Dokumentation zur Lösung der Aufgaben des 23. Bundeswettbewerb Informatik

Marcel Schmittfull
Moritz Beller

14. November 2004

Inhaltsverzeichnis

1 tratsCH trATsch . . . 3
- 1.1 Lösungsidee . . . 3
- 1.2 Programm-Dokumentation . . . 4
- 1.3 Programm-Ablaufprotokoll . . . 6
- 1.4 Quellcode . . . 8

2 Kosmische Schaltjahre . . . 11
- 2.1 Lösungsidee . . . 11
- 2.2 Programm-Dokumentation . . . 16
- 2.3 Programm-Ablaufprotokoll . . . 17
- 2.4 Quellcode . . . 21

3 Schweizer Käse . . . 28
- 3.1 Lösungsidee, Programmdokumentation . . . 28
- 3.2 Quellcode . . . 29

4 Klasse Arbeit . . . 31
- 4.1 Lösungsidee . . . 31
- 4.2 Programm-Dokumentation . . . 32
- 4.3 Programm-Ablaufprotokoll . . . 38
- 4.4 Quellcode . . . 43

5 Zappen und zählen . . . 48
- 5.1 Lösungsidee 1 . . . 48
- 5.2 Programm-Dokumentation 1 . . . 48
- 5.3 Programm-Ablaufprotokoll 1 . . . 50
- 5.4 Quellcode 1 . . . 51
- 5.5 Lösungsidee 2 . . . 51
- 5.6 Programm-Dokumentation 2 . . . 58
- 5.7 Programm-Ablaufprotokoll 2 . . . 60
- 5.8 Quellcode 2 . . . 63

6 Wörterketten . . . 67
- 6.1 Lösungsidee . . . 67
- 6.2 Programmdokumentation . . . 67
- 6.3 Programm-Ablaufprotokoll . . . 70
- 6.4 Code . . . 74

Vorwort

1 tratsCH trATsch

1.1 Lösungsidee

Die Sympathiebeziehungen der Chatonen können als gerichteter Graph modelliert werden, in dem die Knoten für Chatonen stehen und eine Kante von A nach B genau dann existiert, wenn dem Chatonen A der Chatone B sympathisch ist. Ist eine Kante von A nach B vorhanden, so wird A direkter Vorgänger von B und B direkter Nachfolger von A genannt. Gibt es einen Pfad von A nach C[1], dann ist A (nichtdirekter) Vorgänger von C und C (nichtdirekter) Nachfolger von A. Die Beispieltabelle aus der Aufgabenstellung gibt die direkten Nachfolger jedes Chatonen an. Der Graph des Beispiels sieht somit wie folgt aus:

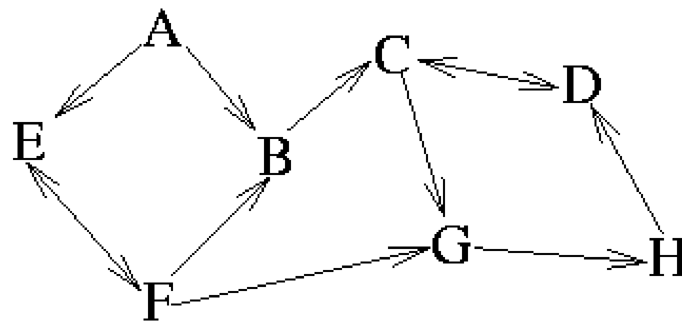


Abbildung 1.1: Modellierung des Beispiels als gerichteter Graph.

Jeder Chaton A spricht mit all den Chatonen, die ihm sympathisch sind, d.h. mit allen seinen direkten Nachfolgern. Die Liste aller Gesprächspartner eines Chatons besteht also schlichtweg aus seinen direkten Nachfolgern. Um Charmefehler zu vermeiden, spricht ein Chaton aber nur über Chatonen, die keine direkten oder indirekten Nachfolger seines Gesprächspartners sind. Das Problem besteht also darin, die direkten und indirekten Nachfolger jedes Chatons zu finden. Dies kann durch eine von jedem Knoten B ausgehende Tiefensuche entlang der Kanten bewerkstelligt werden, indem jeder besuchte Knoten in einer Liste L_B gespeichert wird. Ein Chaton A kann also an B über alle Chatonen schreiben, die sich nicht in L_B befinden und die nicht A oder B sind.

Als Beispiel soll hier Chaton A aus dem Aufgabenbeispiel betrachtet werden. A schreibt an seine direkten Nachfolger, d.h. an E und B. Die Listen aller direkten und indirekten Nachfolger von E und B sind

$$\begin{aligned} L_E &= \{F, G, H, D, C, B\} \\ L_B &= \{C, D, G, H\} \end{aligned}$$

[1] Ein Pfad von A nach C existiert genau dann, wenn die Kanten $A \to B_1, B_1 \to B_2, \ldots, B_{n-1} \to B_n, B_n \to C$ existieren und in der Menge $\{A, C, B_i\}$ kein Knoten doppelt vorkommt.

Es gibt keinen Chatonen, der sich nicht in L_E befindet und nicht A oder E ist, d.h. A kann an E über niemanden schreiben. Dafür kann A an B über E und F schreiben, weil E und F nicht in L_B und nicht A oder B sind.

1.2 Programm-Dokumentation

Als Programmiersprache wurde C gewählt. Der Graph ist mit Hilfe von 10 verketteten Adjazenzlisten implementiert, die jeweils die direkten Nachfolger eines Chatons speichern. Die Adjazenzlisten bestehen aus Elementen vom Typ

```
struct chaton
{ int v; struct chaton *next; };
```

Das erste Element jeder Adjazenzliste wird im Zeiger-Array

```
struct chaton *anfang[10];
```

gespeichert. Alle Elemente nach dem ersten Element innerhalb einer Adjazenzliste werden durch die `*next` Zeiger verkettet. Der Zeiger `*next` im letzten Element ist hierbei ein markierter Zeiger `*z`. In der Variablen `v` wird der Buchstabe des jeweiligen Chatons als ganze Zahl gespeichert.

Eine Eingabedatei mit den Sympathien der Chatonen muss dem in der Aufgabenstellung verwendeten Format entsprechen, z.B.

Beispiel-Eingabedatei tratsch.in

```
A B E
B C
C D G
D C
E F
F B E G
G H
H D
```

Um hieraus die Adjazenzlisten zu erzeugen, werden alle Zeilen durchlaufen. Der erste Buchstabe einer Zeile wird in `x` gespeichert. Die restlichen Buchstaben jeder Zeile werden nacheinandander in `y` gespeichert.[2] Dann wird `y` an den Anfang der Adjazenzliste von `x` gesetzt, d.h. zuerst wird `y→next` zu `anfang[x]`, danach wird der Zeiger `anfang[x]` auf `y` gerichtet. In C bedeutet das:

Einlesen der Eingabe und Erzeugen der Adjazenzlisten

```
while (fgets(zeile, 20, in) != NULL) {
    x = zeile[0]-65;
    for (j = 1; 2*j < strlen(zeile); j++) {
        y = zeile[2*j]-65;
        // Speichere y in Adjazenzliste von x
```

[2] Beim Schreiben der Eingabedaten ist zu beachten, dass nach dem letzten Buchstaben in einer Zeile ohne Leerzeichen direkt ein Newline stehen muss.

```
        t = (struct chaton *) malloc(sizeof *t);
        t->v = y; t->next = anfang[x]; anfang[x] = t;
    }
}
```

Die Adjazenzlisten des Aufgabenbeispiels sehen also wie folgt aus:

$$\begin{array}{lcl}
\texttt{anfang[A]} & \rightarrow & \texttt{E} \rightarrow \texttt{F} \rightarrow \texttt{G} \rightarrow \texttt{H} \rightarrow \texttt{D} \rightarrow \texttt{C} \rightarrow \texttt{B} \rightarrow \texttt{z} \\
\texttt{anfang[B]} & \rightarrow & \texttt{C} \rightarrow \texttt{G} \rightarrow \texttt{H} \rightarrow \texttt{D} \rightarrow \texttt{z} \\
 & \vdots & \\
\texttt{anfang[H]} & \rightarrow & \texttt{D} \rightarrow \texttt{C} \rightarrow \texttt{G} \rightarrow \texttt{z} \\
\texttt{anfang[I]} & \rightarrow & \texttt{z} \\
\texttt{anfang[J]} & \rightarrow & \texttt{z} \\
 & \vdots &
\end{array}$$

Nun können die Listen L_i mit den direkten und indirekten Nachfolgern aller Chatonen i erzeugt werden. Diese Listen werden in einem zweidimensionalen Array `weg` gespeichert, indem das `weg` Array zunächst mit Nullen initialisiert wird und danach `weg[a][b]` genau dann auf 1 gesetzt wird, wenn ein Pfad von Chaton a nach b existiert, d.h. wenn b ein direkter oder indirekter Nachfolger von a ist.

Um alle direkten und indirekten Nachfolger für jeden Chatonen zu finden, wird für jeden Chaton eine Tiefensuche `besuche` ausgeführt. Hierfür wird ein `schonBesucht` Array benötigt, das vor dem Ausführen jeder Tiefensuche mit Nullen gefüllt wird und jeden hinzukommenden besuchten Knoten innerhalb einer Tiefensuche mit einer 1 markiert.[3]

Die Tiefensuche-Prozedur `besuche(a,b)` ist rekursiv implementiert. Das erste Argument gibt an von welchem Chaton die Tiefensuche gestartet wurde. Das zweite Argument enthält den zu besuchenden Chaton. Wenn a und b verschieden sind, d.h. wenn der Chaton, von dem die Tiefensuche ausgeht, nicht der zu besuchende Chaton ist, wird `weg[a][b]` auf 1 gesetzt, da der Chaton b offensichtlich durch die von a ausgehende Tiefensuche besucht wurde. Nun werden alle Chatonen t in der Adjazenzliste von b durchlaufen. Wenn der Chaton t noch nicht besucht wurde und nicht gleich a ist, d.h. wenn `weg[a][t]==0` und `t!=a`, wird `besuche(a,t)` rekursiv aufgerufen. Weil von jedem besuchten Chatonen b alle direkten Nachfolger-Chatonen besucht werden, werden durch diesen Vorgang alle direkten und indirekten Nachfolger von a im `weg` Array gespeichert.

Die Implementierung lautet wie folgt:

Rekursive Tiefensuche mit besuche(a,b)

```
besuche (int a, int b) {
    if (a != b) weg[a][b] = 1;
    struct chaton *t;
    for (t = anfang[b]; t != z; t = t->next)
```

[3] Hierdurch wird sichergestellt, dass kein Chaton eine an ihn gesandte Nachricht weiterschickt, wenn sie zuvor von ihm stammte.

```
        if (weg[a][t->v] == 0 && a != t->v) besuche(a,t->v);
}
```

Nun müssen für jeden Chaton i bzgl. jedem seiner Freunde t die Chatonen j gesucht werden, über die i an t schreiben darf. Dies lässt sich wie folgt implementieren:

Ausgabe

```
// Finde für jeden Chatonen i...
for (i = 0; i < V; i++) {
    // ...bzgl. jedem seiner Freunde t->v...
    for (t = anfang[i]; t != z; t = t->next) {
        // ...die Chatonen, über die i an t->v laestern darf,...
        sprintf(zeile, "");
        for (j = 0; j < V; j++)
            // ...d.h. die Chatonen j fuer die weg[t->v][j] 0 ist
                und die
            // nicht gleich i oder t sind.
            if (weg[t->v][j] == 0 && j != i && j != t->v)
                sprintf(zeile, "%s %c", zeile, j+65);
        if (strlen(zeile) > 0) printf("%c schreibt an %c ueber: %
           s\n", i+65,t->v+65, zeile);
    }
}
```

1.3 Programm-Ablaufprotokoll

tratsch1.in

Zunächst wird das Beispiel aus der Aufgabenstellung getestet:

tratsch1.in

```
A B E
B C
C D G
D C
E F
F B E G
G H
H D
```

Die Ausgabe des Programms ist:

```
$$ tratsch tratsch1.in
A schreibt an B ueber: E F
B schreibt an C ueber: A E F
C schreibt an G ueber: A B E F
C schreibt an D ueber: A B E F
D schreibt an C ueber: A B E F
E schreibt an F ueber: A
F schreibt an G ueber: A B E
```

```
F schreibt an E ueber: A
F schreibt an B ueber: A E
G schreibt an H ueber: A B E F
H schreibt an D ueber: A B E F
```

tratsch2.in

tratsch2.in

```
A
B A C E G
C A D E
D
E F A G
F A G
G A F
```

Die Ausgabe des Programms ist:

```
$$ tratsch tratsch2.in
B schreibt an G ueber: C D E
B schreibt an E ueber: C D
B schreibt an A ueber: C D E F G
C schreibt an E ueber: B D
C schreibt an D ueber: A B E F G
C schreibt an A ueber: B D E F G
E schreibt an G ueber: B C D
E schreibt an A ueber: B C D F G
E schreibt an F ueber: B C D
F schreibt an G ueber: B C D E
F schreibt an A ueber: B C D E G
G schreibt an F ueber: B C D E
G schreibt an A ueber: B C D E F
```

tratsch3.in

tratsch3.in

```
A B C D E F G
B A C D E F G
C A B D E F G
D A B C E F G
E A B C D F G
F A B C D E G
G A B C D E F
H
I J
J I
```

Die Ausgabe des Programms ist:

```
$$ tratsch tratsch3.in
A schreibt an G ueber: H I J
A schreibt an F ueber: H I J
A schreibt an E ueber: H I J
A schreibt an D ueber: H I J
A schreibt an C ueber: H I J
A schreibt an B ueber: H I J
B schreibt an G ueber: H I J
B schreibt an F ueber: H I J
B schreibt an E ueber: H I J
B schreibt an D ueber: H I J
B schreibt an C ueber: H I J
B schreibt an A ueber: H I J
C schreibt an G ueber: H I J
C schreibt an F ueber: H I J
C schreibt an E ueber: H I J
C schreibt an D ueber: H I J
C schreibt an B ueber: H I J
C schreibt an A ueber: H I J
D schreibt an G ueber: H I J
D schreibt an F ueber: H I J
D schreibt an E ueber: H I J
D schreibt an C ueber: H I J
D schreibt an B ueber: H I J
D schreibt an A ueber: H I J
E schreibt an G ueber: H I J
E schreibt an F ueber: H I J
E schreibt an D ueber: H I J
E schreibt an C ueber: H I J
E schreibt an B ueber: H I J
E schreibt an A ueber: H I J
F schreibt an G ueber: H I J
F schreibt an E ueber: H I J
F schreibt an D ueber: H I J
F schreibt an C ueber: H I J
F schreibt an B ueber: H I J
F schreibt an A ueber: H I J
G schreibt an F ueber: H I J
G schreibt an E ueber: H I J
G schreibt an D ueber: H I J
G schreibt an C ueber: H I J
G schreibt an B ueber: H I J
G schreibt an A ueber: H I J
I schreibt an J ueber: A B C D E F G H
J schreibt an I über: A B C D E F G H
```

Hier gibt es also einen Außenseiter H und eine Außenseiter-Gruppe mit den beiden Chatonen I und J.

1.4 Quellcode

```
#include <stdio.h>
```

```
#include <string.h>

#define IN "tratsch.in"
#define OUT "tratsch.out"

/* globale Variablen */
struct chaton
{ int v; struct chaton *next; };
struct chaton *t, *z;
struct chaton *anfang[11];
int weg[11][11]; // weg[a][b]: existiert Weg von a nach b
int schonBesucht[11], id;

/* Rekursive Tiefensuche */
besuche (int a, int b) {
    // wenn a von b verschieden, gibt es einen Weg von a nach b
    if (a != b) weg[a][b] = 1;
    //printf("%c ", b+65);
    // temporaerer chaton struct t
    struct chaton *t;
    // durchlaufe alle chatonen t in Adjazenzliste von b
    for (t = anfang[b]; t != z; t = t->next)
        // wenn t noch nicht besucht wurde, besuche t
        if (weg[a][t->v] == 0 && a != t->v) besuche(a,t->v);
}

/* main Funktion */
int main (int argc, char **argv) {
    // Pruefe auf Argument
    if (argc != 2) {
        printf("Keine Eingabedatei angegeben.\nAufruf: tratsch input-
           file.in\n");
        exit(0);
    }

    /* Variablen */
    FILE *in, *out;
    int i,j,k,x,y,V=0;
    char v1,v2;
    char zeile[20], rest[18];

    // Dateien
    in = fopen(argv[1], "r");
    out = fopen(OUT, "w");

    /* Initialisiere Adjazenzlisten mit auf sich selbst zeigenden
       Zeigern z */
    z = (struct chaton *) malloc(sizeof *z);
    z->next = z;
    for (i = 0; i < 11; i++) anfang[i] = z;

    /* Lese Sympathieliste ein */
    // Durchlaufe jede Zeile
    while (fgets(zeile, 20, in) != NULL) {
```

```
            // Inkrementiere Anzahl Chatonen V
55          V++;
            // x = erster Buchstabe
            x = zeile[0]-65;
            // durchlaufe den Rest der Zeile
            for (j = 1; 2*j < strlen(zeile); j++) {
60              // y = 2*j-ter Buchstabe
                y = zeile[2*j]-65;
                // Speichere y in Adjazenzliste von x
                t = (struct chaton *) malloc(sizeof *t);
                t->v = y; t->next = anfang[x]; anfang[x] = t;
65          }
        }

        /* Initalisiere weg[a][b] Array mit Nullen */
        for (i = 0; i < V; i++)
70          for (j = 0; j < V; j++)
                weg[i][j] = 0;

        /* Tiefensuche fuer jeden Chaton/von allen Knoten aus */
        for (i = 0; i < V; i++) {
75          // fuelle schonBesucht Array mit Nullen
            for (j = 0; j < V; j++) schonBesucht[j] = 0;
            // rekursive Tiefensuche besuche vom Knoten i aus
            besuche(i, i);
            //printf("\n");
80      }

        // Finde fuer jeden Chatonen i...
        for (i = 0; i < V; i++) {
            // ...bzgl. jedem seiner Freunde t->v...
85          for (t = anfang[i]; t != z; t = t->next) {
                // ...die Chatonen, ueber die i an t->v laestern darf,...
                sprintf(zeile, "");
                for (j = 0; j < V; j++)
                    // ...d.h. die Chatonen j fuer die weg[t->v][j] 0 ist
                        und die
90                  // nicht gleich i oder t sind.
                    if (weg[t->v][j] == 0 && j != i && j != t->v)
                        sprintf(zeile, "%s %c", zeile, j+65);
                if (strlen(zeile) > 0) printf("%c schreibt an %c ueber:%s\n
                    ", i+65,t->v+65, zeile);
            }
95      }
        free(z,t);
        fclose(in); fclose(out); exit(0);
    }
```

2 Kosmische Schaltjahre

2.1 Lösungsidee

Der einzige Parameter des Programms besteht in der Umlaufzeit a in X-Tagen. Zunächst wird aus dieser Information die Anzahl der Monate `numMonate` des Kalenders mit

```
numMonate = (int) a/30;
```

angenähert, d.h. es wird angenommen, dass alle Monate 30 Tage haben. Der entstehende Rest[4] zur tatsächlichen Umlaufzeit beträgt

```
rest = (int) a % 30;
```

wobei `%` für den Modulo-Operator steht. Wenn dieser `rest` größer als 15 ist, liegt es nahe, dem Kalender einen Monat hinzuzufügen und einen negativen Rest zu verwenden. Z.B. würde bei $a = 65$ die Anzahl der Monate 2 und der Rest 5 betragen, während bei $a = 88$ ein Kalender mit 3 Monaten zu bevorzugen wäre, da der Rest lediglich -2 statt 28 beträgt, wenn er negativ (d.h. von $3 \cdot 30 = 90$ nach unten statt von $2 \cdot 30 = 60$ nach oben) gezählt wird. Somit wird sichergestellt, dass $|\texttt{rest}| \leq 15$ gilt.

