2 a_{\ell,\alpha} \\
= -1.3219 + 0 - 0.1520 \\
= -1.4739
\]

\[
S_{\beta,1} = \log_2 e_\beta(g) + \max_{\ell \in Q} S_{\ell,0} + \log_2 a_{\ell,\beta} \\
= -2.3219 + 0 - 3.3219 \\
= -5.6438
\]

\[
S_{\alpha,2} = \log_2 e_\alpha(g) + \max_{\ell \in Q} S_{\ell,1} + \log_2 a_{\ell,\alpha} \\
= -1.3219 - 1.4739 - 0.1520 \\
= -2.9478
\]

\[
S_{\beta,2} = \log_2 e_\beta(g) + \max_{\ell \in Q} S_{\ell,1} + \log_2 a_{\ell,\beta} \\
= -2.3219 - 1.4739 - 3.3219 \\
= -7.1177
\]

\[
S_{\alpha,3} = \log_2 e_\alpha(c) + \max_{\ell \in Q} S_{\ell,2} + \log_2 a_{\ell,\alpha} \\
= -3.3219 - 2.9478 - 0.1520 \\
= -6.4217
\]

\[
S_{\beta,3} = \log_2 e_\beta(c) + \max_{\ell \in Q} S_{\ell,2} + \log_2 a_{\ell,\beta} \\
= -1.7370 - 2.9478 - 3.3219 \\
= -8.0067
\]

\[
S_{\alpha,4} = \log_2 e_\alpha(t) + \max_{\ell \in Q} S_{\ell,3} + \log_2 a_{\ell,\alpha} \\
= -3.3219 - 6.4217 - 0.1520 \\
= -9.8956
\]

\[
S_{\beta,4} = \log_2 e_\beta(t) + \max_{\ell \in Q} S_{\ell,3} + \log_2 a_{\ell,\beta} \\
= -1.7370 - 8.0067 - 0.1520 \\
= -9.8957
\]

Actually, we now have \( S_{\alpha,4} = S_{\beta,4} \), though they do not appear to be equal due to round-off errors. We get that the logarithm of the maximum probability of any path for \( x \) is \( S_{\alpha,4} = S_{\beta,4} = -9.8956 \) and the paths that have that maximum probability are \( \pi_1 = \alpha\alpha\alpha\alpha \) and \( \pi_2 = \alpha\alpha\alpha\beta\beta \).
2. In this problem, we analyze a particular aspect of finding motifs with RandomMotifFinder.py. Consult the project assignment for relevant terminology and notation.

We are interested in the log-likelihood value that results from one iteration of the while loop in RandomMotifFinder.py. For simplicity, we assume that there is no planted motif, so all of the input strings are randomly generated according to the simplest null model, where each nucleotide is chosen with probability 0.25. After randomly selecting $s$, RandomMotifFinder.py is then examining an $n \times k$ matrix of nucleotides, where $d$ columns are “don’t cares”. In each column that is not a “don’t care”, it is choosing the most frequent nucleotide to put in that location in $M$.

A. What is the minimum value that $LL(M, s)$ can take, as a function of $k$, $d$, and $n$?

B. What is the maximum value that $LL(M, s)$ can take, as a function of $k$, $d$, and $n$?

C. We now consider the expected (average) value for $LL(M, s)$. Looking at the formulas for $L(M, s)$ and $LL(M, s)$, it is clear that we can concentrate on the distribution of counts in an arbitrary column that is not a “don’t care”. There are $4^n$ possible values for that column, each equally likely. RandomMotifFinder.py chooses the nucleotide in that column that has the maximum count. Hence, the count coming from that column is between $\left\lceil \frac{n}{4} \right\rceil$ and $n$. Write a program called MaxInColumnDistribution to compute the number of times (in the $4^n$ possibilities) that each such maximum count occurs and convert the counts to a probability distribution. For example, the probability distribution for $n = 2$ is

<table>
<thead>
<tr>
<th>count</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
</tr>
</tbody>
</table>

while for $n = 4$, it is

<table>
<thead>
<tr>
<th>count</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.09375</td>
</tr>
<tr>
<td>2</td>
<td>0.703125</td>
</tr>
<tr>
<td>3</td>
<td>0.1875</td>
</tr>
<tr>
<td>4</td>
<td>0.015625</td>
</tr>
</tbody>
</table>

In addition, your program should compute the expected log-likelihood for one column, given this distribution. (You can multiply by $k - d$ to get the expected log-likelihood for the best motif of length $k$ with $d$ “don’t cares”.)

Report the computed distribution and expected log-likelihood for a column for $n = 10$ and $n = 12$. Include the source code of your MaxInColumnDistribution in your PDF.

Such analysis programs can be made part of your toolkit to evaluate the performance of your MotifFinder.
A. For simplicity, assume that $n$ is divisible by 4. Then, the smallest value $\pi_{\alpha_j,j}$ can take is 0.25. (If $n$ is not divisible by 4, then this number should be $\left\lfloor \frac{2}{n} \right\rfloor$). The ratio $\frac{\pi_{\alpha_j,j}}{\rho_{\alpha_j}}$ is then 1, so the logarithm is 0. The minimum log-likelihood is then 0.

