Aufgabe 2: Twist

2.1 Twisten

Lösung

Wir wollen Elvis helfen, seine Texte zu twisten. Dazu müssen wir uns überlegen, wie wir möglichst einfach die inneren Zeichen (also alle Zeichen außer dem Ersten und dem Letzten) eines Wortes mischen können. Dabei wollen wir idealerweise nicht nur irgendwelche Twists finden, sondern eine neue, zufällige Anordnung der inneren Zeichen in den getwisteten Wörtern mit gleicher Wahrscheinlichkeit erreichen.

Der erste Algorithmus, der einem einfallen mag, besteht daraus, alle zu mischenden Zeichen in eine Liste zu packen und dann solange zufällig Elemente aus dieser Liste zu entnehmen, bis die Liste leer ist. In Pseudocode könnte das ungefähr so aussehen:

```
1 mischen(wort w mit Länge n):
2    zeichen := {w[2], w[3], ..., w[n − 1]}
3    ergebnis := w[0]
4    solange zeichen nicht leer:
5       i := zufälliger Index in zeichen
6       ergebnis += zeichen[i]
7       entferne i in zeichen
8    ergebnis += w[n]
9    return Ergebnisse
```

Das hat allerdings zwei entscheidende Nachteile:

1. Man benötigt eine separate Liste für die zu mischenden Zeichen, was zusätzlichen Speicherplatz einnimmt.
2. Aufgrund der üblichen Implementationen einer Liste (Array-Liste und verkettete Liste) hat dieser Algorithmus eine quadratische Laufzeit.

Es gibt mit dem Fisher-Yates-Shuffle[1] jedoch einen Algorithmus, welcher beide Nachteile nicht hat. Er führt zufällige Vertauschungen aus, sodass die Eingabe am Ende gemischt ist[2]. In Pseudocode:

```
1 fisherYates(wort w mit Länge n):
2    für alle i von n − 1 bis 2:
3       j := Zufallszahl zwischen 2 und i
4       vertausche w[i] und w[j]
5    return w
```

Dieser Algorithmus gilt als Standardalgorithmus für das Mischen von Arrays und hat gegenüber unserem ersten Algorithmus wie bereits erwähnt zwei Vorteile:

1. Der Algorithmus arbeitet innerhalb der Liste („in-place“), benötigt also keine zusätzliche Liste.

[1] https://de.wikipedia.org/wiki/Zufällige_Permutation#Fisher-Yates-Verfahren
[2] Dabei sind alle Permutationen sogar gleich wahrscheinlich, was hier jedoch nicht erforderlich wäre

2. Der Algorithmus hat eine lineare Laufzeit, da nur $n-2$ Vertauschungen für jedes Wort der Länge n ausgeführt werden.

Zu beachten sind hierbei noch Randfälle:

- Wörter der Länge 1 oder 2 haben keine inneren Zeichen. Sie sollten deshalb durch das Twisten nicht verändert werden. Ebenso bleiben Wörter der Länge 3 mit nur einem inneren Zeichen unverändert. Wörter der Länge 4 haben zwei innere Zeichen, die entweder miteinander vertauschen werden können oder nicht. In all diesen Fällen ist das Enttwisten der Wörter kein Problem.
- Zahlen, Interpunktionszeichen, Leerzeichen und andere Zeichen müssen gesondert behandelt werden. So sollten z.B. Zahlen nicht getwistet werden und Interpunktionszeichen sollten nicht als Teil von Wörtern behandelt werden.

Um also einen Text zu twisten, werden die folgenden Schritte benötigt:

1. Die Eingabe wird eingelesen und durch einen geeigneten regulären Ausdruck in einzelne Wörter aufgespalten[3].
2. Wörter, die nur aus Buchstaben bestehen (also z.B. keine Jahreszahlen), werden mithilfe des Fisher-Yates-Verfahrens getwistet.
3. Die Wörter werden wieder aneinandergehängt und zusammen mit allen anderen Zeichen in der ursprünglichen Reihenfolge ausgegeben.

