Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Lösungen zu Übungsblatt 4

Aufgabe 1: Der JPEG-Algorithmus

Gegeben sei folgende 8x8-Matrix A im RGB-Format:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 196 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

(a) Was bewirkt die Diskrete Cosinustranformation?

Lösung:

Transformation der Matrix/des Makroblocks aus dem Farb- in den Frequenzraum.

(b) Welche Eigenschaft des menschlichen Auges wird durch die DCT ausgenutzt?

Lösung:

Das menschliche Auge kann niedrige Frequenzen besser erkennen als hohe. Die DCT „sammelt“ die niedrigen Frequenzen in der oberen linken Ecke der Matrix und die hohen unten rechts. Nun kann die unwichtige von der wichtigen Information besser getrennt werden.

(c) Programmieren Sie die DCT wie in der Vorlesung besprochen auf einem Computer.

Lösung:

```
#include <iostream.h>
#include <fstream.h>
```

```
const unsigned int msize = 8;
typedef int matrix[msize][msize];

void init_matrix(matrix &a) {
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      a[i][j] = 0;
}

void read_matrix(char* filename, matrix &a) {
  ifstream in(filename);
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      in >> a[i][j];
}

void matrix_ausgabe(matrix a) {
  int i,j;

  cout << „Ergebnis-Matrix:" << endl;
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      cout.width(5);
      cout << a[i][j];
    }
    cout << endl;
  }
}

void dct(matrix &s, matrix &SS) {
    int x, y, u, v;
    double Cu,Cv;
    double result;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++)
            s[v][u] -= 128;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++) {
            Cu = (u==0) ? 0.7071:1.0;
            Cv = (v==0) ? 0.7071:1.0;

            result = 0.0;
            for (x=0;x<8;x++)
                for (y=0;y<8;y++)
```

```
                    result += s[y][x] *
                        cos((2.0*x+1)*u*3.14159/16.0)*
                        cos((2.0*y+1)*v*3.14159/16.0);
                result *= 0.25*Cu*Cv;
                SS[v][u] = (int) result;
        }
}

void main(int argc, char** argv) {

  matrix a,b;
  int i,j;

  init_matrix(a);
  init_matrix(b);
  read_matrix(„meingabe‟,a);
  matrix_ausgabe(a);
  dct(a,b);
  matrix_ausgabe(b);
}
```

(d) Berechnen Sie mit Hilfe Ihres Programms die Matrix $A' = DCT(A)$.

Lösung:

$$A' = \begin{bmatrix} 337 & -48 & -39 & 10 & -22 & 11 & -63 & 5 \\ 62 & -77 & 15 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -27 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 6 & -7 \\ -9 & 10 & 14 & -10 & 10 & -12 & 9 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{bmatrix}$$

(e) Beschreiben Sie nun den Quantisierungsvorgang.

Lösung:

Bei der Quantisierung wird jeder Wert der Matrix durch einen sogenannten Quantisierungsfaktor geteilt. Der resultierende Wert wird anschließend gerundet. Diese Rundung stellt den eigentlich verlustbehafteten Vorgang dar. Er läßt sich nicht mehr rückgängig machen, erlaubt aber erst eine hohe Kompression.

(f) Berechnen Sie die quantisierte Matrix $A'' = A' \cdot B$, wobei B die Quantisierungsmatrix ist mit $b_{ij} = 30$, i,j=0,..,7 .

Lösung:

$$A'' = \begin{bmatrix} 11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\ 2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) Beschreiben Sie nun den weiteren Kodierungsvorgang, also die Entropiekodierung.

Lösung:

Die Matrix wird nun mit Lauflängenkodierung kodiert, d.h., mehrere gleiche Werte in Folge werden durch einmalige Aufführung des Wertes sowie der Anzahl seiner Vorkommen beschrieben. Hierdurch ensteht die Kompression. Entropiekodierung selbst ist verlustfrei.

(h) Warum wird die Matrix in Zick-Zack-Linie abgetastet?

