Stefan Kurtz **Universität Hamburg**
Christina de Bruyn Kops **Zentrum für Bioinformatik**

Grundlagen der Sequenzanalyse
Wintersemester 2016/2017
Übungen zur Vorlesung: Ausgabe am 17.01.2017

Punkteverteilung: Aufgabe 12.1: 4 Punkte, Aufgabe 12.2: 3 Punkte, Aufgabe 12.3: 6 Punkte

Abgabe bis zum 23.1.

Aufgabe 12.1 Sei $\mathcal{A}$ ein beliebiges Alphabet. Wir definieren eine Scorefunktion σ für alle Ersetzungsoperationen $\alpha \to \beta$ mit $\alpha, \beta \in \mathcal{A}$ wie folgt:

$$\sigma(\alpha \to \beta) = \begin{cases} 1 & \text{falls } \alpha = \beta \\ 0 & \text{sonst} \end{cases}$$

1. Gegeben sei eine Anfragesequenz $w = ggccgc$ und $\mathcal{A} = \{c, g\}$. Konstruieren Sie für $q = 4$ und $k = 3$ die Umgebung $Env_k(w)$ von w gemäß des Blast-Ähnlichkeitsmodells.
2. Charakterisieren Sie die folgenden Umgebungen des Blast-Ähnlichkeitsmodells in Ihren eigenen Worten (für beliebige Alphabete und die obige Scorefunktion σ):
   - Die Umgebung für $k = 0$.
   - Die Umgebung für $k > q$.
   - Die Umgebung für $k = 1$.
   - Die Umgebung für $k = q$.

Hinweis: Bei dieser Aufgabe ist es nicht notwendig, ein Programm zu schreiben!

Aufgabe 12.2 Geben Sie die DP-Rekurrenz an, um das optimale *Multiple Sequenz-Alignment* (*MSA*) von $k = 3$ Sequenzen zu berechnen. D.h. leiten Sie aus der allgemeinen DP-Rekurrenz für k Sequenzen (siehe Skript, Kapitel 5.3) die Rekurrenz für $k = 3$ ab, wobei die jeweiligen Bedingungen für die einzelnen Fälle explizit angegeben werden sollen.

Aufgabe 12.3 Eine positionsspezifische Scoring-Matrix (PSSM) ist eine Repräsentation eines multiplen Sequenzalignments, die zum Beispiel zur Suche nach Sequenzmotiven eingesetzt werden kann.

1. Implementieren Sie eine Datenstruktur (z.B. eine Ruby-Klasse), die eine PSSM speichert. Es soll möglich sein, diese Matrix aus einer Textdatei einlesen zu können. In dieser Textdatei sind die vorkommenden Zeichen des Alphabets in der ersten Spalte angegeben, und in allen anderen Spalten sind die Scores für das jeweilige Zeichen an der entsprechenden Position abgelegt:

```
A -19   5   7 -29 -14 -25   7 -34   7  -7
C  92 -17  -8  99 -22 -34  -8 -27  40  43
D -45  17 -29 -55  14 -25 -25 -44 -16  16
E -49  22 -28 -61  22 -16 -24 -43 -14  -7
```

```
F -30 -28   2 -42 -28 -37 -19 -60 -9  -27
G -36 -15 -25 -45   9 -30 -23 -41 -14 -15
H -38  -7 -10 -47  -8 -15 -22  -8  -6  -9
I -12 -23  25 -31 -26 -36   4 -16 -17 -24
K -41  -8 -23 -52  15  45 -15 -38  14  -5
L -21 -27  -4 -34 -27 -34 -10 -14 -20 -26
M -22 -21  -5 -36 -20 -26  -8 -17 -15 -18
N -40 -26 -25 -49  -7 -18 -19 -39 -10  -6
P -46  18 -32 -56 -26 -35 -29 -51 -24 -25
Q -44  -7 -26 -55  -3  -9 -21 -40 -11  25
R -44 -13 -25 -55  31  49  11 -36  12  13
S -30  -9 -18 -38 -13 -25 -13 -39  15  25
T -25   9  13 -35   5 -26  31 -35   9  -8
V  16 -19  22 -29 -23 -33  31 -21 -13 -21
W -35 -33 -11 -44 -30 -39 -31  -1 -16 -30
Y -34 -25  36 -46 -24 -31 -22  56  20 -24
```

Eine beispielhafte PSSM mit dem Zinkfingermotiv aus dem Skript ist im oben beschriebenen Format in STiNE zu finden. Hinweis: Es ist u.U. möglich, den Parser für Score-Matrizen hier abgewandelt wiederzuverwenden.