Beispiele

Für die online vorgegebenen vier Beispieltexte ergibt das Twisten folgende Ergebnisse:

Eingabe	Ausgabe
twist2.txt	Hat der atle Heneexstemir scih doch enaiml wgbeegeebn! Und nun seolln senie Gietesr auch ncah meienm Wielln leben. Seine Wrot und Wreke mrket ich und den Bucrah, und mit Gteäkrtsissee tu ich Wneudr auch.
twist3.txt	Ein Rarsuaetnt, wlhcees a la ctrae arebtiet, beitet sein Abgenot onhe eine vehorr fggteetelse Mfneeoülgerinhe an. Duardch haben die Gätse zwar mehr Slapeurim bei der Whal ihrer Seisepn, für das Rnatesuart etenhetsn jceodh zhtcäisulzer Awanufd, da weengir Pinahrigsshlcuenet vnarohden ist.
twist4.txt	Atsuuga Ada Byron King, Cnotuses of Lcelaove, war enie bhirtsice Agelide und Mktiiaehtmearn, die als die esrte Pomgriearremirn ürbapheut glit. Btreeis 100 Jarhe vor dem Afmuomken der etrsen Parishoerrgamprcemn enasrn sie enie Rehcen-Mnehicak, der egiine Kpnetoze mdoenerr Prmcmgsraaeeorirhpn vaeowhnrgm.
twist5.txt	Aicle fing an scih zu lwgenliaen; sie saß shocn lnage bei iehrr Scshteewr am Ufer und htate nhtics zu thun. Das Buch, das irhe Sehscwter las, gfieel ihr nihct; dnen es wrean wdeer Bidelr ncoh Gäpscerhe drain. „Und was nüzten Bceühr," dahtce Acile, „ohne Beidlr und Gcsähpree?" [...]

[3]Die meisten Sprachen bieten Möglichkeiten zum Umgang mit regulären Ausdrücken („RegEx"). Damit lassen sich Wortgrenzen ganz einfach mit dem regulären Ausdruck „\b" erkennen, für weitere Informationen siehe z. B. http://www.regexe.de/hilfe.jsp

2.2 Enttwisten

Lösung

Die zweite Aufgabe ist etwas komplizierter: Wir haben ein Wörterbuch zur Verfügung und sollen einen Text mit Hilfe dieses Wörterbuchs wieder enttwisten. Dabei stellt sich die Frage: Wie kann man ein getwistetes Wort möglichst einfach und effizient einem Eintrag im Wörterbuch zuordnen? Dazu müssen wir uns erst einmal klar machen, was ein getwistetes Wort mit dem Originalwort gemeinsam hat:

1. Der erste und der letzte Buchstabe im Wort sind identisch, da sie beim Twisten unverändert bleiben.
2. Beide Wörter haben die gleiche Länge und beinhalten dieselben inneren Zeichen, nur in einer anderen Reihenfolge.

So passt „Prarmgom“ mit dem Originalwort „Programm“ überein, denn beide beginnen mit „P“, enden mit „m“ und enthalten im Inneren die Zeichen „r“ (2 mal), „o“, „g“, „a“ und „m“. „Pmarmgor“ (endet auf „r“) oder „Prermgom“ (enthält ein „e“) würden demnach nicht zu „Programm“ passen.

Man könnte natürlich zu jedem eingegebenen Wort alle möglichen Permutationen der inneren Zeichen generieren und im Wörterbuch suchen; da das allerdings zu einer Laufzeit von $\mathcal{O}(n!)$ führt, ist dieser Ansatz gerade bei längeren Wörtern nicht praktikabel (schon das Wort „Informatiker“ hat über 3,6 Millionen mögliche Twists).

Um eine effizientere Lösung zu finden, machen wir uns folgende Dinge klar:

- Das Twisten ist symmetrisch. Beispiel: Da „Prarmgom“ ein Twist von „Programm“ ist, ist „Programm“ auch ein Twist von „Prarmgom“.
- Das Twisten ist transitiv, also wenn A ein Twist von B und B ein Twist von C ist, dann ist A auch ein Twist von C. Beispiel: Da „Prarmgom“ ein Twist von „Programm“ ist und „Programm“ ein Twist von „Pmargorm“ ist, ist „Prarmgom“ auch ein Twist von „Pmargorm“.

Diese beiden Eigenschaften können wir uns zu nutze machen, indem wir für jedes Wort eine Art „Basisform“ berechnen[4]. Am einfachsten lässt sich dies bewerkstelligen, wenn wir die inneren Zeichen einfach sortieren (diese Funktion nennen wir im folgenden h):

```
1 h(wort w):
2     inner := sort(w[2], w[3], ..., w[n − 1])
3     return (w[1] als Kleinbuchstabe) + inner + w[n]
```

Wir müssen uns außerdem darüber Gedanken machen, wie wir Groß- und Kleinschreibung behandeln. Ein am Anfang eines Satzes stehendes „Dsas“ entspricht sicherlich dem im Wörterbuch enthaltenen „dass“, obwohl sie nach unserer bisherigen Auffassung streng genommen keine Twists voneinander sind. Der einfachste Weg damit umzugehen, ist es, die Groß- und Kleinschreibung des ersten Zeichens bei der Berechnung der Basisform einfach zu ignorieren.