Der ganzzahlige Unterschied `rest` zum eigentlichen Kalender wird durch eine gleichmäßige Verteilung von `rest` auf die einzelnen Monate beseitigt. D.h. im Beispiel von $a = 65$ wird `rest=5` gleichmäßig auf die beiden Monate aufgeteilt, sodass der erste Monat 33 und der zweite Monat 32 X-Tage lang ist. Wenn $\texttt{rest} < 0$ gilt, muss der Rest durch Erniedrigung der Tage pro Monat ausgeglichen werden, d.h. bei $a = 88$ und $\texttt{rest} = -2$ haben der erste und der zweite Monat 29, der dritte Monat 30 Tage.

Die Verteilung der Resttage auf die Monate kann durch eine Iteration oder eine direkte Berechnung bestimmt werden. Die Iteration durchläuft die Monate `rest` Mal und inkrementiert (bzw. dekrementiert für $\texttt{rest} < 0$) bei jedem Schleifendurchlauf die Anzahl der Tage des gerade durchlaufenen Monats (siehe Programm-Dokumentation). Alternativ hierzu lässt sich die Verteilung der Resttage auf die Monate auch berechnen: $|\texttt{rest}| \% \texttt{numMonate}$ Monate haben

$$30 + \frac{\texttt{rest}}{\texttt{numMonate}} + \operatorname{sgn}(\texttt{rest})$$

Tage und $\texttt{numMonate} - (|\texttt{rest}| \% \texttt{numMonate})$ Monate sind

$$30 + \frac{\texttt{rest}}{\texttt{numMonate}}$$

Tage lang, wobei der Bruch $\frac{\texttt{rest}}{\texttt{numMonate}}$ in beiden Fällen zur nächstbetragskleineren Ganzzahl abgerundet[5] werden muss. Im Beispiel von $a = 88$ und $\texttt{rest} = -2$ ergeben sich somit $2\%3 = 2$ Monate mit $30 - 0 - 1 = 29$ Tagen und $3 - (2\%3) = 1$ Monat mit $30 - 0 = 30$ Tagen.

[4] Der Nachkomma-Rest wird später durch die Schaltjahrregelung behandelt. Hier ist zunächst nur der ganzzahlige Rest von Bedeutung.

[5] Dies entspricht dem Abschneiden des Nachkommateils, was in C durch `(int) (...)` ausgedrückt wird.

Somit wird der ganzzahlige Fehler `rest` des Kalenders also ausgeglichen. Um zudem auch den Nachkommateil der Abweichung des Kalenders von a einzubauen, muss eine *Schaltjahrregelung* eingeführt werden. Hierzu wird zunächst der Nachkommarest in der Variablen `schaltRest` gespeichert. Falls `schaltRest` > 0.5 wird der Rest wieder negativ gezählt, d.h. `schaltRest` wird zu `schaltRest` $- 1$ und zu dem ganzzahligen `rest` von oben wird sgn`rest` addiert. Bei z.B. $a = 88,78$ beträgt also `schaltRest` zunächst $0,78$. Wegen `schaltRest` > 0.5 wird aber `schaltRest` zu $-0,22$ und `rest` zu -1, d.h. `schaltRest` wird von 89 nach unten gezählt. Somit gilt |`schaltRest`| $\leq 0,5$.

Nun stellt sich die Frage nach sinnvollen Schaltjahrregelungen und dem Auffinden der geeigneten Regelung für einen bestimmten Wert von `schaltRest`. Wie sich zeigen wird, genügen einige wenige Regelungen, die für verschiedene Zahlenwerte durchlaufen werden. Die Regelung mit der geringsten Abweichung von `schaltRest` wird als Schaltjahrregelung für den Kalender übernommen.

Die folgenden vorgeschlagenen Schaltjahrregelungen sind in der Praxis gut umsetzbar, da sie für die Planetenbewohner recht leicht verständlich und berechenbar sind. Die Regelungen können von höchstens drei Parametern i, j und k abhängen, die jeweils für alle Werte von einschließlich 2 bis einschließlich 10 durchlaufen werden. Konkret lauten die Regeln wie folgt:

1. Jahre, die durch i teilbar sind.
2. Jahre, die durch i oder j teilbar sind.
3. Jahre, die durch i, j oder k teilbar sind.
4. Jahre, die durch i oder j, nicht aber durch k teilbar sind.

Hierbei wird für das Jahr 0 vereinbart, dass es durch keine Zahl teilbar ist.

Jede Regel R führt zu einer bestimmten Häufigkeit $q_R(i, j, k) = \frac{s}{n}$ von Schaltjahren s innerhalb eines Zeitraums von n Kalenderjahren. Bei *Regel 1* ist $q_1(i)$ leicht zu bestimmen, denn wenn jedes durch i teilbare, also jedes i-te Jahr ein Schaltjahr ist, bedeutet das, dass von n Jahren $s = n/i$ Jahre Schaltjahre sind, d.h.

$$q_1(i) = \frac{s}{n} = \frac{1}{i}.$$

Der Wert von $q_2(i, j)$ bei *Regel 2* kann wie folgt berechnet werden. Jedes i-te Jahr und jedes j-te Jahr ist ein Schaltjahr, d.h. von n Jahren sind $s = n/i + n/j$ Jahre Schaltjahre. Hierbei werden jedoch Jahre, die sowohl durch i als auch durch j teilbar sind, doppelt gezählt. Weil also jedes kgV(i,j)-te Jahr doppelt gezählt wird, muss von s noch der Term $n/\text{kgV(i,j)}$ subtrahiert werden, sodass

$$q_2(i, j) = \frac{s}{n} = \frac{1}{i} + \frac{1}{j} - \frac{1}{\text{kgV(i,j)}}.$$

Für z.B. $i = 2$ und $j = 5$ ergeben sich somit in den ersten 20 Jahren folgende Schaltjahre:

	Schaltjahre	*Schaltjahre pro Jahre*
durch $i = 2$ *teilbar*	2 4 6 8 10 12 14 16 18 20	$10/20 = 1/2 = 1/i$
durch $j = 5$ *teilbar*	5 10 15 20	$4/20 = 1/5 = 1/j$
durch $i = 2$ *und* $j = 5$ *teilbar*	10 20	$2/20 = 1/10 = 1/\text{kgV(i,j)}$

Insgesamt gibt es also 12 verschiedene Schaltjahre in den ersten 20 Jahren, d.h. $q_2(2,5) = 12/20 = 0,6$. Dies stimmt mit $q_2(2,5) = 1/2 + 1/5 - 1/10 = 6/10 = 0,6$ überein. Wenn `schaltRest` zufällig $0,6$ ist, wird also Regel 2 mit $i = 2$ und $j = 5$ für den Kalender verwendet, d.h. jedes Jahr, das durch 2 oder 5 teilbar ist, ist ein Schaltjahr.

Die Berechnung von $q_3(i,j,k)$ für *Regel 3* erfolgt ganz ähnlich. Zunächst ist jedes i-te, j-te und k-te Jahr ein Schaltjahr, d.h. $s = n/i + n/j + n/k$ Jahre von n Jahren sind Schaltjahre. Von s müssen analog zu Regel 2 die Terme $n/\text{kgV(i,j)}$, $n/\text{kgV(i,k)}$ und $n/\text{kgV(j,k)}$ subtrahiert werden, damit Jahre, die durch zwei Parameter zugleich teilbar sind, nicht doppelt gezählt werden. Zudem ist jedoch jedes kgV(i,j,k)-te Jahr durch i, j und k teilbar. Ein solches Jahr wird von n/i, n/j und n/k mitgezählt, dafür aber von $n/\text{kgV(i,j)}$, $n/\text{kgV(i,k)}$ und $n/\text{kgV(j,k)}$ wieder zurückgezählt, sodass es am Ende gar nicht gezählt wird. Um solche Jahre dennoch genau ein Mal zu zählen, muss zu s noch der Term $n/\text{kgV(i,j,k)}$ addiert werden. s ist also

$$\begin{aligned} s &= n \cdot \left(\frac{1}{i} + \frac{1}{j} + \frac{1}{k} - \frac{1}{\text{kgV(i,j)}} - \frac{1}{\text{kgV(i,k)}} - \frac{1}{\text{kgV(j,k)}} + \frac{1}{\text{kgV(i,j,k)}} \right) \\ \Rightarrow \quad q_3(i,j,k) &= \frac{s}{n} = \frac{1}{i} + \frac{1}{j} + \frac{1}{k} - \frac{1}{\text{kgV(i,j)}} - \frac{1}{\text{kgV(i,k)}} - \frac{1}{\text{kgV(j,k)}} + \frac{1}{\text{kgV(i,j,k)}} \end{aligned}$$

Für beispielsweise $i = 2$, $j = 4$, $k = 5$ ergibt sich somit für die ersten 20 Jahre folgende Tabelle

	Schaltjahre	*Schaltjahre pro Jahre*
durch $i = 2$ *teilbar*	2 4 6 8 10 12 14 16 18 20	$10/20 = 1/2 = 1/i$
durch $j = 4$ *teilbar*	4 8 12 16 20	$5/20 = 1/4 = 1/j$
durch $j = 5$ *teilbar*	5 10 15 20	$4/20 = 1/5 = 1/k$
durch $i = 2$ *und* $j = 4$ *teilbar*	4 8 12 16 20	$5/20 = 1/4 = 1/\text{kgV(i,j)}$
durch $i = 2$ *und* $k = 5$ *teilbar*	10 20	$2/20 = 1/10 = 1/\text{kgV(i,k)}$
durch $j = 4$ *und* $k = 5$ *teilbar*	20	$1/20 = 1/\text{kgV(j,k)}$
durch $i = 2$, $j = 4$, $k = 5$ *teilbar*	20	$1/20 = 1/\text{kgV(i,j,k)}$

Es gibt also 12 verschiedene Schaltjahre in den ersten 20 Jahren (2, 4, 5, 6, 8, 10, 12, 14, 15, 16, 18, 20), d.h. $q_3(2,4,5) = 12/20 = 0,6$. Zu dem gleichen Ergebnis führt

$$q_3(2,4,5) = \frac{1}{2} + \frac{1}{4} + \frac{1}{5} - \frac{1}{4} - \frac{1}{10} - \frac{1}{20} + \frac{1}{20} = \frac{10+5+4-5-2-1+1}{20} = \frac{12}{20} = 0,6.$$

Schließlich wird noch *Regel 4* betrachtet. Zunächst sind die Jahre Schaltjahre, die durch i oder j teilbar sind, d.h. $s = n \cdot (1/i + 1/j - 1/\text{kgV(i,j)})$. Hiervon werden jetzt

jedoch all die Jahre weggenommen, die durch k teilbar sind. D.h. die Jahre, die durch i und k bzw. j und k teilbar sind, werden von s subtrahiert, also

$$s = n \cdot (1/i + 1/j - 1/\text{kgV(i,j)} - 1/\text{kgV(i,k)} - 1/\text{kgV(j,k)}).$$

Hierbei werden aber Jahre, die durch i, j und k teilbar sind, doppelt subtrahiert, d.h. zu s muss noch der Term $n/\text{kgV(i,j,k)}$ addiert werden, sodass

$$\begin{aligned} s &= n \cdot \left(\frac{1}{i} + \frac{1}{j} - \frac{1}{\text{kgV(i,j)}} - \frac{1}{\text{kgV(i,k)}} - \frac{1}{\text{kgV(j,k)}} + \frac{1}{\text{kgV(i,j,k)}} \right) \\ \Rightarrow \quad q_4(i,j,k) &= \frac{1}{i} + \frac{1}{j} - \frac{1}{\text{kgV(i,j)}} - \frac{1}{\text{kgV(i,k)}} - \frac{1}{\text{kgV(j,k)}} + \frac{1}{\text{kgV(i,j,k)}} \end{aligned}$$

Auf ein Beispiel für diesen Fall wird verzichtet, da er im Prinzip analog zu Regel 3 ist.

Das Ziel einer Schaltjahrregelung ist nun, die Parameter i, j und k zu finden, für die $q_R(i,j,k)$ so nahe wie möglich bei `schaltRest` liegt, d.h. für die die Differenz $d = ||\texttt{schaltRest}| - |q_R(i,j,k)||$ minimal wird. Hierzu werden für i, j und k alle Werte von 2 bis 10 durchlaufen. Wird ein q_R berechnet, das zu einem neuem Minimum von d führt, werden i, j, k und R gespeichert. Das letztlich erhaltene Minimum wird als Schaltjahrregelung verwendet.

Wie oben beschrieben wurde wird `schaltRest` gegebenenfalls negativ gezählt, sodass immer $|\texttt{schaltRest}| \leq 0,5$ gilt. Folglich sind durchlaufene q, die größer als $0,5$ sind, nicht sinnvoll bzw. tragen nicht zu einer Verkleinerung von d bei. Aus diesem Grund wird bei $q > 0,5$ auch q „negativ gezählt“ und mit diesem q die zu optimierende Differenz d berechnet. „Negativ zählen“ bedeutet hier, dass alle vorherigen Nicht-Schaltjahre zu Schaltjahren und alle vorherigen Schaltjahre zu Nicht-Schaltjahren werden, d.h. die Schaltjahre- und Nicht-Schaltjahre-Aufteilung wird invertiert, d.h. die Anzahl Schaltjahre pro Jahre wird zu $1 - q$. Um die negative Zählung zu markieren, wird q noch mit -1 multipliziert, also $q = -1 \cdot (1 - q) = q - 1$. Bei der Ausgabe des Kalenders am Ende des Programms muss das Vorzeichen von q beachtet werden. Ist dieses ein Minus, so werden die Jahre, die Schaltjahre sind, negativ definiert, d.h. z.B. Regel 1 wird zu „Schaltjahre sind Jahre, deren Nummer *nicht* durch i teilbar ist“.

Es muss noch gezeigt werden, dass das Durchlaufen aller Kombinationen von $i, j, k = 2..10$ für alle vier Regeln zu einer ausreichend guten Annäherung des Schaltjahrrest `schaltRest` führt. Hierzu werden einmalig alle durchlaufenen q-Werte in einem Array `qList` gespeichert. Sortiert man `qList`, so können alle Werte geplottet werden (vgl. Abb. 2.2).

Wie zu erkennen ist liefern die vier verwendeten Regeln eine sehr dichte Verteilung von q-Werten, sodass jedes `schaltRest` mit $|\texttt{schaltRest}| \leq 0,5$ sehr genau angenähert werden kann. Die größte Differenz zweier aufeinanderfolgender q-Werte ist $0,05$ von $q = 0$ auf $q = 0,05$. Im schlechtesten Fall ist $\texttt{schaltRest} = 0,025$ und die Abweichung von Kalender zu Realität $d = q - \texttt{schaltRest} = 0,025$ Tage pro Jahr. D.h. im schlechtesten Fall weicht der von dem Programm vorgeschlagene Kalender nach $1/0,025 = 40$ Jahren um einen Tag von der Realität ab[6], was von den Bewohnern der Planeten angesichts

[6] Allg. ist die Anzahl der Jahre, nach der ein Kalender um einen Tag von der Realität abweicht, nichts weiter als der Kehrwert der Tage, die der Kalender in einem Jahr von der Realität abweicht.

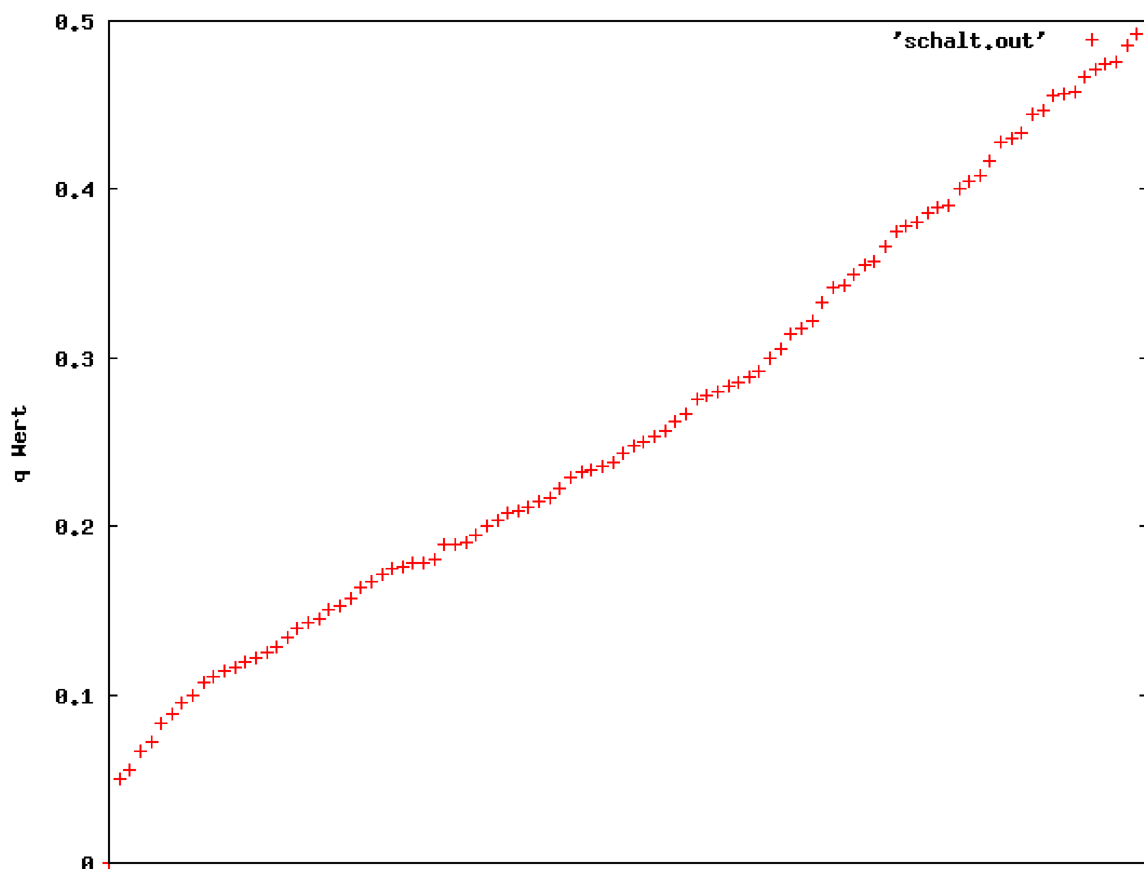


Abbildung 2.2: Verteilung der erzeugbaren q-Werte.

der recht leicht merkbaren und rechenbaren Schaltjahrregelungen sicherlich akzeptiert werden dürfte. Bemerkenswerterweise ist der zweitgrößte Abstand zweier benachbarter q-Werte bereits $0,012$, nämlich von $q = 0,083333$ auf $q = 0,071429$. Die Abweichung von Kalender zu Realität beträgt in diesem Fall höchstens $d = 0,012/2 = 0,006$ Tage pro Jahr, d.h. nach $1/0,006 = 166,6$ oder mehr Jahren beträgt die Abweichung des Kalenders zur Realität einen Tag.

Neben den schlechtmöglichsten Fälle ist es sicher sinnvoll, eine Art durchschnittliche Abweichung $\overline{d}$ der Kalender von der Realität zu finden. Hierzu werden 500000 Werte für `schaltRest` zwischen 0 und $0,5$ durchlaufen und die jeweiligen Abweichungen zum nächstliegenden q aufsummiert. $\overline{d}$ ist dann der Quotient aus der so erhaltenen Summe und 50000. Das Programm ermittelt für $\overline{d}$ den Wert $0,00271$ Tage pro Jahr, d.h. die mittlere Abweichung eines vorgeschlagenen Kalenders von der Realität beträgt einen Tag in $1/\overline{d} \approx 370$ Jahren. Dieses Ergebnis dürfte die Gesamtheit der Bewohner aller Planeten gewiss zufrieden stellen, wenn man unter Betracht zieht, dass wir im Universum als pingelig und kritisch-genau geltenden Erdlinge mit einer etwas größeren Abweichung

sehr gut zu Rande kommen.[7]

2.2 Programm-Dokumentation

Da das Programm nicht mehr als einige grundlegende Rechnungen und Operationen benötigt, wird wieder auf C als Programmiersprache zurückgegriffen. Die Implementation ist hierbei sehr eng an obiger Beschreibung der Lösungsidee gehalten und enthält im Prinzip kaum nennenswerte Besonderheiten. Dennoch soll hier der Vollständigkeit halber kurz auf einige Details eingegangen werden.

