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

Grundlagen der Sequenzanalyse
Wintersemester 2016/2017
Übungen zur Vorlesung: Ausgabe am 13.12.2016

Punkteverteilung: Aufgabe 9.1: 6 Punkte, Aufgabe 9.2: 4 Punkte

Aufgabe 9.1 Implementieren Sie ein Programm `affinealign` zur Berechnung global optimaler Alignments unter dem affinen Gap-Kosten-Modell. Das Programm soll die zu alignierenden Sequenzen aus Dateien oder aus Kommandozeilenargumenten einlesen und als Ausgabe ein optimales Alignment – wie aus den vorherigen Aufgaben bekannt – ausgeben. Zudem sollte Ihr Programm dem Benutzer erlauben, die Kosten zum Starten und zum Erweitern eines Gaps zu definieren:

```
./affinealign <seq1> <seq2> <gap_open_cost> <gap_extend_cost>
```

Verwenden Sie, soweit möglich, den Code aus den vorhergegangenen Aufgaben wieder.

Sie können das Testskript in STiNE (`test_affine.rb`) verwenden, um Ihre Ergebnisse mit der Musterlösung zu vergleichen. Dabei wird die oben angegebene Parameter-Reihenfolge vorausgesetzt; die Sequenzen werden als Kommandozeilenargumente übergeben. Rufen Sie dazu das Testskript mit Ihrem Programmnamen als Parameter auf. Sollte keine Ausgabe erscheinen, so liefert Ihr Programm das gleiche Ergebnis wie die Musterlösung und der exit-code von `test_affine.rb` ist 0.

Aufgabe 9.2 Im folgenden sei δ die Einheitskostenfunktion. Dann liefert $edist_\delta$ die sog. *Einheitskostendistanz* von zwei Sequenzen.

(1) Seien $u = acgacgtag$ und $v = ggacgtgcag$ zwei Sequenzen über dem DNA-Alphabet $\mathcal{A} = \{a, c, g, t\}$. Bestimmen Sie die q-Wort Distanz von u und v für $q = 2$ und $q = 3$.

(2) Bestimmen Sie $edist_\delta(u, v)$ und ein optimales Alignment von u und v.

(3) Ersetzen Sie in u und v die Zeichen a und g jeweils durch r und die Zeichen c und t durch y. Welche Werte erhalten Sie nun für die q-Wort Distanz (für $q = 2$ oder $q = 3$) und für die Einheitskostendistanz?

(4) Geben Sie Beispiele für Paare (u_1, v_1), (u_2, v_2), (u_3, v_3) von nichtleeren Sequenzen und Werte q_1, q_2, q_3 an, für die gilt:

a) $qgdist_{q_1}(u_1, v_1) > edist_\delta(u_1, v_1)$

b) $qgdist_{q_2}(u_2, v_2) = edist_\delta(u_2, v_2)$

c) $qgdist_{q_3}(u_3, v_3) < edist_\delta(u_3, v_3)$

Dabei können Sie $q_i \geq 2$ für $1 \leq i \leq 3$ frei wählen.

Hinweis: Zur Lösung dieser Aufgabe ist es nicht notwendig, ein Programm zu schreiben, aber Sie dürfen natürlich vorhandene Software nutzen.

Bitte die Lösungen zu diesen Aufgaben bis zum 19.12.2016 abgeben. Die Besprechung der Lösungen erfolgt am 20.12.2016.

