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

Grundlagen der Sequenzanalyse
Wintersemester 2016/2017
Übungen zur Vorlesung: Ausgabe am 29.11.2016

Punkteverteilung: Aufgabe 7.1: 4 Punkte, Aufgabe 7.2: 6 Punkte

Abgabe bis zum 5.12.

Aufgabe 7.1 In der letzten Übung haben Sie ein Programm zur Berechnung der Edit-Distanz zweier Sequenzen geschrieben, welches zusätzlich die minimierenden Kanten für jede Zelle der Edit-Matrix speichert. Implementieren Sie nun eine Funktion

```
traceback(dpmatrix, alignment, i, j)
```

die als Parameter eine Referenz auf die DP-Matrix sowie eine Referenz auf eine Alignment-Datenstruktur erhält. Die Parameter `i` und `j` spezifizieren den "Endpunkt" für das Alignment in der Matrix. Die Funktion `traceback` soll ausgehend vom Knoten (i, j) im Editgraphen die minimierenden Pfade rückwärts verfolgen, bis der Knoten $(0, 0)$ erreicht wird. Dabei wird das optimale Alignment der alignierten Sequenzen rückwärts konstruiert.

Sie müssen nicht alle optimalen Alignments berechnen. Es reicht ein optimales Alignment. Falls Sie bei der Auswahl minimierender Kanten mehrere Möglichkeiten haben, dann bevorzugen Sie Ersetzungskanten, dann Löschkanten und schließlich Einfügekanten.

Verwenden Sie Ihre `alignment_evalcost`-Funktion, um zu überprüfen, ob die Kosten des Alignments, welches Ihre `traceback`-Funktion berechnet, mit der Edit-Distanz übereinstimmen. Dabei können Sie die Einheitskostenfunktion mit linearen Gap-Kosten verwenden, um Edit-Operationen zu bewerten.

Benutzen Sie die Funktion `traceback` schließlich in einem Programm, das zwei Sequenzen als Kommandozeilenparameter übergeben bekommt und das entsprechende optimale Alignment bzgl. der Einheitskostenfunktion in folgendem Format ausgibt:

```
acacbb
|  |  ||
a-aabb
```

Dazu bietet sich die Funktion `alignment_show` Ihrer Alignment-Datenstruktur an.

In STiNE finden Sie Material um Ihren Code zu testen. Die Datei `check.rb` können Sie wie folgt aufrufen

```
./check.rb <MeinProgram> aligment-testcases.txt
```

um Ihr Programm `<MeinProgram>` mit 1000 verschiedenen Sequenzpaaren aufzurufen. Vergleichen Sie z.B. mit `diff`, dass die Ausgabe Ihres Programms mit dem Inhalt der Datei `aligment-testcases_out.txt` identisch ist.

Aufgabe 7.2 Ein wichtiges Problem in der Massenspektrometrie besteht darin, alle Proteinsequenzen in einer Datenbank zu finden, die ein bestimmtes Molekulargewicht haben. Formal lässt sich dieses Problem wie folgt formulieren:

Sei $\mathcal{A}$ ein Alphabet und $\sigma : \mathcal{A} \to \mathbb{N} \setminus \{0\}$ eine Gewichtsfunktion, die jedem Symbol aus $\mathcal{A}$ eine positive natürliche Zahl zuordnet. Für alle Sequenzen $u \in \mathcal{A}^*$ definieren wir das *Gewicht* $\sigma(u) = \sum_{i=1}^{|u|} \sigma(u[i])$.

Sei $S \in \mathcal{A}^n$ eine Sequenz der Länge n und $w > 0$ ein vorgegebenes Gewicht. Das *gewichtete Substring-Matching Problem* besteht darin, die folgende Menge zu bestimmen:

$$Sol(\sigma, S, w) = \{(j, l) \mid 1 \leq j \leq n, 1 \leq l \leq n + 1 - j, \sigma\left(S[j \ldots j + l - 1]\right) = w\}$$

d.h. die Menge aller Substrings in S (repräsentiert durch eine Startposition j und die Länge l), deren Gewicht genau w ist.

