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

Grundlagen der Sequenzanalyse
Wintersemester 2016/2017
Übungen zur Vorlesung: Ausgabe am 22.11.2016

Punkteverteilung: Aufgabe 6.1: 3 Punkte, Aufgabe 6.2: 5 Punkte, Aufgabe 6.3: 2 Punkte

Abgabe bis zum 28.11.

Aufgabe 6.1 Erweitern Sie die in einer früheren Aufgabe implementierte Edit-Matrix so, dass jeder Eintrag der berechneten Matrix nicht nur den entsprechenden Distanzwert, sondern auch eine Repräsentation der eingehenden minimierenden Kante(n) speichert. In der Vorlesung wurde gezeigt, wie man diese Information in 3 Bit speichert. Sie können jedoch auch drei boolsche Werte verwenden.

Aufgabe 6.2 In dieser Aufgabe geht es um die Repräsentation von Alignments durch Multi-Editoperationen:

Laut Definition ist ein Alignment eine Folge von Editoperationen. Wir wollen ein Alignment kompakter repräsentieren, in dem wir Multi-Editoperationen verwenden. Wir haben also weiterhin drei verschiedene Typen von Editoperationen. Jede Editoperation wird jedoch mit einer positiven ganzen Zahl i kombiniert, welche die i-malige Anwendung der entsprechenden Editoperation spezifiziert. Das folgende Alignment läßt sich also durch die rechte Liste und die beiden Sequenzen darstellen:

```
acgtaga--tatata-gat
|   |||  ||  | | |                    [R 7,I 2,R 2,D 1,R 3,I 1,R 3]
agaaagaggta-agaggga
```

Dabei wird jede Editoperation durch ihren ersten Buchstaben abgekürzt (Replacement $\mapsto$ `R`, Deletion $\mapsto$ `D`, Insertion $\mapsto$ `I`). Zu einem Alignment gehören nun noch die beiden alignierten Sequenzen sowie die Anzahl der Multi-Editoperationen.

Es soll nun ein strukturierter Datentyp (z.B. eine Struktur in C oder eine Klasse in einer objektorientierten Programmiersprache) implementiert werden. Der strukturierte Datentyp für Multi-Editoperationen soll folgendes leisten:

Speicherung eines Alignments bestehend aus Multi-Editoperationen, den Sequenzen des Alignments und die jeweiligen Sequenz- und Alignment-Längen. Falls Sie ihren Programmcode in C schreiben und die dynamische Speicherverwaltung noch nicht beherrschen, ist es akzeptabel, wenn im Rahmen der Übung eine feste maximale Anzahl von Multi-Editoperationen gewählt wird (z.B. 1000).

Implementieren Sie bitte die folgenden Funktionen bzw. Methoden:

- `alignment_new`: Liefert ein neues leeres Alignment, wobei Verweise auf die alignierten Sequenzen mit angegeben werden sollen. Gegebenenfalls muss hier natürlich auch der Speicher für die Multi-Editoperationen initialisiert werden.

- `alignment_add_replacement`, `alignment_add_deletion`, `alignment_add_insertion`: Fügt eine entsprechende Editoperation zum Alignment hinzu.
- `alignment_show`: Gibt das Alignment aus, so wie im Beispiel oben links.
- `alignment_evalcost`: Berechnet die Kosten eines Alignments für eine gegebene Kostenfunktion (z.B. Einheitskostenfunktion).
- `alignment_delete`: Gibt ggf. für das Alignment reservierten Speicher wieder frei. (nicht jede Sprache erlaubt oder benötigt dies)

Diese Funktionen/Methoden sollen idealerweise für Aufgaben auf dem nächsten Übungsblatt wiederverwendbar sein. Der Programmcode, den Sie abliefern sollte ihre Datenstruktur mit einigen Beispieldaten (z.B. dem obigen Beispiel) testen. Es sollte aus ihrer Abgabe klar werden, dass Sie ihren Code verwendet haben und er funktioniert.

Aufgabe 6.3 Der bekannte DP-Algorithmus zum globalen Alignment kann leicht wie folgt modifiziert werden:

1. statt Kosten werden Scores verwendet, die auch negativ sein können.
2. statt zu minimieren wird in der Rekurrenz maximiert.

