Hochschule Karlsruhe
University of Applied Sciences

Fakultät für **Informatik**
und Wirtschaftsinformatik

Prof. Dr. Carsten Sinz
Lehrgebiet Software-Entwicklung

Labor Betriebssysteme: Aufgabe 3
RLE und Bit-Operationen

Hinweise:

- Bearbeiten Sie die Teilaufgaben in Teams und laden Sie Ihre Antworten in ILIAS hoch. Zusätzlich müssen Sie Ihre Abgaben auch im Labor präsentieren, dabei müssen alle Teammitglieder anwesend sein.
- Für Teilaufgabe 3 finden Sie im Ilias eine Vorlage (`code.zip`), die Sie nur noch vervollständigen müssen.

In diesem Projekt wird das Konzept des Run-Length Encoding (RLE) zur Datenkompression untersucht, grundlegende Kenntnisse über bitweise Operationen aufgefrischt und das Serialisieren einer RLE-Implementierung realisiert.

Teilaufgabe 1 (RLE Grundlagen):

Run-Length Encoding (RLE) ist eine sehr einfache Form der **Datenkomprimierung**, bei der Datenläufe (Sequenzen, in denen derselbe Datenwert in vielen aufeinanderfolgenden Datenelementen vorkommt) als ein einziger Datenwert und eine Zählung gespeichert werden. Dies ist besonders nützlich bei Daten, die viele solcher Läufe enthalten.

Betrachten wir ein einfaches Beispiel mit einer Textkette: `AAAAABBBBCCCC`. Mit RLE könnte diese Zeichenfolge als `5A4B4C` komprimiert werden, wobei die Zahl vor jedem Zeichen die Anzahl der Wiederholungen dieses Zeichens angibt.

RLE ist keine sehr ausgefeilte Komprimierungsmethode und erreicht im Vergleich zu anderen Methoden wie der **Huffman-Kodierung** oder **LZ77/LZ78** keine hohen Kompressionsraten. Es kann jedoch in bestimmten Szenarien verwendet werden, in denen Einfachheit und Geschwindigkeit wichtiger sind als die Kompressionsrate. Es wird häufig in **Grafikdateiformaten** und beim Rendern von Grafiken verwendet. Bitmap-Bilder oder TIFF-Dateien können beispielsweise mit RLE komprimiert werden.

RLE wird auch häufig verwendet, um für die Bildsegmentierung Segmentierungsmasken in Computer-Vision-Anwendungen zu komprimieren. Bei der Bildsegmentierung wird ein Bild in Segmente oder Regionen unterteilt, von denen jeden z. B. verschiedenen Objekten oder Teilen von Objekten im Bild entspricht ($\rightarrow$ *inhaltlich zusammenhängende Regionen*). Eine Segmentierungsmaske ist dabei oft ein binäres Bild, das angibt, welche Pixel zu etwas gehören und welche nicht. In einer solchen binären Segmentierungsmaske werden Pixeln, die zu dem betreffenden Etwas gehören, den Wert 1 zugewiesen, während alle anderen den Wert 0 erhalten. Dies führt zu einer großen Anzahl aufeinanderfolgender 0en und 1en, wodurch die Segmentierungsmaske ein guter Kandidat für RLE ist.

In Abbildung 1 sind verschiedene Arten von Segmentierungen zu sehen, die üblicherweise im Bereich des maschinellen Lernens verwendet werden. Die oft einzelnen binären Segmentierungsmasken werden dabei zur besseren Visualisierung eingefärbt und dann miteinander kombiniert, sodass wieder

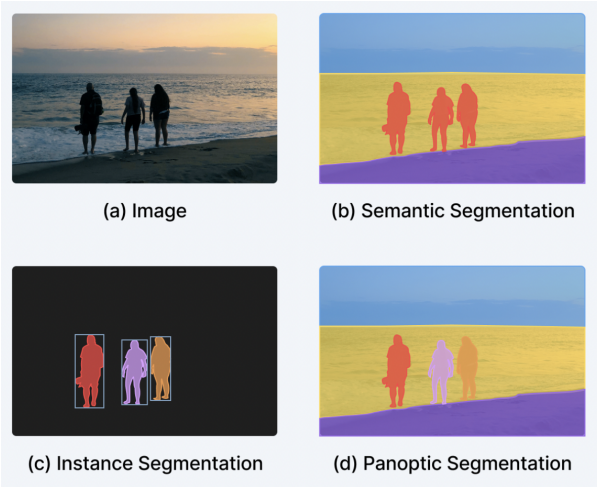

Abbildung 1: Visualisierung der verschiedenen Segmentierungsaufgaben.
Quelle: https://www.v7labs.com/blog/panoptic-segmentation-guide

ein ganzes Bild entsteht. Bei der **Semantic Segmentation** werden verschiedene Menschen oder Objekte nicht unterschieden, bei **Instance Segmentation** werden wiederum alle Teile des Bilds ignoriert, die nicht zählbar sind. **Panoptic Segmentation** ist eine Kombination von Semantic und Instance Segmentation.