Beispiel: Sei $\mathcal{A} = \{a, c, g, t\}$ das DNA-Alphabet, sei $\sigma(a) = 1$, $\sigma(c) = \sigma(t) = 2$ sowie $\sigma(g) = 3$. Für $S = acgtagctc$ ist dann

$$Sol(\sigma, S, 9) = \{(1, 5), (3, 4), (6, 4)\}$$

Diese Menge repräsentiert die Substrings $acgta$, $gtag$ und $gctc$ und es gilt $\sigma(acgta) = \sigma(gtag) = \sigma(gctc) = 9$.

Entwickeln und formulieren Sie einen effizienten Algorithmus, um das gewichtete Substring-Matching Problem zu lösen. Dabei sollten Sie ausnutzen, dass die Gewichte immer positive ganze Zahlen sind. Welche Laufzeit hat Ihr Algorithmus? Verwenden Sie Ihren Algorithmus in einem Programm, das die oben genannten Gewichte für Zeichen aus dem DNA-Alphabet verwendet. Alle anderen Zeichen sollen zu einer Fehlermeldung führen und das Programm beenden. Das Programm soll als Komandozeilenparameter eine Sequenz S und ein Gewicht w übergeben bekommen. Jeder Treffer soll im Format `(j,l)` auf einer eigenen Zeile ausgegeben werden.

Sie finden im Material zu dieser Übung eine Datei `testdata.txt`, in der zeilenweise Sequenzen S mit einem Gewicht w angegeben sind. In der Datei `testdata.out` finden Sie zu jeder dieser Sequenzen die entsprechenden Lösungen des gewichteten Substring-Matching Problems. Das Shell-Skript `runtest.sh` verlangt als ersten Parameter den Namen Ihres ausführbaren Programms und wendet es auf die Sequenzen und Gewichte in `testdata.txt` an. Ihr Programm funktioniert korrekt, wenn `runtest.sh` mit Ihrem Programm die Lösungen aus `testdata.out` liefert, was mit `diff` leicht überprüft werden kann.

Bitte die Lösungen zu diesen Aufgaben bis zum 05.12.2016 abgeben. Die Besprechung der Lösungen erfolgt am 06.12.2016.

Lösung zu Aufgabe 7.1:

Ruby Lösung:

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

$:.unshift File.join(File.dirname(__FILE__))

require 'editgraph'
require 'alignment'
```

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

def tracebackone(matrix, alignment, i, j)
  if i == 0 or j == 0
    if i > 0
      i.downto(1) do
        alignment.add_deletion()
      end
    end
    if j > 0
      j.downto(1) do
        alignment.add_insertion()
      end
    end
    return
  elsif matrix[i][j].replacement_is_min
    tracebackone(matrix, alignment, i-1, j-1)
    alignment.add_replacement()
  elsif matrix[i][j].deletion_is_min
    tracebackone(matrix, alignment, i-1, j)
    alignment.add_deletion()
  elsif matrix[i][j].insertion_is_min
    tracebackone(matrix, alignment, i, j-1)
    alignment.add_insertion()
  end
end

indelcost = 1
dptable = filldptable(u,ulen,v,vlen,indelcost)

a = Alignment.new(u, v, ulen, vlen)
tracebackone(dptable, a, ulen, vlen)
a.show_alignment
cost = a.eval_alignment()
if cost != dptable[ulen][vlen].distvalue
  STDERR.puts "alignment is wrong! should cost: " \
    "#{dptable[ulen][vlen].distvalue}, is: #{cost}"
  exit 1
end

# 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

  def mark_on_min
    @on_min = true
  end

  def is_min?
    return @on_min
  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
  return dptable
end

def tracebackall(dptable, i, j)
  if i == 0 or j == 0
    if i > 0
      i.downto(1) do
        dptable[i][j].mark_on_min
      end
    end
    if j > 0
      j.downto(1) do
        dptable[i][j].mark_on_min
```

```
      end
    end
    return
  else
    dptable[i][j].mark_on_min
    if dptable[i][j].replacement_is_min
      tracebackall(dptable, i-1, j-1)
    end
    if dptable[i][j].deletion_is_min
      tracebackall(dptable, i-1, j)
    end
    if dptable[i][j].insertion_is_min
      tracebackall(dptable, i, j-1)
    end
  end
end

if $0 == __FILE__
  # TODO: adjust this!
  if ARGV.length != 2 then
    STDERR.puts "Usage: #{$0} <u> <v>"
    exit 1
  end
  u = ARGV[0]
  v = ARGV[1]

  m = Array.new(u.length+1) { Array.new(v.length+1) { nil } }
  EdistTable.fillDPtable(m, u, v)
  puts "#{u}\t#{v}\t#{m[u.length][v.length]}"
end

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

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
```