Der so entstandene Algorithmus wird auch als Needleman-Wunsch-Algorithmus bezeichnet (NW), auch wenn der Original-NW-Algorithmus mit allgemeinen Gap-Kosten arbeitet (siehe Vorlesungsskript, Abschnitt 3.8.1).

Sei nun folgende Scorefunktion σ gegeben:

$$\sigma(\alpha \to \beta) = \begin{cases} 2 & \text{falls } \alpha, \beta \in \mathcal{A} \text{ und } \alpha = \beta \\ -4 & \text{falls } \alpha, \beta \in \mathcal{A} \text{ und } \alpha \neq \beta \\ -5 & \text{falls } \alpha = \varepsilon \text{ oder } \beta = \varepsilon \end{cases}$$

Ein Match wird hier also mit Score 2 bewertet, ein Mismatch mit -4 und ein Indel mit -5.

Betrachten Sie nun die folgenden Ausschnitte aus Matrizen optimaler Alignment-Scores, die beim Smith-Waterman (SW), bzw. NW-Algorithmus gefüllt werden:

1)

		$v[j]$	
		$\vdots$	
		2	4
$u[i]$	$\dots$	0	

2)

		$v[j]$	
		$\vdots$	
		0	8
$u[i]$	$\dots$	2	

3)

		$v[j]$	
		$\vdots$	
		2	5
$u[i]$	$\dots$	4	

Welche Scores erhalten die leeren Felder in den folgenden vier Fällen?

(a) $u[i] \neq v[j]$, NW

(b) $u[i] = v[j]$, NW

(c) $u[i] \neq v[j]$, SW

(d) $u[i] = v[j]$, SW

Bitte die Lösungen zu diesen Aufgaben bis zum 28.11.2016 abgeben. Die Besprechung der Lösungen erfolgt am 29.11.2016.

Lösung zu Aufgabe 6.1:

Lösung in Ruby

```ruby
# Dabei speichert distvalue den Distanz-Wert. Die Variablen replacement_is_min,
# deletion_is_min und insertion_is_min zeigen die eingehenden Kanten an: Jede
# Kante, welche zu einem minimierenden Pfad f"uhrt, erh"alt als Wert true.
class DPentry
  attr_reader :distvalue,
    :replacement_is_min,
    :deletion_is_min,
    :insertion_is_min
  def initialize(dist = 0, rp_is_min = false, d_is_min = false,
                 i_is_min = false)
    @distvalue = dist
    @replacement_is_min = rp_is_min
    @deletion_is_min    = d_is_min
    @insertion_is_min   = i_is_min
    @on_min = false
  end

end

def filldptable(u,ulen,v,vlen,indelcost)
  dptable = Array.new(ulen+1) { Array.new(vlen+1) {nil} }
  dptable[0][0] = DPentry.new()

  1.upto(vlen) do |j|
    dptable[0][j] = DPentry.new(dptable[0][j-1].distvalue + indelcost,
                                false, false, true)
  end

  1.upto(ulen) do |i|
    dptable[i][0] = DPentry.new(dptable[i-1][0].distvalue + indelcost,
                                false, true, false)
    1.upto(vlen) do |j|
      ins = dptable[i][j-1].distvalue + indelcost
      del = dptable[i-1][j].distvalue + indelcost
      if u[i-1].chr == v[j-1].chr
        rep = dptable[i-1][j-1].distvalue
      else
        rep = dptable[i-1][j-1].distvalue + 1
      end
      dist = [rep, del, ins].min
      dptable[i][j] = DPentry.new(dist,
                                  rep == dist,
                                  del == dist,
                                  ins == dist)
    end
  end
```