Zunächst wird der Wert von a aus `schalt.in` eingelesen. Mit diesem Wert werden wie oben beschrieben die Anzahl der Monate `numMonate`, der ganzzahlige Rest `rest` und der Schaltjahrrest `schaltRest` ermittelt. Die Aufteilung der Resttage auf die Monate wird wie folgt bewerkstelligt:

Aufteilung von `rest` Tagen auf `numMonate` Monate

```
int monate[numMonate];
// zunaechst haben alle Monate 30 Tage
for (i = 0; i < numMonate; i++) monate[i] = 30;
// der Rest wird der Reihe nach auf die Monate aufgeteilt
for (i = 0; i < absv2(rest); i++)
    monate[i%numMonate] = monate[i%numMonate] + sgn2(rest);
for (i = 0; i < numMonate; i++) printf("Monat %i: %i Tage\n", i
    +1, monate[i]);
```

Alternativ kann die Verteilung des Rests auch direkt angegeben werden:

```
printf("Also %i Monate mit %i Tagen und %i Monate mit %i Tagen.",
absv2(rest)%numMonate, 30+sgn2(rest)+rest/numMonate,
numMonate-(absv2(rest)%numMonate), 30+(rest/numMonate));
```

Die Ermittlung der Schaltjahrregelung erfolgt über Durchlaufen aller möglichen Parameter $i, j, k = 2..10$ mit $i \neq j \neq k$. Hierzu werden drei `for` Schleifen ineinandergesetzt, die i, j und k inkrementieren und bei Gleichheit zweier Parameter abbrechen. Bei jedem Durchlauf wird dann für jede Regel der Wert von $q_R(i, j, k)$ berechnet.[8] Falls der Wert von $q_R(i, j, k)$ zu einer minimalen Abweichung zu `schaltRest` führt, werden die Werte von R, i, j und k, sowie das Vorzeichen von q (um evtl. negatives Zählen des Schaltrestes zu markieren) im `schaltJahrSystem` Array abgelegt. Nach Beenden des Durchlaufens aller möglichen Parameterwerte befindet sich im `schaltJahrSystem` Ar-

[7] Wir können voraussetzen, dass die Erdlinge ihre Schaltjahrregelung nicht sehr gut kennen, d.h. glauben, dass jedes vierte Jahr ein Schaltjahr ist und Jahrhunderregelungen etc. außer Acht lassen. (Erdlinge sind auch sehr widersprüchliche Wesen betrachtet man ihre Pingeligkeit.) Dann ist die Abweichung des Erdkalenders von der Realität $0.25 - 0.24219 = 0.00781$ Tage pro Jahr, d.h. ein Tag in 128 Jahren.

[8] Hierzu wird eine Funktion für den kgV zweier natürlicher Zahlen x und y benötigt. Zunächst wird dazu mit dem Algorithmus von Euklid, der in der Schule besprochen wird und in der Literatur häufig zu finden ist (z.B. Andreas Bartholome, Zahlentheorie für Einsteiger, Vieweg, Braunschweig 2001, S. 46ff.) der ggT von x und y berechnet. Dann ist der kgV von x und y gerade $x \cdot y/ggT(x, y)$.

ray also die Kombination der Parameter mit der geringsten Abweichung von der Realität. Diese Parameterkombination wird dann als Schaltjahrregelung ausgegeben.

Nun wird noch die Abweichung des Kalenders von der Realität ausgegeben, indem von der Abweichung $|q_{opt} - \texttt{schaltRest}|$ Tage pro Jahr der Kehrwert gebildet wird, der aussagt, nach wie vielen Jahren der Kalender um einen Tag von der Realität abweicht.

Um die im vorigen Abschnitt getroffenen Aussagen über die Verteilung der Werte von q und die maximale und durchschnittliche Abweichung von der Realität machen zu können, wird jedes durchlaufene $q_R(i, j, k)$ in einem `qList` Array abgespeichert. Für die grafische Ausgabe dieser Werte mit gnuplot wird `qList` mit Selection Sort aufsteigend sortiert und anschließend in `schalt.out` geschrieben. Anschließend werden noch der mittlere Abstand der q-Werte und die mittlere Abweichung $\overline{d}$ berechnet. Dabei wird ein weiteres Array `qListIndex` in Anspruch genommen, um zu markieren an welchen Indexstellen in der sortierten `qList` zwei Nachbarwerte voneinander verschieden sind. (Viele Werte von q treten mehrmals in `qList` auf.)

2.3 Programm-Ablaufprotokoll

a = 365.24219 X-Tage

Das `schalt` Programm ist so implementiert, dass beim Aufruf ohne Argumente das Beispiel der Erde mit $a = 365.24219$ X-Tagen berechnet wird und einige allgmeine Aussagen getroffen werden, die bei einem normalen Aufruf mit a als Argument nicht angezeigt werden. Die Ausgabe lautet wie folgt:

```
$$ schalt
Kein Argument angegeben.
Aufruf: schalt "a", z.B. schalt "365.24219".
==> Nehme a=365.24219 an und treffe einige allgemeine Aussagen....

a = 365.242190 X-Tage
Zu verteilender Rest: 5 Tage
Monat 1: 31 Tage
Monat 2: 31 Tage
Monat 3: 31 Tage
Monat 4: 31 Tage
Monat 5: 31 Tage
Monat 6: 30 Tage
Monat 7: 30 Tage
Monat 8: 30 Tage
Monat 9: 30 Tage
Monat 10: 30 Tage
Monat 11: 30 Tage
Monat 12: 30 Tage
Also 5 Monate mit 31 Tagen und 7 Monate mit 30 Tagen.

Fuer obige Monatseinteilung ist eine Schaltjahrregelung noetig.
Aufzuteilender Schalt-Rest: 0.242190

Schaltjahrregelung:
```

```
Einer der obigen Monate, der sonst aus 30 Tagen besteht, besteht in
    einem Schaltjahr aus 31 Tagen.
Schaltjahre sind die Jahre, deren Nummer durch 6 oder 7, nicht aber
    durch 10 teilbar ist.
Nach ca. 1498 X-Jahren weicht dieser Kalender um einen X-Tag von den
    astronomischen Realitaeten ab.

Allgemeine Aussagen:
Groesster Abstand zwischen zwei benachbarten q-Werten: 0.05 (von 0 auf
    0.05)
Zweitgroesster Abstand zwischen zwei benachbarten q-Werten: 0.011905 (
    von 0.071429 auf 0.083333)
98 verschiedene Werte fuer q ermittelt.
Der mittlere Abstand zwischen zwei q-Werten ist 0.004592.
Die mittlere Abweichung eines Kalenders von der Realitaet ist 0.002707
    Tage pro Jahr, d.h. ein Tag in 369.360678 Jahren.
```

Zur Überprüfung der Arbeitsweise: $5 \cdot 31 + 7 \cdot 30 = 365$,

$$\begin{aligned} q_4(i=6, j=7, k=10) &= \frac{1}{6} + \frac{1}{7} - \frac{1}{\text{kgV(6,7)}} - \frac{1}{\text{kgV(6,10)}} - \frac{1}{\text{kgV(7,10)}} + \frac{1}{\text{kgV(6,7,10)}} \\ &= \frac{1}{6} + \frac{1}{7} - \frac{1}{42} - \frac{1}{30} - \frac{1}{70} + \frac{1}{210} \\ &= 0,242857 \end{aligned}$$

D.h. $365,242857 = a + 0,000667 = a + 1498^{-1}$.

a = 416,78132 X-Tage

```
$$ schalt "416.78132"
a = 416.781320 X-Tage
Zu verteilender Rest: -3 Tage
Monat 1: 29 Tage
Monat 2: 29 Tage
Monat 3: 29 Tage
Monat 4: 30 Tage
Monat 5: 30 Tage
Monat 6: 30 Tage
Monat 7: 30 Tage
Monat 8: 30 Tage
Monat 9: 30 Tage
Monat 10: 30 Tage
Monat 11: 30 Tage
Monat 12: 30 Tage
Monat 13: 30 Tage
Monat 14: 30 Tage
Also 3 Monate mit 29 Tagen und 11 Monate mit 30 Tagen.

Fuer obige Monatseinteilung ist eine Schaltjahrregelung noetig.
Aufzuteilender Schalt-Rest: -0.218680

Schaltjahrregelung:
```

```
Einer der obigen Monate, der sonst aus 30 Tagen besteht, besteht in
    einem Schaltjahr aus 29 Tagen.
Schaltjahre sind die Jahre, deren Nummer durch 5 oder 9, nicht aber
    durch 4 teilbar ist.
Nach ca. 496 X–Jahren weicht dieser Kalender um einen X–Tag von den
    astronomischen Realitaeten ab.
```

Zur Überprüfung der Arbeitsweise: $3 \cdot 29 + 11 \cdot 30 = 417$,

$$\begin{aligned} q_4(i=5,j=9,k=4) &= \frac{1}{5}+\frac{1}{9}-\frac{1}{\text{kgV(5,9)}}-\frac{1}{\text{kgV(5,4)}}-\frac{1}{\text{kgV(9,4)}}+\frac{1}{\text{kgV(5,9,4)}} \\ &= \frac{1}{5}+\frac{1}{9}-\frac{1}{45}-\frac{1}{20}-\frac{1}{36}+\frac{1}{180} \\ &= 0,21\overline{6} \end{aligned}$$

Also $417 - 0,21\overline{6} = 416,78333 = a + 0,00201 = a + 496^{-1}$.

a = 987.6543 X-Tage

```
$$ schalt "987.6543"
a = 987.654300 X–Tage
Zu verteilender Rest: −2 Tage
Monat 1: 29 Tage
Monat 2: 29 Tage
Monat 3: 30 Tage
Monat 4: 30 Tage
Monat 5: 30 Tage
[...]
Monat 33: 30 Tage
Also 2 Monate mit 29 Tagen und 31 Monate mit 30 Tagen.

Fuer obige Monatseinteilung ist eine Schaltjahrregelung noetig.
Aufzuteilender Schalt−Rest: −0.345700

Schaltjahrregelung:
Einer der obigen Monate, der sonst aus 30 Tagen besteht, besteht in
    einem Schaltjahr aus 29 Tagen.
Schaltjahre sind die Jahre, deren Nummer nicht durch 2, 5 oder 7
    teilbar ist.
Nach ca. 351 X–Jahren weicht dieser Kalender um einen X–Tag von den
    astronomischen Realitaeten ab.
```

Zur Überprüfung der Arbeitsweise: $2 \cdot 29 + 31 \cdot 30 = 988$, Inversion von Regel 3:

$$\begin{aligned} 1-q_3(i=2,j=5,k=7) &= 1-\frac{1}{2}+\frac{1}{5}+\frac{1}{7}-\frac{1}{\text{kgV(2,5)}}-\frac{1}{\text{kgV(2,7)}}-\frac{1}{\text{kgV(5,7)}}+\frac{1}{\text{kgV(2,5,7)}} \\ &= 1-\frac{1}{2}+\frac{1}{5}+\frac{1}{7}-\frac{1}{10}-\frac{1}{14}-\frac{1}{35}+\frac{1}{70} \\ &= 1-0,657143 = 0,342857 \end{aligned}$$

Also $988 - 0,342857 = 987,657143 = a + 0.002843 = a + 351^{-1}$.

a = 779,94 X-Tage (Mars)

Der Mars hat eine synodische Periode von a=779,94 Erdentagen.[9]

```
$$ schalt "779.94"
a = 779.940000 X-Tage
Zu verteilender Rest: 0 Tage
Monat 1: 30 Tage
Monat 2: 30 Tage
Monat 3: 30 Tage
[...]
Monat 26: 30 Tage
Also 26 Monate mit 30 Tagen.

Fuer obige Monatseinteilung ist eine Schaltjahrregelung noetig.
Aufzuteilender Schalt-Rest: -0.060000

Schaltjahrregelung:
Einer der obigen Monate, der sonst aus 30 Tagen besteht, besteht in
    einem Schaltjahr aus 29 Tagen.
Schaltjahre sind die Jahre, deren Nummer durch 6 oder 9, nicht aber
    durch 2 teilbar ist.
Nach ca. 225 X-Jahren weicht dieser Kalender um einen X-Tag von den
    astronomischen Realitaeten ab.
```

Zur Überprüfung der Arbeitsweise: $26 \cdot 30 = 780$, Regel 4:

$$\begin{aligned} q_4(i=6, j=9, k=2) &= \frac{1}{6} + \frac{1}{9} - \frac{1}{\text{kgV(6,9)}} - \frac{1}{\text{kgV(6,2)}} - \frac{1}{\text{kgV(9,2)}} + \frac{1}{\text{kgV(6,9,2)}} \\ &= \frac{1}{6} + \frac{1}{9} - \frac{1}{18} - \frac{1}{6} - \frac{1}{18} + \frac{1}{18} \\ &= 0,0\overline{5} \end{aligned}$$

Also $780 - 0,0\overline{5} = 779,9\overline{4} = a + 0,0044 = a + 225^{-1}$

a = 583,92 X-Tage (Venus)

Die Venus hat eine synodische Periode von a=583,92 Erdentagen.

```
$$ schalt "583.92"
a = 583.920000 X-Tage
Zu verteilender Rest: 14 Tage
Monat 1: 31 Tage
Monat 2: 31 Tage
Monat 3: 31 Tage
Monat 4: 31 Tage
Monat 5: 31 Tage
Monat 6: 31 Tage
Monat 7: 31 Tage
Monat 8: 31 Tage
```

[9] Vgl. http://de.wikipedia.org/wiki/Mars (Planet), die synodische Periode ist die Zeitperiode nach der dieselben Beleuchtungsverhältnisse oder Phasen entstehen. Diese Zeitspanne ist insbesondere sinnvoll für beispielsweise Einteilung in Jahreszeiten und v.a. Kalender.

```
Monat 9: 31 Tage
Monat 10: 31 Tage
Monat 11: 31 Tage
Monat 12: 31 Tage
Monat 13: 31 Tage
Monat 14: 31 Tage
Monat 15: 30 Tage
Monat 16: 30 Tage
Monat 17: 30 Tage
Monat 18: 30 Tage
Monat 19: 30 Tage
Also 14 Monate mit 31 Tagen und 5 Monate mit 30 Tagen.

Fuer obige Monatseinteilung ist eine Schaltjahrregelung noetig.
Aufzuteilender Schalt-Rest: -0.080000

Schaltjahrregelung:
Einer der obigen Monate, der sonst aus 30 Tagen besteht, besteht in
    einem Schaltjahr aus 29 Tagen.
Schaltjahre sind die Jahre, deren Nummer durch 8 oder 9, nicht aber
    durch 4 teilbar ist.
Nach ca. 300 X-Jahren weicht dieser Kalender um einen X-Tag von den
    astronomischen Realitaeten ab.
```

Zur Überprüfung der Arbeitsweise: $13 \cdot 31 + 5 \cdot 30 = 584$, Regel 4:

$$\begin{aligned} q_4(i = 8, j = 9, k = 4) &= \frac{1}{8} + \frac{1}{9} - \frac{1}{\text{kgV(8,9)}} - \frac{1}{\text{kgV(8,4)}} - \frac{1}{\text{kgV(9,4)}} + \frac{1}{\text{kgV(8,9,4)}} \\ &= \frac{1}{8} + \frac{1}{9} - \frac{1}{72} - \frac{1}{8} - \frac{1}{36} + \frac{1}{72} \\ &= 0,08333 \end{aligned}$$

Also $584 - 0,08333 = 583,91666 = a - 0,00333 = a - 300^{-1}$.

