1 Fast Gefunden

In der Bioinformatik (engl. *computational biology*), bei großen Text-Datenbanken und allgemein bei Anwendungen, wo im Text sowie im Suchwort gewisse Fehler, Ungenauigkeiten oder Unbestimmtheiten auftreten können, ist das exakte Suchen nach vorgegebenen Mustern oft nicht das Problem, das man eigentlich lösen möchte. Relevanter wäre hier z.B., die bestmögliche Übereinstimmung zwischen Text und Muster zu bestimmen oder eine Reihe von Transformationen, die den ungenauen Text in den gewünschten umwandeln. Diesmal betrachten wir einige Variationen des Problems, Muster in Texten "fast" zu finden (engl. *approximate pattern matching*). Es werden zwei konkrete Probleme vorgestellt, die mit zwei sehr ähnlichen Algorithmen gelöst werden können. Diese Algorithmen arbeiten nach dem Prinzip des *dynamischen Programmierens*.

Es seien X und Y Wörter aus m bzw. n Zeichen, die wir von 1 ab nummerieren. Es gilt also $X = X[1..m]$ und $Y = Y[1..n]$.

Für zwei gegebene Texte X und Y definieren wir einen "Abstand" zwischen X und Y wie folgt. Die *Edit-Distanz* (engl. *edit distance*) zwischen X und Y, $\text{edit}(X, Y)$, soll die kleinste Anzahl von primitiven "Edit"-Operationen sein, um X in Y umzuwandeln. Bei der Umwandlung bearbeiten wir das Wort X von links nach rechts und erlauben die folgenden Operationen auf dem $i$ten Zeichen $X[i]$ (dabei stellen wir uns vor, daß Y mit dem leeren String initialisiert ist und dann von links nach rechts aufgebaut wird):

- *Copy*: $X[i]$ wird hinten an Y angefügt, danach $i \leftarrow i + 1$.
- *Insert a*: a wird hinten an Y angefügt ($i \leftarrow i$).
- *Delete*: Das Zeichen $X[i]$ wird übersprungen (also nicht an Y angefügt), danach $i \leftarrow i + 1$.
- *Change to a*: Das Zeichen $X[i]$ wird in $a \neq X[i]$ umgewandelt und hinten an Y angefügt, danach $i \leftarrow i + 1$.

Die Edit-Distanz $\text{edit}(X, Y)$ ist dann definiert als die minimal nötige Anzahl von *Insert*-, *Delete*- und *Change*-Operationen (*Copy*-Operationen werden also nicht gezählt) in einer Sequenz von Edit-Operationen aller vier Typen, die X in Y umwandelt. Es ist leicht zu sehen, daß $\text{edit}(X, Y)$ wirklich einen sinnvollen Abstand definiert; es gilt nämlich:

- $\text{edit}(X, Y) = \text{edit}(Y, X)$ (Symmetrie)
- $\text{edit}(X, Y) \leq \text{edit}(X, Z) + \text{edit}(Z, Y)$ (Dreiecks-Ungleichung)
- $\text{edit}(X, Y) = 0$ genau dann, wenn $X = Y$.

Bemerkung: Statt diese Definition der Edit-Distanz zu verwenden, in der das Wort X von links nach rechts abgearbeitet wird, lässt sich die dazu gleichwertige Definition verwenden, in der einfach gefragt wird, wie viele Zeichen man in X an beliebigen Stellen einfügen, ersetzen, oder löschen muß, um Y aus X zu erhalten. Für die Entwicklung eines Algorithmus ist die obige Definition aber hilfreicher.

Hier ein Beispiel, wie sich das Wort `Praktikum` in das Wort `Program` umwandeln lässt:

Operation	$X[i,m]$	Y
	`Praktikum`	
Copy	`raktikum`	`P`
Copy	`aktikum`	`Pr`
Change to `o`	`ktikum`	`Pro`
Change to `g`	`tikum`	`Prog`
Change to `r`	`ikum`	`Progr`
Change to `a`	`kum`	`Progra`
Delete	`um`	`Progra`
Delete	`m`	`Progra`
Copy		`Program`

Aus dieser Sequenz von Edit-Operationen folgt edit(`Praktikum`, `Program`) ≤ 6. Es gibt natürlich viele andere Möglichkeiten, `Praktikum` in `Programm` umzuwandeln. Es kann i.a. auch mehrere verschiedene Sequenzen geben, die X in Y umwandeln und minimal viele *Insert*-, *Delete*- und *Change*-Operationen enthalten.

