Aufgabe3: LaTeX-Dokument

Team: ??? / Name
Team-ID: ???

3. April 2022

Inhaltsverzeichnis

1 Lösungsidee

1.1 Das Problem

Gegeben ist ein String α aus N Hexadezimalziffern, wobei jeder Ziffer eine einzigartige Kombination aus bis zu sieben Segmenten zugewiesen wird. Durch das 'Umlegen' dieser Segmente, welches maximal M erfolgen darf, soll nun der größtmöglichste Hexadezimalstring gebildet werden.
Der Lösungsidee beruht auf folgender Abstraktion des String-Umformungsvorgangs:

1.2 Wie man sich die schrittweise Umformung des Strings vorstellen kann

Um eine Ziffer in eine andere zu transformieren müssen Segmente umgelegt werden. Dabei gibt es für zwei Ziffern x und y, Segmente die in x sind, wo sie in y nicht mehr sein dürfen, und in y sind, ohne in x zu sein.

$$b_{x\to y} := (\text{Anzahl der Segmentpositionen, die nur in x belegt sind,}$$
$$\text{Anzahl der Segmentpositionen, die nur in y belegt sind})$$
$$\implies b_{B\to D} = (1, 1)$$

An dieser Stelle muss hinzugefügt werden, dass die Segmentdarstellung der Ziffern, auch als Binärdarstellung mit 7-bits, gedacht werden kann. Ordnet man die Segmentplätze wie in Abbildung 1, kann jeder Ziffer eine 7 bit Zahl zugeordnet werden. $B \to D$ entspricht dann $0101111 \to 0011111$
Solange die zwei Komponenten von $b_{x\to y}$ gleich sind, intern also an genauso vielen Positionen Segmente entfernt, wie hinzugefügt werden müssen, kann die Transformation ohne Beachtung des Rest des Strings erfolgen. Sie kostet dann $b_{x\to y}[0]$ Umlegungen.
Es gibt aber auch Fälle, wie zum Beispiel $b_{2\to 7} = (2, 1)$, in denen die Segmentanzahlen der zwei Ziffern

nicht gleich sind. Solche Transformationen sind immer noch möglich, da Transformationen mehrerer Ziffern eines Strings sich ausbalancieren können. Beispiel:
“$2F$“ $\rightarrow$ “$C0$“, wobei $b_{2\rightarrow C} = (3, 1)$ und $b_{F\rightarrow 0} = (1, 3)$
Die erste Transformation muss zwei Segmente nach außen abgeben. Sie kann trotzdem stattfinden, da gleichzeitig die zweite Transformation zwei Segmente von außen aufnehmen muss.
Man kann sich für einen Substring γ während des Transformationsvorgangs eine zusäzliche Information g, das Gleichgewicht des Strings, vorstellen. Diese ist anfänglich 0, wird jedoch von Transformationen verändert. Was das heißen soll kann man am Beispiel erkennen:

$$\begin{aligned} \gamma = \text{“}2F\text{“} \quad & g = 0 \\ \xrightarrow{2\rightarrow C} \gamma = \text{“}CF\text{“} \quad & g = 2 \\ \xrightarrow{F\rightarrow 0} \gamma = \text{“}C0\text{“} \quad & g = 0 \end{aligned}$$

Solange das vorgestellte Gleichgewicht nicht null ist, ist der String ungültig, da er teilweise aus nichtexistenten Segmenten besteht, oder Segmente ins Nichts abgelegt wurden. Eine Transformation $x \rightarrow y$ beeinflusst das Gleichgewicht des Systems folgendermaßen:

$$b_{x\rightarrow y} = (w, z) \implies g_{x\rightarrow y} = w - z$$

Man kann sich g als Ablage vorstellen, die positiv ist, wenn der String weniger Segmente als zuvor besitzt. Kann man einer Transformationen Umlegungskosten zuweisen? Die Transformation $x \rightarrow y$ kostet zuerst einmal die inneren Kosten, also $\min(w, z)$. Die Umlegungen, die in ”Zusammenarbeit” mit einer anderen Transformation stattfindeen, werden nur halb gezählt, da sie eben bei beiden Transformationen betrachtet werden. Für die Kosten gilt also:

$$k_{x\rightarrow y} = \min(w, z) + \frac{1}{2}|w - z| \quad (\text{Immer mit } (w, z) = b_{x\rightarrow y})$$