Lösung zu Aufgabe 9.1:

```
class Alignment
# Die Datenstruktur "Alignment"

  def initialize(u, v, m, n)
  # wird bei Alignment.new() aufgerufen, entspricht init_alignment
    @u = u
    @v = v
    @m = m
    @n = n
    @eoplist = []
  end

  def add_item(item, nof_ops)
    if @eoplist.last.nil? or @eoplist.last[0] != item then
      @eoplist.push([item, nof_ops])
    else
      @eoplist.last[1] += nof_ops
    end
  end

  def reverse!
    @eoplist.reverse!
  end

  def add_replacement(nof_ops = 1)
    add_item(:R, nof_ops)
  end

  def add_deletion(nof_ops = 1)
    add_item(:D, nof_ops)
  end

  def add_insertion(nof_ops = 1)
    add_item(:I, nof_ops)
  end

  def show_alignment()
    #first line
    uctr = 0
    @eoplist.each do |eop|
      case eop[0]
        when :R
          0.upto(eop[1]-1) do |i|
            print @u[uctr].chr
            uctr += 1
          end
        when :D
          0.upto(eop[1]-1) do |i|
            print @u[uctr].chr
            uctr += 1
          end
        when :I
```

```
        0.upto(eop[1]-1) do |i|
          print "-"
        end
    end
  end
  print "\n"
  #middle line
  uctr = 0
  vctr = 0
  @eoplist.each do |eop|
    case eop[0]
      when :R
        0.upto(eop[1]-1) do |i|
          if @u[uctr] == @v[vctr] then
            print "|"
          else
            print " "
          end
          uctr += 1
          vctr += 1
        end
      when :D
        0.upto(eop[1]-1) do |i|
          print " "
          uctr += 1
        end
      when :I
        0.upto(eop[1]-1) do |i|
          print " "
          vctr += 1
        end
    end
  end
  print "\n"
  #last line
  vctr = 0
  @eoplist.each do |eop|
    case eop[0]
      when :R
        0.upto(eop[1]-1) do |i|
          print @v[vctr].chr
          vctr += 1
        end
      when :I
        0.upto(eop[1]-1) do |i|
          print @v[vctr].chr
          vctr += 1
        end
      when :D
        0.upto(eop[1]-1) do |i|
          print "-"
        end
    end
  end
  print "\n"
end
```

```ruby
  def eval_alignment(match_cost = 0, mismatch_cost = 1, \
                     del_cost = 1, ins_cost = 1)
    totalcost = 0
    uctr = 0
    vctr = 0
    @eoplist.each do |eop|
      case eop[0]
        when :R
          0.upto(eop[1]-1) do |i|
            if @u[uctr] == @v[vctr] then
              totalcost += match_cost
            else
              totalcost += mismatch_cost
            end
            uctr += 1
            vctr += 1
          end
        when :I
          0.upto(eop[1]-1) do |i|
            totalcost += ins_cost
            vctr += 1
          end
        when :D
          0.upto(eop[1]-1) do |i|
            totalcost += del_cost
            uctr += 1
          end
     end
    end
    totalcost
  end

  def delete_alignment()
    # (entfaellt bei Verwendung von Ruby)
  end
end

if __FILE__ == $0
# Beispielhafte Benutzung dieser Klasse
u = "acgtagatatatagat"
v = "agaaagaggtaagaggga"
a = Alignment.new(u, v, u.length, v.length)
a.add_replacement(7)
a.add_insertion(2)
a.add_replacement(2)
a.add_deletion()
a.add_replacement(3)
a.add_insertion()
a.add_replacement(3)
a.show_alignment()
puts "Gesamtkosten: #{a.eval_alignment()}"
end

#!/usr/bin/env ruby

$:.unshift(File.join(File.dirname(__FILE__), '.'))
```

```
require 'alignment'

# get a useful infinity constant (IEEE 754)
Infinity = 1.0/0.0

class AffineDPentry
  attr_accessor :Rdist, :Idist, :Ddist, :Redge, :Iedge, :Dedge
  def initialize()
    @Rdist = 0
    @Idist = 0
    @Ddist = 0
    @Redge = " "
    @Iedge = " "
    @Dedge = " "
  end
end

def getcost(x,y)
  if x != '' and y != '' and x != y then
    1  #replacement-mismatch
  elsif x != '' and y != '' and x == y then
    0  #replacement-match
  end
end

if ARGV.length != 4 then
  STDERR.puts "Usage: #{$0} <seq1> <seq2> <gap_open_cost> <gap_extension_cost>"
  exit(-1)
end
u = ARGV[0]
ulen = ARGV[0].length
v = ARGV[1]
vlen = ARGV[1].length
gap_open_cost = ARGV[2].to_i
gap_extension_cost = ARGV[3].to_i

dptable = Array.new(ulen+1) { Array.new(vlen+1) { AffineDPentry.new } }

def fillDPtable(dptable, u, v, ulen, vlen, gap_open, gap_extension)
  0.upto(ulen) do |i|
    0.upto(vlen) do |j|
      if i == 0 and j == 0 then
        dptable[i][j].Rdist = 0
        dptable[i][j].Idist = gap_open
        dptable[i][j].Ddist = gap_open
      else
        # compute A_affine(i,j,R)
        if i == 0 or j == 0 then
          dptable[i][j].Rdist = Infinity
        else
          rval = dptable[i-1][j-1].Rdist + getcost(u[i-1].chr, v[j-1].chr)
          dval = dptable[i-1][j-1].Ddist + getcost(u[i-1].chr, v[j-1].chr)
          ival = dptable[i-1][j-1].Idist + getcost(u[i-1].chr, v[j-1].chr)
          dptable[i][j].Rdist = [rval, dval, ival].min
          if rval == dptable[i][j].Rdist then
            dptable[i][j].Redge = :R;
```

```
          elsif dval == dptable[i][j].Rdist then
            dptable[i][j].Redge = :D;
          else
            dptable[i][j].Redge = :I;
          end
        end
        # compute A_affine(i,j,D)
        if i == 0 then
          dptable[i][j].Ddist = Infinity
        else
          rval = dptable[i-1][j].Rdist + gap_open + gap_extension
          dval = dptable[i-1][j].Ddist + gap_extension
          ival = dptable[i-1][j].Idist + gap_open + gap_extension
          dptable[i][j].Ddist = [rval, dval, ival].min
          if rval == dptable[i][j].Ddist then
            dptable[i][j].Dedge = :R;
          elsif dval == dptable[i][j].Ddist then
            dptable[i][j].Dedge = :D;
          else
            dptable[i][j].Dedge = :I;
          end
        end
        # compute A_affine(i,j,I)
        if j == 0 then
          dptable[i][j].Idist = Infinity
        else
          rval = dptable[i][j-1].Rdist + gap_open + gap_extension
          dval = dptable[i][j-1].Ddist + gap_open + gap_extension
          ival = dptable[i][j-1].Idist + gap_extension
          dptable[i][j].Idist = [rval, dval, ival].min
          if rval == dptable[i][j].Idist then
            dptable[i][j].Iedge = :R;
          elsif dval == dptable[i][j].Idist then
            dptable[i][j].Iedge = :D;
          else
            dptable[i][j].Iedge = :I;
          end
        end
      end
    end
  end
end

def traceback(dptable, alignment, i, j)
  # get starting edge to follow
  minval = [dptable[i][j].Rdist, dptable[i][j].Ddist,
            dptable[i][j].Idist].min
  if minval == dptable[i][j].Rdist then
    edge = :R
  end
  if minval == dptable[i][j].Ddist then
    edge = :D
  end
  if minval == dptable[i][j].Idist then
    edge = :I
  end
```

```
  # traceback to (0,0)
  while i > 0 or j > 0 do
    case edge
      when :R
        alignment.add_replacement()
        edge = dptable[i][j].Redge
        i -= 1
        j -= 1
      when :D
        alignment.add_deletion()
        edge = dptable[i][j].Dedge
        i -= 1
      when :I
        alignment.add_insertion()
        edge = dptable[i][j].Iedge
        j -= 1
    end
  end
end

fillDPtable(dptable, u, v, ulen, vlen, gap_open_cost, gap_extension_cost)
a = Alignment.new(u, v, ulen, vlen)
traceback(dptable, a, ulen, vlen)
a.reverse!
a.show_alignment
#puts "alignment cost: #{[dptable[ulen][vlen].Rdist, dptable[ulen][vlen].Ddist,
#                         dptable[ulen][vlen].Idist].min}"
```