Bearbeiten Sie die folgenden Aufgaben und halten Sie die Ergebnisse schriftlich fest.

a) Kodieren Sie `HEEEEELLLLOOOOOOO`, so wie es im Text oben beschrieben wurde. Kodieren Sie dann `ADADADADADAD`. Was fällt Ihnen auf?

b) Stellen Sie sich vor, Sie müssten Segmentierungsmasken in einem RLE-Format speichern und die zu segmentierenden Objekte sind Personen. Wo würden Sie im Bild anfangen (z. B. rechts oben) und würden Sie die Pixel in vertikaler oder horizontaler Richtung durchlaufen? Wenn Sie nun zusätzlich Masken von Autos kodieren müssten, würden Sie nun ihr Verfahren ändern/erweitern? Argumentieren Sie.

Teilaufgabe 2 (Bitweise Operationen):

In der Programmierung werden binäre Zahlen auf der Ebene ihrer einzelnen Bits mit einer bitweisen Operation bearbeitet. Nachfolgend werden die grundlegenden Operationen gezeigt, dabei sind die Zahlen im Binärsystem zu lesen, sofern nicht anders ersichtlich:

- **AND (&)**: Die UND-Operation nimmt zwei Bits und gibt 1 zurück, wenn beide Bits 1 sind. Andernfalls gibt sie 0 zurück.

  `Beispiel: 1 & 1 = 1, 100 & 000 = 000, 000000 & 100111 = 0, 010 & 110 = 010`

- **OR (|)**: Die ODER-Verknüpfung nimmt zwei Bits und gibt 1 zurück, wenn mindestens eines der Bits 1 ist, andernfalls gibt sie 0 zurück.

  `Beispiel: 1 | 0 = 1, 0 | 0 = 0, 1110 | 0011 = 1111, 1101 | 1111 = 1111`

- **XOR (^)**: Die XOR-Verknüpfung (exklusive ODER-Verknüpfung) nimmt zwei Bits und gibt 1 zurück, wenn genau eines der Bits 1 ist. Andernfalls gibt sie 0 zurück.

  ```
  Beispiel: 101 ^ 001 = 100, 110 ^ 001 = 111, 11 ^ 11 = 00, 011 ^ 111 = 100
  ```

- **NOT (~)**: Die NOT-Operation ist eine unäre Operation, die das Bit ihres Operanden umkehrt. Zum Beispiel ergibt die NOT-Operation auf 0 eine 1, und die NOT-Operation auf 1 eine 0. Es gilt zu beachten, dass bei einer 32-Bit Ganzzahl auch alle 32 Bits umgekehrt werden.

  ```
  Beispiel: ~1 = 0, ~0 = 1, ~1011 = 0100
  ```

- **Bitverschiebungen (>>, <<)**: Bitverschiebungen verschieben die Bits einer Binärzahl nach links oder nach rechts. Eine Linksverschiebung (<<) fügt Nullen auf der rechten (niederwertigsten) Seite der Zahl hinzu und multipliziert sie bei jeder Verschiebung effektiv mit 2. Bei einer Rechtsverschiebung (>>) werden Bits von rechts entfernt, wodurch die Zahl bei jeder Verschiebung durch 2 geteilt wird.

  ```
  Beispiel: 13 << 1 = 26 (im Binärformat: 1101 << 1 = 11010),
            13 >> 1 =  6 (im Binärformat: 1101 >> 1 = 110)
  ```

Diese Operationen können verwendet werden, um Bits in einer Binärzahl zu manipulieren, was häufig bei Aufgaben wie Serialisierung oder Komprimierung erforderlich ist.

a) Gehen Sie davon aus, dass Sie die letzten vier Bits auf 0 setzen und die ersten vier Bits unverändert lassen wollen. Wie würden Sie dies mit einer bitweisen Operation tun?

b) Wenn Sie eine 8-Bit-Zahl haben und möchten die ersten 4 Bits mit den letzten 4 Bits vertauschen (z. B. wird aus 01101010 nun 10100110). Wie können Sie dies mit den genannten bitweisen Operationen erreichen?

c) Angenommen, Sie haben eine 8-Bit-Zahl 11010100. Sie möchten das 4. und 5. Bit aus dieser Zahl extrahieren (von rechts gezählt, beginnend mit 1). Wie können Sie dies mit bitweisen Operationen erreichen?

d) Nehmen wir eine 8-Bit-Zahl 10011011. Sie möchten nur das 3. Bit (von rechts, beginnend mit 1) auf 1 setzen. Mit welchen der genannten bitweisen Operationen können Sie dies erreichen und wie?