2.4 Quellcode

```
#include <stdio.h>
#include <string.h>

#define TageMax 2000

/* Kleinstes gemeinsames Vielfaches mit Eulerschem Verfahren*/
double kgv (int x, int y) {
    // besser x>y
    int r, a = x, b = y,c=0;
    do {
        r = a%b;
        a = b; b = r; c++;
    }
    while (r != 0);
    return x*y/a;
}
```

```c
/* Absolutbetrag */
double absv (double x) {
    return (x >= 0) ? x : -x;
}
int absv2 (int x) {
    return (x >= 0) ? x : -x;
}
/* Signum */
double sgn (double x) {
    if (x == 0.) return 0.;
    return x/absv(x);
}
int sgn2 (int x) {
    if (x == 0) return 0;
    return x/absv2(x);
}

/* main Funktion */
int main (int argc, char **argv) {
    /* Variblen */
    double a, x, d, schaltRest, minDiff=100.;
    int i,j,k,numMonate,c=0,c2=0,T=11,rest, allgemein=0;
    int schaltJahrSystem[5]; // 0: negativRegel, 1: system art, 2: arg1
        , 3: arg2, 4: arg3
    double qList[100000]; int qListIndex[100000];

    // Pruefe auf Argument
    if (argc > 2) {
        printf("Zu viele Argumente.\nAufruf: schalt \"a\", z.B. schalt
            \"365.24219\".\n");
        exit(0);
    }
    if (argc == 1) {
        printf("Kein Argument angegeben.\nAufruf: schalt \"a\", z.B.
            schalt \"365.24219\".\n==> Nehme a=365.24219 an und treffe
            einige allgemeine Aussagen....\n\n");
        allgemein = 1;
        a = 365.24219;
    }
    if (argc == 2)
        // Lese Umlaufzeit a in X-Tagen ein
        sscanf(argv[1], "%lf", &a);
    // Gebe Anzahl Tage aus
    printf("a = %lf X-Tage\n",a);

    // Anzahl der Monate ist zunächst a/30 abgerundet
    numMonate = (int) a/30;
    // übriger Rest, wenn alle numMonate Monate 30 Tage hätten
    rest = (int) a % 30;
    // Nachkommateil von a ist schalRest
    schaltRest = a-(int) a;
    // zähle diesen Rest gegebenenfalls nach unten
    if (schaltRest > 0.5) {
```

```
            schaltRest = schaltRest -1;
            // ein Tag mehr muss auf Monate verteilt werden
            rest += sgn2(rest);
70      }
        // Wenn rest größer als 15 Tage ist
        if (rest > 15) {
            // erhöhe die Anzahl der Monate
            numMonate++;
75          // zähle Rest nach unten
            rest = -1*(30-rest);
        }


80      /* Teile rest Tage auf numMonate Monate auf */
        int monate[numMonate];
        // zunaechst haben alle Monate 30 Tage
        for (i = 0; i < numMonate; i++) monate[i] = 30;
        // der Rest wird der Reihe nach auf die Monate aufgeteilt
85      printf("Zu verteilender Rest: %i Tage\n",rest);
        for (i = 0; i < absv2(rest); i++)
            monate[i%numMonate] = monate[i%numMonate] + sgn2(rest);
        for (i = 0; i < numMonate; i++) printf("Monat %i: %i Tage\n", i+1,
           monate[i]);
        // alternativ kann die Verteilung des Rest auch direkt angegeben
           werden:
90      if (rest == 0)
            printf("Also %i Monate mit %i Tagen.\n", numMonate-(absv2(rest)
                %numMonate), 30+(rest/numMonate));
        else
            printf("Also %i Monate mit %i Tagen und %i Monate mit %i Tagen
                .\n", absv2(rest)%numMonate, 30+sgn2(rest)+rest/numMonate,
                numMonate-(absv2(rest)%numMonate), 30+(rest/numMonate));


95
        /** Ermittlung einer sinnvollen Schaltjahrregelung **/

        if (schaltRest != 0) {
            printf("\nFuer obige Monatseinteilung ist eine
                Schaltjahrregelung noetig.\nAufzuteilender Schalt-Rest: %lf\
                n",schaltRest);
100
            /* Durchlaufe mögliche Schaltjahrkombinationen */
            for (i = 2; i < T; i++) {
                // Jahre, die durch i teilbar sind
                x = 1./i;
105             if (absv(absv(x)-absv(schaltRest)) <= minDiff) {
                    minDiff = absv(absv(x)-absv(schaltRest));
                    schaltJahrSystem[0] = sgn(x); // Schaltjahre/Jahre q
                    schaltJahrSystem[1] = 0; // Systemtyp
                    schaltJahrSystem[2] = i; // Argument 1
110             }
                qList[++c] = x;
                for (j = 2; j < T; j++) {
                    // Jahre, die durch i oder j teilbar sind
```

```
                if (i == j) continue;
115             x = 1./i+1./j-1./kgv(j,i);
                if (x > 0 && x < 1) {
                    if (x > 0.5) { x = x-1; c2++;}
                    d = absv(absv(x)-absv(schaltRest));
                    if (d < minDiff || (d == minDiff && 1 <
                        schaltJahrSystem[1])) {
120                     minDiff = absv(absv(x)-absv(schaltRest));
                        schaltJahrSystem[0] = sgn(x);
                        schaltJahrSystem[1] = 1;
                        schaltJahrSystem[2] = i;
                        schaltJahrSystem[3] = j;
125                 }
                    qList[++c] = x;
                }
                for (k = 2; k < T; k++) {
                    // Jahre, die durch i,j oder k teilbar sind
130                 if (k == i || k == j) continue;
                    x = 1./i+1./j+1./k - 1./kgv(j,i)-1./kgv(k,i)-1./kgv
                        (k,j)+1./kgv(k, kgv(j,i));
                    if (x > 0 && x < 1) {
                        if (x > 0.5) { x = x-1; c2++; }
                        d = absv(absv(x)-absv(schaltRest));
135                     if (d < minDiff || (d == minDiff && 2 <
                            schaltJahrSystem[1])) {
                            minDiff = absv(absv(x)-absv(schaltRest));
                            schaltJahrSystem[0] = sgn(x);
                            schaltJahrSystem[1] = 2;
                            schaltJahrSystem[2] = i;
140                         schaltJahrSystem[3] = j;
                            schaltJahrSystem[4] = k;
                        }
                        qList[++c] = x;
                    }
145                 // Jahre, die durch i oder j, nicht aber durch k
                        teilbar sind
                    x = 1./i+1./j-1./kgv(j,i) - (1./kgv(k,i)+1./kgv(k,j
                        )-1./kgv(k, kgv(j,i)));
                    if (x > 0 && x < 1) {
                        if (x > 0.5) { x = x-1; c2++;}
                        d = absv(absv(x)-absv(schaltRest));
150                     if (d < minDiff) {
                            minDiff = absv(absv(x)-absv(schaltRest));
                            schaltJahrSystem[0] = sgn(x);
                            schaltJahrSystem[1] = 3;
                            schaltJahrSystem[2] = i;
155                         schaltJahrSystem[3] = j;
                            schaltJahrSystem[4] = k;
                        }
                        qList[++c] = x;
                    }
160             }
            }
        }
```

```
            /* Ausgabe der ermittelten Schaltjahrregelung */
165         // Wenn es mind. 1 Monat mit numMonate-(rest%numMonate) Tagen
                gibt..
            if (numMonate-(absv2(rest)%numMonate) > 0) {
                if (rest == 0)
                    printf("\nSchaltjahrregelung:\nEiner der obigen Monate,
                         der sonst aus %i Tagen besteht, besteht in einem
                        Schaltjahr aus %i Tagen.\n", 30+rest/numMonate, 30+
                        rest/numMonate+(int) sgn(schaltRest));
                else
170                 printf("\nSchaltjahrregelung:\nEiner der obigen Monate,
                         der sonst aus %i Tagen besteht, besteht in einem
                        Schaltjahr aus %i Tagen.\n", 30+rest/numMonate, 30+
                        rest/numMonate+(int) sgn(schaltRest));
            }
            else
                printf("\nSchaltjahrregelung:\nEiner der obigen Monate, der
                     sonst aus %i Tagen besteht, besteht in einem Schaltjahr
                     aus %i Tagen.\n", 30+sgn2(rest)+rest/numMonate, 30+rest
                    /numMonate+sgn2(rest)+(int) sgn(schaltRest));
            printf("Schaltjahre sind die Jahre, deren Nummer ");
175         if (schaltJahrSystem[0] == -1) printf("nicht ");
            switch (schaltJahrSystem[1]) {
            case 0: printf("durch %i teilbar ist.\n", schaltJahrSystem[2]);
                 break;
            case 1: printf("durch %i oder %i teilbar ist.\n",
                schaltJahrSystem[2], schaltJahrSystem[3]); break;
            case 2: printf("durch %i, %i oder %i teilbar ist.\n",
                schaltJahrSystem[2], schaltJahrSystem[3], schaltJahrSystem
                [4]); break;
180         case 3:
                printf("durch %i oder %i, ",schaltJahrSystem[2],
                    schaltJahrSystem[3]);
                if (schaltJahrSystem[0] == 1) printf("nicht ");
                printf("aber durch %i teilbar ist.\n", schaltJahrSystem[4])
                    ;
                break;
185         }
        }

        // Berechne Abweichung
        if (schaltRest == 0.)
190         printf("Der Kalender weicht nicht von den astronomischen
                Realitaeten ab.");
        else if (minDiff < 10e-9)
            printf("Der Kalender weicht in 10^9 X-Jahren weniger als einen
                X-Tag von den astronomischen Realitaeten ab.");
        else
            printf("Nach ca. %i X-Jahren weicht dieser Kalender um einen X-
                Tag von den astronomischen Realitaeten ab.\n",(int) (1./
                minDiff));
195
```

```
    if (allgemein == 1) {
        printf("\nAllgemeine Aussagen:\n");
        FILE *out;
200     out = fopen("schalt.out", "w");

        // Sortiere qList Array für grafische Ausgabe mit Selection
            Sort
        int min;
        for (i=0; i<c; i++) {
205         min = i;
            for (j=i+1;j<c; j++)
                if (qList[j]<qList[min]) min = j;
            x=qList[min]; qList[min] = qList[i]; qList[i]=x;
        }
210     // schreibe qList Array in schalt.out
        fprintf(out,"%lf\n",qList[0]);
        for (i=1; i<c; i++)
            if (qList[i]<1. && qList[i]>0. && qList[i]-qList[i-1]>10e
                -6) fprintf(out,"%lf\n",qList[i]);

215     x = 0.;
        for (i=0; i<c; i++)
            if (qList[i+1]-qList[i] != 0. && qList[i] > 0.05 && qList[i
                +1] > 0.05 && qList[i+1]-qList[i] > x)
            { x = qList[i+1]-qList[i]; j=i;}
        printf("Groesster Abstand zwischen zwei benachbarten q-Werten:
            0.05 (von 0 auf 0.05)\n");
220     printf("Zweitgrößter Abstand zwischen zwei benachbarten q-
            Werten: %lf (von %lf auf %lf)\n",x,qList[j],qList[j+1]);

        // mittlerer Abstand der q-Werte
        j = 0; x=0;
        for (i=0; i<c; i++)
225         if (qList[i+1]-qList[i]>10.e-6 && qList[i] > 10.e-6 &&
                qList[i+1] > 10.e-6)
            { x += qList[i+1]-qList[i]; qListIndex[++j]=i; }
        printf("%i verschiedene Werte für q ermittelt.\nDer mittlere
            Abstand zwischen zwei q-Werten ist %lf.\n",j,x/j);

        // mittlere Abweichung d_quer
230     j = 0; x=0;
        for (i=0; i<500000; i++) {
            //if (j>96) printf("%i %lf %lf   %lf < %lf  add %lf\n",j,i
                /1000000.,qList[qListIndex[j]],i/1000000.-qList[
                qListIndex[j]],i/1000000.-qList[qListIndex[j+1]],i
                /1000000.-qList[qListIndex[j]]);
            if (absv(i/1000000.-qList[qListIndex[j]])<absv(i/1000000.-
                qList[qListIndex[j+1]]))
            { x += absv(i/1000000.-qList[qListIndex[j]]); printf(""); }
235         else { x += absv(i/1000000.-qList[qListIndex[j]]); j++;
                printf("");}
        }
        printf("Die mittlere Abweichung eines Kalenders von der
            Realitaet ist %lf Tage pro Jahr, d.h. ein Tag in %lf Jahren
```

```
                .\n",x/500000.,500000./x);

            fclose(out);
240     }
        printf("\n");
        exit(0);
    }
```

3 Schweizer Käse

3.1 Lösungsidee, Programmdokumentation

Der Würfel kann als 3-dimensionales boolean Array `wuerfel[20][20][20]` modelliert werden, wenn `true` für Luft und `false` für Käse steht. Das vorgegebene Käse-Luft-Verhältnis wird mit einem Zufallssimulator erreicht:

```
/* Erzeuge Würfel nach vorgegebenem p */
for (i=0;i<20;i++)
  for (j=0;j<20;j++)
    for (k=0;k<20;k++) {
      if(random(1)>p)            // Zufallszahl von 0 bis 1
        wuerfel[i][j][k] = true; // true = Luft, false = Käse
      else
        wuerfel[i][j][k] = false; // true = Luft, false = Käse
    }
```

Um herauszufinden, ob durch den erzeugten Würfel Wasser fließen kann, wird ein Weg bzw. Pfad durch den Würfel gesucht. Begonnen wird hierbei auf der obersten Ebene des Würfels mit allen $0 \leq i, j < 20$ mit `wuerfel[i][j][0] == true`. Für jedes solcher Luftkästchen auf der obersten Ebene wird die rekursive `besuche(i,j,0)` Funktion aufgerufen. Diese überpüft ausgehend von dem ihr als Argument übergebenen Luftkästchen alle Nachbarkästchen auf Luftgehalt. Findet sie ein Luftkästchen in der Nachbarschaft so wird dieses natürlich sofort ebenfalls rekursiv besucht. D.h.

```
bool besuche(int x, int y, int z) {
  wuerfel[x][y][z] = false;      // Markiere als besucht, also
     Käse
  if(z == 19)
    return true;
  if(wuerfel[x][y][z+1] == true)
    if(besuche(x,y,z+1) == true)
      return true;
 if(wuerfel[x][y][z-1] == true)
    if(besuche(x,y,z-1) == true)
      return true;
  if(wuerfel[x][y+1][z] == true)
    if(besuche(x,y+1,z))
      return true;
  if(wuerfel[x][y-1][z] == true)
    if(besuche(x,y-1,z))
      return true;
  if(wuerfel[x+1][y][z] == true)
    if(besuche(x+1,y,z))
      return true;
  if(wuerfel[x-1][y][z] == true)
    if(besuche(x-1,y,z) == true)
      return true;
```

```
    return false;
  }
```

Sobald eine der am Anfang für die oberste Ebene aufgerufene `besuche` Funktion den Wert `true` zurückgibt, existiert ein Luftweg durch den Würfel.

3.2 Quellcode

```
#include <iostream>
using namespace std;

double random(int);                  // Aus random.cpp
bool besuche(int,int,int);

bool wuerfel[20][20][20];            // Dimensionen Würfel: x, y, z

int main(int argc, char** argv) {
  srand(time(0));                    // Initialisiere Zufallsgenerator
  int i, j, k, l = 0;                // Counterhilfsvariablen
  bool succeeded = false;            // Einen Weg durch den Würfel gefunden
  double p = 0.6;
  if(argc > 1) sscanf(argv[1], "%lf", &p);

  /* Erzeuge Würfel nach vorgegebenem p */
  for (i=0;i<20;i++)
    for (j=0;j<20;j++)
      for (k=0;k<20;k++) {
        if(random(1)>p)              // Zufallszahl von 0 bis 1
          wuerfel[i][j][k] = true; // true = Luft, false = Käse
        else
          wuerfel[i][j][k] = false; // true = Luft, false = Käse
      }

  for (i=0;i<20 && succeeded==false;i++)
    for (j=0;j<20 && succeeded==false;j++)
      if(wuerfel[i][j][0] == true) // Wenn aktuelles Kästchen = Luft
        if(besuche(i,j,0)) {
          succeeded = true;
        }

  if(succeeded)
    cout << "Es existiert ein Luftweg durch den Würfel."
         << endl;

  return 0;
}

bool besuche(int x, int y, int z) {
  wuerfel[x][y][z] = false;          // Markiere als besucht, also Käse
  if(z == 19)
    return true;
```

```
    if(wuerfel[x][y][z+1] == true)
45    if(besuche(x,y,z+1) == true)
        return true;
   if(wuerfel[x][y][z-1] == true)
      if(besuche(x,y,z-1) == true)
        return true;
50  if(wuerfel[x][y+1][z] == true)
      if(besuche(x,y+1,z))
        return true;
    if(wuerfel[x][y-1][z] == true)
      if(besuche(x,y-1,z))
55      return true;
    if(wuerfel[x+1][y][z] == true)
      if(besuche(x+1,y,z))
        return true;
    if(wuerfel[x-1][y][z] == true)
60    if(besuche(x-1,y,z) == true)
        return true;

    return false;
  }
```

```
#include <cstdlib>                  // For random number generator
#include <ctime>                    // For time function
using namespace std;

5 double random(int count) {
    return static_cast<double>((count*static_cast<double>(rand()))/(
        RAND_MAX+1.0));
  }
```

4 Klasse Arbeit

4.1 Lösungsidee

Die Eingabedatei gibt die in einem Band enthaltenen Aufgaben an. So ist es möglich, für jede Aufgabe *i* eine Liste `aList[i]` von Bänden zu erstellen, in denen diese Aufgabe enthalten ist. In dem Aufgabenbeispiel enthält die Liste für Aufgabe 1 also die Bänder 1 und 6. Für die Aufgaben, die bereits von älteren Schülern gesammelt wurden, ist eine derartige Liste natürlich überflüssig, d.h. sie wird in diesem Fall nicht angelegt. Für alle anderen Aufgaben wird die Liste jedoch aufgebaut, in dem Aufgabenbeispiel:

Aufgabe	in Bänden
3	2, 6
5	2, 4
9	1, 8
11	4, 5

Aufgabe	in Bänden
13	3, 7
15	1, 2, 5
17	6
19	4, 7

Anhand dieses Beispiels ist die Lösungsidee auch schon recht schnell ersichtlich. Band 6 muss auf jeden Fall gekauft werden, da ein Kauf von Band 6 die einzige Möglichkeit darstellt, Aufgabe 17 zu erhalten. Um Aufgabe 3 zu erhalten, muss Band 2 oder 6 gekauft werden, für Aufgabe 5 benötigt man Band 2 oder 4. D.h. für jede Aufgabe der obigen Tabelle muss mind. ein Band gekauft werden. Mögliche Kaufkombinationen für Bände, die alle Aufgaben abdecken, bestehen also aus *Pfaden* durch die Bände obiger Tabelle. Ein möglicher solcher Pfad wäre z.B.

$$2 \to 2 \to 8 \to 5 \to 7 \to 1 \to 6 \to 4.$$

Das Problem besteht nun darin, alle möglichen Pfade zu durchlaufen und dabei den Pfad mit minimalen Kosten, d.h. möglichst wenigen voneinander verschiedenen Bänden, aufzufinden. Um alle Pfade abzuarbeiten, wird die obige Tabelle als *Wald* mit einem oder mehreren *Bäumen* aufgefasst, die als Knoten die Bände auf den verschiedenen Aufgabenebenen besitzen. Die obige Tabelle würde als Wald mit zwei Bäumen also wie folgt aussehen:[10]

Aufgabe	*Baum 1*	*Baum 2*
3	2	6
5	2 4	2 4
9	1 8 1 8	1 8 1 8
11	4 5 4 5 4 5 4 5	4 5 4 5 4 5 4 5
...	...	...

[10] In der Tabelle denke man sich Verbindungen von jedem Knoten zu den mittig darunter liegenden Nachfolger-Knoten.

Um die Gesamtanzahl der Knoten zu verringern, werden die Aufgaben, die in wenigen Bänden liegen, möglichst weit oben in die Bäume gesetzt.[11] In dem Programm werden die Aufgabenlisten mit den sie enthaltenden Bänden also sortiert und anschließend nach ansteigender Anzahl an Bänden innerhalb der Listen in die Bäume geschrieben, im Aufgabenbeispiel:

Aufgabe	*Baum 1*
17	6
3	2 6
5	2 4 2 4
9	1 8 1 8 1 8 1 8
...	...

Aufgabe 15 wird in diesem Baum als letzte Aufgabe eingebaut, da sie in drei Bänden enthalten ist.

Ist der Baum erst einmal konstruiert muss er nur noch durchlaufen werden. Das Problem hierbei besteht jedoch darin, dass es nicht ganz einfach ist die Kosten eines Pfades zu berechnen, die durch die Anzahl der auf ihm liegenden, voneinander verschiedenen Bänder bestimmt sind. Mit Hilfe einer rekursiven Traversierungsprozedur `visit` ist die Berechnung der Kosten aber auf recht elegante Art und Weise möglich. Zunächst wird `visit` aus dem Hauptprogramm für die Stämme aller Bäume aufgerufen. `visit(b)` arbeitet dann wie folgt: Zuerst wird der Besucht-Status des gerade besucht werdenden Bandes b um 1 inkrementiert. Anschließend wird `visit` für alle Baumnachfolger von b rekursiv aufgerufen. Existieren keine Baumnachfolger, so werden die Kosten des gerade entstandenen Pfads anhand der Werte im Besucht-Status Array berechnet und gegebenenfalls der Pfad gespeichert. Zuletzt wird der Besucht-Status von b wieder um 1 dekrementiert.

Nach Beenden der Traversierung aller Pfade des Waldes steht der Pfad mit minimalen Kosten fest. Die im kostengünstigsten Fall zu kaufenden Bänder sind dann die Knoten dieses Pfades und können als Ergebnis ausgegeben werden.

4.2 Programm-Dokumentation

Das Format der Eingabedatei muss folgendes Format verwenden.

Beispiel-Eingabedatei klasse.in

```
1 2 4 6 7 8 10 12 14 16 18 20
1: 1 9 12 15 18
2: 2 3 5 14 15
```

[11] Gemeint ist hier nicht nur die Anzahl der Knoten in der untersten Zeile der Bäume, sondern die Summe der Knoten über alle Zeilen, die $\sum_{i=1}^{N} \prod_{j=1}^{i} \omega_j$ beträgt, wenn ω_i die Anzahl der Bände ist, in denen die Aufgabe auf Ebene i des Baumes enthalten ist. Sind die ω_i mit kleineren i kleiner als ω_i mit größeren i, so ist das Produkt über das summiert wird kleiner als es bei unsortierten ω_i ist.

```
3: 4 7 13 16 18
4: 5 11 16 19 20
5: 6 8 11 14 15
6: 1 3 6 17 20
7: 2 8 10 13 19
8: 4 9 10 12 16
```

In der ersten Zeile werden also die bereits vorhandenen Bücher eingetragen. In den darauf folgenden Zeilen werden die in einem Band enthaltenen Aufgaben angegeben, z.B. sind in Band 6 die Aufgaben 1, 3, 6, 17 und 20.

Um zu markieren, dass die bereits gesammelten Aufgaben in der ersten Zeile nicht mehr gekauft werden müssen, werden diese schon vorhandenen Aufgaben i mit dem Status `aufgabeStatus[i]` = 1 versehen.

