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

Grundlagen der Sequenzanalyse
Wintersemester 2016/2017
Übungen zur Vorlesung: Ausgabe am 15.11.2016

Punkteverteilung: Aufgabe 5.1: 4 Punkte, Aufgabe 5.2: 3 Punkte, Aufgabe 5.3: 3 Punkte

Abgabe bis zum 21.11.

Aufgabe 5.1 Implementieren Sie den in der Vorlesung vorgestellten Algorithmus zur Lösung des Edit-Distanz-Problems für die Einheitskostenfunktion. Sie können dabei davon ausgehen, dass die Sequenzen maximal die Länge 100 haben, so dass die DP-Matrix, die den Distanzwert speichert, als statisches zweidimensionales Array implementiert werden kann.

Schreiben Sie dazu eine Funktion `fillDPtable`, welche die DP-Matrix E_δ für zwei Sequenzen u und v berechnet. Als Parameter erhält `fillDPtable` die Matrix, die Sequenzen u und v sowie (falls nötig) deren Längen $m = |u|$ und $n = |v|$. Als Kostenfunktionen können Sie die Einheitskostenfunktion verwenden. Beachten Sie, dass in der E_δ-Rekurrenz aus der Vorlesung auf die Strings 1-basiert zugegriffen wird, während in den gängigen Programmiersprachen (z.B. C und Ruby) Strings 0-basiert indiziert werden.

Benutzen Sie die Funktion `fillDPtable` schließlich in einem Programm, das zwei Sequenzen als Kommandozeilenparameter erhält und deren Edit-Distanz in folgendem Format ausgibt:

```
seq1 seq2 edist
```

Dabei sind `seq1` und `seq2` die beiden Sequenzen und `edist` die Editdistanz bzgl. der Einheitskostenfunktion. Diese drei Werte sind durch einen Tabulator getrennt.

Ihr Programm muss das (Ruby) Testskript `checkedist.rb` fehlerfrei durchlaufen, d.h. die Ergebnisse in `edist-testcases.tsv` müssen von Ihrem Programm reproduziert werden (Material siehe STiNE).

Aufgabe 5.2 Sei σ eine Score-Funktion. Wir definieren:

$$\mathrm{min}_\sigma(u,v) = \min\{\sigma(A) \mid A \text{ ist Alignment von } u \text{ und } v\}$$
$$\mathrm{max}_\sigma(u,v) = \max\{\sigma(A) \mid A \text{ ist Alignment von } u \text{ und } v\}$$

Sei $x, y \in \mathbb{R}$ und δ wie folgt definiert:

$$\delta(\alpha \to \beta) = \begin{cases} x \cdot \sigma(\alpha \to \beta) + y & \text{falls } \alpha \to \beta \text{ ein Indel ist} \\ x \cdot \sigma(\alpha \to \beta) + 2 \cdot y & \text{sonst} \end{cases}$$

Zeigen Sie, dass für alle Alignments A von u und v die folgenden Eigenschaften gelten:

1. Falls $x > 0$, dann ist $\delta(A) = \mathrm{max}_\delta(u,v) \iff \sigma(A) = \mathrm{max}_\sigma(u,v)$.
2. Falls $x < 0$, dann ist $\delta(A) = \mathrm{min}_\delta(u,v) \iff \sigma(A) = \mathrm{max}_\sigma(u,v)$.

Hier noch zwei Hinweise:

- Überlegen Sie, wie sich die Summe aller Scores eines Alignments in Scores für Indels und Ersetzungsoperationen aufspalten lässt.
- Sie können in Ihrem Beweis die Tatsache ausnutzen, dass für ein Alignment von zwei Sequenzen u und v mit i Indels und r Ersetzungsoperationen die Gleichung $|u|+|v| = i+2\cdot r$ gilt, siehe Lemma 2 des Vorlesungsskriptes.

Aufgabe 5.3 Seien u und v zwei Sequenzen der Längen m bzw. n. Sei s die Länge der längsten gemeinsamen Subsequenz von u und v.[1] Sei die Kostenfunktion δ für alle Editoperationen $\alpha \to \beta$ wie folgt definiert:

$$\delta(\alpha \to \beta) = \begin{cases} 0 & \text{if } \alpha = \beta \\ 1 & \text{if } \alpha = \varepsilon \text{ or } \beta = \varepsilon \\ \infty & \text{otherwise} \end{cases}$$

Zeigen Sie, dass die folgende Gleichung gilt:

$$s = \frac{m + n - edist_\delta(u, v)}{2} \tag{1}$$

Beispiel: Sei $u = FREIZEIT$ und $v = ZEITGEIST$. Dann ist $EIEIT$ die längste gemeinsame Subsequenz von u und v und daher ist $s = 5$. Weiterhin gilt $edist_\delta(u, v) = 7$, denn

```
F-REI-Z-EI-T
-Z-EIT-GEIST
```

ist ein optimales Alignment. Damit ist $\frac{m+n-edist_\delta(u,v)}{2} = \frac{8+9-7}{2} = \frac{17-7}{2} = 5 = s$.

Hinweis: Auch in diesem Beweis können Sie die Tatsache ausnutzen, dass für ein Alignment von zwei Sequenzen u und v mit i Indels und r Ersetzungsoperationen die Gleichung $|u|+|v| = i+2\cdot r$ gilt, siehe Lemma 2 des Vorlesungsskriptes.

Bitte die Lösungen zu diesen Aufgaben bis zum 21.11.2016 abgeben. Die Besprechung der Lösungen erfolgt am 22.11.2016.

