Algorithmen auf Sequenzen

Vorlesung von Prof. Dr. Sven Rahmann
im Sommersemester 2008

Kapitel 5
Approximative Textsuche

Webseite zur Vorlesung
http://ls11-www.cs.tu-dortmund.de/people/rahmann/teaching/ss2008/AlgorithmenAufSequenzen

Sprechstunde
Mo 16-17 in OH14, R214

Übersicht

- Abstandsmaße zwischen Strings
- Approximatives Suchproblem
- Sellers' und Ukkonen's Algorithmus
- Scorematrizen, Ähnlichkeiten
- Positionsspezifische Scorematrizen

- Sequenz-Alignments
- Längster gemeinsamer Teilstring
- Längste gemeinsame Teilsequenz

Abstandsmaße auf Strings

Definiere Metriken auf
- A^n (Menge aller Strings der Länge n über A), separat für jedes n
- A^* (Menge aller endlichen Strings über A)

Hamming-Distanz

Die Hamming-Distanz (Anzahl der Unterschiede zwischen zwei gleichlangen Strings) ist eine Metrik auf A^n.

Edit-Distanz

Seien s,t beliebige endliche Strings über A. Man überführt s in t,
indem man auf s von links nach rechts folgende Operationen anwendet
- das nächste Zeichen aus s kopieren (Kosten 0)
- das nächste Zeichen aus s durch ein anderes ersetzen (Kosten 1)
- das nächste Zeichen aus s löschen (überspringen; Kosten 1)
- ein zusätzliches Zeichen einfügen (Kosten 1)

Die Edit-Distanz (unter Einheitskosten) zwischen s und t
ist definiert als die Operationsfolge mit minimalen Kosten, die s in t überführt.
Statt Kosten 0/1 können für jede Operation spezifische Kosten festgelegt werden.
Dies definiert eine Metrik auf A^* (Beweis: Definitheit, Symmetrie, Dreiecksungleichung?).

Abstandsmaße auf Strings

Indel-Distanz

wie Edit-Distanz, aber Ersetzungen sind nicht zugelassen.
Dies definiert eine Metrik auf A^*.

q-gram-Distanz

Zähle wie oft jedes q-gram z (Wort der Länge q, Element von A^q) jeweils in s,t vorkommt:
Die Anzahlen seien $N_z(s)$, $N_z(t)$.
Definiere $d_q(s,t) := \sum_{z \in A^q} |N_z(s) - N_z(t)|$
Dies definiert im allgemeinen keine Metrik auf A^*.
(Welche Eigenschaft einer Metrik wird verletzt?)

Beispiele

```
s = ABRACADABRA
t = CANDELABRAS
```

Hamming-Distanz: 11 (alle Buchstaben sind verschieden)

```
ABRACADABRA
CANDELABRAS
ssssssssss
```

Edit-Distanz: 8

```
ABRACA-DABRA-
--CANDELABRAS
ddscssisccccci
111 1111    1
```

Indel-Distanz: 8

```
ABRACA-D--ABRA-
----CANDELABRAS
ddddcciciiccccci
1111  1 11    1
```

q-gram-Distanz für *q*=1,2: Übung

Legende der Operationen:

`c` = copy, Kopieren, Kosten 0
`s` = substitute, Ersetzen, Kosten 1
`d` = delete, Löschen, Kosten 1
`i` = insert, Einfügen, Kosten 1

Visualisierung als Alignment:

`d` und `i` werden durch Gaps (`-`) in jeweils einer Sequenz dargestellt

Alignments

Definition
Ein Alignment A von zwei Strings s,t über dem Alphabet Σ ist ein String über dem Alphabet der Paare $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$, so dass line1($A$)=$s$ und line2(A)=t.

Dabei ist line1(A) der durch
line1(a,b) := line1(a,$-$) := a und line1($-$,b) := ϵ
definierte String-Homomorphismus, line2 entsprechend.

(Mit anderen Worten, die erste Zeile des Alignments, ohne Gaps gelesen, muss s ergeben, die zweite t.)

Die Kosten eines Alignments sind definiert
als Summe der Kosten seiner Symbole (Spalten),
d.h. die Summe der Kosten der repräsentierten Operationen.

```
s = ABRACADABRA
t = CANDELABRAS
```

Alignment 1 mit Kosten 8
```
ABRACA-DABRA-
--CANDELABRAS
```

Alignment 2 mit Kosten 8
```
ABRACA-D--ABRA-
----CANDELABRAS
```

Alignment 3 mit Kosten 12
```
-ABR--AC-ADABRA
CANDELABRAS----
```

Zur Berechnung der Edit-Distanz von s,t sucht man also nach den minimalen Kosten aller Alignments von s und t unter den erlaubten Edit-Operationen.
Ein Alignment mit minimalen Kosten heißt optimales Alignment.