Lösung:

In den meisten Fällen sammeln sich Nullwerte in der rechten unteren Ecke der Matrix an. Durch ein Ablaufen der Matrix im Zick-Zack erreicht man dann längere Nullfolgen und damit eine bessere Kompression.

(i) Wie lautet das kodierte String für die Matrix A‘‘?

Lösung:

Jeweils zwei Werte gehören zusammen. der erste Wert gibt an, wie oft der zweite Wert vorkommt.

1 11 1 2 1 2 -1 1 -2 1 0 1 -1 5 0 1 2 3 0 1 -1 2 0 1 1 1 0 1 -2 2 0 1 -1 11 0 1 -1 11 0 1 -1 15 0

(j) Dekodieren Sie nun den String vollständig und geben Sie die entstehende RGB-Matrix an.

Lösung:

$$C = \begin{bmatrix} 169 & 133 & 124 & 128 & 128 & 128 & 124 & 128 \\ 128 & 115 & 127 & 128 & 123 & 128 & 127 & 127 \\ 128 & 125 & 140 & 128 & 127 & 128 & 128 & 128 \\ 132 & 128 & 131 & 123 & 128 & 128 & 128 & 128 \\ 129 & 132 & 128 & 127 & 128 & 128 & 128 & 128 \\ 128 & 129 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{bmatrix}$$

Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Lösungen zu Übungsblatt 4

Aufgabe 1: Der JPEG-Algorithmus

Gegeben sei folgende 8x8-Matrix A im RGB-Format:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 196 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

(a) Was bewirkt die Diskrete Cosinustranformation?

Lösung:

Transformation der Matrix/des Makroblocks aus dem Farb- in den Frequenzraum.

(b) Welche Eigenschaft des menschlichen Auges wird durch die DCT ausgenutzt?

Lösung:

Das menschliche Auge kann niedrige Frequenzen besser erkennen als hohe. Die DCT „sammelt“ die niedrigen Frequenzen in der oberen linken Ecke der Matrix und die hohen unten rechts. Nun kann die unwichtige von der wichtigen Information besser getrennt werden.

(c) Programmieren Sie die DCT wie in der Vorlesung besprochen auf einem Computer.

Lösung:

```
#include <iostream.h>
#include <fstream.h>
```

```
const unsigned int msize = 8;
typedef int matrix[msize][msize];

void init_matrix(matrix &a) {
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      a[i][j] = 0;
}

void read_matrix(char* filename, matrix &a) {
  ifstream in(filename);
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      in >> a[i][j];
}

void matrix_ausgabe(matrix a) {
  int i,j;

  cout << „Ergebnis-Matrix:" << endl;
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      cout.width(5);
      cout << a[i][j];
    }
    cout << endl;
  }
}

void dct(matrix &s, matrix &SS) {
    int x, y, u, v;
    double Cu,Cv;
    double result;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++)
            s[v][u] -= 128;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++) {
            Cu = (u==0) ? 0.7071:1.0;
            Cv = (v==0) ? 0.7071:1.0;

            result = 0.0;
            for (x=0;x<8;x++)
                for (y=0;y<8;y++)
```

```
                    result += s[y][x] *
                        cos((2.0*x+1)*u*3.14159/16.0)*
                        cos((2.0*y+1)*v*3.14159/16.0);
                result *= 0.25*Cu*Cv;
                SS[v][u] = (int) result;
        }
}

void main(int argc, char** argv) {

  matrix a,b;
  int i,j;

  init_matrix(a);
  init_matrix(b);
  read_matrix(„meingabe",a);
  matrix_ausgabe(a);
  dct(a,b);
  matrix_ausgabe(b);
}
```

(d) Berechnen Sie mit Hilfe Ihres Programms die Matrix $A' = DCT(A)$.

Lösung:

$$A' = \begin{bmatrix} 337 & -48 & -39 & 10 & -22 & 11 & -63 & 5 \\ 62 & -77 & 15 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -27 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 6 & -7 \\ -9 & 10 & 14 & -10 & 10 & -12 & 9 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{bmatrix}$$

(e) Beschreiben Sie nun den Quantisierungsvorgang.