2. Implementieren Sie den einfachen Algorithmus zur Ermittlung von PSSM-Treffern in einer gegebenen Sequenz. Der Schwellwert th soll vom Benutzer angegeben werden können. Schließlich soll zeilenweise eine Liste von Startpositionen von Treffern in der Anfragesequenz und die jeweiligen Score-Werte ausgegeben werden und zwar im folgenden Format, in dem Werte in einer Zeile durch einen Tabulator getrennt werden.

   ```
   1 44
   3 12
   6 436
   ```

3. Implementieren Sie den effizienten *lookahead-scoring*-Algorithmus aus der Vorlesung.

4. Verwenden Sie Ihre Implementierung der beiden Algorithmen in einem Programm. Dieses soll 4 Parameter erhalten:

   `<s|l> <PSSM-Datei> <Sequenz> <Threshold>`

   dabei steht `s` bzw. `l` für die Auswahl des Algorithmus und `Sequenz` ist der zu durchsuchende String über dem Alphabet der PSSM, die Sie aus der Datei PSSM-Datei lesen.

5. Führen Sie das Bash-Skript `test.sh` (siehe Material in STiNE) aus, um Ihren Code zu testen. Das einzige Argument von `test.sh` ist der Name Ihres Programms. Die Ausgabe Ihres Programms soll identisch mit dem Inhalt der Datei `PSSM.matches` aus den Materialien sein.

Bitte die Lösungen zu diesen Aufgaben bis zum 23.01.2017 abgeben. Die Besprechung der Lösungen erfolgt am 24.01.2017.

Lösung zu Aufgabe 12.1:

1.

$$Env_k(\texttt{ggccgc}) = \left\{ \begin{array}{lll} (\texttt{cgcc},1)_3, & (\texttt{gccc},1)_3, & (\texttt{ggcc},1)_4, \\ (\texttt{ggcg},1)_3, & (\texttt{gggc},1)_3, & (\texttt{cccg},2)_3, \\ (\texttt{gccc},2)_3, & (\texttt{gccg},2)_4, & (\texttt{gcgg},2)_3, \\ (\texttt{ggcg},2)_3, & (\texttt{cccc},3)_3, & (\texttt{ccgc},3)_4, \\ (\texttt{ccgg},3)_3, & (\texttt{cggc},3)_3, & (\texttt{gcgc},3)_3 \end{array} \right\}$$

2.
- $k = 0$: D.h. die Anzahl der matchenden Zeichen muss mindestens 0 sein. Das gilt für alle q-Worte über $\mathcal{A}$. Damit ist

$$Env_k(w) = \{(s,i) \mid s \in \mathcal{A}^q, i \in [1, |w| - q + 1]\}$$

- $k > q$: D.h. die Anzahl der matchenden Zeichen muss grösser sein als q. Das gilt für kein q-Wort. Damit ist $Env_k(w) = \emptyset$
- $k = 1$: $Env_k(w)$ enthält alle Paare (s,i), so dass $1 \leq i \leq |w| - q + 1$ und $s \in \mathcal{A}^q$ stimmt an mindestens einer Position mit $w[i \ldots i+q-1]$ überein.
- $k = q$: D.h. die Anzahl der matchenden Zeichen muss genau q sein. Es müssen also alle Zeichen der q-Worte passen, d.h. die Umgebung besteht aus den q-Worten von w. Es gilt also:

$$Env_k(w) = \{(w[i \ldots i+q-1], i) \mid i \in [1, |w| - q + 1]\}$$

Lösung zu Aufgabe 12.2:

Recall the original recurrence for arbitrary $k > 1$.

$$\begin{aligned} M(q_1, \ldots, q_k) = \min\{ & M(q_1 - d_1, \ldots, q_k - d_k) + \\ & colcost(\varphi(q_1, d_1), \ldots, \varphi(q_k, d_k)) \mid (d_1, \ldots, d_k) \in \{0,1\} \times \cdots \times \{0,1\}, \\ & (d_1, \ldots, d_k) \neq (0, \ldots, 0), \\ & q_1 \geq d_1, \ldots, q_k \geq d_k\} \end{aligned}$$

where

$$\varphi(q_i, d_i) = \left\{ \begin{array}{ll} s_i[q_i] & \text{if } d_i = 1 \\ - & \text{otherwise} \end{array} \right.$$

and $colcost$ is a function assigning costs to a column of k elements from $\mathcal{A}'$, depending on the chosen model.