Anschließend wird für jede Aufgabe eine verketette Liste mit den Bänden angelegt, in denen die jeweilige Aufgabe enthalten ist. Wie in Aufgabe 1 wird hierzu ein Zeiger-Array `anfang[aufgabenMAX]` angelegt, das auf die Köpfe der verketteten Listen zeigt. Zunächst zeigen alle Köpfe auf sich selbst. Nach Verarbeiten der 1. Zeile zeigen `anfang[1]`, `anfang[9]`, `anfang[12]`, `anfang[15]` und `anfang[18]` auf Band 1. Ein Band wird ähnlich zu Aufgabe 1 per struct implementiert

```
struct band
{ int v; struct band *next; };
```

In v wird die Bandnummer gespeichert und `*next` zeigt auf den folgenden Band innerhalb der verketteten Liste. Kommt nun für eine Aufgabe a ein weiterer Band b hinzu, der a enthält, so wird ein neuer `band struct t` mit `t->v=b` erzeugt und dann an den Anfang der verketteten Liste hinter `anfang[a]` gesetzt, d.h. `t->next=anfang[a]` und `anfang[i]=t`. Für Aufgabe 15 aus dem Beispiel oben verändert sich die verkettete Liste also wie folgt:

nach Zeile 1: `anfang[15]` $\rightarrow 1$
nach Zeile 2: `anfang[15]` $\rightarrow 2 \rightarrow 1$

Bei jedem Hinzufügen eines neuen Bandes zu einer verketteten Liste für Aufgabe a wird das a-te Element des Arrays `int laenge[aufgabenMAX]` inkrementiert, um anschließend die verketteten Listen nach ihrer Länge aufsteigend zu sortieren. Dies erfolgt mit Selection Sort:

Selection Sort der verketteten Listen

```
for (i = 0; i < AufgabenMAX; i++)
    if (laenge[i] == 0) laenge[i] = 1000;
for (i = 0; i < AufgabenMAX; i++) {
    min = i;
    for (j = i+1; j < AufgabenMAX; j++)
        if (laenge[indexAufg[j]] < laenge[indexAufg[min]])
            min = j;
    x=indexAufg[i]; indexAufg[i]=indexAufg[min]; indexAufg[min]=x
        ;
```

```
}
```

Man beachte, dass verkettete Listen mit der Länge 0 ans Ende gesetzt werden, indem ihnen vor dem Sortiervorgang die Länge 1000 zugewiesen wird. Das Sortieren selbst erfolgt indirekt über ein `indexAufg` Array, dass die Indizes der verketteten Listen speichert. In unserem Beispiel ist `indexAufg[0]=17`, weil die Liste von Aufgabe 17 nur einen Band enthält. `indexAufg[1]` ist dann 3, `indexAufg[2]` 5 usw. (vgl. Tabelle auf S. 32).

Jetzt können die Bäume erzeugt werden. Zunächst benötigen wir dazu einige Stämme `roots`:

Erzeugen der Stämme `roots`

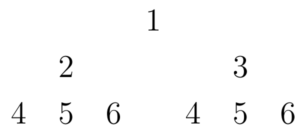
```
for (t = anfang[indexAufg[0]], x = 0; t != z; t = t->next, x++) {
    roots[x] = (struct baumBand *) malloc(sizeof *roots[x]);
    roots[x]->v = t->v;
    erzeugeBaumBand(roots[x], 1, 1);
}
```

Die `for` Schleife durchläuft die verkettete Liste von Aufgabe `indexAufg[0]`, d.h. die Liste mit der geringsten Länge, vom Kopf `anfang[indexAufg[0]]` bis zum Ende `z`. Die Stämme selbst sind vom Typ

```
struct baumBand
{ int v; struct baumBand *l; struct baumBand *r; };
struct baumBand *roots[BuecherMAX];
```

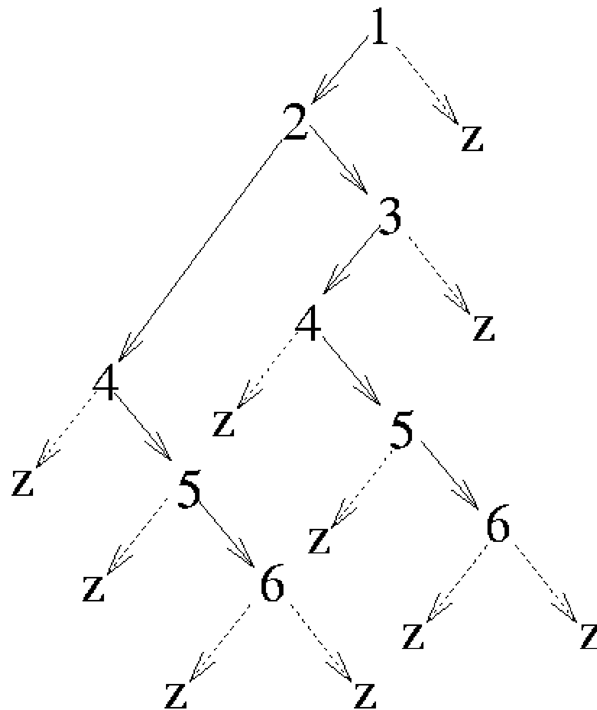
In `v` wird wieder die Bandnummer gespeichert, die Zeiger `*l` und `*r` werden für die Baumstruktur benötigt.

Die im vorigen Abschnitt gezeigten Bäume werden als binäre Bäume *bB* implementiert, d.h. jeder bB-Knoten bzw. jeder Band innerhalb eines b-Baums hat genau einen linken und genau einen rechten bB-Nachfolger. Der linke bB-Nachfolger eines bB-Knotens soll immer auf ein Kind dieses Knotens zeigen, ein rechter bB-Nachfolger auf einen Bruder. Zur Veranschaulichung dieser Implementierung soll die Struktur des binären Baums an Hand eines Beispiels betrachtet werden. Der umzuwandelnde Baum sei



Dann kann dieser Baum als binärer Baum bB wie folgt dargestellt werden (Abb. 4.2).

Wie oben beschrieben, stellen die linken bB-Nachfolger Kinder dar, während die rechten bB-Nachfolger für Brüder stehen. Die `z` markieren die Enden des Baums. Um nun von einem Knoten k ausgehend alle direkten Nachfolger im Sinne des ursprünglichen Baums zu durchlaufen, müssen der linke bB-Nachfolger `k->l` und alle rechten bB-Nachfolger von `k->l` besucht werden, d.h. `k->l->r, k->l->r->r, k->l->r->r->r` usw. bis man auf ein `z2`, also ein Ende des Baums, stößt. In unserem Beispiel wären alle Nachfolger von `k=3` im Sinne des ursprünglichen Baum die Knoten `k->l=4, k->l->r=5` und `k->l->r->r=6`. Diese Feststellung wird bei der rekursiven Traversierung der Bäume benötigt werden.



Doch vorerst zur Konstruktion der Bäume selbst. Wie im Codestück zur Erzeugung der Stämme zu sehen ist wird die Funktion `erzeugeBaumBand` zunächst für jeden Stamm-Knoten aufgerufen. Das Argument 1 gibt dabei die Generation des hinzuzufügenden Knotens an, die 0 sagt aus, dass vom hinzuzufügenden Knoten aus keine rechten Brüder erzeugt werden sollen. Die `erzeugeBaumBand` sieht implementiert dann wie folgt:

Funktion `erzeugeBaumBand` zum Bäumchenbauen

```
int erzeugeBaumBand (struct baumBand *curNode, int generation,
    int erzeugeRechteBrueder) {
    if (erzeugeRechteBrueder == 1) {
        // Rechte Brueder
        t3 = curNode;
        for (t = anfang[indexAufg[generation]]->next; t != z; t =
            t->next) {
            t4 = (struct baumBand *) malloc(sizeof t4);
            t4->v = t->v;
            t3->r = t4;
            printf("%i hat als rechten bruder %i (generation %i)\
                n",t3->v,t3->r->v, generation);
            erzeugeBaumBand(t4, generation, 0);
            t3 = t4;
        }
        t3->r = z2; t4->r = z2;
    }
    //Linkestes Kind
```

```
        if (laenge[indexAufg[generation+1]] < 1000) {
            t4 = (struct baumBand *) malloc(sizeof t4);
            t4->v = anfang[indexAufg[generation+1]]->v;
            curNode->l = t4;
20          printf("%i hat als linkestes Kind %i (generation %i)\n",
               curNode->v,curNode->l->v, generation);
            erzeugeBaumBand(curNode->l, generation+1, 1);
        }
        else curNode->l = z2;
    }
```

Die Argumente der Funktion sind der zu erzeugende Knoten `curNode`, die Generation von `curNode` und der Wert `erzeugeRechteBrueder`, der angibt, ob die rechten Brüder von `curNode` erzeugt werden sollen. Die Funktion selbst arbeitet dann wie folgt. Wenn die rechten Brüder erzeugt werden sollen, wird zunächst `curNode` in der temporären Variable `t3` gespeichert. Wenn `curNode` Brüder hat, d.h. wenn die verkettete Liste der Aufgabengeneration von `curNode`[12] noch weitere Bände nach `curNode` selbst enthält, werden alle diese Bände mittels einer `for` Schleife durchlaufen. Für jeden Band wird ein neuer Zeiger `t4` auf ein `baumBand struct` erzeugt mit der Bandnummer `v` des gerade durchlaufenen Bandes. Da `t4` rechter bB-Nachfolger von `t3` ist, wird `t3->r=t4` gesetzt. Danach wird `erzeugeBaumBand` für Band `t4` aufgerufen. Die Generation ist dabei die gleiche wie die Generation von `curNode` (rechte Nachfolger sind Brüder, keine Kinder). Zuletzt wird `t3=t4` gesetzt und in der verketteten Liste der nächste Bruder vom ursprünglichen `curNode` ausgewählt und durchlaufen. Da alle rechten Brüder bereits von der `for` Schleife aufgerufen werden, werden die `erzeugeBaumBand` Aufrufe innerhalb der `for` Schleife mit einer 0 als drittes Argument ausgestattet, d.h. diese Aufrufe von `erzeugeBaumBand` werden die rechten Brüder nicht noch einmal erzeugen.

Der zweite Teil der `erzeugeBaumBand` Funktion sorgt für das Erzeugen linker bB-Nachfolger, also Kinder. Wenn `curNode` ein Kind hat, d.h. wenn die Länge der verketteten Liste der Nachfolger-Generation von `curNode` nicht 1000[13] beträgt, wird ein neuer Zeiger `t4` erzeugt mit `t4->v=anfang[indexAufg[generation+1]]->v`, d.h. mit der Buchnummer des Kopfes der Kinderliste. `curNode` hat dann als linken bB-Nachfolger `t4`, `curNode->l = t4`. Anschließend kann `erzeugeBaumBand` für `curNode->l` aufgerufen werden. Die Generation wird hierbei um 1 inkrementiert und das Erzeugen von rechten Brüdern wird zugelassen. Falls `curNode` kein Kind hatte, wird statt der Erzeugung eines neuen Knotens `curNode->l=z2` gesetzt, um das Ende des bB-Baumes an dieser Stelle zu markieren.

Nun kann der erzeugte Baum mit der rekursiven Funktion `visit` wie in der Lösungsidee beschrieben traversiert werden. Zunächst wird `visit` aus dem Hauptprogramm für die Stämme aller Bäume aufgerufen:

Aufrufen von `visit` für alle Stämme

```
min = 10000;
for (i = 0; i < x; i++) {
```

[12]Gemeint ist die verkettete Liste mit dem Kopf `anfang[indexAufg[generation]]`.

[13]Eine Länge von 1000 markiert wie oben beschrieben, dass die zugehörige Liste leer ist.

```
    for (j = 0; j < BuecherMAX; j++)
        bandStatus[j] = 0;
5   visit(roots[i]);
}
```

`min` speichert hierbei die minimalen Pfadkosten, das Array `bandStatus` gibt den Besucht-Status eines Bandes an, d.h. `bandStatus[b]=n`, wenn Band b n Mal auf einem Pfad besucht wurde. Die Implementation der `visit` Funktion setzt die Beschreibung der Lösungsidee direkt um:

Rekursive Traversierungsfunktion `visit`

```
void visit (struct baumBand *tn) {
    // Inkrementiere Bandstatus von tn
    bandStatus[tn->v]++;
    // Wenn tn Nachfolger hat
5   if (tn->l != z2) {
        // rufe visit fuer alle Nachfolger von tn auf
        visit(tn->l);
        for (t2 = tn->l->r; t2 != z2; t2 = t2->r) visit(t2);
    }
10  // wenn tn keine Nachfolger hat
    else {
        // Kosten fuer gefundenen Pfad
        for (i = 0, x = 0; i < BuecherMAX; i++)
            if (bandStatus[i] > 0) x++;
15      // Falls dieser Pfad ein Kostenminimum ist
        if (x < min) {
            // speichere minimale Kosten
            min = x;
            // und kopiere Bandstatus-Array
20          for (i = 0; i < BuecherMAX; i++)
                minBandStatus[i] = bandStatus[i];
        }
    }
    // Dekrementiere Bandstatus von tn
25  bandStatus[tn->v]--;
}
```

Zuerst wird also der `bandStatus` des zu besuchenden Bandes `tn` um 1 inkrementiert. Wenn `tn` Nachfolger hat, d.h. wenn `tn->l!=z2` gilt, wird `visit` für alle Nachfolger von `tn` aufgerufen. Die Nachfolger sind zunächst `tn->l` als linkes Kind von `tn` und dann alle rechten Brüder von `tn->l`, d.h. `tn->l->r, tn->l->r->r` usw. bis man auf `z2` stößt (vgl. oben). Wenn `tn` keine Nachfolger hatte, befindet man sich am Ende eines Pfades und kann die Kosten dieses Pfades berechnen. Dazu werden alle Bandstatus-Werte durchlaufen und der Wert der Kosten x um 1 inkrementiert, wenn ein Bandstatus-Wert größer als 0 ist, d.h. wenn ein Band mind. ein Mal in dem Pfad vorkommt. Wenn dieser Pfad ein Kostenminimum ist, d.h. wenn `x<min` gilt, werden die minimalen Kosten

in `min` gespeichert und die Bandstatus-Werte in ein `minBandStatus` Array kopiert. Bevor die Funktion beendet werden kann muss am Ende der `bandStatus` von `tn` wieder um 1 dekrementiert werden.

Nach dem Traversieren aller möglichen Pfade durch die gebauten Bäume befinden sich in `min` der Wert der minimalen Kosten und im `minBandStatus` Array steht geschrieben, wie viele Aufgaben im optimalen Fall aus jedem Band verwendet werden. All die Bände, aus denen mindestens eine Aufgabe verwendet wird, müssen gekauft werden. Dieses Ergebnis lässt sich nun einfach ausgeben.

4.3 Programm-Ablaufprotokoll

klasse1.in

Zunächst soll das in der Aufgabenstellung abgedruckte Beispiel betrachtet werden. Die Eingabedatei sieht wie folgt aus:

klasse1.in

```
1 2 4 6 7 8 10 12 14 16 18 20
1: 1 9 12 15 18
2: 2 3 5 14 15
3: 4 7 13 16 18
4: 5 11 16 19 20
5: 6 8 11 14 15
6: 1 3 6 17 20
7: 2 8 10 13 19
8: 4 9 10 12 16
```

Die Ausgabe lautet in Kurform[14]:

```
$$ klasse klasse1.in
Lese Input ein...
Band 1 in Adjazenzliste von Aufgabe 9
Band 1 in Adjazenzliste von Aufgabe 15
Band 2 in Adjazenzliste von Aufgabe 3
Band 2 in Adjazenzliste von Aufgabe 5
Band 2 in Adjazenzliste von Aufgabe 15
Band 3 in Adjazenzliste von Aufgabe 13
Band 4 in Adjazenzliste von Aufgabe 5
Band 4 in Adjazenzliste von Aufgabe 11
Band 4 in Adjazenzliste von Aufgabe 19
Band 5 in Adjazenzliste von Aufgabe 11
Band 5 in Adjazenzliste von Aufgabe 15
Band 6 in Adjazenzliste von Aufgabe 3
Band 6 in Adjazenzliste von Aufgabe 17
Band 7 in Adjazenzliste von Aufgabe 13
Band 7 in Adjazenzliste von Aufgabe 19
Band 8 in Adjazenzliste von Aufgabe 9

Sortiere Listen nach ihrer Maechtigkeit
```

[14] Eine sehr viel detailliertere Ausgabe erhält man mit dem Argument `-v` beim Aufruf des Programms.

```
   Index 0: Aufgabe 17 (Laenge 1)
   Index 1: Aufgabe 3 (Laenge 2)
   Index 2: Aufgabe 5 (Laenge 2)
   Index 3: Aufgabe 9 (Laenge 2)
25 Index 4: Aufgabe 11 (Laenge 2)
   Index 5: Aufgabe 13 (Laenge 2)
   Index 6: Aufgabe 19 (Laenge 2)
   Index 7: Aufgabe 15 (Laenge 3)

30 Baue Baeume
   .................................................................
   .................................................................
   .................................................................
   .................................................................
35 ...........................................................[ok]

   Traversiere Baeume
   ........
   NEUES MINIMUM: 5 Buecher: 4 5 6 7 8
40 ............................................................
   NEUES MINIMUM: 4 Buecher: 1 4 6 7
   .................................................................
   .................................................................
   .................................................................
45 ........................................................[ok]

   LETZTES MINIMUM: 4 Buecher (40 Euro)
   Zu kaufende Buecher: 1 4 6 7
```

klasse2.in

Ein zweites Beispiel:

klasse2.in

```
   1 2 3 4 5
   1: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
   2: 2 4 6 8 10 12 14
   3: 3 6 9 12 15
5  4: 4 8 12 16
   5: 5 10 15
   6: 1 6 12
   7: 2 7 14
   8: 1 8 16
10 9: 9
   10: 5 10
```

Die Ausgabe lautet:

```
$$ klasse klasse2.in
Lese Input ein...
Band 1 in Adjazenzliste von Aufgabe 6
```

```
    Band 1 in Adjazenzliste von Aufgabe 7
5   Band 1 in Adjazenzliste von Aufgabe 8
    Band 1 in Adjazenzliste von Aufgabe 9
    Band 1 in Adjazenzliste von Aufgabe 10
    Band 1 in Adjazenzliste von Aufgabe 11
    Band 1 in Adjazenzliste von Aufgabe 12
10  Band 1 in Adjazenzliste von Aufgabe 13
    Band 1 in Adjazenzliste von Aufgabe 14
    Band 1 in Adjazenzliste von Aufgabe 15
    Band 2 in Adjazenzliste von Aufgabe 6
    Band 2 in Adjazenzliste von Aufgabe 8
15  Band 2 in Adjazenzliste von Aufgabe 10
    Band 2 in Adjazenzliste von Aufgabe 12
    Band 2 in Adjazenzliste von Aufgabe 14
    Band 3 in Adjazenzliste von Aufgabe 6
    Band 3 in Adjazenzliste von Aufgabe 9
20  Band 3 in Adjazenzliste von Aufgabe 12
    Band 3 in Adjazenzliste von Aufgabe 15
    Band 4 in Adjazenzliste von Aufgabe 8
    Band 4 in Adjazenzliste von Aufgabe 12
    Band 4 in Adjazenzliste von Aufgabe 16
25  Band 5 in Adjazenzliste von Aufgabe 10
    Band 5 in Adjazenzliste von Aufgabe 15
    Band 6 in Adjazenzliste von Aufgabe 6
    Band 6 in Adjazenzliste von Aufgabe 12
    Band 7 in Adjazenzliste von Aufgabe 7
30  Band 7 in Adjazenzliste von Aufgabe 14
    Band 8 in Adjazenzliste von Aufgabe 8
    Band 8 in Adjazenzliste von Aufgabe 16
    Band 9 in Adjazenzliste von Aufgabe 9
    Band 10 in Adjazenzliste von Aufgabe 10
35
    Sortiere Listen nach ihrer Maechtigkeit
    Index 0: Aufgabe 11 (Laenge 1)
    Index 1: Aufgabe 13 (Laenge 1)
    Index 2: Aufgabe 7 (Laenge 2)
40  Index 3: Aufgabe 16 (Laenge 2)
    Index 4: Aufgabe 9 (Laenge 3)
    Index 5: Aufgabe 14 (Laenge 3)
    Index 6: Aufgabe 15 (Laenge 3)
    Index 7: Aufgabe 8 (Laenge 4)
45  Index 8: Aufgabe 10 (Laenge 4)
    Index 9: Aufgabe 6 (Laenge 4)
    Index 10: Aufgabe 12 (Laenge 5)

    Baue Baeume
50  ..................................................................
    ..................................................................
    ..................................................................
    ..................................................................
    ..................................................................
55  ..................................................................
    ..................................................................
    ..................................................................
```

```
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
....................................[ok]

Traversiere Baeume
...........
NEUES MINIMUM: 7 Buecher: 1 5 6 7 8 9 10
.............
NEUES MINIMUM: 6 Buecher: 1 5 6 7 8 9
................................................................
................................................................
................................................................
...................................
NEUES MINIMUM: 5 Buecher: 1 3 5 7 8
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
................................................................
```

```
........................................[ ok ]

LETZTES MINIMUM: 5 Buecher (50 Euro)
Zu kaufende Buecher: 1 3 5 7 8
```

klasse3.in

Und noch ein letztes Beispiel:

klasse3.in

```
9 10
1: 1 3 5 7 9
2: 2 3
3: 4 6 8 10
4: 7 9 10
5: 3 4 5
6: 6 4 5
7: 3 5 10
```