2 Bestimmung der Edit-Distanz

Wir möchten jetzt die Edit-Distanz zwischen zwei vorgegebenen Texten X und Y berechnen. Wir betrachten Teil-Wörter $X[1..i]$ und $Y[1..j]$, $0 \leq i \leq m$ und $0 \leq j \leq n$, und versuchen, eine Tabelle mit Lösungen für alle Teil-Probleme $\mathrm{edit}(X[1..i], Y[1..j])$ zu konstruieren:

$$\mathtt{EDIT}[i,j] = \mathrm{edit}(X[1..i], Y[1..j])$$

Die Lösung zum ursprünglichen Problem kann dann direkt in $\mathtt{EDIT}[m,n]$ abgelesen werden. Teile der `EDIT`-Tabelle können einfach initialisiert werden: es ist nämlich klar, daß $\mathtt{EDIT}[0,j] = j$ für alle $j \leq n$, und $\mathtt{EDIT}[i,0] = i$ für alle $i \leq m$. Im ersten Fall müssen wir Y von einem leeren Wort konstruieren, welches mit einer Sequenz von j *Insert*-Operationen geht, und im zweiten Fall müssen wir ein leeres Wort Y konstruieren, welches mit einer Sequenz von i *Delete*-Operationen geht.

Ist die letzte Operation in einer optimalen Sequenz, die $X[1..i]$ in $Y[1..j]$ transformiert, eine *Insert*-Operation, gilt offensichtlich $\mathtt{EDIT}[i,j] = \mathtt{EDIT}[i,j-1]+1$. Ist die letzte Operation eine *Delete*-Operation, gilt umgekehrt $\mathtt{EDIT}[i,j] = \mathtt{EDIT}[i-1,j]+1$. Für die *Copy*-Operation, die nur im Fall $X[i] = Y[j]$ relevant ist, gilt $\mathtt{EDIT}[i,j] = \mathtt{EDIT}[i-1,j-1]$, und schließlich für die *Change*-Operation $\mathtt{EDIT}[i,j] = \mathtt{EDIT}[i-1,j-1]+1$. Mittels der Gleichung

$$\mathtt{EDIT}[i,j] = \min(\mathtt{EDIT}[i,j-1]+1, \mathtt{EDIT}[i-1,j]+1, \mathtt{EDIT}[i-1,j-1]+\delta(X[i],Y[j])),$$

wobei $\delta(a,b) = 1$, wenn $a \neq b$, und $= 0$, wenn $a = b$ (weil die *Copy*-Operation nichts zum Abstand beiträgt), können wir die Tabelle $\mathtt{EDIT}[i,j]$ aufbauen. Ein naives Programm mit zwei verschachtelten Schleifen läuft in $O(mn)$ Zeit, verwendet aber auch $O(mn)$ Speicherplatz für die ganze Tabelle. Es ist aber leicht zu sehen daß der Platz-Verbrauch sich zu $O(\min(m,n))$ reduzieren lässt (wie?).

Dieser Algorithmus ist ein typisches Beispiel für *dynamisches Programmieren*: die Lösung des gegebenen Problems wird aus bereits gefundenen und gespeicherten Lösungen zu Teil-Problemen (der gleichen Art) zusammengesetzt.

2.1 Alternative Sichtweise der Berechnung

Eine alternative Sichtweise der Methode aus dem vorigen Abschnitt, die Edit-Distanz zu berechnen, ergibt sich, wenn man das Problem auf ein Graphen-Problem zurückführt. Seien die Paare (i, j) für $0 \leq i \leq m$ und $0 \leq j \leq n$ die Knoten in einem Graphen G. Der Knoten (i, j) repräsentiert das Paar der Teil-Wörter $X[1..i]$ und $Y[1..j]$.

Die Kanten in G sollen so gewählt werden, daß jeder Pfad von (i, j) zu (i', j'), $i \leq i'$ und $j \leq j'$, einer Sequenz von Edit-Operationen entspricht, die $X[i+1..i']$ in $Y[j+1..j']$ transformiert. (Zur Erinnerung: $X[a..b]$ ist per Konvention der leere String, falls $a > b$.)

Das Gewicht eines Pfades (Summe der Kantengewichte) soll die Anzahl von *Insert*-, *Delete*- und *Change*-Operationen sein. Wie oben erklärt, ist diese Eigenschaft erfüllt, wenn G alle Kanten $(i-1, j) \to (i, j)$ mit Gewicht 1, alle Kanten $(i, j-1) \to (i, j)$ mit Gewicht 1, und Kanten $(i-1, j-1) \to (i, j)$ mit Gewicht $\delta(X[i], Y[j])$ enthält. Die Edit-Distanz zwischen den Wörtern X und Y, die ja die minimale Anzahl von *Insert*-, *Delete*- und *Change*-Operationen in einer Sequenz von Edit-Operationen zur Umwandlung von X in Y ist, kann berechnet werden als die Länge eines kürzesten Pfades von $(0, 0)$ zu (m, n) in G. Ein Pfad von $(0, 0)$ nach (m, n) entspricht ja dem Abarbeiten des Wortes X von links nach rechts, wobei für jedes Zeichen $X[i]$ eine oder mehrere Edit-Operationen ausgeführt werden. Der Graph G ist offensichtlich azyklisch, so daß sich diese Berechnung in linearer Zeit in der Größe von G ausführen lässt (siehe Skript zu Blatt 7). Da der Graph $O(mn)$ Knoten und Kanten hat, folgt sofort, daß Speicherplatz und Zeitaufwand für die Berechnung der Edit-Distanz mit dieser Methode beide $O(mn)$ sind.

