Exercise 3

1. Create a function that takes a PWM and string as input and finds the best match of the PWM in the string.

2. Create a function that determines the similarity (edit distance) of two PWMs.

3. Exercise 4.17 in Jones and Pevzner
### Naïve string search

<table>
<thead>
<tr>
<th>PWM</th>
<th>A 1 0 6 0 0 1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C 2 7 0 6 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>G 4 0 1 1 7 0 7 6</td>
</tr>
<tr>
<td></td>
<td>T 0 0 0 0 0 5 0 1</td>
</tr>
</tbody>
</table>

| Consensus  | G C A C G T G G |

**Input string:**
```
acgcggccaatcagaggccACGCA CG AG GTggagggcaagtccgcctctcgggagg
```

**Scan:**
```
gc acg tgg
gac acg tgg
gca cg tgg
gcac gtgg
gca cgtgg
gcac gtgg
gcac gtgg
...
```
Naïve string search algorithm

NaiveStringSearch(p, s):

1. for i in range(len(s)):
   1. match = True
   2. for j in range(len(p)):
      1. match = match and (s[i + j] == p[j])
   3. if match:
      1. return i

2. return -1
Extending `NaiveStringSearch` to find position with most matches

```python
StringSearchMostMatch(p, s):
    bestIndex = 0
    bestCount = 0
    for i in range(len(s)):
        count = 0
        for j in range(len(p)):
            if s[i + j] == p[j]:
                count = count + 1
        if count > bestCount:
            bestIndex = i
            bestCount = count
    return bestIndex
```

```python
NaiveStringSearch(p, s):
    for i in range(len(s)):
        match = True
        for j in range(len(p)):
            match = match and (s[i + j] == p[j])
        if match:
            return i
    return -1
```
Extending NaiveStringSearch to find position with most matches

StringSearchMostMatch(p, s):
1. bestIndex = 0
2. bestCount = 0
3. for i in range(len(s)):
   1. count = 0
   2. for j in range(len(p)):
      1. if s[i + j] == p[j]:
         1. count = count + 1
   3. if count > bestCount:
      1. bestIndex = i
      2. bestCount = count
4. return bestIndex

NaiveStringSearch(p, s):
1. for i in range(len(s)):
   1. match = True
   2. for j in range(len(p)):
      1. match = match and (s[i + j] == p[j])
   3. if match:
      1. return i
2. return -1
Scanning with PWM is straightforward extension

PWMscan(pwm, s):
1. bestIndex = 0
2. bestCount = 0
3. for i in range(len(s)):
   1. count = 0
   2. for j in range(len(p)):
      1. count = count + pwm[j][s[i + j]]
   3. if count > bestCount:
      1. bestIndex = i
      2. bestCount = count
4. return bestIndex

PWM:
A 1 0 6 0 0 1 0 0
C 2 7 0 6 0 1 0 0
G 4 0 1 1 7 0 7 6
T 0 0 0 0 0 5 0 1
String:
What if insertions in string is allowed?

PWM:
A 1 0 6 0 0 1 0 0
C 2 7 0 6 0 1 0 0
G 4 0 1 1 7 0 7 6
T 0 0 0 0 0 5 0 1

Insertion cost: -1

Input string:
acgcggccaatcagaggccACGCA
CGAGGtggggcaagtccgctctcggaggg

Scan:

GCACGAGG = 44
GCACG- -TGG = 45
Aligning PWM and string

- Initialization?
- Recurrence?
Aligning PWM and string (PA)

- Initialization
  - Insertion cost: -1

- Recurrence?

<table>
<thead>
<tr>
<th>PWM:</th>
<th>ε</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>String:</td>
<td>ε</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Aligning PWM and string (PA)

- **Initialization**
  - Insertion cost: -1

- **Recurrence**
  - \( \max\{PA(S_{i-1},P_{j-1}) + PWM[j][S_i], \ PA(S_{i-1},P_j) - 1\} \)

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varepsilon)</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-4</td>
<td>-5</td>
<td>-6</td>
<td>-7</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Exercise 3

1. Create a function that takes a PWM and string as input and finds the best match of the PWM in the string.

2. Create a function that determines the similarity (edit distance) of two PWMs.

3. Exercise 4.17 in Jones and Pevzner
**Similarity of two PWMs?**

- **Similarity of two PWM columns**

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Total: 8

\[
d(P_{1i}, P_{2j}) = \sum_{a \in DNA} \left| P_{1i,a} - P_{2j,a} \right|
\]
Aligning two PWMs

- **Initialization?**
- **Recurrence?**

\[ d(P_{1i}, P_{2j}) = \sum_{a \in DNA} |P_{1i,a} - P_{2j,a}| \]
Aligning two PWMs

- Initialization?
  - Insertion cost: \( k \)
- Recurrence?

\[
d(P_{1i}, P_{2j}) = \sum_{a \in \text{DNA}} |P_{1i,a} - P_{2j,a}|
\]
Aligning two PWMs

- Initialization?
  - Insertion cost: $k$

- Recurrence?
  - $\min\{PP(P_{1i-1}, P_{2j-1}) + d(P_{1i}, P_{2j}), PP(P_{1i-1}, P_{2j}) + k, PP(P_{1i}, P_{2j-1}) + k\}$

$$d(P_{1i}, P_{2j}) = \sum_{a \in \text{DNA}} |P_{1i,a} - P_{2j,a}|$$
Exercise 3

1. Create a function that takes a PWM and string as input and finds the best match of the PWM in the string.
2. Create a function that determines the similarity (edit distance) of two PWMs.
3. Exercise 4.17 in Jones and Pevzner
Branch and bound median string

- Internal nodes represent word prefixes
  - Skip subtrees where prefix has higher distance than current best
    - optimisticDistance = \( d_H(\text{word}_i, s) \)

Best distance = 10

\[
\begin{align*}
    d_H(“aaac”, s) &= 9 \\
    d_H(“aaag”, s) &= 10 \\
    d_H(“aat”, s) &= 11
\end{align*}
\]

Tighter bound possible? (Hint: “aaacaa” = “aaac” + “aa”)
Hamming distance for given positions can be subdivided into substrings

\[
    \text{Score}(s) = \sum_{j=1}^{l} M_{P(s)}(j) = lt - d_H(w, s) = lt - \sum_{i=1}^{t} d_H(w, s_i) = lt - \sum_{i=1}^{t} d_H(u, s_i) - \sum_{i=1}^{t} d_H(v, s_{i+|u|})
\]

\[
d_H(u, s) + d_H(v, s) \neq d_H(w, s)
\]
Solving equation means evaluating all subtrees

\[
\text{Score}(s) = \sum_{j=1}^{l} M_{P(s)}(j) = lt - d_H(w,s) = lt - \sum_{i=1}^{l} d_H(w,s_i) = lt - \sum_{i=1}^{l} d_H(u,s_i) - \sum_{i=1}^{l} d_H(v,s_{i+|u|})
\]

\[
d_H(u,s) + d_H(v,s) \leq d_H(w,s)
\]