M114-Arbeit - 2023-11-24

Verfasser: Noé Farese und Aurel Schmid

Aufgabe 1

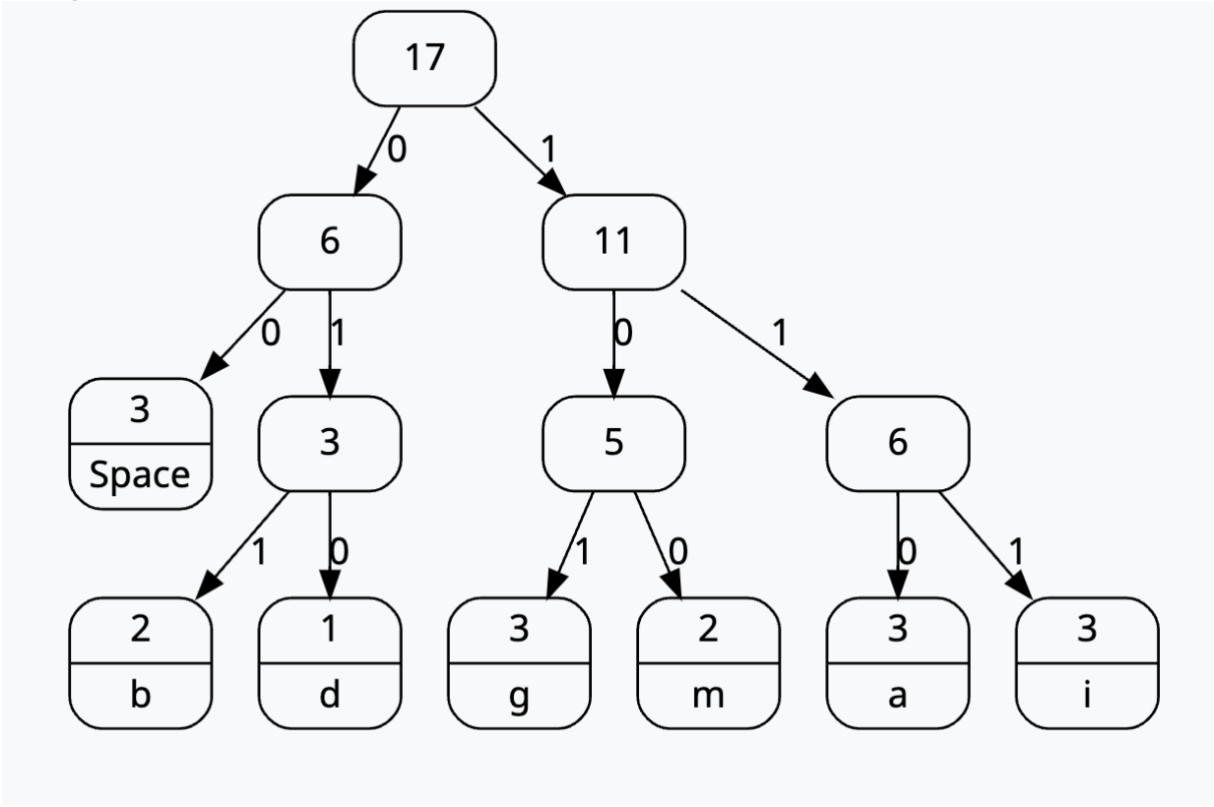
Es handelt sich dabei um einen Strichcode, der aus einer Gruppe von fünf Elementen besteht, von denen zwei aktiv sind.

Hier ist eine kurze Erläuterung des 2-aus-5-Codes:
Aufbau: Jedes Zeichen des 2-aus-5-Codes besteht aus fünf Strichen, von denen zwei aktiv sind (dunkel) und drei inaktiv (hell). Die Position der aktiven Striche variiert, um unterschiedliche Zeichen darzustellen.

Hamming-Abstand: Der Hamming-Abstand ist die Anzahl der Bits, die sich zwischen zwei Codewörtern unterscheiden. Im 2-aus-5-Code beträgt der minimale Hamming-Abstand 2. Das bedeutet, dass zwei verschiedene Zeichen im Code mindestens an zwei Positionen unterschiedliche aktive Striche haben.

Redundanz: Der 2-aus-5-Code hat eine gewisse Redundanz, da nicht alle möglichen Kombinationen von fünf Strichen für Zeichen genutzt werden. Dies ermöglicht die Erkennung von Fehlern, wenn zum Beispiel durch Beschädigungen ein Strich nicht korrekt gelesen wird. Die Redundanz ist jedoch begrenzt.