Ein Vorteil dieser Lösung ist, daß sich eine optimale Sequenz von Operationen, die X in Y umwandelt, leicht rekonstruieren lässt; man muß einfach den Pfad von $(0, 0)$ nach (m, n) durchlaufen. Man kann auch die Kantengewichte modifizieren, so daß sie gewisse Kosten für die entsprechenden Operationen angeben. Beide Verallgemeinerungen lassen sich aber auch leicht in die Lösung mittels dynamischen Programmierens einarbeiten. Hier muß man natürlich aufpassen, daß sich jede Berechnung eines Tabelleneintrags $\texttt{EDIT}[i, j]$ in konstanter Zeit ausführen lässt.

3 Längste gemeinsame Teil-Sequenz

Ein der Berechnung der Edit-Distanz sehr verwandtes Problem ist die Berechnung der längsten gemeinsamen Teil-Sequenz (engl. *longest common subsequence*) von Wörtern X und Y, d.h. des größten Wertes k, so daß es Folgen $i_1 < i_2 < \cdots < i_k$ und $j_1 < j_2 < \cdots < j_k$ gibt mit $X[i_1] = Y[j_1], X[i_2] = Y[j_2], \ldots, X[i_k] = Y[i_k]$. Man kann wie beim Problem der Edit-Distanz leicht eine Gleichung angeben, die direkt zu einer Lösung mittels dynamischen Programmierens führt. Dafür bezeichne $\text{longest}(X, Y)$ die Länge der längsten gemeinsamen Teil-Sequenz von X und Y; wir konstruieren eine Tabelle $\texttt{LCS}$ mit $\texttt{LCS}[i, j] = \text{longest}(X[1..i], Y[1..j])$.

Es besteht ein direkter Zusammenhang zwischen der Edit-Distanz und der längsten gemeinsamen Teil-Sequenz. Wenn man keine *Change*-Operationen erlaubt, so sollte nämlich klar sein, daß man $\mathrm{longest}(X, Y)$ bestimmen kann als die Anzahl von *Copy*-Operationen in einer kürzesten Sequenz von *Copy*-, *Delete*- und *Insert*-Operationen, die X in Y umwandelt. Bezeichnen wir daher mit $\mathrm{edit}_{\mathrm{cdi}}(X, Y)$ die Edit-Distanz zwischen X und Y, wo nur *Copy*-, *Delete*- und *Insert*-Operationen erlaubt sind, und wobei die *Copy*-Operationen wieder nichts zum Abstand beitragen ($\mathrm{edit}_{\mathrm{cdi}}(X, Y)$ kann analog zum vorigen Abschnitt berechnet werden). Dann gilt:

$$\mathrm{longest}(X, Y) = \frac{m + n - \mathrm{edit}_{\mathrm{cdi}}(X, Y)}{2} \tag{1}$$

Die Begründung hierfür ist wie folgt: Jede Sequenz von *Copy*-, *Delete*- und *Insert*-Operationen, die X in Y umwandelt und c *Copy*-Operationen enthält, muß außerdem genau $m - c$ *Delete*-Operationen (für die $m - c$ Zeichen von X, die nicht kopiert werden) und $n - c$ *Insert*-Operationen (für die $n - c$ Zeichen von Y, die nicht durch Kopieren entstehen) enthalten, und ihre Kosten sind damit $m + n - 2c$. Somit hat eine solche Sequenz genau dann minimale Kosten, wenn maximal viele *Copy*-Operationen enthalten sind. Damit muß $\mathrm{edit}_{\mathrm{cdi}}(X, Y) = m + n - 2 \cdot \mathrm{longest}(X, Y)$ gelten, woraus sofort Gleichung (1) folgt.

Eine andere Herleitung der Gleichung $\mathrm{edit}_{\mathrm{cdi}}(X, Y) = m + n - 2 \cdot \mathrm{longest}(X, Y)$ ergibt sich aus der Betrachtung des Gitter-Graphen G aus Abschnitt 2.1. Die *Copy*-Operationen verbinden direkt Knoten (i, j) und $(i+1, j+1)$ und sind diagonale Kanten in G. Ersetzen wir jede solche Kante, die ja Kosten 0 hat, auf einem kürzesten Pfad von $(0, 0)$ zu (m, n) durch *zwei* Kanten $(i, j) \to (i + 1, j)$ und $(i + 1, j) \to (i + 1, j + 1)$ mit jeweils Kosten 1, so wird der Pfad um genau $2c$ teurer und hat Länge $m + n$.