Lösung in C

Header

edist_table.h

```
#ifndef EDIST_TABLE_H
```

```
#define EDIST_TABLE_H

#include <stdbool.h>

/* DP-Matrix entry
Dabei speichert \lstinline!distvalue! den Distanz-Wert. Die Variablen
\lstinline!min_replacement!, \lstinline!min_deletion! und
\lstinline!min_insertion! zeigen die eingehenden Kanten an: Jede Kante,
welche zu einem minimierenden Pfad f"uhrt, erh"alt als Wert \lstinline!true!.
*/
typedef struct
{
  unsigned long distvalue;
  bool min_replacement,
       min_deletion,
       min_insertion;
} DPentry;

/* fill <dptable> of size (<ulen> + 1) x (<vlen> + 1) with values of the edit
   distance (unit cost) of <u> and <v>. <dptable> must be initialized to 0
   (and/or false). */
void fillDPtable(DPentry **dptable,
                 const char *u, unsigned long ulen,
                 const char *v, unsigned long vlen);

#endif
```

Code

edist_table.c

```
#include <stdlib.h>
#include <assert.h>
#include "edist_table.h"

#define MIN(X,Y) (((X) < (Y)) ? (X) : (Y))

void fillDPtable(DPentry **dptable,
                 const char *u, unsigned long ulen,
                 const char *v, unsigned long vlen)
{
  unsigned long i, j,
                repvalue, delvalue, insvalue, minvalue;

  assert(dptable != NULL &&
         u != NULL &&
         ulen != 0 &&
         v != NULL &&
         vlen != 0);

  for (i = 1; i <= ulen; i++) {
    dptable[i][0].distvalue = i;
    dptable[i][0].min_deletion = true;
  }
  for (j = 1; j <= vlen; j++) {
```

```
    dptable[0][j].distvalue = j;
    dptable[0][j].min_insertion = true;

    for (i = 1; i <= ulen; i++) {
      repvalue = dptable[i-1][j-1].distvalue + ((u[i-1] == v[j-1]) ? 0 : 1);
      delvalue = dptable[i-1][j].distvalue + 1;
      insvalue = dptable[i][j-1].distvalue + 1;

      minvalue = MIN(MIN(repvalue, delvalue), insvalue);

      dptable[i][j].distvalue = minvalue;
      dptable[i][j].min_replacement = minvalue == repvalue;
      dptable[i][j].min_deletion    = minvalue == delvalue;
      dptable[i][j].min_insertion   = minvalue == insvalue;
    }
  }
}
```

Programm

edist.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "edist_table.h"

/* Allocates a new 2-dimensional array with dimensions
   <ROWS> x <COLS> and assigns a pointer to the newly
   allocated space to <ARR2DIM>. Size of each element is
   determined from type of the <ARR2DIM> pointer. */
#define array2dim_malloc(ARR2DIM, ROWS, COLS)\
        {\
          unsigned long idx;\
          ARR2DIM = malloc(sizeof (*ARR2DIM) * (ROWS));\
          (ARR2DIM)[0]\
            = malloc(sizeof (**ARR2DIM) * (ROWS) * (COLS));\
          for (idx = 1UL; idx < (ROWS); idx++) \
            (ARR2DIM)[idx] = (ARR2DIM)[idx-1] + (COLS);\
        }

#define array2dim_delete(ARR2DIM)\
        if ((ARR2DIM) != NULL)\
        {\
          free((ARR2DIM)[0]);\
          free(ARR2DIM);\
        }

void usage(int error, char *progname)
{
  fprintf(stderr, "%s <u> <v>\n", progname);
  exit(error);
}
```

```c
int main(int argc, char *argv[])
{
  DPentry **dp_table = NULL;
  char *u, *v;
  unsigned int u_len, v_len;
  if (argc != 3)
    usage(0, argv[0]);

  u = argv[1];
  v = argv[2];
  u_len = strlen(u);
  v_len = strlen(v);

  array2dim_malloc(dp_table, u_len + 1, v_len + 1);


  fillDPtable(dp_table, u, u_len, v,  v_len);

  printf("%s\t%s\t%lu\n", u, v, dp_table[u_len][v_len].distvalue);
  return 0;
}
```

Lösung zu Aufgabe 6.2:

Siehe dazu die Klasse `alignment.[ch]` von *GenomeTools* (`http://genometools.org`).

Header

```c
#ifndef ALIGNTYPE_H
#define ALIGNTYPE_H

typedef struct Alignment Alignment;
typedef struct Multieop Multieop;