Die Ausgabe lautet:

```
$$ klasse klasse3.in
Lese Input ein...
Band 1 in Adjazenzliste von Aufgabe 1
Band 1 in Adjazenzliste von Aufgabe 3
Band 1 in Adjazenzliste von Aufgabe 5
Band 1 in Adjazenzliste von Aufgabe 7
Band 2 in Adjazenzliste von Aufgabe 2
Band 2 in Adjazenzliste von Aufgabe 3
Band 3 in Adjazenzliste von Aufgabe 4
Band 3 in Adjazenzliste von Aufgabe 6
Band 3 in Adjazenzliste von Aufgabe 8
Band 4 in Adjazenzliste von Aufgabe 7
Band 5 in Adjazenzliste von Aufgabe 3
Band 5 in Adjazenzliste von Aufgabe 4
Band 5 in Adjazenzliste von Aufgabe 5
Band 6 in Adjazenzliste von Aufgabe 6
Band 6 in Adjazenzliste von Aufgabe 4
Band 6 in Adjazenzliste von Aufgabe 5
Band 7 in Adjazenzliste von Aufgabe 3
Band 7 in Adjazenzliste von Aufgabe 5

Sortiere Listen nach ihrer Maechtigkeit
Index 0: Aufgabe 1 (Laenge 1)
Index 1: Aufgabe 2 (Laenge 1)
Index 2: Aufgabe 8 (Laenge 1)
Index 3: Aufgabe 6 (Laenge 2)
Index 4: Aufgabe 7 (Laenge 2)
Index 5: Aufgabe 4 (Laenge 3)
Index 6: Aufgabe 3 (Laenge 4)
Index 7: Aufgabe 5 (Laenge 4)
```

```
Baue Baeume
..............................................................
...............................[ok]

Traversiere Baeume
........
NEUES MINIMUM: 6 Buecher: 1 2 3 4 6 7
......................
NEUES MINIMUM: 5 Buecher: 1 2 3 6 7
..............................................................
NEUES MINIMUM: 4 Buecher: 1 2 3 5
.[ok]

LETZTES MINIMUM: 4 Buecher (40 Euro)
Zu kaufende Buecher: 1 2 3 5
```

4.4 Quellcode

```
#include <stdio.h>
#include <string.h>

// maximale Anzahlen an Buechern und Aufgaben
#define BuecherMAX 30
#define AufgabenMAX 30

/* Variablen */
FILE *in;
// Counter
int i,j,k,l,b1;
int x,y,min;
int verbose = 0;
// Zeichenketten
char c, zeile[50];
// Arrays fuer den Algorithmus
int aufgabeStatus[AufgabenMAX], laenge[AufgabenMAX], indexAufg[
    AufgabenMAX];
int bandStatus[BuecherMAX], minBandStatus[BuecherMAX];

/* Structs */
struct band
{ int v; struct band *next; };
struct band *t, *z;
struct band *anfang[AufgabenMAX];
struct baumBand
{ int v; struct baumBand *l; struct baumBand *r; };
struct baumBand *t2, *z2, *t3, *t4;
struct baumBand *roots[BuecherMAX];

/* Funktion zum erzeugen des Baums */
int erzeugeBaumBand (struct baumBand *curNode, int generation, int
    erzeugeRechteBrueder) {
    if (!verbose) printf(".");
```

```
    if (erzeugeRechteBrueder == 1) {
        // Rechte Brueder
35      t3 = curNode;
        for (t = anfang[indexAufg[generation]]->next; t != z; t = t->
            next) {
            t4 = (struct baumBand *) malloc(sizeof t4);
            t4->v = t->v;
            t3->r = t4;
40          if (verbose) printf("%i hat als rechten bruder %i (
                generation %i)\n",t3->v,t3->r->v, generation);
            erzeugeBaumBand(t4, generation, 0);
            t3 = t4;
        }
        t3->r = z2; t4->r = z2;
45  }
    //Linkestes Kind
    if (laenge[indexAufg[generation+1]] < 1000) {
        t4 = (struct baumBand *) malloc(sizeof t4);
        t4->v = anfang[indexAufg[generation+1]]->v;
50      curNode->l = t4;
        if (verbose) printf("%i hat als linkestes Kind %i (Generation %
            i)\n",curNode->v,curNode->l->v, generation);
        erzeugeBaumBand(curNode->l, generation+1, 1);
    }
    else curNode->l = z2;
55 }

/* Rekursive Traversierungsprozedur des Baumes */
void visit (struct baumBand *tn) {
    if (!verbose) printf(".");
60  if (verbose) printf("%i  Start visit(%i) ==> buchstat[%i]=%i\n",tn
        ->v,tn->v,tn->v, bandStatus[tn->v]+1);
    // Inkrementiere Bandstatus von tn
    bandStatus[tn->v]++;
    // Wenn tn Nachfolger hat
    if (tn->l != z2) {
65      // rufe visit fuer alle Nachfolger von tn auf
        if (verbose) printf("%i  Start linkes Kind exisitiert, besuche
            %i\n",tn->v,tn->l->v);
        visit(tn->l);
        for (t2 = tn->l->r; t2 != z2; t2 = t2->r) visit(t2);
    }
70  // wenn tn keine Nachfolger hat
    else {
        //  berechne Kosten fuer den gefunden Pfad
        if (verbose) printf("%i  kein linkes Kind, pruefe ob Minimum \n
            ",tn->v);
        x = 0;
75      for (i = 0; i < BuecherMAX; i++)
            if (bandStatus[i] > 0) x++;
        if (verbose) printf("NEUER PFAD %i: %i  Buecher: ",b1++,x);
        if (verbose)
            for (i = 0; i < BuecherMAX; i++)
80              if (bandStatus[i] > 0) printf("%i ",i);
```

```
            if (verbose) printf("\n");
            // Falls dieser Pfad einem Kostenminimum entspricht, speichere
                minimale
            // Kosten und kopiere Bandstatus-Array
            if (x < min) {
85              min = x;
                printf("\nNEUES MINIMUM: %i Buecher: ",min);
                for (i = 0; i < BuecherMAX; i++) {
                    minBandStatus[i] = bandStatus[i];
                    if (minBandStatus[i] > 0) printf("%i ",i);
90              }
                printf("\n");
            }
        }
        // Dekrementiere Bandstatus von tn
95      bandStatus[tn->v]--;
        if (verbose) printf("%i  Ende visit(%i) ==> buchstat[%i]=%i\n",tn->
            v,tn->v,tn->v,bandStatus[tn->v]);
    }

    /* Hauptfunktion main */
100 int main (int argc, char **argv) {
        // Pruefe auf Argument
        if (!(argc==2 || argc==3)) {
            printf("Keine oder zu viele Parameter angegeben.\nAufruf:
                klasse [-v] input-file.in.\n");
            exit(0);
105     }
        // Dateien
        if (argc == 3) {
            if (strcmp(argv[1], "-v")==0) verbose = 1;
            else { printf("Aufruf: klasse [-v] input-file.in.\n"); exit(0);
                }
110     }
        in = fopen(argv[argc-1], "r");

        // Initalisiere Adjazenzlisten mit auf sich selbst zeigenden
            Zeigern z
        z = (struct band *) malloc(sizeof *z);
115     z->next = z;
        z2 = (struct baumBand *) malloc(sizeof *z2);
        z2->l = z2; z2->r = z2;
        // Initialisiere einige Arrays
        for (i = 0; i < AufgabenMAX; i++) {
120         anfang[i] = z; // Adjazenzlistenkopf
            laenge[i] = 0;
            indexAufg[i] = i;
            aufgabeStatus[i] = 0;
        }
125     // Initialisiere bandStatus Array
        for (i = 0; i < BuecherMAX; i++) {
            bandStatus[i] = 0;
        }
```

```
130     /* Input */
        // Bereits vorhandene Aufgaben
        // Zeilenformat wie "1, 2, 4, 5, 6, ...\n"
        printf("Lese Input ein...\n");
        fgets(zeile, 50, in);
135     for (i = -2; zeile[++i] != '\n' && zeile[i+1] != '\n'; ) {
            sscanf(zeile, "%i", &x);
            // Status von Aufgabe x ist 1
            aufgabeStatus[x] = 1;
            // Ueberschreibe x im zeile String mit Leerzeichen
140         zeile[++i] = ' ';
            if (x > 9) zeile[++i] = ' ';
        }

        // Band b enthaelt Aufgaben a_i
145     // b: a_1, a_2, a_3, ...\n
        while (fgets(zeile, 50, in) != NULL) {
            sscanf(zeile, "%i", &x); // Band x
            zeile[0] = zeile[1] = zeile[2] = ' ';
            for (i = ((x < 10) ? 1 : 2); zeile[++i] != '\n' && zeile[i+1]
                != '\n'; ) {
150             sscanf(zeile, "%i", &y); // Aufgabe y
                zeile[++i] = ' ';
                if (y > 9) zeile[++i] = ' ';
                // wenn Aufgabe noch nicht vorhanden
                if (aufgabeStatus[y] == 0) {
155                 // Speichere Band x in Adjazenzliste von Aufgabe y
                    printf("Band %i in Adjazenzliste von Aufgabe %i\n",x,y)
                        ;
                    t = (struct band *) malloc(sizeof *t);
                    t->v = x; t->next = anfang[y]; anfang[y] = t;
                    laenge[y]++;
160             }
            }
        }

        // Setze 0-Laengen auf 1000 fuer Sortieren
165     for (i = 0; i < AufgabenMAX; i++)
            if (laenge[i] == 0) laenge[i] = 1000;
        // Sortiere Adjazenzlisten nach aufsteigender Laenge, leere
           Adjazenzlisten
        // ans Ende (selection sort)
        for (i = 0; i < AufgabenMAX; i++) {
170         min = i;
            for (j = i+1; j < AufgabenMAX; j++)
                if (laenge[indexAufg[j]] < laenge[indexAufg[min]])
                    min = j;
            x=indexAufg[i]; indexAufg[i]=indexAufg[min]; indexAufg[min]=x;
175     }

        printf("\nSortiere Listen nach ihrer Maechtigkeit\n");
        for (i = 0; i < AufgabenMAX; i++)
            if (laenge[indexAufg[i]] < 1000)
```

```
180             printf("Index %i: Aufgabe %i (Laenge %i)\n",i,indexAufg[i],
                    laenge[indexAufg[i]]);

        /* Baue Baeume */
        printf("\nBaue Baeume\n");
        x = 0;
185     // Erzeuge Staemme roots
        for (t = anfang[indexAufg[0]], x = 0; t != z; t = t->next, x++) {
            roots[x] = (struct baumBand *) malloc(sizeof *roots[x]);
            roots[x]->v = t->v;
            erzeugeBaumBand(roots[x], 0, 0);
190         x++;
        }
        if (!verbose) printf("[ok]");

        /* Traversiere Baeume */
195     printf("\n\nTraversiere Baeume\n");
        b1=0;
        min = 10000;
        for (i = 0; i < x; i++) {
            for (j = 0; j < BuecherMAX; j++)
200             bandStatus[j] = 0;
            //bandStatus[roots[i]->v]++;
            visit(roots[i]);
        }
        if (!verbose) printf("[ok]");
205
        printf("\n\nLETZTES MINIMUM: %i Buecher (%i Euro)\nZu kaufende
            Buecher: ",min,min*10);
        for (i = 0; i < BuecherMAX; i++)
            if (minBandStatus[i] > 0) printf("%i ",i,minBandStatus[i]);

210     printf("\n");
        free(z,z2,t,t4,roots);
        fclose(in); exit(0);
    }
```

5 Zappen und zählen

5.1 Lösungsidee 1

Für Teilaufgabe 1 bedarf es im Prinzip keiner Lösungsidee, da in der Programm-Dokumentation beschrieben wird, wie man mit dem Paket `XGraphics` direkt Rechtecke und Raster in ein Fenster zeichnen kann.

Als Beispieldatensätze wurden folgende Rechtecke gewählt:

$$\begin{aligned} R3 &= \{((2;8),(7;13)),((5;3),(10,11)),((6;5),(14;15))\} \\ R4 &= \{((7;2),(28;18)),((10;1),(19,15)),((6;10),(17;13)),((8;4),(15;9))\} \\ R5 &= \{((2;8),(30;16)),((10;1),(19,18)),((6;10),(17;13)), \\ &\quad ((8;4),(19;9)),((18;5),(20;19))\} \end{aligned}$$

5.2 Programm-Dokumentation 1

Das Format der Eingabedatei ist so gewählt, dass die Koordinaten je eines Rechtecks in je einer Zeile stehen, wobei die vier Koordinaten in der Reihenfolge `x1 y1 x2 y2` angegeben werden müssen, wenn $P(x1|y1)$ der linke untere und $P(x2|y2)$ der rechte obere Punkt des Rechtecks ist. Das Rechteck $R2$ aus der Aufgabenstellung wird also wie folgt eingegeben.

Beispiel einer Eingabedatei zapp2.in für $R2$

```
2 6 13 12
8 3 18 18
5 10 23 16
```

Die Rechtecke werden in dem Programm im `re` Array gespeichert:

```
struct rechteck { int x1,y1, x2,y2; };
struct rechteck *re[rMax];
```

`rMax` steht hier für die maximale Anzahl an eingebbaren Rechtecken. Der linkere untere Punkt des zweiten Rechtecks ist also (`re[1]->x1`|`re[1]->y1`), der rechte obere Punkt (`re[1]->x2`|`re[1]->y2`). Das Einlesen der Input-Datei und Füllen des `re` Arrays erfolgt also schlichtweg mit

```
in = fopen(argv[1], "r");
for (i = 0; !feof(in) && i < rMax; i++) {
    re[i] = (struct rechteck *) malloc(sizeof re);
    fscanf(in, "%i %i %i %i", &re[i]->x1, &re[i]->y1, &re[i]->x2,
        &re[i]->y2);
}
```

Für die Grafikausgabe wird das Paket `XGraphics`[15] von Martin Lüders verwendet. Die Initialisierung der Grafik erfolgt durch folgenden Code

Grafikinitialisierung mit `XGraphics`

```
#include "Xgraphics.h"
...
int main (int argc, char **argv) {
    InitX();
    mywindow = CreateWindow(700,400,"Zapp");
    myworld = CreateWorld(mywindow,0,0,600,400,-1,21,31,-1,0,0);
    InitButtons(mywindow, "t,Kommandos:,,b,Ende,e,b,weiter,w",
        100);
    ShowWindow(mywindow);
    ...
}
```

Es wird also ein Fenster `mywindow` der Größe 700x400 erzeugt. In dieses Fenster wird eine Welt `myworld` mit den Maßen 600x400 gesetzt. Der linken oberen Ecke der Welt entspricht $(-1|21)$, der rechten unteren $(31|-1)$.[16] Dann werden noch zwei Buttons „Ende“ und „weiter“ eingefügt, worauf das Fenster schließlich angezeigt wird.

Am Ende der `main` Funktion wird eine Endlosschleife zur Event-Überwachung eingebaut, die beim Klick auf den Button „Ende“ das Programm beendet und beim Starten oder Verschieben des Fensters die Funktion `redraw` aufruft. Diese ist für das Zeichnen des Rasters und der Rechtecke zuständig:

Raster und Rechtecke zeichnen mit `redraw`

```
void redraw () {
    ClearWorld(myworld);
    for (i = 1; i <= 30; i++)
        WDrawLine(myworld, i,0, i,20, 5);
    for (i = 1; i <= 20; i++)
        WDrawLine(myworld, 0,i, 30,i, 5);
    WDrawLine(myworld, 0,0, 30,0, 1);
    WDrawLine(myworld, 0,0, 0,20, 1);
    WDrawString(myworld, -0.5,-0.5, "0", 1);
    WDrawString(myworld, 29.7,-0.8, "30", 1);
    WDrawString(myworld, -1,19.7, "20", 1);
    for (i = 0; i < numR; i++)
        WDrawRectangle(mywaorld, re[i]->x1, re[i]->y2, re[i]->x2,
            re[i]->y1, 4);
}
```

Zunächst wird der alte Inhalt der Welt mit `ClearWorld` gelöscht. Dann werden die horizontalen und vertikalen Linien des Rasters gezeichnet. Zuerst wird für i von 1 bis 30

[15] GPL, Copyright ©1996 Martin Lüders, e-mail: lueders@physik.uni-wuerzburg.de. Für weitere Informationen siehe...

[16] Für ein schöneres Erscheinungsbild der Ausgabe wird jeweils eine Einheit als Rand hinzugefügt.

eine vertikale Linie von $(i|0)$ nach $(i|20)$ gezogen.[17] Anschließend zeichnet das Programm eine horizontale Linie von $(0|i)$ nach $(30|i)$ für i von 1 bis 20. Die nächsten beiden Linien stellen die Achsen nach rechts und oben dar (Farbe 1=schwarz). Die folgenden drei Strings beschriften die beiden Achsen. Zuletzt werden alle Rechtecke `re[i]` durchlaufen und mit `WDrawRectangle` in der Farbe 4=blau gezeichnet. Zu beachten ist hierbei, dass die an `WDrawRectangle` übergebenenen Koordinaten für den linken oberen und den rechten unteren Punkt des zu zeichnenden Rechtecks stehen:

Xgraphics Funktion zum Zeichnen eines Rechtecks `WDrawRectangle`

```
void WDrawRectangle (World world, double x1, double y1, double x2
    , double y2, int c)
```

Mit $(x1, y1)$ als linker oberer und $(x2, y2)$ als rechter unterer Ecke. Deshalb erfolgt der Zeichenaufruf in dem Programm mit

```
WDrawRectangle(mywaorld, re[i]->x1, re[i]->y2, re[i]->x2, re[i]->
    y1, 4);
```

5.3 Programm-Ablaufprotokoll 1

Beispieldatensatz zapp3.in

```
2 8 7 13
5 3 10 11
6 5 14 15
```

`zapp zapp3.in`:

Beispieldatensatz zapp4.in

```
7 2 28 18
10 1 19 15
6 10 17 13
8 4 15 9
```

`zapp zapp4.in`:

Beispieldatensatz zapp5.in

```
2 8 30 16
10 1 19 18
6 10 17 13
8 4 19 9
18 5 20 19
```

`zapp zapp5.in`:

[17] Das letzte Argument in den Funktionen `WDraw...` steht für die Farbe, hier 5=gelb.