Lösung zu Aufgabe 5.1:

Ruby solution

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

module EdistTable
  # Unit cost
  def EdistTable.delta(a,b)
    if a == b then
      0
    else
```

[1] Zur Erinnerung: Der Begriff der längsten gemeinsamen Subsequenz wurde in Aufgabe 2.3. definiert.

```
      1
    end
  end

  def EdistTable.fillDPtable(matrix, u, v)
    0.upto(u.length) do |i|
      0.upto(v.length) do |j|
        if i == 0 then
          matrix[i][j] = j
        elsif j == 0 then
          matrix[i][j] = i
        else
          newval = [matrix[i-1][j] + 1,
                    matrix[i][j-1] + 1,
                    matrix[i-1][j-1] + delta(u[i-1], v[j-1])].min
          matrix[i][j] = newval
        end
      end
    end
    matrix
  end

  private_class_method :delta
end

if $0 == __FILE__
  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) { 0 } }
  EdistTable.fillDPtable(m, u, v)
  puts "#{u}\t#{v}\t#{m[u.length][v.length]}"
end
```

C solution

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);
}

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 5.2:

In the following we want to sum scores of all indels and of all replacements in an alignment A. This will be denoted by

$$\sum_{\alpha \to \beta \in A \cap \mathsf{ID}} \cdots \quad \text{and} \quad \sum_{\alpha \to \beta \in A \cap \mathsf{R}} \cdots$$

As above, let id be the number of insertions and deletions, and r be the number of replacements in A. We have

$$\begin{aligned}
\delta(A) &= \sum_{\alpha \to \beta \in A} \delta(\alpha \to \beta) \\
&= \sum_{\alpha \to \beta \in A \cap \mathsf{ID}} \delta(\alpha \to \beta) + \sum_{\alpha \to \beta \in A \cap \mathsf{R}} \delta(\alpha \to \beta) \\
&= \sum_{\alpha \to \beta \in A \cap \mathsf{ID}} (x \cdot \sigma(\alpha \to \beta) + y) + \sum_{\alpha \to \beta \in A \cap \mathsf{R}} (x \cdot \sigma(\alpha \to \beta) + 2y) \\
&= id \cdot y + r \cdot 2y + \sum_{\alpha \to \beta \in A \cap \mathsf{ID}} x \cdot \sigma(\alpha \to \beta) + \sum_{\alpha \to \beta \in A \cap \mathsf{R}} x \cdot \sigma(\alpha \to \beta) \\
&= y \cdot (id + 2 \cdot r) + x \cdot \left(\sum_{\alpha \to \beta \in A \cap \mathsf{ID}} \sigma(\alpha \to \beta) + \sum_{\alpha \to \beta \in A \cap \mathsf{R}} \sigma(\alpha \to \beta) \right) \\
&= y \cdot (id + 2 \cdot r) + x \cdot \sum_{\alpha \to \beta \in A} \sigma(\alpha \to \beta) \\
&= y \cdot (m + n) + x \cdot \sigma(A)
\end{aligned}$$

Note that for deriving the last equality we have applied the Lemma labeled Relatesequence-length2editoperations from the section „The edit distance model“ in the GSA-Script (Lemma 2). Let $\mathcal{A}(u, v)$ denote the set of alignments of u and v.

1. Let $x > 0$ and $A' \in \mathcal{A}(u, v)$. Then

$$\begin{aligned}
\delta(A) \geq \delta(A') &\iff y \cdot (m + n) + x \cdot \sigma(A) \geq y \cdot (m + n) + x \cdot \sigma(A') \\
&\iff x \cdot \sigma(A) \geq x \cdot \sigma(A') \\
&\iff \sigma(A) \geq \sigma(A')
\end{aligned}$$

Hence $\delta($$A$ is maximal with respect to δ if and only if A is maximal with respect to σ.

2. Let $x < 0$ and $A' \in \mathcal{A}(u, v)$. Then

$$\begin{aligned} \delta(A) \leq \delta(A') &\iff y \cdot (m+n) + x \cdot \sigma(A) \leq y \cdot (m+n) + x \cdot \sigma(A') \\ &\iff x \cdot \sigma(A) \leq x \cdot \sigma(A') \\ &\iff \sigma(A) \geq \sigma(A') \end{aligned}$$

Hence A is minimal with respect to δ if and only if A is maximal with respect to σ.

Lösung zu Aufgabe 5.3:

Beweis: Wir betrachten das optimale Alignment A von u und v. Annahme: A enthält mindestens eine Ersetzungsoperationen $\alpha \to \beta$ mit $\alpha \neq \beta$. Diese Ersetzungsoperation hat die Kosten ∞ und könnte durch die beiden Edit-Operationen $\alpha \to \varepsilon$ und $\varepsilon \to \beta$ ersetzt werden, die zusammen nur Kosten 2 haben. Damit ist A kein optimales Alignment, d.h. die obige Annahme war falsch. Also enthält A nur Lösch-Operationen, Einfüge-Operationen und Matches. Da die Kosten für Matches 0 sind, hat ein optimales Alignment eine maximale Anzahl von Matches. Ein Folge von Matches ist aber eine Subsequenz und eine maximale Folge von Matches eine Subsequenz maximaler Länge. Daher ist die Anzahl r der Matches in A identisch mit s. Die Edit-Distanz von u und v ist dann genau i, wobei i die Anzahl der Indels in A ist. Damit folgt $m + n = i + 2r = edist_\delta(u, v) + 2s$, aus der sich Aussage (1) direkt ergibt.