Lösung zu Aufgabe 9.2:

Sei $u = acgacgtag$ und $v = ggacgtgcag$. Dann gilt $edist_\delta(u, v) = 4$ und es ergibt sich das folgende optimale Alignment:

```
acgacgt--ag
g-gacgtgcag
```

Die Qwortdistanz für $q = 2$ ist 7 und für $q = 3$ ist sie 9.

Wir definieren nun eine Abbildung $\zeta : \mathcal{A} \to \{r, y\}$ durch $\zeta(a) = \zeta(g) = r$ und $\zeta(c) = \zeta(t) = y$. $\zeta(u)$ sei die Sequenz, die sich ergibt, wenn man ζ auf die einzelnen Buchstaben von u anwendet. Sei nun $u' = \zeta(u)$ und $v' = \zeta(v)$. Dann ergibt sich $edist_\delta(u', v') = 2$ und das folgende optimale Alignment:

```
ryr-ryryrr
rrryryryrr
```

Die Qwortdistanz für $q = 2$ ist 1 und für $q = 3$ ist sie 3.

Wir wählen u, v, u', und v' wie oben. Dann gilt:

$$
\begin{aligned}
qgdist_2(u,v) &= 7 > edist_\delta(u,v) \\
qgdist_3(u,v) &= 9 \\
qgdist_2(u',v') &= 1 < edist_\delta(u',v') \\
qgdist_3(u',v') &= 3
\end{aligned}
$$

Außerdem gilt:

$$qgdist_{10}(gtatagataaat, cgctgataataa) = edist_\delta(gtatagataaat, cgctgataataa) = 6$$