Teilaufgabe 3 (Serialisierung von RLE): Es ist eine **RLE-Implementierung** gegeben, die im Endeffekt eine **verkettete Liste** darstellt, deren Listenelemente jeweils eine Anzahl beinhaltet. Dadurch, dass nur zwei Werte, nämlich 0 oder 1, gezählt werden, gibt der Index des Listenelements das zu kodierende Bit an. Das erste Listenelement steht dabei repräsentativ für die Anzahl an 0-Bits, das zweite für die Anzahl an 1-Bits, das dritte wieder für 0-Bits, das vierte für 1-Bits usw. Ist das allererste Bit eine 1, besitzt die Liste trotzdem als ersten Eintrag die Anzahl an 0en (in diesem Fall dann 0). So wird sichergestellt, dass die Bits immer richtig interpretiert werden können, da das zu zählende Bit nicht extra gespeichert wird.

Ihre Aufgabe besteht darin, diese **RLE-Struktur** zu **serialisieren**. **Serialisierung** ist der Prozess der Umwandlung von Datenstrukturen oder Objektzuständen in ein Format, das gespeichert oder über ein Netzwerk übertragen werden kann und später durch einen **Deserialisierungsprozess** wieder in

eine In-Memory-Datenstruktur zurückverwandelt werden kann.

Ziel der Serialisierung ist es, **komplexe Datenstrukturen**, wie beispielsweise Listen, Bäume oder Graphen, so zu speichern oder zu übertragen, dass sie an einem anderen Ort oder zu einem späteren Zeitpunkt genau in ihrer ursprünglichen Form wiederhergestellt werden können. Das ist besonders nützlich in Anwendungen wie **verteilten Systemen**, wo Daten zwischen verschiedenen Prozessen oder Computern ausgetauscht werden müssen, oder beim **Speichern des Anwendungsstatus** auf der Festplatte, um später wieder geladen zu werden.

Eine naheliegende Möglichkeit wäre es, einfach alle Werte der gezählten Bits, die z. B. als 8-Byte unsigned Integer vorliegen, genau so wieder in eine Datei zu schreiben. Bei häufigeren Wechseln der Bits, steigt die entstehende Dateigröße aber recht stark an. Um diesem Umstand etwas entgegenzuwirken, sollen Sie daher die Zählungen auf eine bestimmte Weise serialisieren.

Gehen Sie mit einer **kleinsten Einheit von 4-Bit** aus. Das Bit ganz links gibt dabei an, welche Art von Bit (0 oder 1) gerade kodiert wird. Das Bit rechts daneben steuert die Anzahl der zur Verfügung stehenden Bits, ist es auf 0 werden nur die nächsten zwei Bits berücksichtigt, ist es jedoch auf 1, wird zusätzlich die nächste 4-Bit Einheit zum Zählen hinzugenommen. Die übrigen zwei (oder sechs) Bits geben an, wie viele des ersten Bits hiermit kodiert werden.

```
Beispiel 1:
  Daten:   0001 1000
  Kodiert: 0011 1010 0011

Beispiel 2:
  Daten:   0000 0011 1101 1111 1111 1111 1111 1111
  Kodiert: 0100 0110 1100 0100 0001 1101 0101
```

a) Welche Probleme können bei dieser Serialisierung auftreten? Sind diese lösbar oder nicht ohne größere Änderungen vermeidbar?

b) Im Ilias finden Sie eine Code-Vorlage. Implementieren Sie für das oben vorgestellte Verfahren die Serialisierung und Deserialisierung in dieser Vorlage. Sie müssen keine Änderungen am Rest des Codes vornehmen, lediglich die beiden zuständigen Methoden implementieren.

c) Testen Sie das Serialisieren und Deserialisieren mit beliebigen Dateien. Stellen Sie sicher, dass die dekomprimierten Dateien wieder exakt dem Original entsprechen. Bei größeren Dateien können Sie z. B. die md5-Summen berechnen lassen (`md5sum file1.txt file2.txt`, unter Mac `md5 file1.txt file2.txt`, unter Windows geht jeweils eine Datei mittels `CertUtil -hashfile file1.txt MD5`) und dann die Werte miteinander vergleichen.