Problem der Approximativen Textsuche

Gegeben ein Text T und ein Muster p über demselben Alphabet,
sowie ein Abstandsmaß d (entweder auf $A^{|P|}$ oder A^*) und ein Schwellenwert k,
finde die (Start- und) End-Positionen aller Teilstrings t von T mit $d(p,t) <= k$.

Für k=0 erhält man das Problem der einfachen Textsuche.

Das Problem lässt sich auf Mengen von Strings P als Muster verallgemeinern:
Gesucht sind dann die Teilstrings t von T,
so dass es einen String p in P gibt mit $d(p,t) <= k$.

Statt die Menge P explizit anzugeben,
kann man wieder z.B. verallgemeinerte Strings verwenden.

Später betrachten wir noch ein weiteres Modell für Muster:
Positionsspezifische Scorematrizen (PSSMs)

Bemerkung

Das Problem der approximativen Textsuche lässt sich im Prinzip lösen,
indem man die Menge aller passenden Strings aufzählt,
und darauf die Methoden für Stringmengen anwendet (Kapitel 3).
Für viele Distanzmaße ist die Größe der Menge jedoch exponentiell in k.

DP-Algorithmus von Sellers für die Edit-Distanz

Idee

Löse das Problem für alle Präfixe und T und p,
tabelliere Lösungen der kleineren Teilprobleme (dynamic programming)

Eingaben

Muster p, Text T,
Ersetzungkosten für jedes Zeichenpaar $d(a,b)>=0$ mit $d(a,a)=0$ für alle a,
Zeichenspezifische Löschkosten $d(a,\epsilon)$ und Einfügekosten $d(\epsilon,a)$

Definition

Sei p^i das Präfix der Länge i von p, analog T^j.
Sei $D(i,j)$ die minimale Edit-Distanz zwischen p^i und allen Suffixen von T^j.

Rekurrenzen

$D(0,0) = 0$ [leere Strings]
$D(i,0) = D(i\text{-}1,0) + d(p_i,\epsilon)$ für $i>0$ [ganz p^i muss gelöscht werden]
$D(0,j) = 0$ für alle j [leeres Musterpräfix gegen leeres Suffix von T^j]
$D(i,j) = ???$

Wie kann man $D(i,j)$ aus D-Werten für kleinere Argumente berechnen?

Kern des Algorithmus von Sellers

Beobachtung

Jedes Alignment von p^i und jedem Suffix von T^j
muss auf genau eine von drei Arten enden: Sei p_i = a und T_j = b.

```
(1)  ...a      (2)  ...a      (3)  ..a-
     ...b           ..b-           ...b
```

In (1) wird (a,b) einem existierenden Alignment von p^{i-1} und T^{j-1} hinzugefügt.
In (2) wird (a,−) einem existierenden Alignment von p^{i-1} und T^j hinzugefügt.
In (3) wird (−,b) einem existierenden Alignment von p^i und T^{j-1} hinzugefügt.

Behauptung

Das optimale Alignment von p^i und T^j ergibt sich
als beste der drei genannten Möglichkeiten,
d.h. als kostengünstigste Verlängerung der drei optimalen Präfix-Alignments.
Also $D(i,j) = \min \{ D(i-1,j-1) + d(p_i, T_j),$
$D(i-1, j) + d(p_i, \epsilon),$
$D(i, j-1) + d(\epsilon, T_j) \}$

Beweis

durch Induktion und Widerspruch – angenommen, es gäbe ein besseres Alignment;
dann erhielte man ein besseres Präfix-Alignment, was nach Annahme nicht sein kann.

Algorithmus von Sellers

Übersicht

Man berechnet spaltenweise die Matrix D.
Spalte 0: Initialisierung wie angegeben
Spalte j: $D(0,j)=0$ wie angegeben, danach $D(i,j)$ durch Minimum über 3 Fälle für $i>0$
Ist der letzte Eintrag $D(|p|,j)$ in Spalte j kleiner gleich k,
dann haben p und ein Text-Teilstring, der an Position j endet,
eine Edit-Distanz von höchstens k.

