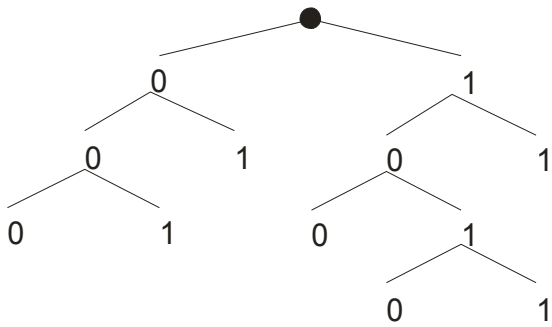


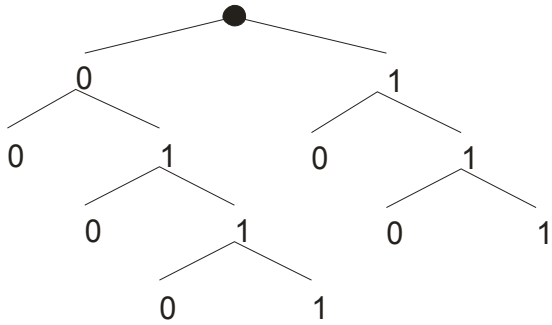
1. Überprüfen Sie, ob die Codes, die aus den folgenden Worten bestehen, Präfixcodes sind und zeichnen Sie gegebenenfalls den zugehörigen binären Baum.:

(a) 11, 1010, 1011, 100, 01, 001, 000

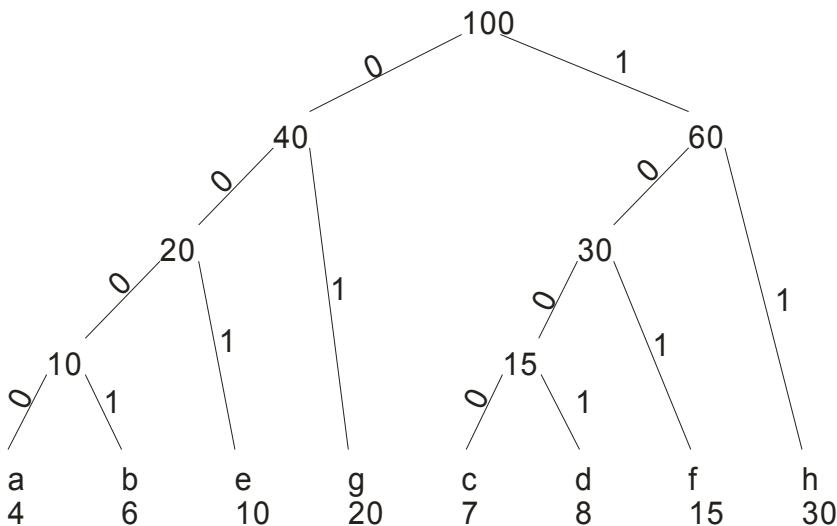


(b) 00, 010, 011, 10, 110, 111, 0110, 0111

Dies ist kein Präfix-Code, weil einer der Codes (011) auch der Anfang zwei weiterer Codes (0110 und 0111) ist



2. Konstruieren Sie mit dem Huffman-Algorithmus einen Präfixcode für das Alphabet {a, b, c, d, e, f, g, h} bei der Häufigkeitsverteilung {4, 6, 7, 8, 10, 15, 20, 30}, (a = 4%, b = 6%, ..., h = 30%).

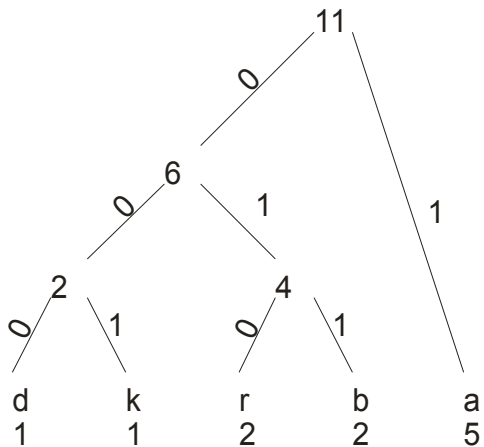


3. Kodieren Sie „abrakadabra“ mit dem Huffman-Algorithmus!

Zunächst werden die Häufigkeiten ermittelt:

- a 5
- b 2
- r 2
- k 1
- d 1

Es folgt der Bau eines Code-Baumes:



Nun kann die Codierung erfolgen:

a=1, b=011, r=010, k=001, d=000

Daraus folgt: 1 011 010 1 001 1 000 1 011 010 1 = 10110101001100010110101

4. Bestimmen Sie die Lauflängenkodierung für die Zeichenfolge  
AAAAGGGGGbbbbbllllllllllkkLLLLLo

(5, A) (5, G) (5, b) (10, l) (2, k) (5, L) o

5. Bestimmen Sie einen möglichst kompakten Lauflängencode für  
000000111111000111111111111111

Wenn vereinbart ist, daß die Werte alternierend und immer zuerst mit einem 0-Block angegeben sind, genügt 6 6 3 15

