UA RUHR

Algorithmen auf Sequenzen

Four-Russians-Methode: Subquadratische Berechnung der Edit-Distanz

Sven Rahmann

Genominformatik
Universitätsklinikum Essen
Universität Duisburg-Essen
Universitätsallianz Ruhr

Fragestellung

- Gegeben $r \in \Sigma^m$, $s \in \Sigma^n$ mit $m = \Theta(n)$, Alphabetgröße $\sigma = |\Sigma|$
- $d(r, s)$ (Edit-Distanz) kann in $\mathcal{O}(n^2)$ berechnet werden.
  Rekurrenz für Edit-Distanz $D[i, j]$ aller Präfixe der Längen i, j:
  - $D[0, 0] = 0$.
  - $D[i, 0] = i$.
  - $D[0, j] = j$.
  - $$D[i, j] = \min \begin{cases} D[i-1, j-1] + d(s[i-1], t[j-1]), \\ D[i-1, j] + 1, \\ D[i, j-1] + 1. \end{cases}$$

Fragestellung

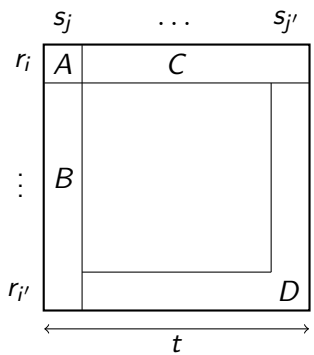
- Gegeben $r \in \Sigma^m$, $s \in \Sigma^n$ mit $m = \Theta(n)$, Alphabetgröße $\sigma = |\Sigma|$
- $d(r, s)$ (Edit-Distanz) kann in $\mathcal{O}(n^2)$ berechnet werden.
  Rekurrenz für Edit-Distanz $D[i, j]$ aller Präfixe der Längen i, j:
  - $D[0, 0] = 0$.
  - $D[i, 0] = i$.
  - $D[0, j] = j$.
  - $$D[i, j] = \min \begin{cases} D[i-1, j-1] + d(s[i-1], t[j-1]), \\ D[i-1, j] + 1, \\ D[i, j-1] + 1. \end{cases}$$
- Reduktion auf subquadratische Zeit?

Fragestellung

- Gegeben $r \in \Sigma^m$, $s \in \Sigma^n$ mit $m = \Theta(n)$, Alphabetgröße $\sigma = |\Sigma|$
- $d(r, s)$ (Edit-Distanz) kann in $\mathcal{O}(n^2)$ berechnet werden.
  Rekurrenz für Edit-Distanz $D[i, j]$ aller Präfixe der Längen i, j:
  - $D[0, 0] = 0$.
  - $D[i, 0] = i$.
  - $D[0, j] = j$.
  - $$D[i, j] = \min \begin{cases} D[i-1, j-1] + d(s[i-1], t[j-1]), \\ D[i-1, j] + 1, \\ D[i, j-1] + 1. \end{cases}$$
- Reduktion auf subquadratische Zeit?
- Ja: mit der **Method of Four Russians** (1970),
  benannt nach ihren Erfindern Arlazarov, Dinic, Kronrod und Faradzev,

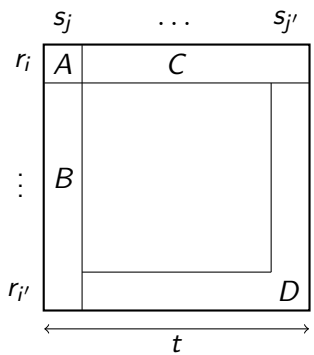
Grundidee

- Innerhalb eines Rechtecks der Matrix hängen die Einträge D nur von den Einträgen A, B, C und von den Teilstrings $r[i..i'], s[j..j']$ ab.
- Definition: Ein t-Block ist ein $t \times t$ großer Ausschnitt der Matrix



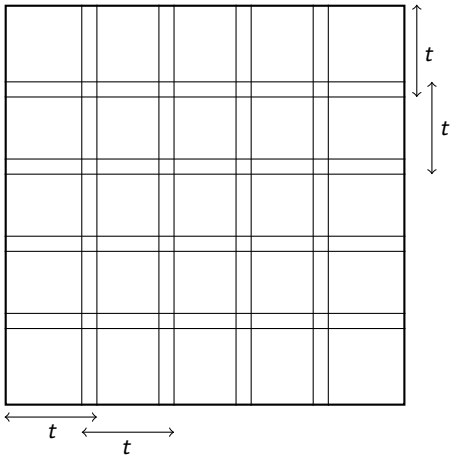
Grundidee

- Innerhalb eines Rechtecks der Matrix hängen die Einträge D nur von den Einträgen A, B, C und von den Teilstrings $r[i..i'], s[j..j']$ ab.
- Definition: Ein t-Block ist ein $t \times t$ großer Ausschnitt der Matrix
- Idee: Matrix in t-Blöcke aufteilen, Werte im Bereich D für alle Kombinationen von (A, B, C, r', s') vorberechnen
- Redundanzen vermeiden



Grundidee

Matrix D in überlappende t-Blöcke aufteilen:



Beobachtungen

Lemma

Unter Einheitskosten unterscheiden sich die Werte in zwei benachbarten Feldern (horizontal, vertikal oder diagonal) der Edit-Matrix um höchstens 1.