Details

- Treffer treten in Clumps (Intervallen) auf – betrachte in solchen Intervallen nur Minima.
- Laufzeit $O(|p||T|)=O(mn)$; jeder Tabelleneintrag benötigt konstante Zeit.
- Speicherbedarf $O(|p|)$, man braucht immer nur 2 Spalten gleichzeitig
- Man erhält nur die Endpunkte der passenden Teilstrings,
  nicht die Anfangspunkte oder die Alignments.

Wie bekommt man die Anfangspunkte und Alignments?

Man merkt sich in jedem Schritt,
welche der drei Vorgängerzellen zur optimalen Wahl geführt hat,
bzw. in welcher Spalte deren optimaler Anfangspunkt lag, etwa start(i,j) = ...

Tafelbeispiel

Beispiel

Gegeben: p = BAABA, T = AABCABAABBABAABA
Einheitskosten, k = 1

			0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
		T		A	A	B	C	A	B	A	A	B	B	A	B	A	A	B	A
p	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	B	1	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0	1
	2	A	2	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	0
	3	A	3	2	1	2	2	1	2	1	0	1	2	1	1	1	0	1	1
	4	B	4	3	2	1	2	2	1	2	1	0	1	2	1	2	1	0	1
	5	A	5	4	3	2	2	2	2	**1**	2	**1**	**1**	**1**	2	**1**	2	**1**	**0**

Farblich hervorgehoben:
Pfade (ggf. mehrere) durch die Matrix, die zu den Einträgen <= 1 der letzten Zeile führen.
Der hellblaue Pfad stellt eine Abzweigung des dunkelblauen Pfades dar.

Übung

Wie sehen die zugehörigen Alignments aus?

Verbesserung: Algorithmus von Ukkonen

Gegeben: p = BAABA, T = AABCABAABBABAABA
Einheitskosten, k = 1

			0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6
		T		A	A	B	C	A	B	A	A	B	B	A	B	A	A	B	A
p	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	B	1	1	1	0	1	1	0	1	1	0	0	1	0	1	1	0	1
	2	A		1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	0
	3	A			1	2	2	1	2	1	0	1	2	1	1	1	0	1	1
	4	B				1	2		1	2	1	0	1	2	1	2	1	0	1
	5	A					2			**1**	2	**1**	**1**	**1**	2	**1**	2	**1**	**0**

Idee

In jeder Spalte j, betrachte last(j), die tiefste Zeile mit Wert <= k (unterstrichen).
Die genauen Werte darunter sind irrelevant und müssen nicht betrachtet werden.
Da Kosten nichtnegativ sind, führen sie nicht zu kleineren Werten weiter rechts.

In Spalte j+1 muss zunächst bis Zeile last(j)+1 berechnet werden,
da möglicherweise die Diagonale Kosten 0 aufweist.
Ist der Wert D(last(j)+1, j+1) > k, so sucht man last(j+1) oberhalb (z.B. Spalte 4)
Ist der Wert noch <= k , ergeben sich die Werte unterhalb nur durch vertikale Kosten;
man fügt diese so lange hinzu, bis erstmalig k überschritten wird.

Algorithmus von Ukkonen

Bemerkung

Bei Einheitskosten ist das vertikale Weitersuchen nicht notwendig,
da sich benachbarte Zellinhalte um 0 oder 1 unterscheiden.

Analyse

Man kann zeigen, dass auf zufälligen Strings gilt: last(j) = O(k),
so dass sich eine Laufzeit von O(kn) ergibt,
was insbesondere bei $k << m$ vorteilhaft gegenüber O(mn) ist.

Definition

Den Quotienten $e := k/m$ nennt man zulässige Fehlerrate (error level).
Wichtig in der Praxis sind kleine Fehlerraten (z.B. 1%, 5%, 10%).

Anwendung: Mapping von Sequenzfragmenten

Das menschliche Genom ist seit ca. 2000 weitgehend sequenziert
und besteht insgesamt aus ca. 3 Gigabasen DNA.
Die öffentlich verfügbare Genomsequenz ist eine Art „Durchschnitt“,
oder Konsensus-Sequenz, aus mehreren Individuen.

Weniger gut erforscht sind individuelle Varianten und deren Bedeutung.
In den letzten Jahren ist DNA-Sequenziertechnologie sehr viel
effizienter (weil paralleler) und billiger geworden.
Man kann jetzt prinzipiell zufällig ausgewählte DNA-Bereiche aus zahlreichen Individuen sequenzieren und in der Konsensus-Sequenz lokalisieren;
durch das Alignment werden die Unterschiede zum Konsensus deutlich.