/* returns a new object of type Alignment, <seq1> and <seq2> will be used for
   alignment show, and there determines if a replacement is a match or mismatch.
   Both sequences should be long enough for the alignment. */
Alignment*    alignment_new(const char *seq1,
                            const char *seq2) ;

/* adds corresponding edit operation to <algn> */
void          alignment_add_replacement(Alignment *algn);
void          alignment_add_deletion(Alignment *algn);
void          alignment_add_insertion(Alignment *algn);

/* prints a readable representation of <algn> to stdout,
   <width> = 0 means no linebreaking, otherwise the alignment is wrapped at
   width */
void          alignment_show(const Alignment *algn,
                             unsigned int width);

/* returns the cost of <algn> assuming unit cost function */
unsigned long alignment_evalcost(const Alignment *algn);

/* frees all space of <algn> */
```

```
void          alignment_delete(Alignment *algn);

#endif /* ALIGNTYPE_H */
```

code

```
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <stdio.h>
#include <stdbool.h>
#include "aligntype.h"

typedef enum
{
  Replacement = 1,
  Deletion,
  Insertion
} Eoptype;

struct Multieop
{
  Eoptype eop;       /* the edit operation */
  unsigned int steps;/* number of multiples of eop */
};

struct Alignment
{
  Multieop *multieoplist;
  char *seq1,
       *seq2;
  unsigned long seq1len,
                seq2len,
                numofmultieops,
                multieopssize;
};

Alignment *alignment_new(const char *seq1, const char *seq2)
{
  Alignment *algn = calloc((size_t) 1, sizeof(*algn));

  algn->seq1 = strdup(seq1);
  algn->seq2 = strdup(seq2);
  algn->seq1len = strlen(seq1);
  algn->seq2len = strlen(seq2);

  return algn;
}

/* 0 -> 20 -> 60 -> 140 -> 300 ... */
#define RESIZE(X) (((X) + 10) << 1)

static void alignment_add_editop(Alignment *algn, Eoptype eop)
{
  if(algn->numofmultieops != 0 &&
     algn->multieoplist[algn->numofmultieops - 1].eop == eop) {
```

```
    algn->multieoplist[algn->numofmultieops - 1].steps++;
    assert(algn->multieoplist[algn->numofmultieops - 1].steps != 0);
  }
  else {
    if(algn->numofmultieops == algn->multieopssize) {
      algn->multieopssize = RESIZE(algn->multieopssize);
      algn->multieoplist =
        realloc(algn->multieoplist,
                algn->multieopssize * sizeof(*algn->multieoplist));

    }
    algn->multieoplist[algn->numofmultieops].eop = eop;
    algn->multieoplist[algn->numofmultieops].steps = 1;
    algn->numofmultieops++;
  }
}

void alignment_add_replacement(Alignment *algn)
{
  alignment_add_editop(algn, Replacement);
}

void alignment_add_deletion(Alignment *algn)
{
  alignment_add_editop(algn, Deletion);
}

void alignment_add_insertion(Alignment *algn)
{
  alignment_add_editop(algn, Insertion);
}

#define LINEWIDTH 80
struct Alignment_lines
{
  char *seq1line,
       *seq2line,
       *middline;
  unsigned long linepos, length;
  bool breaking;
};

static struct Alignment_lines *alignment_lines_new(unsigned int width)
{
  struct Alignment_lines *lines = calloc((size_t) 1, sizeof(*lines));

  if (width == 0) {
    lines->length = LINEWIDTH;
    lines->breaking = false;
  }
  else {
    lines->length = width;
    lines->breaking = true;
  }
  lines->seq1line = calloc(lines->length + 1, sizeof(*lines->seq1line));
  lines->seq2line = calloc(lines->length + 1, sizeof(*lines->seq2line));
```