C Lösung:

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

/*
#define SHOWCALL              printf("call %s\n",__func__)
*/

#define SHOWCALL              /* Nothing */

#define MAXSEQUENCELENGTH     100     // maximal length of input sequences

/*
  The access to the entries in the DP-table is via the following
  macro.
*/

#define ACCESS(I,J) ((I) * (MAXSEQUENCELENGTH+1) + (J))

#define ALWIDTH               30      // width of alignment
#define TMPDISPLAYWIDTH       (2 * MAXSEQUENCELENGTH)

#define DISPLAYSIZE           (3 * 2 * MAXSEQUENCELENGTH +\
                               3 * (1+MAXSEQUENCELENGTH/ALWIDTH))

#define COPYTOOUTBUF(COMP)\
        memcpy(&display[outoffset],&tmpdisplay-> COMP ## _line[lineoffset],\
               sizeof(unsigned char) * width);\
        outoffset += width;\
        display[outoffset++] = (unsigned char) '\n'
```

```
/*
  Echo line of the alignment has length at most
  2 * \texttt{MAXSEQUENCELENGTH}. There are 3 lines to show.
  The number of line-locks is 1+\texttt{MAXSEQUENCELENGTH}/\texttt{ALWIDTH}.
  After each line we show one newline.
*/

#define DISPLAYRAW "RDI"

#define FILLTMPDISPLAY(UPP,MID,LOW)\
        if(tmpdisplay->columnnum >= TMPDISPLAYWIDTH)\
        {\
          fprintf(stderr,"access tmpdisplay at index %u >= %u\n",\
                   tmpdisplay->columnnum,\
                   TMPDISPLAYWIDTH);\
          exit(EXIT_FAILURE);\
        }\
        tmpdisplay->upper_line[tmpdisplay->columnnum] = (unsigned char) (UPP);\
        tmpdisplay->middle_line[tmpdisplay->columnnum] = (unsigned char) (MID);\
        tmpdisplay->lower_line[tmpdisplay->columnnum] = (unsigned char) (LOW)

#define ADDNEWEOP(EOP)\
        if(al->numofmultieops == 0)\
        {\
          al->multieoplist[0].eop = EOP;\
          al->multieoplist[0].steps = 1;\
          al->numofmultieops = 1;\
        } else\
        {\
          if(al->multieoplist[al->numofmultieops-1].eop == (EOP))\
          {\
            al->multieoplist[al->numofmultieops-1].steps++;\
          } else\
          {\
            al->multieoplist[al->numofmultieops].eop = EOP;\
            al->multieoplist[al->numofmultieops].steps = 1;\
            al->numofmultieops++;\
          }\
        }

#define SUBLASTEOP(EOP)\
        if(al->numofmultieops == 0)\
        {\
          fprintf(stderr,"program error, line %d: numofmultieops=0\n",\
                          __LINE__);\
          exit(EXIT_FAILURE);\
        }\
        if(al->multieoplist[al->numofmultieops-1].eop != (EOP))\
        {\
          fprintf(stderr,"program error, line %d: multieop[%u]=%c != %c\n",\
                          __LINE__,\
                          al->numofmultieops-1,\
                          DISPLAYRAW[al->multieoplist[\
                                     al->numofmultieops-1].eop],\
                          DISPLAYRAW[EOP]);\
```

