Vorwort: Für alle zu implementierenden Aufgaben werden entsprechende Demonstrationsmöglichkeiten erwartet.

1 Editierdistanz

1.1 Was ist eine Editierdistanz?

Als *Editierdistanz* bezeichnet man die Anzahl der Operationen, die man benötigt, um einen String s in einen String t umzuwandeln. Was eine Operation ist, hängt vom jeweils verwendeten Algorithmus ab. In manchen Fällen werden auch die benötigten Operationen selbst angegeben.

1.2 Levenshtein-Distanz

Die Levenshtein-Distanz (in ihrer Grundform) beinhaltet folgende Operationen:

- `insert(i,c):` fügt den Buchstaben `c` an der Stelle `i` ein.
- `delete(i):` entfernt den Buchstaben an der Stelle `i`.
- `replace(i,c):` ersetzt den Buchstaben an der Stelle `i` durch `c`.

1.2.1 Algorithmus

1. Zunächst wird aus den beiden Strings s und t der Länge n und m eine Matrix M der Größe $n+1 \times m+1$ gebildet. An einer Achse werden die Buchstaben von s, auf der anderen Achse die Buchstaben von t aufgetragen (jeweils um eine Stelle nach hinten verschoben. Die Position (0|0) bleibt also zunächst leer!). In der ersten Spalte bzw. Zeile werden die Werte 0 bis n, bzw. 0 bis m eingetragen. Das Ergebnis sollte wie in 1 aussehen.

2. Danach wird zeilenweise durch die freien Felder $(i|j)$ gegangen. Für jedes Feld werden die folgende Werte berechnet:

   (a)
   $$c := \begin{cases} 0, & s[i] = t[j] \\ 1, & \text{sonst} \end{cases}$$

   (b) rep = M[i-1][j-1] + c

   (c) ins = M[i][j-1] + 1

   (d) del = M[i-1][j] + 1

   Und dann ist $M[i][j] := min(rep, ins, del)$

		i	n	t	e	r	e	s	t	s
	0	1	2	3	4	5	6	7	8	9
i	1									
n	2									
d	3									
u	4									
s	5									
t	6									
r	7									
y	8									

Figure 1: Leere Martrix für $s =$ “industry” und $t =$ “interests”.

3. Füllt man die Matrix konsequent nach diesem Schema aus, befindet sich im letzten Feld $M[n + 1][m + 1]$ die Editierdistanz (siehe auch 2, rot dargestellt).

4. *Optional:* verfolgt man den Weg durch die Matrix von $M[n + 1][m + 1]$ zurück nach $M[0][0]$ und wählt dabei als nächstes jeweils das benachbarte Feld, mit den billigsten Kosten, so ergeben sich daraus die Operationen:

   (a) Geht man nach links, muss man das Zeichen $s[i]$ *löschen*.

   (b) Geht man nach oben, muss man das Zeichen $t[j]$ an der Stelle $s[i]$ *einfügen*.

   (c) Geht man nach links oben, muss man das Zeichen $s[i]$ durch $t[j]$ ersetzen, falls die Zeichen sich unterscheiden

   Lässt man diesen Schritt weg, ergibt sich eine Performanceoptimierung. Anstatt Speicher für die gesamte Matrix zu allokieren, genügt es, zwei Reihen zu reservieren und dann die Werte immer wieder zu überschreiben. Diese optimierung wirkt sich in verschiedenen Sprachen verschieden stark auf die Performance aus, je nach dem wie die verwendete Sprache mit Speicherreservierungen umgeht.

1.3 Aufgaben

Im RELAX findet ihr zwei Dateien `scrambled.txt` und `reference.txt`.
`scrambled.txt` beinhaltet händisch gepflegte Informationen eines Sammlers über seine *Magic: The Gathering* Kartensammlung. Jeder Eintrag beinhaltet den Namen der Karte, ihre Manakosten, die Kummulativen Manakosten ihren Typ, und wie oft der Sammler die Karte besitzt, jeweils durch eine Pipe (“`|`”) voneinander getrennt. Beim Umzug auf eine neue Festplatte wurde die Datei allerdings beschädigt, sodass einige Zeichen der Namen unleserlich wurden (“`?`”) und andere Zeichen sogar den Platz getauscht haben.
`reference.txt` beinhaltet eine vollständige Referenzliste aller Kartennamen.

		i	n	t	e	r	e	s	t	s
	0	1	2	3	4	5	6	7	8	9
i	1	0	1	2	3	4	5	6	7	8
n	2	1	0	1	2	3	4	5	6	7
d	3	2	1	1	2	3	4	5	6	7
u	4	3	2	2	2	3	4	5	6	7
s	5	4	3	3	3	3	4	4	5	6
t	6	5	4	3	4	4	4	5	4	5
r	7	6	5	4	4	4	5	5	5	5
y	8	7	6	5	5	5	5	6	6	6

Figure 2: Levenshtein-Distanz von s = “industry” und t = “interests”

Figure 3: Eine *Magic: The Gathering*-Spielkarte. Rot: Name der Karte, Blau: der Typ der Karte, Grün: die Manakosten (hier: `{3}{G}{B}`) und zugleich die *cumulative mana costs* (cmc), berechnet aus der Summe der Manakosten, wobei Buchstaben als 1 zählen (also hier: 5).

Eure Aufgabe ist es nun, mithilfe der Referenzliste und der Levenshtein-Distanz die Datenbank des Sammlers wiederherzustellen.

1. Implementiert den Algorithmus zur Berechnung der Levenshtein-Distanz (LD) (5P).
2. Implementiert eine Datenstruktur `Card`, welche die Attribute `name`, `mana`, `cmc`, `type` und `count` beinhaltet (5P). Berücksichtigt dabei die Prinzipien der Objektorientierung.
3. Implementiert eine Methode, um aus `scrambled.txt` eine Liste von `Card`s zu generieren. Haltet euch dabei an das Format, das im Einleitungstext beschrieben ist (3P).
4. Implementiert eine Methode, um `Card`s wieder im oben angegebenen Format in eine Datei zu speichern (3P).
5. Implementiert eine Methode, um die Namen der Karten aus der Referenzliste zu lesen. In jeder Zeile steht genau ein Name (2P).
6. Implementiert eine Methode, welche den Namen einer `Card` c wiederherstellt. Berechnet dazu c so lange die LD aus deren Namen und den Namen aus der Referenzliste, **bis** die LD kleiner als 26,75% der Länge des Kartennamens ist. Findet ihr so einen Eintrag, ersetzt ihr den Namen von c durch den Referenzwert (5P).
7. Schreibt die reparierten Karten wieder in eine Datei (2P).
8. Erstellt eine Dokumentation als PDF, die die Lösung als vollständiges UML Klassendiagramm darstellt, das mit Sparx Enterprise Architect erstellt wurde (4P). Gebt die Metriken Lines of Code pro Klasse und Gesamt, sowie Documentation per Class an (1P) und schreibt eine kurze Beschreibung der Funktionsweise der Lösung (3P), (Insges. 8P)