Lösung:

Bei der Quantisierung wird jeder Wert der Matrix durch einen sogenannten Quantisierungsfaktor geteilt. Der resultierende Wert wird anschließend gerundet. Diese Rundung stellt den eigentlich verlustbehafteten Vorgang dar. Er läßt sich nicht mehr rückgängig machen, erlaubt aber erst eine hohe Kompression.

(f) Berechnen Sie die quantisierte Matrix $A'' = A' \cdot B$, wobei B die Quantisierungsmatrix ist mit $b_{ij} = 30$, i,j=0,..,7 .

Lösung:

$$
A'' = \begin{bmatrix}
11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\
2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

(g) Beschreiben Sie nun den weiteren Kodierungsvorgang, also die Entropiekodierung.

Lösung:

Die Matrix wird nun mit Lauflängenkodierung kodiert, d.h., mehrere gleiche Werte in Folge werden durch einmalige Aufführung des Wertes sowie der Anzahl seiner Vorkommen beschrieben. Hierdurch ensteht die Kompression. Entropiekodierung selbst ist verlustfrei.

(h) Warum wird die Matrix in Zick-Zack-Linie abgetastet?

Lösung:

In den meisten Fällen sammeln sich Nullwerte in der rechten unteren Ecke der Matrix an. Durch ein Ablaufen der Matrix im Zick-Zack erreicht man dann längere Nullfolgen und damit eine bessere Kompression.

(i) Wie lautet das kodierte String für die Matrix A‘‘?

Lösung:

Jeweils zwei Werte gehören zusammen. der erste Wert gibt an, wie oft der zweite Wert vorkommt.

1 11 1 2 1 2 -1 1 -2 1 0 1 -1 5 0 1 2 3 0 1 -1 2 0 1 1 1 0 1 -2 2 0 1 -1 11 0 1 -1 11 0 1 -1 15 0

(j) Dekodieren Sie nun den String vollständig und geben Sie die entstehende RGB-Matrix an.

Lösung:

$$C = \begin{bmatrix} 169 & 133 & 124 & 128 & 128 & 128 & 124 & 128 \\ 128 & 115 & 127 & 128 & 123 & 128 & 127 & 127 \\ 128 & 125 & 140 & 128 & 127 & 128 & 128 & 128 \\ 132 & 128 & 131 & 123 & 128 & 128 & 128 & 128 \\ 129 & 132 & 128 & 127 & 128 & 128 & 128 & 128 \\ 128 & 129 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{bmatrix}$$

Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Lösungen zu Übungsblatt 4

Aufgabe 1: Der JPEG-Algorithmus

Gegeben sei folgende 8x8-Matrix A im RGB-Format:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 196 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

(a) Was bewirkt die Diskrete Cosinustranformation?

Lösung:

Transformation der Matrix/des Makroblocks aus dem Farb- in den Frequenzraum.

(b) Welche Eigenschaft des menschlichen Auges wird durch die DCT ausgenutzt?

Lösung:

Das menschliche Auge kann niedrige Frequenzen besser erkennen als hohe. Die DCT „sammelt“ die niedrigen Frequenzen in der oberen linken Ecke der Matrix und die hohen unten rechts. Nun kann die unwichtige von der wichtigen Information besser getrennt werden.

(c) Programmieren Sie die DCT wie in der Vorlesung besprochen auf einem Computer.