Beobachtungen

Wenn die Werte in den Bereichen A, B, C in zwei t-Blöcke sich jeweils um dieselbe Differenz v unterscheiden und die Teilstrings identisch sind, dann unterscheiden sich die Werte in D auch um v.

	a	b	b	a
b	5	6	5	4
a	6	6	6	5
b	6	6	6	6
a	7	7	7	6

	a	b	b	a
b	2	3	2	1
a	3	3	3	2
b	3	3	3	3
a	4	4	4	3

Vorberechnung

Um zu verhindern, dass man D für (unendlich) viele Kombinationen von A, B, C vorberechnen muss, genügt es einen Offsetvektor V_B, V_C aus B und C zu erzeugen.
Sei $V_B[0] := 0$, $V_B[i] := B[i] - B[i-1]$;
sei $V_C[0] := 0$, $V_C[j] := C[j] - C[j-1]$

Vorberechnung

Um zu verhindern, dass man D für (unendlich) viele Kombinationen von A, B, C vorberechnen muss, genügt es einen Offsetvektor V_B, V_C aus B und C zu erzeugen.
Sei $V_B[0] := 0$, $V_B[i] := B[i] - B[i-1]$;
sei $V_C[0] := 0$, $V_C[j] := C[j] - C[j-1]$

	a	b	b	a
b	5	6	5	4
a	6			
b	6			
a	7			

$\rightarrow$

	a	b	b	a
b	0	1	-1	-1
a	1			
b	0			
a	1			

$$B = [5, 6, 6, 7] \quad \rightarrow \quad V_B = [0, 1, 0,]1$$
$$C = [5, 6, 5, 4] \quad \rightarrow \quad V_C = [0, 1, -1, -1]$$

Vorberechnung

- Weil $V_B[0], V_C[0] = 0$ sind und die anderen Einträge nur die Werte $\{-1, 0, 1\}$ annehmen, gibt es maximal $3^{2(t-1)}$ Kombinationen für (V_B, V_C).

Vorberechnung

- Weil $V_B[0], V_C[0] = 0$ sind und die anderen Einträge nur die Werte $\{-1, 0, 1\}$ annehmen, gibt es maximal $3^{2(t-1)}$ Kombinationen für (V_B, V_C).
- Es gibt $\sigma^{2(t-1)}$ verschiedene Substringkombinationen.
- Die Vorberechnung eines t-Blocks dauert $\mathcal{O}((t-1)^2)$ Zeit.

Vorberechnung

- Weil $V_B[0], V_C[0] = 0$ sind und die anderen Einträge nur die Werte $\{-1, 0, 1\}$ annehmen, gibt es maximal $3^{2(t-1)}$ Kombinationen für (V_B, V_C).
- Es gibt $\sigma^{2(t-1)}$ verschiedene Substringkombinationen.
- Die Vorberechnung eines t-Blocks dauert $\mathcal{O}((t-1)^2)$ Zeit.
- Gesamtlaufzeit: $\mathcal{O}\big(3^{2(t-1)}\sigma^{2(t-1)}(t-1)^2\big)$.

Vorberechnung

- Weil $V_B[0], V_C[0] = 0$ sind und die anderen Einträge nur die Werte $\{-1, 0, 1\}$ annehmen, gibt es maximal $3^{2(t-1)}$ Kombinationen für (V_B, V_C).
- Es gibt $\sigma^{2(t-1)}$ verschiedene Substringkombinationen.
- Die Vorberechnung eines t-Blocks dauert $\mathcal{O}((t-1)^2)$ Zeit.
- Gesamtlaufzeit: $\mathcal{O}\big(3^{2(t-1)}\sigma^{2(t-1)}(t-1)^2\big)$.
- Wähle $t = 1 + \log_{3\sigma}(n)/2$:
- Laufzeit wird $\mathcal{O}\big(n \cdot (\log_{3\sigma} n)^2\big)$

Vorberechnung

- Weil $V_B[0], V_C[0] = 0$ sind und die anderen Einträge nur die Werte $\{-1, 0, 1\}$ annehmen, gibt es maximal $3^{2(t-1)}$ Kombinationen für (V_B, V_C).
- Es gibt $\sigma^{2(t-1)}$ verschiedene Substringkombinationen.
- Die Vorberechnung eines t-Blocks dauert $\mathcal{O}((t-1)^2)$ Zeit.
- Gesamtlaufzeit: $\mathcal{O}\big(3^{2(t-1)}\sigma^{2(t-1)}(t-1)^2\big)$.
- Wähle $t = 1 + \log_{3\sigma}(n)/2$:
- Laufzeit wird $\mathcal{O}\big(n \cdot (\log_{3\sigma} n)^2\big)$
- Speicherverbrauch für D-Region eines t-Blocks: $\mathcal{O}(t)$ Bits (differenzcodiert)
- Insgesamt: $\mathcal{O}(n \log n)$ Bits, machbar