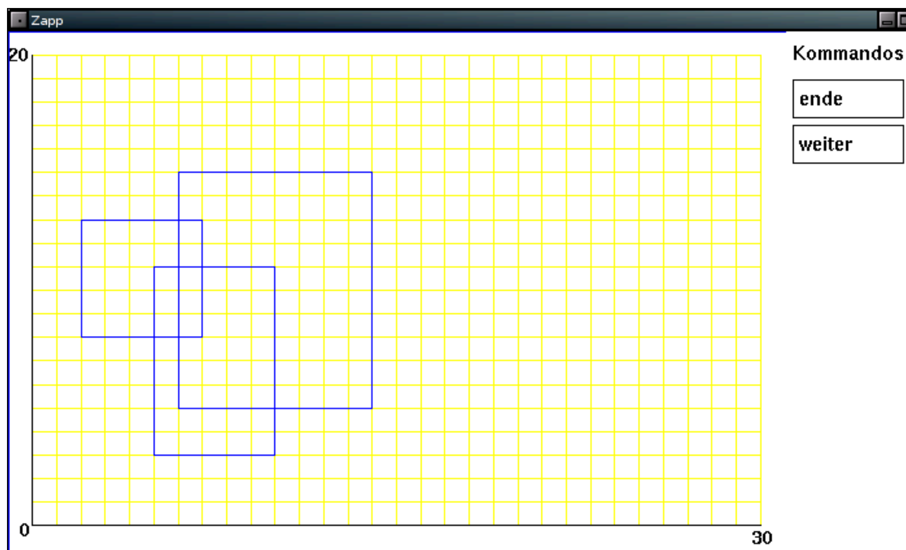


Abbildung 5.1: Beispieldatensatz zapp3.in für drei Rechtecke.

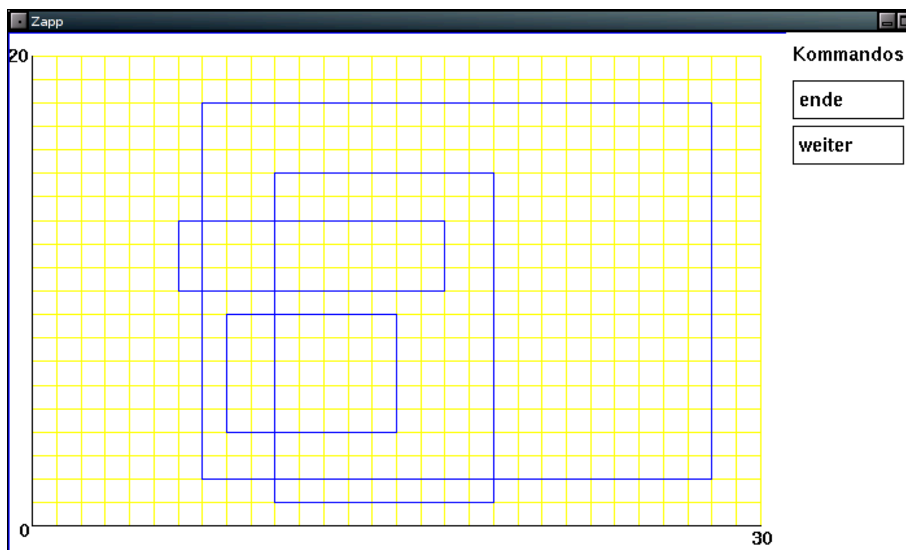


Abbildung 5.2: Beispieldatensatz zapp4.in für vier Rechtecke.

5.4 Quellcode 1

Der Code für Teilaufgabe 1 ist im Code für Teilaufgabe 2 enthalten und wird deshalb im Code-Abschnitt von Teilaufgabe 2 angegeben.

5.5 Lösungsidee 2

Die Lösungsidee der Aufgabe besteht darin, zunächst alle Schnittpunkte der Rechtecke untereinander zu finden und anschließend mit Hilfe dieser Information alle sich ergeben-

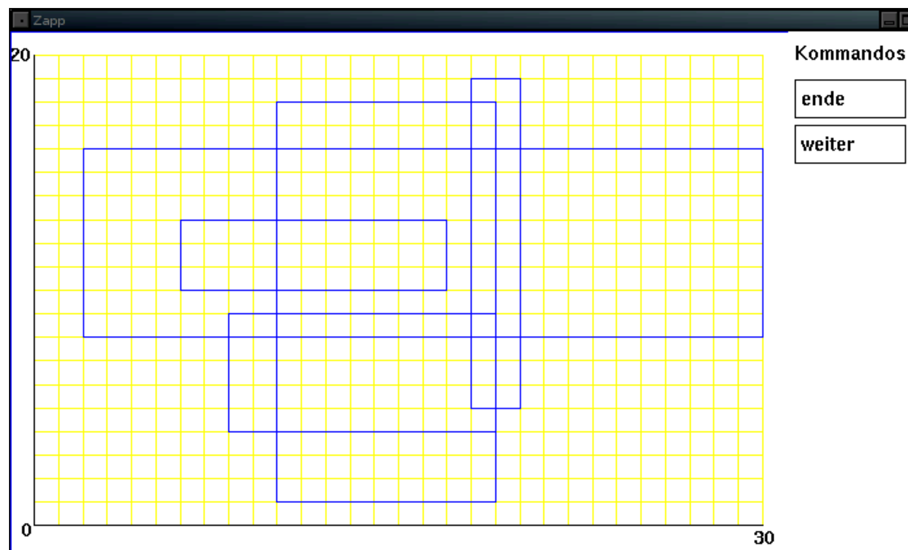


Abbildung 5.3: Beispieldatensatz zapp5.in für fünf Rechtecke.

den Rechtecke zu durchlaufen und auszugeben.

Um alle Schnittpunkte der Rechtecke zu finden, wird die in Abb. 5.4 gezeigte Beschriftung für zwei Rechtecke i und j gewählt. Dabei gilt für die Eckpunkte jeweils

$$A = (x1|y1) \;\; B = (x2|y1) \;\; C = (x2|y2) \;\; D = (x1|y2).$$

Es gibt nun vier unterschiedliche Fälle für mögliche Schnittpunkte von zwei Rechtecken i und j.

A1 *d_j schneidet a_i (linke Kante von j schneidet untere Kante von i), Abb. 5.5.*

Zunächst muss geklärt unter welchen Bedingungen dieser Fall eintritt. Wie in Abb. 5.5 zu erkennen ist lauten diese beiden Bedingungen:

1. x_1 von j ist zwischen x_1 und x_2 von i,
2. y_1 von i ist zwischen y_1 und y_2 von j.

Treffen beide Bedingungen zu, so schneidet d_j a_i im Punkt $(j_{x1}|i_{y1})$.

A2 *d_j schneidet c_i (linke Kante von j schneidet obere Kante von i), Abb. 5.6.*

Die Bedingungen für das Eintreten dieses Falles sind (vgl. Abb. 5.6):

1. x_1 von j ist zwischen x_1 und x_2 von i,

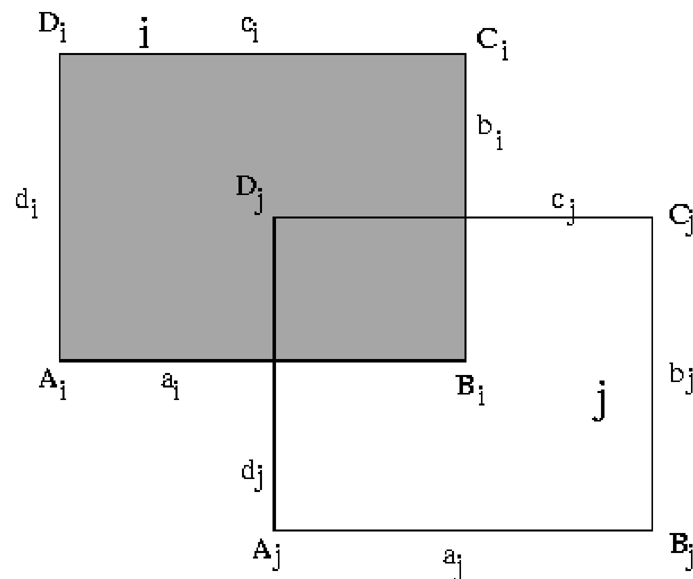


Abbildung 5.4: Allg. zwei Rechtecke

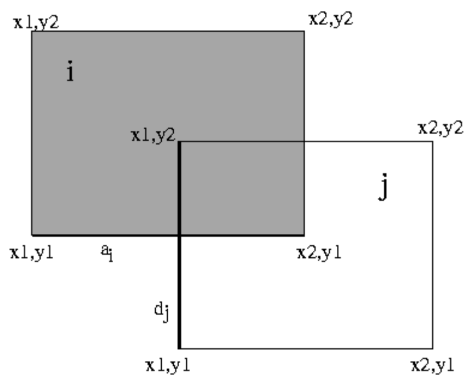


Abbildung 5.5: Schnittpunkt Fall A1, d_j schneidet a_i.

2. y_2 von i ist zwischen y_1 und y_2 von j.

Treffen beide Bedingungen zu, so schneidet d_j c_i im Punkt $(j_{x1}|i_{y2})$.

B1 *b_j schneidet a_i (rechte Kante von j schneidet untere Kante von i), Abb. 5.7.*

Die Bedingungen für das Eintreten dieses Falles sind (vgl. Abb. 5.7):

1. x_2 von j ist zwischen x_1 und x_2 von i,
2. y_1 von i ist zwischen y_1 und y_2 von j.

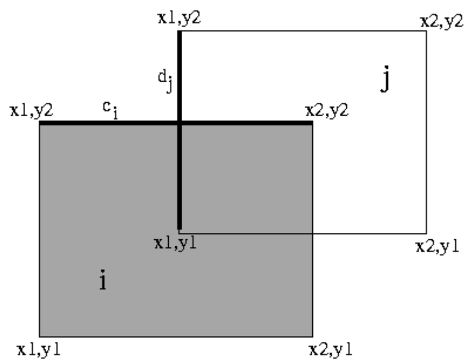


Abbildung 5.6: Schnittpunkt Fall A2, d_j schneidet c_i.

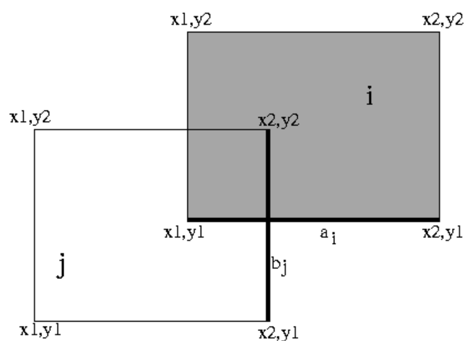


Abbildung 5.7: Schnittpunkt Fall B1, b_j schneidet a_i.

Treffen beide Bedingungen zu, so schneidet b_j a_i im Punkt $(j_{x2}|i_{y1})$.

B2 *b_j schneidet c_i (rechte Kante von j schneidet obere Kante von i), Abb. 5.8.*

Die Bedingungen für das Eintreten dieses Falles sind (vgl. Abb. 5.8):

1. x_2 von j ist zwischen x_1 und x_2 von i,
2. y_2 von i ist zwischen y_1 und y_2 von j.

Treffen beide Bedingungen zu, so schneidet b_j a_i im Punkt $(j_{x2}|i_{y2})$.

Um alle Schnittpunkte zu finden, werden für i und j alle Anfangsrechtecke aus der Eingabedatei durchlaufen, die obigen Bedingungen für jeden Fall überprüft und gegebenfalls die jeweiligen Schnittpunkt gespeichert. Da natürlich neben den Schnittpunkten

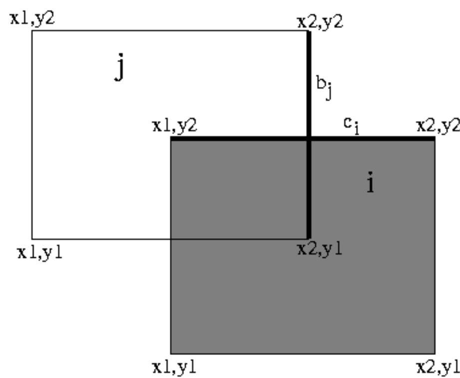
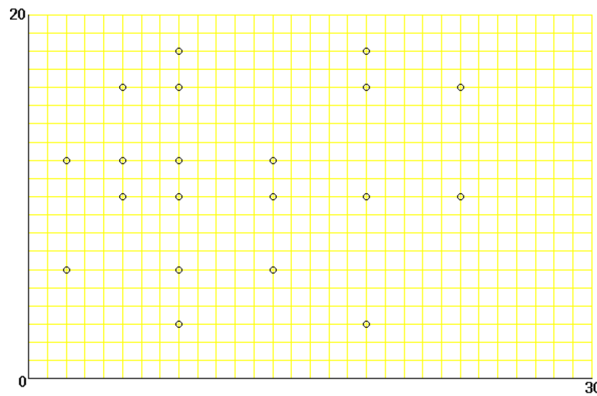


Abbildung 5.8: Schnittpunkt Fall B1, b_j schneidet c_i.

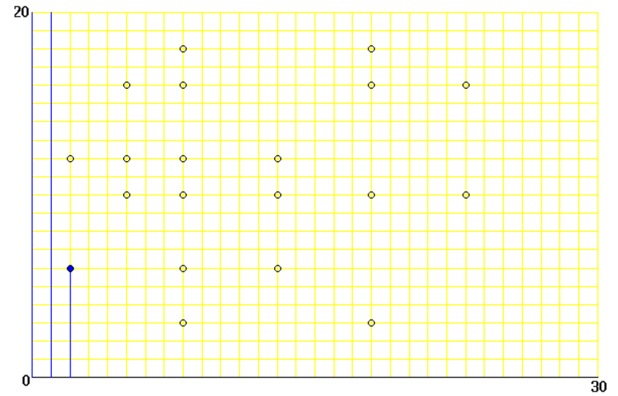
der Anfangsrechtecke untereinander auch die Eckpunkte der Anfangsrechtecke als Eckpunkte für später entstehende Rechtecke dienen können, wird „zwischen“ in den obigen Bedingungen so interpretiert, dass die beiden Grenzen eingeschlossen sind, und beim Durchlaufen der Rechtecke wird auch für den Fall $i = j$ genauso wie für zwei unterschiedliche Rechtecke i und j verfahren. D.h. im Fall $i = j$ treffen alle vier obige Bedingungen zu und die Eckpunkte des Anfangsrechtecks $i = j$ werden ebenfalls als „Schnittpunkte“ gespeichert.

Da wir nun alle potentiellen Eckpunkte für Rechtecke kennen, können wir alle Rechtecke suchen. Hierzu wird das Raster von links nach rechts und von unten nach oben „abegescannt“, genauer: die Linie von $(i|j = 0)$ nach $(i|j = 20)$ wird für aufsteigende $i = 0..30$ nach Schnittpunkten, d.h. potentiellen Rechteckeckpunkten durchsucht. Weil das Raster von unten nach oben und von links nach rechts durchsucht wird, werden die gefundenen Schnittpunkte als linke untere Eckpunkte eines potentiellen Rechtecks angenommen. Von einem gefundenen linken Eckpunkt $(x_l|y_u)$ aus wird versucht das Rechteck zu vervollständigen, d.h. die Linie von $(x_l|y_u)$ bis $(30|y_u)$ wird nach Schnittpunkten durchsucht, die als rechten untere Eckpunkte dienen sollen. Wird ein solcher Schnittpunkt bei $(x_r|y_u)$ gefunden, so wird die Linie von $(x_r|y_u)$ bis $(x_r|20)$ nach Schnittpunkten für den rechten oberen Punkt des Rechtecks gescannt. Wenn sich bei $(x_r|y_o)$ der rechte obere Eckpunkt des Rechtecks befindet, muss der linke obere Eckpunkt bei $(x_l|y_o)$. Ist an diesem Punkt ein vorher gefundener Schnittpunkt, d.h. potentieller Eckpunkt, so haben wir ein Rechteck mit den Eckpunkten $(x_l|y_u)$, $(x_r|y_u)$, $(x_r|y_o)$ und $(x_l|y_o)$ gefunden. Andernfalls wurde kein vollständiges Rechteck entdeckt und das Scannen wird fortgesetzt.

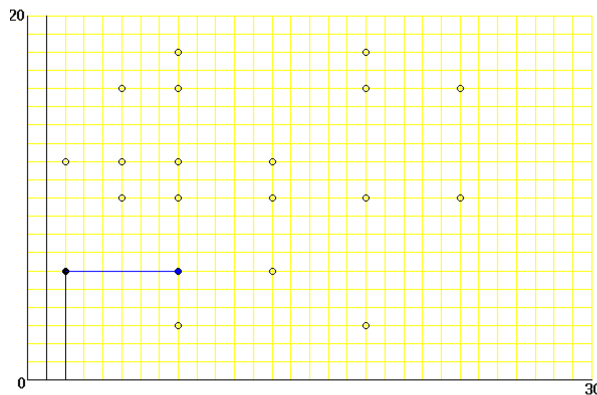
Die folgenden Abbildungen sollen die Arbeitsweise des Algorithmus veranschaulichen.



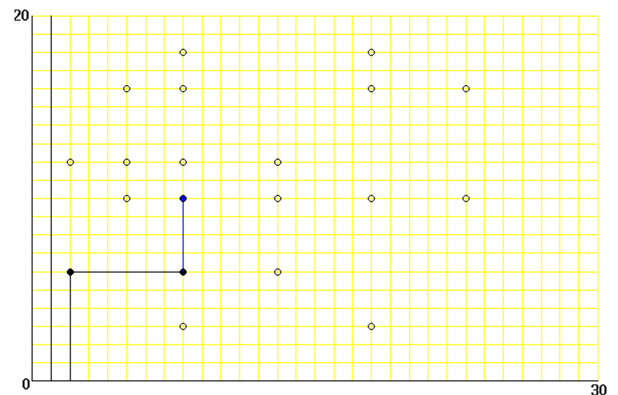
1. Schnittpunkte der Rechtecke aus der Eingabedatei = potentielle Eckpunkte für entstehende Rechtecke.



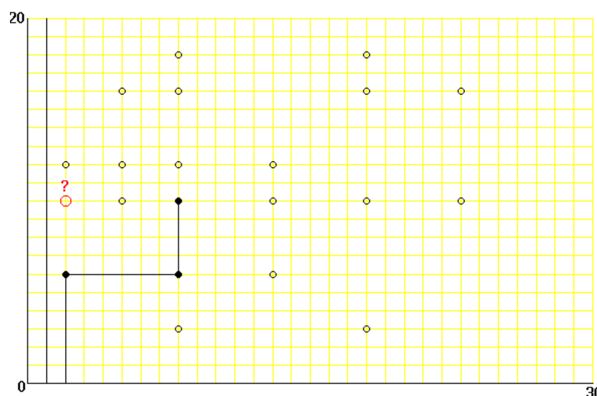
2. Scanne von unten nach oben und links nach rechts. Antreffen eines Schnittpunkts ⇒ linker unterer Eckpunkt.



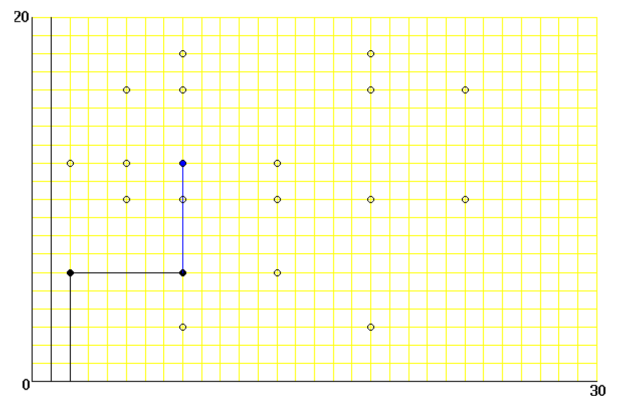
3. Scanne nach rechts bis Schnittpunkt ⇒ rechter unterer Punkt.