Unterschiede können mehrere Ursachen haben:
- Sequenzierfehler
- tatsächliche individuelle Unterschiede, z.B. SNPs (single nucleotide polymorphisms)

Je nach verwendeter Technologie sind die sequenzierten DNA-Stücke (reads)
sehr kurz (25 – 35 nt; Solid, Illumina/Solexa) oder kurz (100 – 250 nt, 454/Roche).
Längere reads erlauben eher eine Zuordnung (mapping) zu einem eindeutigen Bereich auf der Konsensus-Sequenz.
Hier lässt sich Ukkonnen's Algorithmus mit $e = 5\%$ anwenden.

NFAs für die approximative Textsuche

Anderer Ansatz (nur für Einheitskosten!):
Wir konstruieren einen NFA, der alle Vorkommen des Musters mit <= *k* Fehlern erkennt.
Wir betrachten dann verschiedene Methoden, diesen NFA bit-parallel zu simulieren.

Design-Ideen

Der NFA hat (*k*+1)(*m*+1) Zustände und besteht aus (*k*+1) Kopien des Muster-NFAs.
Diese Kopien müssen geeignet verbunden werden.

Nach Lesen eines Textzeichens ist Zustand (*e*,*i*) mit 0<=*e*<=*k*, 0<=*i*<=*m*=|*p*| aktiv, wenn das Präfix p^i mit <= *e* Fehlern gleich einem Suffix des bisher gelesenen Textes ist.

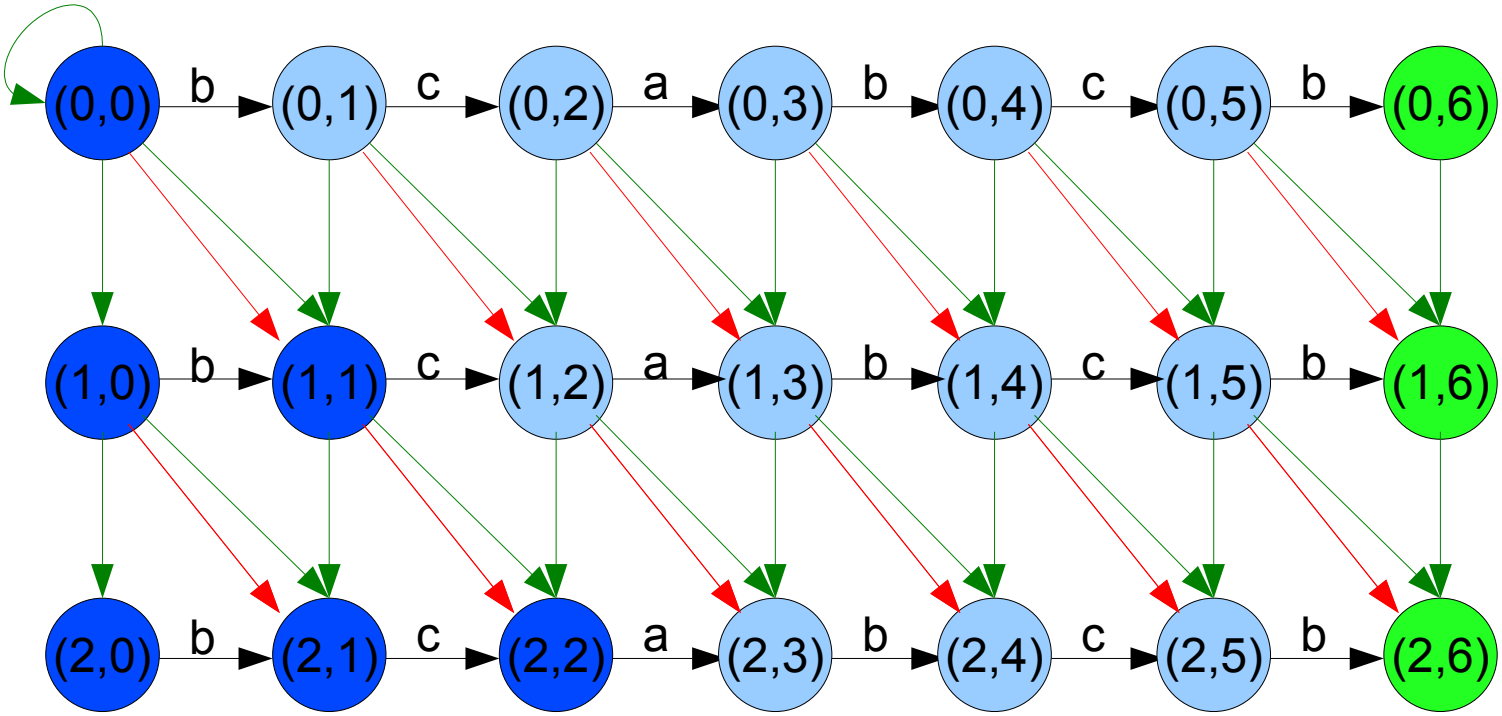
Wann immer ein Zustand (*e*,*m*) aktiv ist, wurde ein Vorkommen mit *e* Fehlern gefunden.
Von Interesse ist immer nur das kleinste *e*, für das (*e*,*m*) aktiv ist.

