UTF-8 Kodierung

UTF-8 ist eine Methode, die Nummern von Unicode-Zeichen zu kodieren (d.h. effektiv zu speichern). Der Vorteil dieser Kodierung ist, dass ASCII-Zeichen (woraus der grösste Teil Westeuropäischer Texte besteht) zu ASCII kompatibel in nur einem Byte gespeichert werden.

Für Nummern, die mehr als 7 Bits benötigen, wird folgendes Schema gebraucht. Ein Startbyte gibt an, mit wie vielen Bytes die Nummer kodiert wird. Die Folgebytes sind ebenfalls speziell gekennzeichnet und tragen nur 6 Bit an effektiver Information.

Startbyte 0b110x'xxxx (1 Folgebyte), 0b1110'xxxx (2 Folgebytes), 0b1111'0xxx (3 Folgebytes). Die 'x' stehen für die effektive Information in Big-Endian.

Folgebyte 0b10xx'xxxx. Die 'x' stehen für die effektive Information.

Beispiele:

Eine 9-Bit lange Nummer a'bcde'fghi wird wie folgt kodiert (wobei 1 führendes Null eingefügt wird):

0b110'**0**abc, 0b10de'fghi

Eine 12-Bit lange Nummer abcd'efgh'ijkl wird wie folgt kodiert (wobei 5 führende Nullen eingefügt werden):

0b1110'**0000**, 0b1**0**ab'cdef, 0b10gh'ijkl

Umrechnung Binär und Hexadezimal

Das Hexadezimalsystem ist das 16er-System, mit Ziffern 0 bis 15, wobei die Ziffern 10 bis 15 mit den Buchstaben 'a' bis 'f' dargestellt werden.

Hexadezimalzahlen werden mit dem Prefix '0x' gekennzeichnet, Binärzahlen mit '0b'.

Mit 4 Bits können die Ziffern 0=0x0=0b0000 bis 15=0xf=0b1111 dargestellt werden, d.h. ein 4er-Paket entspricht genau eine Hexadezimalziffer.

Aufgabe 0.1 Schreiben Sie die Bytes auf, mit der das Unicode-Zeichen mit der Nummer 0x1f4a9 in UTF-8 kodiert wird. Geben Sie die Bytes einerseits binär an, wobei die codierenden Bits markiert werden soll, anderseits auch hexdezimal.

0.1 Bitweise Operationen

`&` für and, `|` für or, für not, `^` für xor, `<<` für shift left, `>>` für shift right

Aufgabe 0.2 Schreiben Sie eine Funktion `achtBisElf` in Python oder JavaScript, die eine Unicode-Nummer zwischen 8 und 11 Bit in UTF-8 umrechnet. Das Resultat soll ein Array mit den entsprechenden Ganzzahlen sein.

Zur Kontrolle:

```
achtBisElf(281) liefert [196, 153] = ['0b11000100', '0b10011001']
achtBisElf(513) liefert [200, 129] = ['0b11001000', '0b10000001']
achtBisElf(1234) liefert [211, 146] = ['0b11010011', '0b10010010']
achtBisElf(2022) liefert [223, 166] = ['0b11011111', '0b10100110']
```

0.2 Lösungen

Hinweise zu den Symbolen:

⚒ Diese Aufgaben könnten (mit kleinen Anpassungen) an einer Prüfung vorkommen. Für die Prüfungsvorbereitung gilt: “If you want to nail it, you’ll need it”.

✸ Diese Aufgaben sind wichtig, um das Verständnis des Prüfungsstoffs zu vertiefen. Die Aufgaben sind in der Form aber eher nicht geeignet für eine Prüfung (zu grosser Umfang, nötige «Tricks», zu offene Aufgabenstellung, etc.). **Teile solcher Aufgaben können aber durchaus in einer Prüfung vorkommen!**.

Diese Aufgaben sind dazu da, über den Tellerrand hinaus zu schauen und/oder die Theorie in einen grösseren Kontext zu stellen.

Lösung zu Aufgabe 0.1 ex-unicode-to-utf8

0x1f4a9 = 0b 1’1111’0100’1010’1001, es werden also 17 kodierende Bits benötigt.

Pro Folgebyte können 6 Bits kodiert werden, d.h. es werden mindestens 2 benötigt. Im Startbyte für 2 Folgebytes können aber nur 4 Bits kodiert werden, es braucht also 3 Folgebytes. Die Kodierung ist also

Binär: 0b1111’0**000**, 0b10**01’1111**, 0b10**01’0010**’, 0b10**10’1001**

Hexadezimal: 0xf09f92a9

Lösung zu Aufgabe 0.2 ex-acht-bis-elf-bit-unicode-in-utf8

```python
def achtBisElf(u):
    folgeByte = 0b10000000 | (u & 0b111111)
    startByte = 0b11000000 | ((u >> 6) & 0b11111)
    return [startByte, folgeByte]

# Test der Funktion
for u in [281, 513, 1234, 2022]:
    b = achtBisElf(u)
    bb = [bin(x) for x in b]
    print("achtBisElf("+str(u)+") liefert "+str(b)+" = "+str(bb))
```

```javascript
window.addEventListener('load', function() {
  let achtBisElf = function(u) {
    let folgeByte = 0b10000000 | (u & 0b111111);
    let startByte = 0b11000000 | ((u >> 6) & 0b11111);
    return [startByte, folgeByte];
  };

  // Test der Funktion
  for (const u of [281, 513, 1234, 2022]) {
    let b = achtBisElf(u);
    let bb = b.map(x => "0b"+x.toString(2));
    console.log(`achtBisElf(${u}) liefert [${b}] = [${bb}]`);
  }
});
```