```
          exit(EXIT_FAILURE);\
        }\
        if(al->multieoplist[al->numofmultieops-1].steps == 1)\
        {\
          al->numofmultieops--;\
        } else\
        {\
          al->multieoplist[al->numofmultieops-1].steps--;\
        }

#define READSEQ(IND,ARGC)\
        al.seq##IND##len = (unsigned int) strlen(argv[ARGC]);\
        if(al.seq##IND##len > MAXSEQUENCELENGTH)\
        {\
          fprintf(stderr,"%s sequence \"%s\" is too long, "\
                         " maximal length is %d\n",\
                         argv[0],argv[ARGC],MAXSEQUENCELENGTH);\
          return EXIT_FAILURE;\
        }\
        memcpy(al.seq##IND,argv[ARGC],sizeof(unsigned char) * al.seq##IND##len)

#define SETEDGES(I,J,RV,DV,IV)\
        dptable[ACCESS(I,J)].minreplacementedgein = RV;\
        dptable[ACCESS(I,J)].mindeletionedgein = DV;\
        dptable[ACCESS(I,J)].mininsertionedgein = IV

typedef enum
{
  Replacement,
  Deletion,
  Insertion
} Eoptype;

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

typedef struct
{
  unsigned char seq1[MAXSEQUENCELENGTH],     // first sequence
        seq2[MAXSEQUENCELENGTH];     // second sequence
  unsigned int seq1len,                       // length of first sequence
       seq2len,                      // length of second sequence
       numofmultieops;               // number of multistep editoperation
  Multieop multieoplist[2*MAXSEQUENCELENGTH];
} Alignment;

typedef struct
{
  unsigned int distvalue;
  bool minreplacementedgein,
       mindeletionedgein,
       mininsertionedgein;
} DPentry;
```

```
typedef struct
{
  unsigned int columnnum;
  unsigned char upper_line[TMPDISPLAYWIDTH],
        middle_line[TMPDISPLAYWIDTH],
        lower_line[TMPDISPLAYWIDTH];
} Tmpdisplay;

static void reversealignment(Multieop *multieoplist, unsigned int numofmultieops)
{
  unsigned int i, j;
  Multieop eop;

  SHOWCALL;
  for(i=0, j=numofmultieops-1; i<j; i++,j--)
  {
    eop = multieoplist[i];
    multieoplist[i] = multieoplist[j];
    multieoplist[j] = eop;
  }
}

void showmultieoplist(const Alignment *al)
{
  unsigned int i;

  SHOWCALL;
  printf("[");
  for(i=0; i < al->numofmultieops; i++)
  {
    printf("%c %u",DISPLAYRAW[(unsigned int) al->multieoplist[i].eop],
                   al->multieoplist[i].steps);
    if(i==al->numofmultieops-1)
    {
      printf("]\n");
    } else
    {
      printf(",");
    }
  }
}

void showmultieoplistreverse(const Alignment *al)
{
  unsigned int i = al->numofmultieops;

  SHOWCALL;
  i = al->numofmultieops-1;
  printf("[");
  while (true)
  {
    printf("%c %u",DISPLAYRAW[(unsigned int) al->multieoplist[i].eop],
                   al->multieoplist[i].steps);
    if(i==0)
    {
```

```
      printf("]\n");
      break;
    }
    printf(",");
    i--;
  }
}

static void filltmpalignment(Tmpdisplay *tmpdisplay,const Alignment *al)
{
  unsigned int i, j, i1 = 0, i2 = 0;