Übergänge von (*e*,*i*) aus:
Durch Lesen des nächsten Muster-Zeichens p_{i+1} gelangt man nach (*e*,*i*+1) [Match].
Durch Lesen eines beliebigen Zeichens gelangt man nach (*e*+1, *i*+1) [Substitution].
Man gelangt auch direkt (Epsilon-Transition) nach (*e*+1, *i*+1) [Löschung von p_{i+1}].
Durch Lesen eines beliegiben Zeichens gelangt man nach (*e*+1, *i*) [Einfügung vor p_{i+1}].
Es gibt wie immer eine Alphabet-Schleife in (0,0) [Warten auf den Beginn von *p*].

NFA - Beispiel

Beispiel

p = bcabcb, k = 2



Startzustände (dunkelblau): Epsilon-Abschluss der ersten Spalte

Epsilon-Transitionen

Σ-Transitionen

Zusammenhang NFA und DP-Algorithmus

Nach Lesen des *j*-ten Textzeichens gilt:

D(*i*,*j*) <= *k* genau dann wenn D(*i*,*j*) = min { *e*: NFA-Zustand (*e*,*i*) ist aktiv }.

D(*i*,*j*) > *k* genau dann wenn keiner der NFA-Zustände in Spalte *i* aktiv ist.

Zeilenweise bitparallele Simulation

Repräsentiere Zeile *e* in Bitvektor active[*e*], ohne Spalte 0, für *e* = 0 .. *k* (wie Shift-And)
Definiere Masken mask[*c*] für alle Buchstaben *c* wie bei Shift-And.

Initialisierung (Aktivierung der Startzustände)
active[0] = 0...000
active[1] = 0...001
active[2] = 0...011
...

Schritt: Lesen des nächsten Text-Zeichens *c*
Zum Update der active[*e*]-Bits zu active+[*e*] benötigt man
sowohl die alten Werte active[*e*-1] als auch die neuen Werte active+[*e*-1] der Vorzeile.
e = 0:
  active+[0] := (active[0] << 1 | 1) & mask[*c*] (normales Shift-And für *e*=0)
Für alle *e* > 0:
  active+[*e*] := ((active[*e*] << 1) & mask[*c*])
                  | active[*e*-1] | (active[*e*-1] << 1) | (active+[*e*-1] << 1)
Test, ob das *m*-te Bit in einem active+-Vektor gesetzt ist
active[*e*] := active+[*e*] für alle *e*.

Laufzeit
O(*nk m*/*w*), dabei *w* die Wortlänge (32 oder 64), praktisch O(*nk*)

Verallgemeinerungen

Die zeilenweise bit-parallele Simulation des *k*-Fehler NFA lässt sich auf verallgemeinerte Strings verallgemeinern (Kapitel 2):

- Zeichenklassen, über die Masken mask[c]
- Lücken beschränkter Länge (x(min,max)-Notation)
- Optionale Zeichen (c?)
- Wiederholbare Zeichen (c+, c*)

Vorgehen

Man wendet die NFA-Techniken aus Kapitel 2 auf jede der k+1 Kopien des NFA an.

Übung

Wie sieht der Automat zu C – x(2,4) – A – B? – C+ – [AB] mit *k*=1 Fehler aus?
Wiederhole die Methoden zur bit-parallelen Simulation von x(min,max), ?, +, *

Diagonale bit-parallele Simulation

Andere Sichtweise:
Der Automat besteht aus m-k+1 vollständigen Diagonalen (0 .. m – k) der Länge k + 1.
Wir repräsentieren jede Diagonale durch k+2 Bits (das erste ist jeweils konstant 0).
Epsilon-Transitionen treten nur innerhalb einer Diagonale auf.
(Keine Details)