Now specialize the recurrence for $k = 3$:

$$\begin{aligned} M(q_1, q_2, q_3) = \min\{ & M(q_1 - d_1, q_2 - d_2, q_3 - d_3) + \\ & colcost(\varphi(q_1, d_1), \varphi(q_2, d_2), \varphi(q_3, d_3)) \mid (d_1, d_2, d_k) \in \{0,1\} \times \{0,1\} \times \{0,1\}, \\ & (d_1, d_2, d_3) \neq (0,0,0), \\ & q_1 \geq d_1, \ldots, q_k \geq d_k\} \end{aligned}$$

Assume that $q_i \geq 1$ for $1 \leq i \leq 3$. Then we can simplify the previous right hand side as follows:

$$
\begin{aligned}
&= \min\{\, M(q_1 - d_1, q_2 - d_2, q_3 - d_3) + colcost(\varphi(q_1, d_1), \varphi(q_2, d_2), \varphi(q_3, d_3)) \mid \\
&\qquad (d_1, d_2, d_k) \in \{(0,0,1),(0,1,0),(0,1,1),(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}\} \\
&= \min \quad \{M(q_1 - 0, q_2 - 0, q_3 - 1) + colcost(\varphi(q_1, 0), \varphi(q_2, 0), \varphi(q_3, 1))\} \cup \\
&\qquad \{M(q_1 - 0, q_2 - 1, q_3 - 0) + colcost(\varphi(q_1, 0), \varphi(q_2, 1), \varphi(q_3, 0))\} \cup \\
&\qquad \{M(q_1 - 0, q_2 - 1, q_3 - 1) + colcost(\varphi(q_1, 0), \varphi(q_2, 1), \varphi(q_3, 1))\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 0, q_3 - 0) + colcost(\varphi(q_1, 1), \varphi(q_2, 0), \varphi(q_3, 0))\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 0, q_3 - 1) + colcost(\varphi(q_1, 1), \varphi(q_2, 0), \varphi(q_3, 1))\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1, q_3 - 0) + colcost(\varphi(q_1, 1), \varphi(q_2, 1), \varphi(q_3, 0))\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1, q_3 - 1) + colcost(\varphi(q_1, 1), \varphi(q_2, 1), \varphi(q_3, 1))\} \\
&= \min \quad \{M(q_1, q_2, q_3 - 1) + colcost(-, -, s_3[q_3])\} \cup \\
&\qquad \{M(q_1, q_2 - 1, q_3) + colcost(-, s_2[q_2], -)\} \cup \\
&\qquad \{M(q_1, q_2 - 1, q_3 - 1) + colcost(-, s_2[q_2], s_3[q_3])\} \cup \\
&\qquad \{M(q_1 - 1, q_2, q_3) + colcost(s_1[q_1], -, -)\} \cup \\
&\qquad \{M(q_1 - 1, q_2, q_3 - 1) + colcost(s_1[q_1], -, s_3[q_3])\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1, q_3) + colcost(s_1[q_1], s_2[q_2], -)\} \cup \\
&\qquad \{M(q_1 - 1, q_2 - 1, q_3 - 1) + colcost(s_1[q_1], s_2[q_2], s_3[q_3])\}
\end{aligned}
$$

In case $q_i = 0$ for some i, $1 \leq i \leq 3$, the corresponding terms with $q_i - 1$ do not appear in the minimum. Thus for any choice of q_i, we can derive the following equality:

$$
M(q_1, q_2, q_3) = \min \left\{ \begin{array}{lcl}
\{M(q_1, q_2, q_3 - 1) & + & colcost(-, -, s_3[q_3]) \mid q_3 \geq 1\} \cup \\
\{M(q_1, q_2 - 1, q_3) & + & colcost(-, s_2[q_2], -) \mid q_2 \geq 1\} \cup \\
\{M(q_1, q_2 - 1, q_3 - 1) & + & colcost(-, s_2[q_2], s_3[q_3]) \mid q_2, q_3 \geq 1\} \cup \\
\{M(q_1 - 1, q_2, q_3) & + & colcost(s_1[q_1], -, -) \mid q_1 \geq 1\} \cup \\
\{M(q_1 - 1, q_2, q_3 - 1) & + & colcost(s_1[q_1], -, s_3[q_3]) \mid q_1, q_3 \geq 1\} \cup \\
\{M(q_1 - 1, q_2 - 1, q_3) & + & colcost(s_1[q_1], s_2[q_2], -) \mid q_1, q_2 \geq 1\} \cup \\
\{M(q_1 - 1, q_2 - 1, q_3 - 1) & + & colcost(s_1[q_1], s_2[q_2], s_3[q_3]) \mid q_1, q_2, q_3 \geq 1\}
\end{array} \right\}
$$