  SHOWCALL;
  tmpdisplay->columnnum = 0;
  for(i=0; i < al->numofmultieops; i++)
  {
    if(al->multieoplist[i].eop == Replacement)
    {
      for(j=0; j<al->multieoplist[i].steps; j++)
      {
        FILLTMPDISPLAY(al->seq1[i1],
                       (al->seq1[i1] != al->seq2[i2]) ?
                         '|' : ' ',
                       al->seq2[i2]);
        i1++;
        i2++;
        tmpdisplay->columnnum++;
      }
    } else
    {
      if(al->multieoplist[i].eop == Deletion)
      {
        for(j=0; j<al->multieoplist[i].steps; j++)
        {
          FILLTMPDISPLAY(al->seq1[i1],
                         '|',
                         '-');
          i1++;
          tmpdisplay->columnnum++;
        }
      } else
      {
        for(j=0; j<al->multieoplist[i].steps; j++)
        {
          FILLTMPDISPLAY('-',
                         '|',
                         al->seq2[i2]);
          i2++;
          tmpdisplay->columnnum++;
        }
      }
    }
  }
}

static void formattmpdisplay(Tmpdisplay *tmpdisplay)
```

```
{
  unsigned int width, outoffset = 0, lineoffset = 0,
               remain = tmpdisplay->columnnum;
  unsigned char display[DISPLAYSIZE];

  while(remain > 0)
  {
    if(remain > ALWIDTH)
    {
      width = ALWIDTH;
      remain -= ALWIDTH;
    } else
    {
      width = remain;
      remain = 0;
    }
    COPYTOOUTBUF(upper);
    COPYTOOUTBUF(middle);
    COPYTOOUTBUF(lower);
    display[outoffset++] = (unsigned char) '\n';
    lineoffset += width;
  }
  fwrite(&display[0],sizeof(unsigned char),outoffset,stdout);
}

static unsigned int evalalignment(unsigned int *indels,
                                  unsigned int *mismatches, const Alignment *al)
{
  unsigned int i, j, i1 = 0, i2 = 0, sumcost = 0;

  SHOWCALL;
  *indels = 0;
  *mismatches = 0;
  for(i=0; i < al->numofmultieops; i++)
  {
    if (al->multieoplist[i].eop == Replacement)
    {
      for(j=0; j<al->multieoplist[i].steps; j++)
      {
        if(al->seq1[i1] != al->seq2[i2])
        {
          (*mismatches)++;
          sumcost++;
        }
        i1++;
        i2++;
      }
    } else
    {
      (*indels) += al->multieoplist[i].steps;
      if(al->multieoplist[i].eop == Deletion)
      {
        for(j=0; j<al->multieoplist[i].steps; j++)
        {
          sumcost++;
          i1++;
```

```
        }
      } else
      {
        for(j=0; j<al->multieoplist[i].steps; j++)
        {
          sumcost++;
          i2++;
        }
      }
    }
  }
  return sumcost;
}

static void checkanddisplay(Alignment *al,unsigned int edist)
{
  Tmpdisplay tmpdisplay;
  Alignment tmpal;
  unsigned int evaledist, indels, mismatches;

  tmpal = *al;
  reversealignment(tmpal.multieoplist,tmpal.numofmultieops);
  evaledist = evalalignment(&indels,&mismatches,&tmpal);
  if(evaledist != edist)
  {
    fprintf(stderr,"evaledist = %u != %u = edist\n",evaledist,edist);
    exit(EXIT_FAILURE);
  }
  filltmpalignment(&tmpdisplay,&tmpal);
  formattmpdisplay(&tmpdisplay);
  printf("indels: %u, mismatches: %u\n",indels,mismatches);
}

static void fillDPtable(DPentry *dptable,const unsigned char *useq,unsigned int ulen,
                        const unsigned char *vseq,unsigned int vlen)
{
  unsigned int i, j, tmpvalue, minvalue;

