week3 pseudo-code

B363 Fall lab

September 2016

0.1 Compute the Hamming Distance Between Two Strings

Input: two string a, b with same length
Output: total count of mismatches between two string

Algorithm 1 Hamming Distance

```
count ← 0
for i = 0 to |a| do
    if a[i] ≠ b[i] then
        count ← count + 1
return count
```

0.2 Find Frequent Words with Mismatches and Reverse Complements

Input: A DNA string Text as well as integers k(length of pattern) and d(maximum hamming distance)
Output: All k-mers Pattern maximizing the count of k-mers with mismiatches and reverse complements

Algorithm 2 FindingFrequentWordsWithMismatchesAndReverseComplementsBySorting(Text, k, d)

FrequentPatterns ← an empty set
Neighborhoods ← an empty list
for $i = 0$ to $|\text{Text}| - \text{k}$ **do**
    add NEIGHBORS(Text(i,k), d) to *Neighborhoods*
    add NEIGHBORS($\overline{Text}(i, k), d$)*to Neighborhoods*
form an array *NeighborhoodArray* holding all strings in Neighborhoods
for $i = 0$ to $|Neighborhoods| - 1$ **do**
    Pattern ← *NeighborhoodArray(i)*
    Index(i) ← PATTERNTONUMBER(pattern)
    Count(i) ← 1
SortedIndex ← SORT(*Index*)
for $i = 0$ to $|Neighborhoods| - 2$ **do**
    if *SortedIndex(i) = SortedIndex(i+1)* **then**
        Count(i+1) ← *Count(i)+1*
maxCount ← maximum value in array *Count*
for $i = 0$ to $|Neighborhoods| - 1$ **do**
    if *Count(i) = maxCount* **then**
        Pattern ← NUMBERTOPATTERN(*SortedIndex(i)*, k)
        add Pattern to *FrequentPatterns*
return *FrequentPatterns*

Algorithm 3 NEIGHBORS(Pattern, d)

```
if d = 0 then
    return {Pattern}
if |Pattern| = 1 then
    return {A,C,G,T}
Neighborhood ← an empty set
SuffixNeighbors ← NEIGHBORS(Suffix(Pattern), d)
for each string Text from SuffixNeighbors do
    if HAMMINGDISTANCE(Suffix(Pattern), Text) < d then
        for each nuceotide x do  add x + Text to Neighborhood
    else
        add FirstSymbol(Pattern) + Text to Neighborhood
return Neighborhood
```

Input: A DNA string Text
Output: the reverse complement string of Text

Algorithm 4 Reverse Complement

```
rText ← reverse Text
Text̄ ← an empty string
for i = 0 to |rText| − 1 do
    Text̄(i) ← complement of rText(i)
return Text̄
```

Algorithm 5 PATTERNTONUMBER(Pattern)

```
if Pattern contains no symbols then
    return 0
symbol ← LastSymbol(Pattern)
Prefix ← PREFIX(Pattern)
return 4·PATTERNTONUMBER(Prefix) + SYMBOLTONUMBER(symbol)
```

Algorithm 6 NUMBERTOPATTERN(index, k)

```
if k = 1 then
    return NUMBERTOSYMBOL(index)
prefixIndex ← QUOTIENT(index, 4)
r ← REMAINDER(index, 4)
PrefixPattern ← NUMBERTOPATTERN(prefixIndex, k-1)
return concatenation of PrefixPattern with symbol
```

Reference: Compeau, Phillip, and Pavel Pevzner. Bioinformatics algorithms: an active learning approach. Active Learning Publishers, 2015.