```
  lines->middline = calloc(lines->length + 1, sizeof(*lines->middline));
  return lines;
}

static void alignment_lines_print(struct Alignment_lines *lines)
{
  lines->seq1line[lines->linepos] = '\0';
  lines->seq2line[lines->linepos] = '\0';
  lines->middline[lines->linepos] = '\0';
  printf("%s\n%s\n%s\n",
         lines->seq1line,
         lines->middline,
         lines->seq2line);
}

static void alignment_extend_lines(struct Alignment_lines *lines,
                                   char c1, char c2, char cm)
{
  lines->seq1line[lines->linepos] = c1;
  lines->seq2line[lines->linepos] = c2;
  lines->middline[lines->linepos] = cm;
  lines->linepos++;
  if (lines->linepos == lines->length) {
    if (lines->breaking) {
      alignment_lines_print(lines);
      lines->linepos = 0;
    }
    else {
      lines->length += LINEWIDTH;
      lines->seq1line = realloc(lines->seq1line, sizeof(*lines->seq1line) *
                                (lines->length + 1));
      lines->seq2line = realloc(lines->seq2line, sizeof(*lines->seq2line) *
                                (lines->length + 1));
      lines->middline = realloc(lines->middline, sizeof(*lines->middline) *
                                (lines->length + 1));
    }
  }
}

static void alignment_lines_delete(struct Alignment_lines *lines)
{
  if (lines) {
    free(lines->seq1line);
    free(lines->seq2line);
    free(lines->middline);
    free(lines);
  }
}

void alignment_show(const Alignment *algn, unsigned int width)
{
  unsigned long i, j,
                seq1idx = 0,
                seq2idx = 0;
  struct Alignment_lines *lines = alignment_lines_new(width);
```

```
  for (i=0; i < algn->numofmultieops; i++) {
    if (algn->multieoplist[i].eop == Replacement) {
      for (j = 0; j<algn->multieoplist[i].steps; j++) {
        char bar_or_space =
          algn->seq1[seq1idx] != algn->seq2[seq2idx] ?  ' ' : '|';
        alignment_extend_lines(lines,
                               algn->seq1[seq1idx++],
                               algn->seq2[seq2idx++],
                               bar_or_space);
        assert(seq1idx <= algn->seq1len);
        assert(seq2idx <= algn->seq2len);
      }
    }
    if (algn->multieoplist[i].eop == Deletion) {
      for (j = 0; j < algn->multieoplist[i].steps; j++) {
        alignment_extend_lines(lines,
                               algn->seq1[seq1idx++],
                               '-', ' ');
        assert(seq1idx <= algn->seq1len);
      }
    }
    if (algn->multieoplist[i].eop == Insertion)
    {
      for (j = 0; j < algn->multieoplist[i].steps; j++)
      {
        alignment_extend_lines(lines,
                               '-',
                               algn->seq2[seq2idx++],
                               ' ');
        assert(seq2idx <= algn->seq2len);
      }
    }
  }
  alignment_lines_print(lines);
  alignment_lines_delete(lines);
}

void alignment_delete(Alignment *algn)
{
  if (algn != NULL) {
    free(algn->multieoplist);
    free(algn->seq1);
    free(algn->seq2);
    free(algn);
  }
}

unsigned long alignment_evalcost(const Alignment *algn)
{
  unsigned long i, j,
                seq1idx = 0,
                seq2idx = 0,
                sumcost = 0;

  for(i=0; i < algn->numofmultieops; i++) {
    if(algn->multieoplist[i].eop == Replacement) {
```

```
      for(j=0; j<algn->multieoplist[i].steps; j++) {
        if(algn->seq1[seq1idx] != algn->seq2[seq2idx])
          sumcost++;
        seq1idx++;
        seq2idx++;
        assert(seq1idx <= algn->seq1len);
        assert(seq2idx <= algn->seq2len);
      }
    }
    else {
      if(algn->multieoplist[i].eop == Deletion) {
        for(j=0; j<algn->multieoplist[i].steps; j++) {
          sumcost++;
          seq1idx++;
          assert(seq1idx <= algn->seq1len);
        }
      }
      else {
        for(j=0; j<algn->multieoplist[i].steps; j++) {
          sumcost++;
          seq2idx++;
          assert(seq2idx <= algn->seq2len);
        }
      }
    }
  }
  return sumcost;
}
```

Ruby

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

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 $0 == __FILE__
  # 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
```

Lösung zu Aufgabe 6.3:

1)				2)				3)			
(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)	(a)	(b)	(c)	(d)
-1	4	0	4	3	3	3	3	0	4	0	4