Berechnung

	–	a	b	b	a	b	a
–							
b							
b							
a							
a							
b							
a							

Berechnung

1. Initialisiere 0-te Zeile und Spalte

$V_B =$
$V_C =$
$A =$
$D =$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:

$V_B =$
$V_C =$
$A = 0$
$D =$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D =$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup:
   $D = F[V_B, V_C,$
   $r[i' : i''], s[j' : j'']]$.

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D = 2, 2, 2, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1						
b	2						
a	3						
a	4						
b	5						
a	6						

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup:
   $D = F[V_B, V_C,$
   $r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array

$V_B = 0, 1, 1, 1$
$V_C = 0, 1, 1, 1$
$A = 0$
$D = 2, 2, 2, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			
b	2			1			
a	3	2	2	2			
a	4						
b	5						
a	6						

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup:
   $D = F[V_B, V_C, r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array

$V_B = 0, 1, 1, 1$
$V_C = 0, -1, 0, 0$
$A = 3$
$D = 2, 1, 1, 0, 0$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			
b	2			1			
a	3	2	2	2			
a	4			3			
b	5			3			
a	6	5	4	4			

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup:
   $D = F[V_B, V_C,$
   $r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array

$V_B = 0, -1, -1, 1$
$V_C = 0, 1, 1, 1$
$A = 3$
$D = -2, -1, 0, 1, 2$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			5
b	2			1			4
a	3	2	2	2	1	2	3
a	4			3			
b	5			3			
a	6	5	4	4			

Berechnung

1. Initialisiere 0-te Zeile und Spalte
2. Für alle $i, j < n/t$:
3. Berechne V_B, V_C aus B, C.
4. Lookup:
   $D = F[V_B, V_C,$
   $r[i' : i''], s[j' : j'']]$.
5. Addiere Wert A zum D-Array

$V_B = 0, 1, 0, 1$
$V_C = 0, -1, 1, 1$
$A = 2$
$D = 1, 1, 0, 1, 0$

	–	a	b	b	a	b	a
–	0	1	2	3	4	5	6
b	1			2			5
b	2			1			4
a	3	2	2	2	1	2	3
a	4			3			2
b	5			3			3
a	6	5	4	4	3	3	2

Laufzeit

Ein Block:

- Berechnen der Offset-Vektoren: $\mathcal{O}(t)$
- Lookup in F: $\mathcal{O}(t)$ zur Berechnung des Index
- Addition von A: $\mathcal{O}(t)$
- Laufzeit pro t-Block also $\mathcal{O}(t)$

Laufzeit

Ein Block:

- Berechnen der Offset-Vektoren: $\mathcal{O}(t)$
- Lookup in F: $\mathcal{O}(t)$ zur Berechnung des Index
- Addition von A: $\mathcal{O}(t)$
- Laufzeit pro t-Block also $\mathcal{O}(t)$

Gesamtanalyse:

- Anzahl der t-Blöcke: n^2/t^2

Laufzeit

Ein Block:

- Berechnen der Offset-Vektoren: $\mathcal{O}(t)$
- Lookup in F: $\mathcal{O}(t)$ zur Berechnung des Index
- Addition von A: $\mathcal{O}(t)$
- Laufzeit pro t-Block also $\mathcal{O}(t)$

Gesamtanalyse:

- Anzahl der t-Blöcke: n^2/t^2
- Laufzeit insgesamt: $\mathcal{O}(t \cdot n^2/t^2) = \mathcal{O}(n^2/\log n)$

Zusammenfassung: Four Russians Methode

- Mit dem Four-Russians Trick (Differenzcodierung, Vorberechnung kleiner Blöcke) kann man die Edit-Distanz in subquadratischer Zeit berechnen.
- Vorberechnung: $\mathcal{O}\big(n\,(\log n)^2\big)$ Zeit
- Berechnung selbst: $\mathcal{O}(n^2/\log n)$ Zeit

Zusammenfassung: Four Russians Methode

- Mit dem Four-Russians Trick (Differenzcodierung, Vorberechnung kleiner Blöcke) kann man die Edit-Distanz in subquadratischer Zeit berechnen.
- Vorberechnung: $\mathcal{O}\big(n\,(\log n)^2\big)$ Zeit
- Berechnung selbst: $\mathcal{O}(n^2/\log n)$ Zeit
- Wegen der hohen Basis im Logarithmus (3σ) ist die Methode für kleine Alphabete (DNA: $\sigma = 4$) erst ab ca. $n = 10^4$ sinnvoll.
- Für große Alphabete wird hingegen viel Speicher benötigt.
- Daher wird diese Methode in der Praxis selten genutzt.