Lösung:

```
#include <iostream.h>
#include <fstream.h>
```

```
const unsigned int msize = 8;
typedef int matrix[msize][msize];

void init_matrix(matrix &a) {
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      a[i][j] = 0;
}

void read_matrix(char* filename, matrix &a) {
  ifstream in(filename);
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      in >> a[i][j];
}

void matrix_ausgabe(matrix a) {
  int i,j;

  cout << „Ergebnis-Matrix:“ << endl;
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      cout.width(5);
      cout << a[i][j];
    }
    cout << endl;
  }
}

void dct(matrix &s, matrix &SS) {
    int x, y, u, v;
    double Cu,Cv;
    double result;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++)
            s[v][u] -= 128;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++) {
            Cu = (u==0) ? 0.7071:1.0;
            Cv = (v==0) ? 0.7071:1.0;

            result = 0.0;
            for (x=0;x<8;x++)
                for (y=0;y<8;y++)
```

```
                    result += s[y][x] *
                        cos((2.0*x+1)*u*3.14159/16.0)*
                        cos((2.0*y+1)*v*3.14159/16.0);
                result *= 0.25*Cu*Cv;
                SS[v][u] = (int) result;
        }
}

void main(int argc, char** argv) {

  matrix a,b;
  int i,j;

  init_matrix(a);
  init_matrix(b);
  read_matrix(„meingabe“,a);
  matrix_ausgabe(a);
  dct(a,b);
  matrix_ausgabe(b);
}
```

(d) Berechnen Sie mit Hilfe Ihres Programms die Matrix $A' = DCT(A)$.

Lösung:

$$A' = \begin{bmatrix} 337 & -48 & -39 & 10 & -22 & 11 & -63 & 5 \\ 62 & -77 & 15 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -27 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 6 & -7 \\ -9 & 10 & 14 & -10 & 10 & -12 & 9 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{bmatrix}$$

(e) Beschreiben Sie nun den Quantisierungsvorgang.

Lösung:

Bei der Quantisierung wird jeder Wert der Matrix durch einen sogenannten Quantisierungsfaktor geteilt. Der resultierende Wert wird anschließend gerundet. Diese Rundung stellt den eigentlich verlustbehafteten Vorgang dar. Er läßt sich nicht mehr rückgängig machen, erlaubt aber erst eine hohe Kompression.

(f) Berechnen Sie die quantisierte Matrix $A'' = A' \cdot B$, wobei B die Quantisierungsmatrix ist mit $b_{ij} = 30$, i,j=0,..,7.

Lösung:

$$A'' = \begin{bmatrix} 11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\ 2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) Beschreiben Sie nun den weiteren Kodierungsvorgang, also die Entropiekodierung.

Lösung:

Die Matrix wird nun mit Lauflängenkodierung kodiert, d.h., mehrere gleiche Werte in Folge werden durch einmalige Aufführung des Wertes sowie der Anzahl seiner Vorkommen beschrieben. Hierdurch ensteht die Kompression. Entropiekodierung selbst ist verlustfrei.

(h) Warum wird die Matrix in Zick-Zack-Linie abgetastet?

Lösung:

In den meisten Fällen sammeln sich Nullwerte in der rechten unteren Ecke der Matrix an. Durch ein Ablaufen der Matrix im Zick-Zack erreicht man dann längere Nullfolgen und damit eine bessere Kompression.

(i) Wie lautet das kodierte String für die Matrix A‘‘?

Lösung:

Jeweils zwei Werte gehören zusammen. der erste Wert gibt an, wie oft der zweite Wert vorkommt.

1 11 1 2 1 2 -1 1 -2 1 0 1 -1 5 0 1 2 3 0 1 -1 2 0 1 1 1 0 1 -2 2 0 1 -1 11 0 1 -1 11 0 1 -1 15 0

(j) Dekodieren Sie nun den String vollständig und geben Sie die entstehende RGB-Matrix an.

Lösung:

$$
C = \begin{bmatrix}
169 & 133 & 124 & 128 & 128 & 128 & 124 & 128 \\
128 & 115 & 127 & 128 & 123 & 128 & 127 & 127 \\
128 & 125 & 140 & 128 & 127 & 128 & 128 & 128 \\
132 & 128 & 131 & 123 & 128 & 128 & 128 & 128 \\
129 & 132 & 128 & 127 & 128 & 128 & 128 & 128 \\
128 & 129 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\
128 & 128 & 128 & 128 & 128 & 128 & 128 & 128
\end{bmatrix}
$$

Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Lösungen zu Übungsblatt 4

Aufgabe 1: Der JPEG-Algorithmus

Gegeben sei folgende 8x8-Matrix A im RGB-Format:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 196 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

(a) Was bewirkt die Diskrete Cosinustranformation?