  SHOWCALL;
  SETEDGES(0,0,false,false,false);
  dptable[ACCESS(0,0)].distvalue = 0;
  for(i=1; i <= ulen; i++)
  {
    dptable[ACCESS(i,0)].distvalue = i;
    SETEDGES(i,0,false,true,false);
  }
  for(j=1; j <= vlen; j++)
  {
    dptable[ACCESS(0,j)].distvalue = j;
    SETEDGES(0,j,false,false,true);
    for(i=1; i <= ulen; i++)
    {
      if(useq[i-1] == vseq[j-1])
      {
        minvalue = dptable[ACCESS(i-1,j-1)].distvalue;
      } else
```

```
      {
        minvalue = dptable[ACCESS(i-1,j-1)].distvalue + 1;
      }
      SETEDGES(i,j,true,false,false); // set rep edge
      tmpvalue = dptable[ACCESS(i-1,j)].distvalue + 1; // check del edge
      if(tmpvalue == minvalue)
      {
        dptable[ACCESS(i,j)].mindeletionedgein = true;
      } else
      {
        if(tmpvalue < minvalue)
        {
          SETEDGES(i,j,false,true,false);
          minvalue = tmpvalue;
        }
      }
      tmpvalue = dptable[ACCESS(i,j-1)].distvalue + 1;
      if(tmpvalue == minvalue)
      {
        dptable[ACCESS(i,j)].mininsertionedgein = true;
      } else
      {
        if(tmpvalue < minvalue)
        {
          SETEDGES(i,j,false,false,true);
          minvalue = tmpvalue;
        }
      }
      dptable[ACCESS(i,j)].distvalue = minvalue;
    }
  }
}

static void traceback(const DPentry *dptable,unsigned int edist,Alignment *al,
                      unsigned int i,unsigned int j)
{
  SHOWCALL;
  if(dptable[ACCESS(i,j)].minreplacementedgein)
  {
    ADDNEWEOP(Replacement);
    traceback(dptable,edist,al,i-1,j-1);
  } else
  {
    if(dptable[ACCESS(i,j)].mindeletionedgein)
    {
      ADDNEWEOP(Deletion);
      traceback(dptable,edist,al,i-1,j);
    } else
    {
      if(dptable[ACCESS(i,j)].mininsertionedgein)
      {
        ADDNEWEOP(Insertion);
        traceback(dptable,edist,al,i,j-1);
      } else
      {
        checkanddisplay(al,edist);
```

```
      }
    }
  }
}

static unsigned int tracebackall(const DPentry *dptable,
                                 unsigned int edist,
                                 /*@out@*/ Alignment *al,
                                 unsigned int i,
                                 unsigned int j)
{
  unsigned int countoptal = 0;
  bool backtrace = false;
  SHOWCALL;
  if(dptable[ACCESS(i,j)].minreplacementedgein)
  {
    backtrace = true;
    ADDNEWEOP(Replacement);
    countoptal += tracebackall(dptable,edist,al,i-1,j-1);
    SUBLASTEOP(Replacement);
  }
  if(dptable[ACCESS(i,j)].mindeletionedgein)
  {
    backtrace = true;
    ADDNEWEOP(Deletion);
    countoptal += tracebackall(dptable,edist,al,i-1,j);
    SUBLASTEOP(Deletion);
  }
  if(dptable[ACCESS(i,j)].mininsertionedgein)
  {
    backtrace = true;
    ADDNEWEOP(Insertion);
    countoptal += tracebackall(dptable,edist,al,i,j-1);
    SUBLASTEOP(Insertion);
  }
  if(!backtrace)
  {
    countoptal++;
    checkanddisplay(al,edist);
  }
  return countoptal;
}

int main(int argc,char *argv[])
{
  Alignment al;
  unsigned int countoptal = 0, edist;
  bool allalg = false;
  DPentry dptable[(MAXSEQUENCELENGTH+1) * (MAXSEQUENCELENGTH+1)];

  if(argc != 4)
  {
    fprintf(stderr,"Usage: %s [all|one] seq1 seq2\n",argv[0]);
    return EXIT_FAILURE;
  }
  if(strcmp(argv[1],"all") == 0)
```