[4] Man kann hier auch von einer Hashfunktion sprechen, also einer Funktion, die eine große Menge an Eingaben auf eine kleine Menge an Ausgaben abbildet. Dabei bekommen „gleiche“ (also in unserem Fall ineinander twistbare) Wörter auch gleiche Hashwerte.

Insgesamt ist für unser Beispielwort „Programm“ dann $h(\text{"}Programm\text{"}) = \text{"}pagmorrm\text{"}$ und auch $h(\text{"}Prarmgom\text{"}) = \text{"}pagmorrm\text{"}$. Da die beiden Basisformen gleich sind, wissen wir dank Symmetrie und Transitivität auch, dass „Prarmgom“ ein Twist von „Programm“ ist.

Genau diesen „Umweg“ über das innen sortierte Wort können wir nun benutzen, um einen Index aus dem uns zur Verfügung gestellten Wörterbuch aufzubauen. Wir benötigen dafür eine Lookup-Datenstruktur (in den meisten Sprachen als Hashtabelle implementiert), in der wir jedes Wort w mit $h(w)$ als Schlüssel abspeichern. Das könnte dann folgendermaßen aussehen:

Schlüssel	Werte
...	...
pagmorrm	programm
aektur	akuter, akteur
hadelnte	handelte, haltende
haefr	hafer
...	...

Um nun ein Wort w zu enttwisten, müssen wir einfach nur seine Basisform $h(w)$ berechnen und im Index nachschauen, welche Wörter aus dem Wörterbuch dieselbe Basisform haben. So würden wir beim Wort „Prrmgoam“ die Basisform $h(\text{"}Prrmgoam\text{"}) = \text{"}pagmorrm\text{"}$ erhalten und über den Index dann das Wörterbuchwort „Programm“ finden. Dabei muss man auf die entsprechende Behandlung der Fälle achten, in denen der Schlüssel gar nicht oder mehrfach im Index vorkommt. Außerdem muss auch hierbei wieder auf Sonderzeichen geachtet werden.

Insgesamt müssen wir also beim Enttwisten, nachdem wir das Wörterbuch eingelesen und indiziert haben, wie beim Twisten die Eingabe in einzelne Wörter aufteilen, dann diese mit Hilfe ihrer Basisform enttwisten und die resultierenden Wörter in der ursprünglichen Reihenfolge ausgeben.

Laufzeitverhalten

Sei m die Länge des Wörterbuchs und n die Länge der Eingabe. Der erste Teil des Enttwisten-Algorithmus liest das Wörterbuch ein und indiziert es. Dafür werden die Basisformen berechnet (wofür wir sortieren müssen, was $\mathcal{O}(w \log w)$ Zeit benötigt) und diese in eine Lookup-Datenstruktur eingetragen. Mit der Annahme, dass diese Datenstruktur als Hashtabelle implementiert ist, ergibt sich dadurch eine Laufzeit von $\mathcal{O}(m \log m)$, da das Wörterbuch potentiell aus einem einzigen Wort der Länge m bestehen kann.

Für das Enttwisten müssen wir wieder für alle Wörter der Eingabe die Basisform berechnen und im Index danach suchen. Das ergibt auch hier eine Laufzeit von $\mathcal{O}(n \log n)$, womit das gesamte Verfahren eine Laufzeit von $\mathcal{O}(m \log m + n \log n)$ besitzt.

Unter der Annahme, dass die Wörter aus einer natürlichen Sprache stammen und damit in ihrer Länge konstant beschränkt sind, fallen die logarithmischen Faktoren weg; die Laufzeit ist in diesem Fall linear in m und n: $\mathcal{O}(m+n)$.