Lösung:

Transformation der Matrix/des Makroblocks aus dem Farb- in den Frequenzraum.

(b) Welche Eigenschaft des menschlichen Auges wird durch die DCT ausgenutzt?

Lösung:

Das menschliche Auge kann niedrige Frequenzen besser erkennen als hohe. Die DCT „sammelt“ die niedrigen Frequenzen in der oberen linken Ecke der Matrix und die hohen unten rechts. Nun kann die unwichtige von der wichtigen Information besser getrennt werden.

(c) Programmieren Sie die DCT wie in der Vorlesung besprochen auf einem Computer.

Lösung:

```
#include <iostream.h>
#include <fstream.h>
```

```
const unsigned int msize = 8;
typedef int matrix[msize][msize];

void init_matrix(matrix &a) {
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      a[i][j] = 0;
}

void read_matrix(char* filename, matrix &a) {
  ifstream in(filename);
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      in >> a[i][j];
}

void matrix_ausgabe(matrix a) {
  int i,j;

  cout << „Ergebnis-Matrix:" << endl;
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      cout.width(5);
      cout << a[i][j];
    }
    cout << endl;
  }
}

void dct(matrix &s, matrix &SS) {
    int x, y, u, v;
    double Cu,Cv;
    double result;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++)
            s[v][u] -= 128;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++) {
            Cu = (u==0) ? 0.7071:1.0;
            Cv = (v==0) ? 0.7071:1.0;

            result = 0.0;
            for (x=0;x<8;x++)
                for (y=0;y<8;y++)
```

```
                    result += s[y][x] *
                        cos((2.0*x+1)*u*3.14159/16.0)*
                        cos((2.0*y+1)*v*3.14159/16.0);
                result *= 0.25*Cu*Cv;
                SS[v][u] = (int) result;
        }
}

void main(int argc, char** argv) {

  matrix a,b;
  int i,j;

  init_matrix(a);
  init_matrix(b);
  read_matrix(„meingabe",a);
  matrix_ausgabe(a);
  dct(a,b);
  matrix_ausgabe(b);
}
```

(d) Berechnen Sie mit Hilfe Ihres Programms die Matrix $A' = DCT(A)$.

Lösung:

$$A' = \begin{bmatrix} 337 & -48 & -39 & 10 & -22 & 11 & -63 & 5 \\ 62 & -77 & 15 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -27 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 6 & -7 \\ -9 & 10 & 14 & -10 & 10 & -12 & 9 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{bmatrix}$$

(e) Beschreiben Sie nun den Quantisierungsvorgang.

Lösung:

Bei der Quantisierung wird jeder Wert der Matrix durch einen sogenannten Quantisierungsfaktor geteilt. Der resultierende Wert wird anschließend gerundet. Diese Rundung stellt den eigentlich verlustbehafteten Vorgang dar. Er läßt sich nicht mehr rückgängig machen, erlaubt aber erst eine hohe Kompression.

(f) Berechnen Sie die quantisierte Matrix $A'' = A' \cdot B$, wobei B die Quantisierungsmatrix ist mit $b_{ij} = 30$, i,j=0,..,7 .

Lösung:

$$A'' = \begin{bmatrix} 11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\ 2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) Beschreiben Sie nun den weiteren Kodierungsvorgang, also die Entropiekodierung.

Lösung:

Die Matrix wird nun mit Lauflängenkodierung kodiert, d.h., mehrere gleiche Werte in Folge werden durch einmalige Aufführung des Wertes sowie der Anzahl seiner Vorkommen beschrieben. Hierdurch ensteht die Kompression. Entropiekodierung selbst ist verlustfrei.

(h) Warum wird die Matrix in Zick-Zack-Linie abgetastet?

Lösung:

In den meisten Fällen sammeln sich Nullwerte in der rechten unteren Ecke der Matrix an. Durch ein Ablaufen der Matrix im Zick-Zack erreicht man dann längere Nullfolgen und damit eine bessere Kompression.

(i) Wie lautet das kodierte String für die Matrix A‘‘?