```
  {
    allalg = true;
  } else
  {
    if(strcmp(argv[1],"one") == 0)
    {
      allalg = false;
    } else
    {
      fprintf(stderr,"%s: first argument must be \"all\" or \"one\"\n",
              argv[0]);
      return EXIT_FAILURE;
    }
  }
  READSEQ(1,2);
  READSEQ(2,3);
  fillDPtable(&dptable[0],al.seq1,al.seq1len,al.seq2,al.seq2len);
  edist = dptable[ACCESS(al.seq1len,al.seq2len)].distvalue;
  printf("edist=%u\n",edist);
  al.numofmultieops = 0;
  if(allalg)
  {
    countoptal = tracebackall(&dptable[0],edist,&al,al.seq1len,al.seq2len);
    printf("number of optimal alignments: %u\n",countoptal);
  } else
  {
    traceback(&dptable[0],edist,&al,al.seq1len,al.seq2len);
  }
  return EXIT_SUCCESS;
}
```

Lösung zu Aufgabe 7.2:

Effizientere Lösung, $O(|s|)$ in Ruby:

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

#define weight function
weights = {?a => 1, ?c => 2, ?g => 3, ?t => 2}
weights.default = 0

if ARGV.length != 2 then
  STDERR.puts "Usage: #{$0} <string> <weight>"
  exit 1
end
s = ARGV[0].downcase
begin
  w = Integer(ARGV[1])
rescue e
  STDERR.puts e
  STDERR.puts "Usage: #{$0} <string> <weight>"
  exit 1
end

puts "#{s}\t#{w}"
l, r, current_weight = 0, 0, 0
while r <= s.length do
```

```
  if current_weight < w              # enlarge window by adding next unread symbol
    if r < s.length and weights[s[r]] == 0
      STDERR.puts "Found unweighted symbol: #{s[r]}, this is an error."
      exit 1
    end
    current_weight += weights[s[r]] if r < s.length
    r += 1
  else
    if current_weight == w then      # query weight in window, save solution
      puts "(#{l + 1},#{r - l})"
    end
    current_weight -= weights[s[l]] # shrink window by discarding leftmost symbol
    l += 1
  end
end
```

Effizientere Lösung, $O(|s|)$ in C:

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

#include "int_conv.h"

const int *const get_weights() {
  static int weights[256] = {0};
  if (weights[(int) 'a'] == 0) {
    weights[(int) 'a'] =
      weights[(int) 'A'] = 1;
    weights[(int) 'c'] =
      weights[(int) 't'] =
      weights[(int) 'C'] =
      weights[(int) 'T'] = 2;
    weights[(int) 'g'] =
      weights[(int) 'G'] = 3;
  }
  return weights;
}

void usage(const char *progname)
{
  fprintf(stderr, "USAGE: %s <sequence> <weight>\n", progname);
  fprintf(stderr, "<sequence> is the \"data base\", <weight> the integer weigth"
          " > 0 of the substrings to search\n");
  exit(EXIT_FAILURE);
}

int main(int argc, char *argv[])
{
  int w;
  const int * const weights = get_weights();
  if (argc != 3)
    usage(argv[0]);

  w = get_int(argv[2]);

  if (w <= 0)
```

```
    usage(argv[0]);
  else {
    unsigned int r=0, l=0, len, win=0;
    len = strlen(argv[1]);
    printf("%s\t%d\n",argv[1],w);
    while(r <= len) {
      if (win < w) {
        if (r < len &&
            weights[(int) argv[1][r]] == 0) {
          fprintf(stderr, "Found unweighted symbol %c in %s, this is an error",
                  argv[1][r], argv[1]);
          exit(EXIT_FAILURE);
        }
        win += weights[(int) argv[1][r++]];
      }
      else {
        if (win == w) {
          printf("(%u,%u)\n", l + 1, r - l);
        }
        win -= weights[(int) argv[1][l++]];
      }
    }
  }
  return EXIT_SUCCESS;
}
```