Beispiele

Für das auf dem Aufgabenblatt enthaltene Beispiel „twist1“ sowie drei weitere Beispiele ergibt das Enttwisten folgende Ergebnisse:

Eingabe	Ausgabe
twist1	der Twsit? (englisch tiwst? = drehung, verdrehung) war ein Mdaotenz? im 4/4-takt, der in den [frühen,führen] 1960er jahren populär wurde und zu rock'n'Roll?, Ryhthm? and blues oder spezieller Twsit?-musik getanzt wird.
beispiel2	es existiert ein verfahren, womit man einen text maschinell unlesbar machen kann, sodass ihn ein mensch trotzdem noch lesen kann. dabei werden die zeichen, bis auf das erste und das letzte, jedes einzelnen wortes in eine beliebige reihenfolge gebracht. dadurch wird es einem computer erschwert, den text mithilfe von wortlisten zu analysieren, gleichzeitig kann ein mensch den text beim lesen trotzdem noch intuitiv verstehen.
beispiel3	das Z ist der letzte buchstabe des lateinischen alphabets. er kommt in deutschen texten relativ selten vor, findet aber in [anreden,anderen] sprachen, wie zum beispiel englisch oder polnisch, größere verwendung. er ist verwandt mit dem griechischen zeta und hat sich ursprünglich aus dem symbol einer stichwaffe entwickelt.
beispiel4	beim maschinellen lernen wird in der regel versucht, automatisiert aus einer menge an beispielen einen aneleemgin? zusammenhang herauszufinden. die theoretische grundlage dieser verfahren bildet das berechnen von empirischen wahrscheinlichkeitsverteilungen, aus denen sich dann die gewünschten informationen [schleißen,schließen] lassen.

2.3 Bewertungskriterien

Die Bewertungskriterien (Fettdruck) vom Bewertungsbogen werden hier näher erläutert (Punktabzug in []).

- (1) [-1] **Dokumentation sehr unverständlich bzw. unvollständig**.
- (2) [-1] **Vorgegebenes Dateneingabeformat mangelhaft umgesetzt**:
  Das vorgegebene Eingabeformat sollte eingehalten und in eine geeignete interne Darstellung umgesetzt werden. Insbesondere sind lineare Datenstrukturen für die Darstellung der Texte als Folgen von Wörtern geeignet. Auch sollten die Groß- und Kleinschreibung von Wörtern berücksichtigt werden sowie die Interpunktionszeichen und andere (Sonder-)Zeichen im Text korrekt behandelt werden.
- (3) [-1] **Lösungsverfahren fehlerhaft**:
  Die Verfahren zum Twisten und Enttwisten sollten beide korrekt sein. Die Anfangs- und Endbuchstaben der Wörter sollten an ihrer ursprünglichen Position bleiben, alle anderen Buchstaben im Inneren der Wörter müssen zufällig gemischt werden. Sollte statt einer zufälligen Umordnung (wie gemäß der Aufgabenstellung) die inneren Buchstaben nur umsortiert werden, so gibt dies ohne ausreichende Begründung (z. B. hinsichtlich des ähnlichen Informationsverlust im Wort) einen Punkt Abzug. Es ist grundsätzlich in Ordnung, wenn z.B. „Dass“ und „dass“ wegen Groß-/Kleinschreibung beim Enttwisten als verschieden aufgefasst werden. Eine Aussage über die Qualität des Verfahrens zum Enttwisten wird nicht erwartet.
- (4) [-1] **Strategie für Ähnlichkeitsvergleich mangelhaft / fehlt**:
  Es wurde eine Strategie für das Enttwisten angewandt, die den Ähnlichkeitsvergleich zweier Wörter sinnvoll bewerkstelligt.

- (5) [-1] **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**:
  Nur besonders umständliche oder ineffiziente Verfahren bzw. Implementierungen führen zu Punktabzug. Da der sehr einfache Algorithmus zum Twisten mittels der Datenstruktur einer Liste quadratische Laufzeit besitzt, sollte das Verfahren fürs Twisten auf keinen Fall langsamer sein. Fürs Enttwisten sollte das Generieren und Überprüfen aller Permutationen vermieden werden. Das Durchsuchen eines Wörterbuchs in linearer Laufzeit ist jedoch in Ordnung.
- (6) [-1] **Programmausgabe ungeeignet**:
  Eine geeignete Ausgabe der Ergebnistexte des Twisten bzw. Enttwisten sollte vorhanden sein. Wenn ein Wort nicht enttwistet werden konnte, muss dies in der enttwisteten Lösung nicht kenntlich gemacht sein.
- (7) [-1] **Beispiele fehlerhaft bzw. zu wenige (mind. 2/4 und 1)**:
  Es wird die Dokumentation der Ergebnisse von mind. 2 der 4 vorgegebenen Beispiele erwartet. Außerdem muss mind. 1 Beispiel für das Entwisten aussagekräftig genug sein, um einen Punktabzug zu vermeiden. Das 5. Beispiel darf aufgrund seiner Textlänge stark verkürzt in der Dokumentation enthalten sein.