Lösung:

Jeweils zwei Werte gehören zusammen. der erste Wert gibt an, wie oft der zweite Wert vorkommt.

1 11 1 2 1 2 -1 1 -2 1 0 1 -1 5 0 1 2 3 0 1 -1 2 0 1 1 1 0 1 -2 2 0 1 -1 11 0 1 -1 11 0 1 -1 15 0

(j) Dekodieren Sie nun den String vollständig und geben Sie die entstehende RGB-Matrix an.

Lösung:

$$C = \begin{bmatrix} 169 & 133 & 124 & 128 & 128 & 128 & 124 & 128 \\ 128 & 115 & 127 & 128 & 123 & 128 & 127 & 127 \\ 128 & 125 & 140 & 128 & 127 & 128 & 128 & 128 \\ 132 & 128 & 131 & 123 & 128 & 128 & 128 & 128 \\ 129 & 132 & 128 & 127 & 128 & 128 & 128 & 128 \\ 128 & 129 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{bmatrix}$$

Universität Mannheim
Lehrstuhl für Praktische Informatik IV
Dr. Stefan Fischer
Tel.: 292-1407
Email: stefis@pi4.informatik.uni-mannheim.de

Lösungen zu Übungsblatt 4

Aufgabe 1: Der JPEG-Algorithmus

Gegeben sei folgende 8x8-Matrix A im RGB-Format:

$$A = \begin{pmatrix} 150 & 170 & 132 & 185 & 147 & 190 & 215 & 220 \\ 165 & 185 & 130 & 190 & 175 & 196 & 223 & 199 \\ 155 & 163 & 180 & 220 & 202 & 173 & 197 & 170 \\ 143 & 154 & 160 & 170 & 211 & 185 & 190 & 166 \\ 130 & 140 & 172 & 190 & 193 & 150 & 180 & 140 \\ 135 & 164 & 198 & 180 & 177 & 141 & 172 & 135 \\ 170 & 190 & 163 & 140 & 165 & 132 & 160 & 140 \\ 160 & 200 & 145 & 135 & 170 & 199 & 190 & 129 \end{pmatrix}$$

(a) Was bewirkt die Diskrete Cosinustranformation?

Lösung:

Transformation der Matrix/des Makroblocks aus dem Farb- in den Frequenzraum.

(b) Welche Eigenschaft des menschlichen Auges wird durch die DCT ausgenutzt?

Lösung:

Das menschliche Auge kann niedrige Frequenzen besser erkennen als hohe. Die DCT „sammelt" die niedrigen Frequenzen in der oberen linken Ecke der Matrix und die hohen unten rechts. Nun kann die unwichtige von der wichtigen Information besser getrennt werden.

(c) Programmieren Sie die DCT wie in der Vorlesung besprochen auf einem Computer.

Lösung:

```
#include <iostream.h>
#include <fstream.h>
```

```
const unsigned int msize = 8;
typedef int matrix[msize][msize];

void init_matrix(matrix &a) {
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      a[i][j] = 0;
}

void read_matrix(char* filename, matrix &a) {
  ifstream in(filename);
  int i,j;

  for (i=0;i<8;i++)
    for (j=0;j<8;j++)
      in >> a[i][j];
}

void matrix_ausgabe(matrix a) {
  int i,j;

  cout << „Ergebnis-Matrix:" << endl;
  for (i=0;i<8;i++) {
    for (j=0;j<8;j++) {
      cout.width(5);
      cout << a[i][j];
    }
    cout << endl;
  }
}

void dct(matrix &s, matrix &SS) {
    int x, y, u, v;
    double Cu,Cv;
    double result;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++)
            s[v][u] -= 128;

    for (v=0;v<8;v++)
        for (u=0;u<8;u++) {
            Cu = (u==0) ? 0.7071:1.0;
            Cv = (v==0) ? 0.7071:1.0;

            result = 0.0;
            for (x=0;x<8;x++)
                for (y=0;y<8;y++)
```

```
                    result += s[y][x] *
                        cos((2.0*x+1)*u*3.14159/16.0)*
                        cos((2.0*y+1)*v*3.14159/16.0);
                result *= 0.25*Cu*Cv;
                SS[v][u] = (int) result;
        }
}

void main(int argc, char** argv) {

  matrix a,b;
  int i,j;

  init_matrix(a);
  init_matrix(b);
  read_matrix(„meingabe",a);
  matrix_ausgabe(a);
  dct(a,b);
  matrix_ausgabe(b);
}
```

(d) Berechnen Sie mit Hilfe Ihres Programms die Matrix $A' = DCT(A)$.