Aufgabe 2



g	101
a	110
b	011
i	111
-	00
m	100
d	010

Siehe auch Aufgabe 2 auf dem Blatt.

Aufgabe 3

UTF-16 ist 3KB größer als ASCII, weil UTF-16 mehr Bits verwendet, um Zeichen zu repräsentieren. ASCII (American Standard Code for Information Interchange) verwendet 7 Bits, was eine Begrenzung auf 128 verschiedene Zeichen ermöglicht. Später wurde ASCII auf 8 Bits erweitert, was als Extended ASCII bekannt ist, und ermöglichte die Darstellung von 256 Zeichen.

UTF-16 (Unicode Transformation Format, 16-Bit) hingegen verwendet 16 Bits (2 Bytes) für die Darstellung von Zeichen. Dies ermöglicht eine viel größere Anzahl von Zeichen (bis zu 65,536), was notwendig ist, um eine Vielzahl von Sprachen und Sonderzeichen darzustellen, die in der Welt existieren.

Die größere Größe von UTF-16 im Vergleich zu ASCII kann dazu führen, dass UTF-16-codierte Texte mehr Speicherplatz benötigen, insbesondere wenn der Text hauptsächlich aus ASCII-Zeichen besteht. Der Vorteil von UTF-16 liegt jedoch in seiner Fähigkeit, eine breitere Palette von Zeichen und Sprachen zu unterstützen, was besonders wichtig in globalen Anwendungen ist.

Der Kompressionsfaktor (K) lässt sich wie folgt berechnen:

K = Ursprüngliche Grösse/Komprimierte Grösse

Werte einsetzten:

K= 9 KB / 833 Bytes

Einheiten konvertieren. 1 KB entspricht 1024 Bytes.

K=9×1024 / 833

Berechnen Sie den Wert:

K≈10.92

Der Kompressionsfaktor beträgt etwa 10,92. Dies bedeutet, dass die Datei nach der Kompression nur etwa 1/10,92 der Größe der ursprünglichen Datei hat. Es zeigt, wie effektiv die ZIP-Kompression in diesem Fall war.


B1_UTF-16 LE.txt
B1_ASCII.txt
P1: ? 1 0 1 0 0

Aufgabe 4

Um den CRC-Code zu berechnen, führt man eine XOR-Operation zwischen den Datenbits und den CRC-Bits durch. Der Prozess wird wiederholt, bis alle Datenbits durchlaufen sind. Das resultierende Bitmuster ist der CRC-Code.

Datenbitmuster ('f'): 01100110
Anzahl der CRC-Bits - 1 (in diesem Fall 3) Nullen hinzu: 01100110000

CRC-Code mit Nullen initialisieren: 000

XOR zwischen dem CRC-Code und den ersten drei Bits des erweiterten Datenmusters durchführen: 000 XOR 011 = 011

Wiederhole den Vorgang für jedes Bit im erweiterten Datenmuster:
011 XOR 001 = 010
010 XOR 000 = 010
010 XOR 110 = 100
100 XOR 000 = 100
100 XOR 000 = 100

Das **resultierende Bitmuster** ist der CRC-Code: **100**

Aufgabe 5

Um den Hamming-Code zu überprüfen und zu korrigieren, müssen wir zunächst feststellen, welche Paritätsbits falsch sind. Der Hamming-Code ist so strukturiert, dass die Positionen der Paritätsbits (2^0, 2^1, 2^2, ...) Bits überprüfen. Hier sind die Paritätsbits auf den Positionen 1, 2, 4, 8, 16:

P	1 P 2 3 P 4 5 6 7 P 8 9
Hamming-Code	1 0 1 1 0 0 0 0 1 1 1 0

Paritäten berechnen:

P	1 P 2 3 P 4 5 6 7 P 8 9
Hamming-Code	1 0 1 1 0 0 0 0 1 1 1 0
Berechnete Parität	1 1 0 1 0 0 1 0 0 1 1 0

Die falschen Paritäten sind an den Positionen 2, 4 und 8. Die Positionen sind 2^1, 2^2, und 2^3. Daher haben wir Fehler in den Bits 2, 4 und 8.

Korrigieren von Fehler:
Korrigierter Code: 1 1 1 1 0 0 0 0 1 1 1 0

Nun müssen wir den dezimalen Wert dieses korrigierten Codes finden, um das ASCII-Zeichen zu erhalten. Der dezimale Wert ist:

Dezimaler Wert = $2^0 + 2^1 + 2^3 + 2^4 + 2^7 + 2^8 + 2^{11}$ = 1+2+8+16+128+256+2048 = 2463