with $\min(\emptyset) = 0$.

Lösung zu Aufgabe 12.3:

```ruby
#!/usr/bin/env ruby

class PositionSpecificScoreMatrix

  def initialize(scorefile)
    if !File.exists?(scorefile)
      raise "The specified file '#{scorefile}' does not exist."
    end
    @mlen, @score_matrix, @alphabet = read_pssm(scorefile)
    @sigma = precalculate_thresholds(@mlen,@score_matrix)
  end

  # naive algorithm to determine positions with score over threshold
  # O(|M||w|) (quadratic) asymptotic time complexity
  def match_simple(w, threshold)
    positions = Array.new()
    0.upto(w.length - @mlen) do |pos|
      score = 0
      0.upto(@mlen - 1) do |matchpos|
```

```
        score += @score_matrix[w[pos + matchpos]][matchpos]
      end
      if score >= threshold then
        positions.push([pos, score])
      end
    end
    return positions
  end

  # smart lookahead-scoring algorithm
  def match_lookahead(w, threshold)
    intermediate_thresholds = Array.new()
    0.upto(@mlen - 1) do |d|
      intermediate_thresholds[d] = threshold - @sigma[d]
    end
    positions = Array.new()
    0.upto(w.length - @mlen) do |pos|
      score = 0
      0.upto(@mlen - 1) do |matchpos|
        score += @score_matrix[w[pos + matchpos]][matchpos]
        break if score < intermediate_thresholds[matchpos]
      end
      if score >= threshold then
        positions.push([pos, score])
      end
    end
    return positions
  end

private
  def read_pssm(scorefile)
    mlen = nil
    score_matrix = Hash.new()
    alphabet = Array.new()
    File.open(scorefile) do |f|
      i = 0
      lines = f.each_line do |ln|
        i += 1
        tokens = ln.split
        character = tokens.shift
        alphabet.push(character)
        if mlen.nil? then
          mlen = tokens.length
        elsif mlen != tokens.length then
          raise "line #{i} has different number of items than the rest!"
        end
        score_matrix[character] = Array.new()
        tokens.each do |value|
          begin
            score_matrix[character].push Integer(value)
          rescue => err
            raise "line #{i} contains non numerical value #{value} (#{err})"
          end
        end
      end
    end
```

```
    return mlen, score_matrix, alphabet
  end

  def precalculate_thresholds(mlen,score_matrix)
    max = Array.new(mlen, 0)
    0.upto(mlen - 1) do |d|
      score_matrix.each do |k, v|
        max[d] = [v[d], max[d]].max
      end
    end

    sigma = Array.new(mlen, 0)
    intermediate_thresholds = Array.new()
    0.upto(mlen - 1) do |d|
      (d + 1).upto(mlen - 1) do |h|
        sigma[d] += max[h]
      end
    end
    return sigma
  end

end

USAGE="#{$0} [s|l] <PSSM file> <sequence> <threshold>\n"\
  "s or l defines which algorithm should be used"
if ARGV.length != 4 then
  STDERR.puts "wrong number of arguments"
  STDERR.puts USAGE
  exit 1
end

if ARGV[0] != "s" and ARGV[0] != "l" then
  STDERR.puts "first argument has to be 's' or 'l'"
  STDERR.puts USAGE
  exit 1
end

begin
  pssm = PositionSpecificScoreMatrix.new(ARGV[1])
  results = nil
  if ARGV[0] == 's' then
    results = pssm.match_simple(ARGV[2], Integer(ARGV[3]))
  else
    results = pssm.match_lookahead(ARGV[2], Integer(ARGV[3]))
  end
  results.each do |pos, score|
    puts "#{pos}\t#{score}"
  end
rescue Exception => msg
  STDERR.puts "Error: #{msg}"
end
```