Lösung:

$$A' = \begin{bmatrix} 337 & -48 & -39 & 10 & -22 & 11 & -63 & 5 \\ 62 & -77 & 15 & -12 & 34 & 23 & -5 & -41 \\ 7 & -7 & 75 & 26 & -27 & -12 & -10 & -43 \\ -30 & 1 & 7 & -40 & 5 & 12 & 3 & 3 \\ -9 & -34 & 0 & 24 & -10 & -7 & 6 & -7 \\ -9 & 10 & 14 & -10 & 10 & -12 & 9 & 10 \\ 10 & 2 & -29 & -20 & -7 & 14 & 9 & 0 \\ -8 & 14 & -8 & -16 & 15 & 13 & -2 & 2 \end{bmatrix}$$

(e) Beschreiben Sie nun den Quantisierungsvorgang.

Lösung:

Bei der Quantisierung wird jeder Wert der Matrix durch einen sogenannten Quantisierungsfaktor geteilt. Der resultierende Wert wird anschließend gerundet. Diese Rundung stellt den eigentlich verlustbehafteten Vorgang dar. Er läßt sich nicht mehr rückgängig machen, erlaubt aber erst eine hohe Kompression.

(f) Berechnen Sie die quantisierte Matrix $A'' = A' \cdot B$, wobei B die Quantisierungsmatrix ist mit $b_{ij} = 30$, i,j=0,..,7.

Lösung:

$$A''=\begin{bmatrix} 11 & -1 & -1 & 0 & 0 & 0 & -2 & 0 \\ 2 & -2 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) Beschreiben Sie nun den weiteren Kodierungsvorgang, also die Entropiekodierung.

Lösung:

Die Matrix wird nun mit Lauflängenkodierung kodiert, d.h., mehrere gleiche Werte in Folge werden durch einmalige Aufführung des Wertes sowie der Anzahl seiner Vorkommen beschrieben. Hierdurch ensteht die Kompression. Entropiekodierung selbst ist verlustfrei.

(h) Warum wird die Matrix in Zick-Zack-Linie abgetastet?

Lösung:

In den meisten Fällen sammeln sich Nullwerte in der rechten unteren Ecke der Matrix an. Durch ein Ablaufen der Matrix im Zick-Zack erreicht man dann längere Nullfolgen und damit eine bessere Kompression.

(i) Wie lautet das kodierte String für die Matrix A‘‘?

Lösung:

Jeweils zwei Werte gehören zusammen. der erste Wert gibt an, wie oft der zweite Wert vorkommt.

1 11 1 2 1 2 -1 1 -2 1 0 1 -1 5 0 1 2 3 0 1 -1 2 0 1 1 1 0 1 -2 2 0 1 -1 11 0 1 -1 11 0 1 -1 15 0

(j) Dekodieren Sie nun den String vollständig und geben Sie die entstehende RGB-Matrix an.

Lösung:

$$C = \begin{bmatrix} 169 & 133 & 124 & 128 & 128 & 128 & 124 & 128 \\ 128 & 115 & 127 & 128 & 123 & 128 & 127 & 127 \\ 128 & 125 & 140 & 128 & 127 & 128 & 128 & 128 \\ 132 & 128 & 131 & 123 & 128 & 128 & 128 & 128 \\ 129 & 132 & 128 & 127 & 128 & 128 & 128 & 128 \\ 128 & 129 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \\ 128 & 128 & 128 & 128 & 128 & 128 & 128 & 128 \end{bmatrix}$$