B. The maximum occurs when every $\pi_{\alpha_j,j} = 1$. Then, the ratio $\frac{\pi_{\alpha_j,j}}{\rho_{\alpha_j}} = 4$. Its logarithm is 2, and there are $k - d$ positions contributing that to the logarithm. Hence, the maximum log-likelihood is $2(k - d)$.

C. A sample Python 3 program for this task is in Figures 1 through 4.

The probability distribution for $n = 10$ is

<table>
<thead>
<tr>
<th>count</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.208282470703125</td>
</tr>
<tr>
<td>4</td>
<td>0.48065185546875</td>
</tr>
<tr>
<td>5</td>
<td>0.23215484619140625</td>
</tr>
<tr>
<td>6</td>
<td>0.06488800048828125</td>
</tr>
<tr>
<td>7</td>
<td>0.012359619140625</td>
</tr>
<tr>
<td>8</td>
<td>0.001544952392578125</td>
</tr>
<tr>
<td>9</td>
<td>0.00011444091796875</td>
</tr>
<tr>
<td>10</td>
<td>3.814697265625 $\times 10^{-6}$</td>
</tr>
</tbody>
</table>

The expected log-likelihood is 0.7159835655012098.

The probability distribution for $n = 12$ is

<table>
<thead>
<tr>
<th>count</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.022029876708984375</td>
</tr>
<tr>
<td>4</td>
<td>0.38001537322998047</td>
</tr>
<tr>
<td>5</td>
<td>0.38067626953125</td>
</tr>
<tr>
<td>6</td>
<td>0.16026735305786133</td>
</tr>
<tr>
<td>7</td>
<td>0.04588508605957031</td>
</tr>
<tr>
<td>8</td>
<td>0.009559392929077148</td>
</tr>
<tr>
<td>9</td>
<td>0.0014162063598632812</td>
</tr>
<tr>
<td>10</td>
<td>0.00014162063598632812</td>
</tr>
<tr>
<td>11</td>
<td>8.58306884765625 $\times 10^{-6}$</td>
</tr>
<tr>
<td>12</td>
<td>2.384185791015625 $\times 10^{-7}$</td>
</tr>
</tbody>
</table>

The expected log-likelihood is 0.6706569670078446.
#!/usr/bin/env python3

# Copyright (c) 2014 Lenwood S. Heath. All rights reserved.
# This program or module is free software: you can redistribute it and/or
# modify it under the terms of the GNU General Public License as published
# by the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version. It is provided for educational
# purposes and is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
# General Public License for more details.

""
This program takes a number n as input, generates all 4^n columns of
nucleotides with n rows, and accumulates the maximum nucleotide count
in each column.
""

import argparse
import logging
import math
import sys

Figure 1: The program MaxInColumnDistribution.py, part 1.
MaxInColumnDistribution generates all $4^n$ columns and accumulates the max counts. Finally, it computes the probability distribution.

```python
def MaxInColumnDistribution(n):
    four_to_n = 4**n
    distribution = [0.0 for i in range(n+1)]
    column = ['A' for i in range(n)]
    for j in range(four_to_n):
        counts = {x:0 for x in {'A','C','G','T'}}
        for nucleotide in column:
            counts[nucleotide] += 1
        maximum = max(counts.values())
        distribution[maximum] += 1.0
    for i in range(n):
        nucleotide = column[i]
        if nucleotide == 'A':
            nucleotide = 'C'
        elif nucleotide == 'C':
            nucleotide = 'G'
        elif nucleotide == 'G':
            nucleotide = 'T'
        else:
            nucleotide = 'A'
        column[i] = nucleotide
        if nucleotide != 'A':
            break
    for i in range(n+1):
        distribution[i] /= four_to_n
    return distribution
```

Figure 2: The program `MaxInColumnDistribution.py`, part 2.
def main():
    
    The main program.
    Uses argparse to parse the command line arguments.
    Uses logging to record progress.
    Then call MaxInColumnDistribution to compute the distribution.
    Finally, print the distribution to standard output.
    
    # Set up basic logging
    logging.basicConfig(filename="MaxInColumnDistribution.log",filemode="w",
                        level=logging.INFO,format="%(levelname)s: %(asctime)s: %(message)s")
    logging.info("Starting MaxInColumnDistribution")

    # Process command line arguments
    argparser = argparse.ArgumentParser(
        description="Compute distribution of max nucleotide count in a column.")
    argparser.add_argument("-n",dest="n",type=int,default=10,
        help="Number of rows in a column")
    args = argparser.parse_args()
    logging.info("Arguments sucessfully parsed")
    n = args.n
    logging.info("Number of rows in a column is " + str(n))
    if n <= 0:
        raise ValueError("n must be at least 1")

    Figure 3: The program MaxInColumnDistribution.py, part 3.
# Compute distribution
distribution = MaxInColumnDistribution(n)

# Report distribution
print(distribution)

# Report expected log-likelihood for random DNA
expected = 0.0
for i in range(n):
    j = i+1
    log_likelihood = math.log2((float(j)/n)/0.25)
    expected += distribution[j]*log_likelihood
print("Expected = {}".format(expected))

if __name__ == "__main__":
    main()

Figure 4: The program MaxInColumnDistribution.py, part 4.