Es muss an dieser Stelle hinzugefügt werden, dass das gerade erklärte System mit den Transformationen, der vorgestellten ”Segmentbank” und den rationalen Kosten, nicht beschreibt wie am Schluss die Segmente tatsächlich umgelegt werden. Es wird nur im Verfahren den Ergebnisstring zu finden benötigt.
Damit ein String nach mehrern Transformationen, die dazu führen, dass $g = 0 \wedge k \leq M$, auch im Sinne der Aufgabenstellung gültig ist muss noch folgendes gezeigt werden.

1.3 Weshalb eine Ziffer nie komplett leer ist

Es soll gezeigt werden, dass es immer möglich ist Strings gleicher Segmentanzahlen umzuformen, ohne dabei eine Ziffer je komplett zu leeren.

1.4 Ansatz

Die Lösung wird Ziffer für Ziffer zusammengesetzt. Damit der resultierende String so groß wie möglich ist werden in absteigender Reihenfolge die möglichen bedeutendsten Ziffern getestet. Der allererste Schritt für $\alpha =$ “$D24$“ ist also zu bestimmen, was die minimalen Kosten für “$D24$“ $\rightarrow$ “F“ sind. Der Suffix ist unbestimmt, da seine lexikographische Größe weniger wichtig als die der bedeutendsten Stelle ist. $D \rightarrow F$ resultiert in g = 1 für den Suffix. Dieser Suffix muss also ausgehend von g=1 "balanciert"werden, also mit minimalen Kosten so umgeformt werden, dass g zu 0 wird. Die minimalen Kosten für die balancierung des Suffixes sein $f(i, g)$ wobei i die bedeutendste Stelle des Suffixes ist und g das Ungleichgewicht. Wenn $k_{D\rightarrow F} + f(1, 1) \leq M$ wäre, könnte man jetzt die 0. Stelle der Lösung auf F festlegen, doch im Beispiel ist das nicht der Fall. Also geht man weiter zu Umformung in die nächstkleinere Ziffer E und so weiter.
Zusammengefasst wird die Lösung also Ziffer für Ziffer, beginnend bei der bedeutendsten Stelle, zusammengesetzt. Ausgehend von den Kosten für die bisherigen Transformationen und dem aus diesen resultierten Ungleichgewicht, wird die aktuelle Stelle in die größtmöglichste Ziffer transformiert. Dass die Kosten am Schluss nicht M übersteigen und dass der resultierende String gleich viele Segmente wie das Original enthält, wird von f garantiert. Da jedesmal die größtemöglichste Ziffer gewählt wird, ist auch das Ergebnis maximal.

1.5 f(i, g)

$$f(N,g) = \begin{cases} 0, \text{ falls } g = 0 \\ \infty, \text{ ansonsten} \end{cases} \qquad \text{0-basierte Indexe}$$

$$f(i,g) = \min\left(\infty, \min_{y \in Z}\left(k_{x \to y} + f(i+1, g + g_{x \to y})\right)\right)$$

Z ist die Menge der Ziffern $\quad x$ ist die Ziffer im Original

Da die Definitionsmenge von f relativ begrenzt ist, kann man mit dynamic programming die Ergebnisse Memoisieren. Der Speicherbedarf ist dann $O(N^2)$ und die Laufzeit $O(N^2 \cdot |Z|)$.

1.6 Zusammenfassung

Für den gesamten Algorithmus ergibt sich somit für das Finden des Lösungsstrings die Laufzeit $O(N^2 \cdot |Z|)$ Hat man einmal den Lösungsstring kann man in linearer Laufzeit die Umlegungsschritte berechnen.

2 Umsetzung

Für die Umsetzung wurde Rust verwendet

3 Erweiterungen

4 Beispiele

5 Quellcode