6. Machen Sie sich (auf Papier) die Wirkungsweise des LZW-Algorithmus klar, verwenden Sie zum Beispiel: „blablablabla“!

Zeichenkette	gefundener Eintrag	Ausgabe	neuer Eintrag
blablablabla	b	b	bl <1>
lablablabla	l	l	la <2>
ablablabla	a	a	ab <3>
blablabla	bl	<1>	bla <4>
ablabla	ab	<3>	abl <5>

lablabla	la	<2>	lab <6>
blabla	bla	<4>	blab <7>
bla	bla	<7>	-

7. Experimentieren Sie unter Linux mit gimp und verschiedenen Kompressionsstufen! Auf dem Skripteserver (Verzeichnis Bilder) liegen verschiedene zu komprimierende Dateien:

lena.rgb, camera.rgb, figure.rgb, figure.gif

Vergleichen Sie:

(a) Was bringt die bei Abspeicherung als rgb-Datei mögliche Lauflängenkodierung?

Sie bringt eine Platzersparnis dort, wo mindestens ein Kanal größere Flächen abdeckt.

(b) Lassen sich alle Bilder gleich gut mit JPEG komprimieren? Vergleichen Sie Qualität und Speicherplatz!

Besonders Schwarzweißbilder und Bilder mit Rauschen lassen sich schlechter komprimieren. Auch Bilder mit feinen, wiederkehrenden Mustern lassen sich schlecht komprimieren.

(c) Was macht JPEG die meisten Schwierigkeiten? Wo treten die meisten Artefakte auf?

Die meisten Artefakte treten bei mittleren und starken Farbwechseln (Übergangskanten) und schmalen/schrägen Kanten auf.

(d) Können Sie sich den Größenunterschied des "figure"-Bildes als \*gif und als JPEG-Datei erklären?

Grund für dieses "Phänomen": Da im Bild viele sich wiederholende Muster vorkommen, die über den LZW-Algorithmus komprimiert werden können, kann das Bild als \*gif mit weniger (redundanten) Informationen gesichert werden.

## 8. Matlab

Was tut das Programm bv\_jpegprinzip\_noc.m ? Versehen Sie die Zeilen des Programms mit aussagekräftigen Kommentaren!

Den Inhalt aller Variablen löschen

```
clear
```

alle Bilder schließen

```
close all
```

Variable `thresh` den Wert 0.02 zuweisen

```
thresh = 0.02
```

Variable `I` das Image "lena\_std\_sw.png" zuweisen

```
I = imread('lena_std_sw.png');
```

Das Image, das sich hinter `I` verbirgt, anzeigen

```
figure('Name','Originalbild Lena'), imshow(I);
```

Imagewerte aus `I` in `Double` konvertieren

```
I = im2double(I);
```

Variable `T` eine diskrete Cosinus-Transformationsmatrix der Größe 8 zuweisen

```
T = dctmtx(8);
```

Array `B` mit der Ergebnismatrix aus der Matrixmultiplikation von `T`, `I` und `T` transponiert

8 8 zeigt an, daß 8x8-Blöcke verwendet werden

```
B = blkproc(I, [8 8], 'P1*x*P2', T,T');
```

```
%ecke = B(1:1:8, 1:1:8)
```

**Matrix `mask` mit den vorgegebenen Werten definieren**

```
mask = [ 1 1 1 1 0 0 0 0  
        1 1 1 0 0 0 0 0  
        1 1 0 0 0 0 0 0  
        1 0 0 0 0 0 0 0  
        0 0 0 0 0 0 0 0  
        0 0 0 0 0 0 0 0  
        0 0 0 0 0 0 0 0  
        0 0 0 0 0 0 0 0 ];
```

**Array `B2` mit der Ergebnismatrix aus der einzelnen Multiplikation von jedem Element aus `B` mit jedem Element aus `mask`**

```
B2 = blkproc(B, [8 8], 'P1.*x', mask);
```

```
%ecke = B2(1:1:8,1:1:8)
```

**Array `I2` mit der Ergebnismatrix aus der Matrixmultiplikation von `T`, `B2` und `T` transponiert**

```
I2 = blkproc(B2, [8 8], 'P1*x*P2', T',T);
```

**Bild `I2` anzeigen**

```
figure('Name','Lena geändert 1'), imshow(I2);
```

**Jedes Element aus `B2`, das  $< 0,2$  und  $> -0,2$  ist, wird auf 0 gesetzt**

```
B2(abs(B2)<treshold) = 0;
```

```
%ecke = B2(1:1:8,1:1:8)
```

**Array `I3` mit der Ergebnismatrix aus der Matrixmultiplikation von `T`, `B2` und `T` transponiert**

```
I3 = blkproc(B2, [8 8], 'P1*x*P2', T',T);
```

**Bild `I3` anzeigen**

```
figure('Name','Lena geändert 2'), imshow(I3);
```