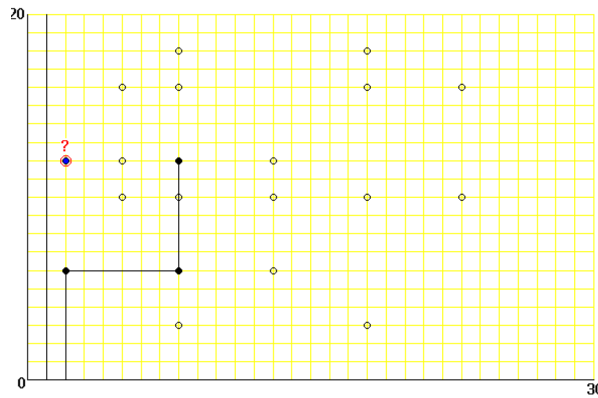
4. Scanne nach oben bis Schnittpunkt ⇒ rechter oberer Punkt.



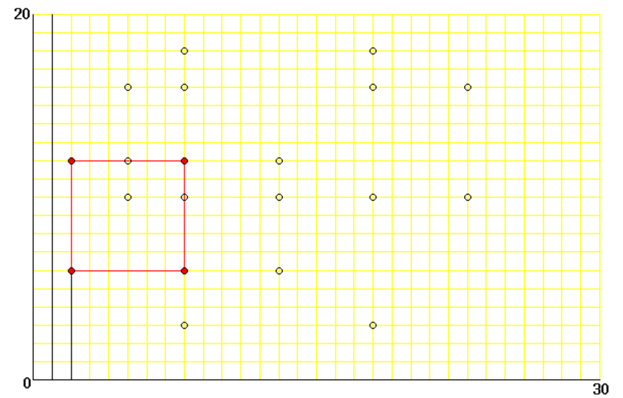
5. Überprüfe linken oberen Punkt des Rechtecks. Kein Schnittpunkt ⇒ kein Rechteck.



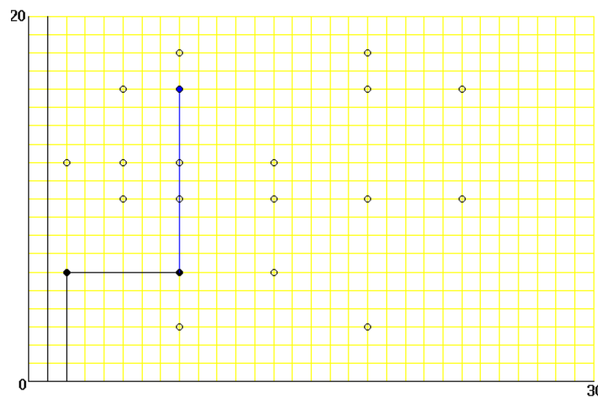
6. Scanne weiter nach oben bis Schnittpunkt ⇒ rechter oberer Punkt.



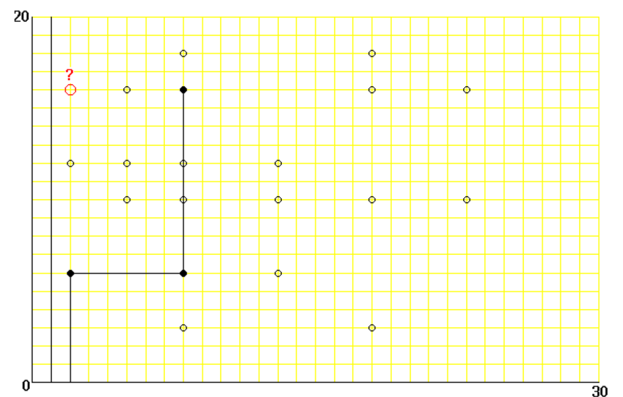
7. Überprüfe linken oberen Punkt des Rechtecks. Schnittpunkt vorhanden ⇒ vollständiges Rechteck gefunden.



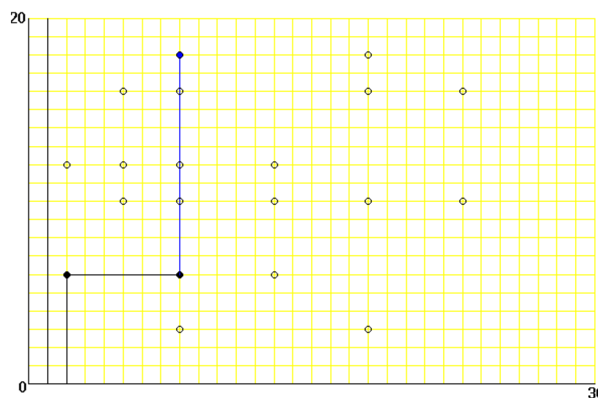
8. Gefundenes Rechteck.



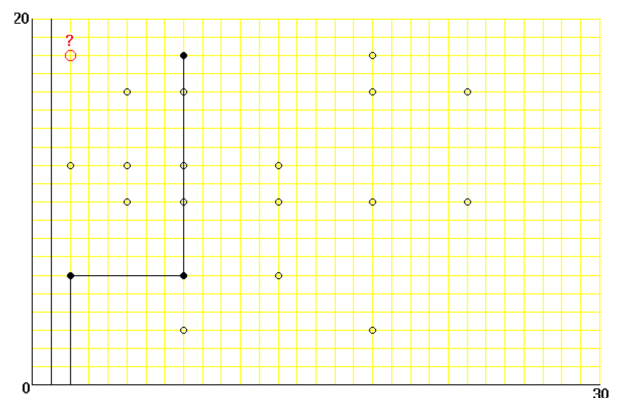
9. Scanne weiter nach oben bis Schnittpunkt ⇒ rechter oberer Punkt.



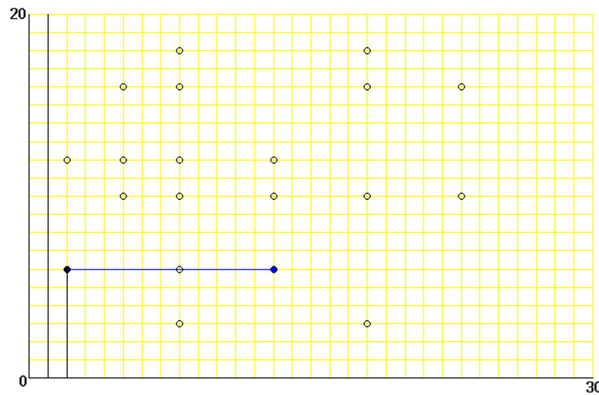
10. Überprüfe linken oberen Punkt des Rechtecks. Kein Schnittpunkt ⇒ kein Rechteck.



11. Scanne weiter nach oben bis Schnittpunkt ⇒ rechter oberer Punkt.



12. Überprüfe linken oberen Punkt des Rechtecks. Kein Schnittpunkt ⇒ kein Rechteck.



13. Scanne weiter nach rechts bis Schnittpunkt ⇒ rechter unterer Punkt.

usw. . . bis „Hauptscan“ von unten nach oben und links nach rechts im Punkt (30|20) angelangt ist.

Die gefundenen Rechtecke werden gezählt, gespeichert und schließlich ausgegeben und gezeichnet.

5.6 Programm-Dokumentation 2

Die Grafikinitialisierung, das Einlesen und Speichern des Inputs und das Zeichnen des Rasters und der Rechtecke wird aus Teilaufgabe 1 übernommen.

Die Schnittpunkte werden im Array `sp[x][y]` gespeichert, nachdem dieses mit Nullen initialisiert wurde.

```
int sp[xMax+1][yMax+1];
for (i = 0; i < xMax+1; i++)
    for (j = 0; j < yMax+1; j++)
        sp[i][j] = 0;
```

Das Array muss für x- bzw. y-Achse `xMax+1=31` bzw. `yMax+1=21` Elemente haben, weil es in dem 20x30 Raster 21 bzw. 31 mögliche Schnittpunkte gibt, wenn man die Ränder der Achsen jeweils als mögliche Schnittpunkte mitzählt.[18]

Nun folgt der Algorithmus zum Finden aller Schnittpunkte. Die Implementierung lautet wie folgt:

Algorithmus zum Finden aller Schnittpunkte

```
for (i = 0; i < numR; i++)
    for (j = 0; j < numR; j++) {
        // A
        if (re[j]->x1 >= re[i]->x1 && re[j]->x1<=re[i]->x2) {
5           // A1
            if (re[i]->y1>=re[j]->y1 && re[i]->y1<=re[j]->y2)
                sp[re[j]->x1][re[i]->y1] = 1;
            // A2
```

[18] Bei der x-Achse gibt es also Schnittpunkte von `sp[0][y]` bis `sp[30][y]`, für die y-Achse von `sp[x][0]` bis `sp[x][20]`.

```
                if (re[i]->y2>=re[j]->y1 && re[i]->y2<=re[j]->y2)
10                  sp[re[j]->x1][re[i]->y2] = 1;
            }
            // B
            if (re[j]->x2>=re[i]->x1 && re[j]->x2<=re[i]->x2) {
                // B1
15              if (re[i]->y1>=re[j]->y1 && re[i]->y1<=re[j]->y2)
                    sp[re[j]->x2][re[i]->y1] = 1;
                // B2
                if (re[i]->y2>=re[j]->y1 && re[i]->y2<=re[j]->y2)
                    sp[re[j]->x2][re[i]->y2] = 1;
20          }
        }
```

Wie oben ausgeführt wurde müssen alle Rechteckpaare i, j durchlaufen werden, also beginnt der Code mit zwei `for` Schleifen mit jeweils `numR` Durchläufen und inkrementierendem Rechteckindex. (`numR` ist die Anzahl der eingegebenen Rechtecke.) Für jedes Paar i, j müssen die vier Fälle A1, A2, B1 und B2 unterschieden werden. Offenbar haben die beiden Fälle A1 und A2 ihre 1. Bedingung gemeinsam, nämlich x_1 von j liegt zwischen x_1 und x_2 von i (Zeile 4). Die 2. Bedingung von A1 lautet: y_1 von i liegt zwischen y_1 und y_2 von i (Zeile 6). Der Schnittpunkt ist dann $(j_{x1}|i_{y1})$ (Zeile 7). Die 2. Bedingung von A2 ist: y_2 von i liegt zwischen y_1 und y_2 von i (Zeile 9). Der Schnittpunkt ist dann $(j_{x1}|i_{y2})$ (Zeile 10). Die Fälle B1 und B2 haben die Bedingung gemeinsam, das x_2 von j zwischen x_1 und x_2 von i liegt (Zeile 13). Die zweite Bedingung von B1 ist: y_1 von i liegt zwischen y_1 und y_2 von j (Zeile 15). Der Schnittpunkt ist dann bei $(j_{x2}|i_{y1})$ (Zeile 16). Die zweite Bedingung von B2 ist: y_2 von i ist zwischen y_1 und y_2 von j (Zeile 18). Der Schnittpunkt liegt bei $(j_{x2}|i_{y2})$ (Zeile 19).

Der nächste Schritt besteht im Auffinden aller Rechtecke an Hand der gefundenen Schnittpunkte. Die Implementierung des oben ausführlich beschriebenen Algorithmus lautet wie folgt:

Algorithmus zum Finden aller Rechtecke

```
   m = 0;
   for (i = 0; i < xMax+1; i++)
       for (j = 0; j < yMax+1; j++)
           if (sp[i][j] == 1)
5              for (k = i+1; k < xMax+1; k++)
                   if (sp[k][j] == 1)
                       for (l = j+1; l < yMax+1; l++)
                           if (sp[k][l] == 1)
                               if (sp[i][l] == 1) {
10                                 printf("Rechteck%i (%i|%i),(%i|%i
                                       )\n",m,i,j,k,l);
                                   rechteck[m] = (struct rechteck *)
                                        malloc(sizeof rechteck);
                                   rechteck[m]->x1 = i; rechteck[m
                                       ]->y1 = j;
```

```
                        rechteck[m]->x2 = k; rechteck[m
                            ]->y2 = l;
                        m++;
15                  }
    numGefundeneRechtecke = m;
    printf("Es gibt %i Rechtecke.\n", numGefundeneRechtecke);
```

Es werden also alle Punkte des Rasters von unten nach oben und links nach rechts durchlaufen. Wenn ein Schnittpunkt gefunden wird, wird die horizontale Linie rechts dieses Schnittpunkts nach weiteren Schnittpunkten gescannt (Zeile 5). Wenn ein Schnittpunkt gefunden wird, wird die vertikale oberhalb dieses Punkts nach weiteren Schnittpunkt abgesucht (Zeile 7). Ist dann der linke obere Punkt des zu konstruierenden Rechtecks ein Schnittpunkt (Zeile 9), so ist ein vollständiges Rechteck gefunden worden. D.h. die Koordinaten des Rechtecks werden ausgegeben und im Array `rechteck` abgespeichert. Zuletzt wird noch die Anzahl `m` der Rechtecke ausgegeben.

Die `redraw` Funktion und der Event Handler werden nun noch so angepasst, dass beim Klicken auf den „weiter“ Button das nächste gefundene Rechteck hervorgehoben (gehighlighted) wird. Im Event Handler wird bei jedem Klick auf „weiter“ dieser Code ausgeführt:

```
hlight++;
hlight = hlight % (numGefundeneRechtecke);
redraw(hlight, rechteck[hlight]);
```

`redraw` verarbeitet diese beiden Argumente für das hervorzuhebende Rechteck dann wie folgt:

```
void redraw (int hlightnr, struct rechteck *hrec) {
    ...
    if (hlightnr > -1) {
        WFillRectangle(myworld, hrec->x1, hrec->y2, hrec->x2,
            hrec->y1, 3);
5       sprintf(str, "Rechteck %i", hlightnr+1);
        WDrawString(myworld, 26,1, str, 3);
    }
    ...
}
```

D.h. das hervorzuhebende Rechteck wird mit der Farbe 3=grün gefüllt und rechts unten im Fenster wird die Nummer dieses Rechtecks angezeigt. `hlightnr` $= -1$ wird beim ersten Starten des Programms aufgerufen, um in diesem Fall kein Rechteck hervorzuheben.

5.7 Programm-Ablaufprotokoll 2

Die Eingabedateien aus Teilaufgabe 1 werden nun vom vollständigen `zapp` Programm aufgerufen. Die Ausgaben des Programms im einzelnen sind:

```
$$ zapp zapp1.in
Rechteck0 (3|2),(6|5)
Rechteck1 (3|2),(8|5)
Rechteck2 (3|2),(13|5)
Rechteck3 (6|2),(8|5)
Rechteck4 (6|2),(8|10)
Rechteck5 (6|2),(13|5)
Rechteck6 (6|5),(8|10)
Rechteck7 (8|2),(13|5)
Es gibt 8 Rechtecke.

$$ zapp zapp2.in
Rechteck0 (2|6),(8|12)
Rechteck1 (2|6),(13|12)
Rechteck2 (5|10),(8|12)
Rechteck3 (5|10),(8|16)
Rechteck4 (5|10),(13|12)
Rechteck5 (5|10),(18|16)
Rechteck6 (5|10),(23|16)
Rechteck7 (5|12),(8|16)
Rechteck8 (8|3),(18|10)
Rechteck9 (8|3),(18|16)
Rechteck10 (8|3),(18|18)
Rechteck11 (8|6),(13|10)
Rechteck12 (8|6),(13|12)
Rechteck13 (8|10),(13|12)
Rechteck14 (8|10),(18|16)
Rechteck15 (8|10),(18|18)
Rechteck16 (8|10),(23|16)
Rechteck17 (8|16),(18|18)
Rechteck18 (18|10),(23|16)
Es gibt 19 Rechtecke.

$$ zapp zapp3.in
Rechteck0 (2|8),(6|13)
Rechteck1 (2|8),(7|13)
Rechteck2 (5|3),(10|11)
Rechteck3 (5|8),(6|11)
Rechteck4 (5|8),(7|11)
Rechteck5 (6|5),(10|11)
Rechteck6 (6|5),(14|15)
Rechteck7 (6|8),(7|11)
Rechteck8 (6|8),(7|13)
Rechteck9 (6|11),(7|13)
Es gibt 10 Rechtecke.

$$ zapp zapp4.in
Rechteck0 (6|10),(7|13)
Rechteck1 (6|10),(10|13)
Rechteck2 (6|10),(17|13)
Rechteck3 (7|2),(10|10)
Rechteck4 (7|2),(10|13)
```

```
Rechteck5 (7|2) ,(28|18)
Rechteck6 (7|10) ,(10|13)
Rechteck7 (7|10) ,(17|13)
Rechteck8 (8|4) ,(10|9)
Rechteck9 (8|4) ,(15|9)
Rechteck10 (10|1) ,(19|2)
Rechteck11 (10|1) ,(19|15)
Rechteck12 (10|2) ,(19|15)
Rechteck13 (10|4) ,(15|9)
Rechteck14 (10|10) ,(17|13)
Es gibt 15 Rechtecke.

$$ zapp zapp5.in
Rechteck0 (2|8) ,(10|16)
Rechteck1 (2|8) ,(18|16)
Rechteck2 (2|8) ,(19|16)
Rechteck3 (2|8) ,(20|16)
Rechteck4 (2|8) ,(30|16)
Rechteck5 (6|10) ,(10|13)
Rechteck6 (6|10) ,(17|13)
Rechteck7 (8|4) ,(10|8)
Rechteck8 (8|4) ,(10|9)
Rechteck9 (8|4) ,(19|8)
Rechteck10 (8|4) ,(19|9)
Rechteck11 (8|8) ,(10|9)
Rechteck12 (8|8) ,(18|9)
Rechteck13 (8|8) ,(19|9)
Rechteck14 (10|1) ,(19|4)
Rechteck15 (10|1) ,(19|8)
Rechteck16 (10|1) ,(19|9)
Rechteck17 (10|1) ,(19|16)
Rechteck18 (10|1) ,(19|18)
Rechteck19 (10|4) ,(19|8)
Rechteck20 (10|4) ,(19|9)
Rechteck21 (10|4) ,(19|16)
Rechteck22 (10|4) ,(19|18)
Rechteck23 (10|8) ,(18|9)
Rechteck24 (10|8) ,(18|16)
Rechteck25 (10|8) ,(18|18)
Rechteck26 (10|8) ,(19|9)
Rechteck27 (10|8) ,(19|16)
Rechteck28 (10|8) ,(19|18)
Rechteck29 (10|8) ,(20|16)
Rechteck30 (10|8) ,(30|16)
Rechteck31 (10|9) ,(18|16)
Rechteck32 (10|9) ,(18|18)
Rechteck33 (10|9) ,(19|16)
Rechteck34 (10|9) ,(19|18)
Rechteck35 (10|10) ,(17|13)
Rechteck36 (10|16) ,(18|18)
Rechteck37 (10|16) ,(19|18)
Rechteck38 (18|5) ,(19|8)
Rechteck39 (18|5) ,(19|9)
Rechteck40 (18|5) ,(19|16)
```

```
Rechteck41 (18|5),(19|18)
Rechteck42 (18|5),(20|8)
Rechteck43 (18|5),(20|16)
Rechteck44 (18|5),(20|19)
Rechteck45 (18|8),(19|9)
Rechteck46 (18|8),(19|16)
Rechteck47 (18|8),(19|18)
Rechteck48 (18|8),(20|16)
Rechteck49 (18|8),(20|19)
Rechteck50 (18|8),(30|16)
Rechteck51 (18|9),(19|16)
Rechteck52 (18|9),(19|18)
Rechteck53 (18|16),(19|18)
Rechteck54 (18|16),(20|19)
Rechteck55 (19|5),(20|8)
Rechteck56 (19|5),(20|16)
Rechteck57 (19|8),(20|16)
Rechteck58 (19|8),(30|16)
Rechteck59 (20|8),(30|16)
Es gibt 60 Rechtecke.
```

5.8 Quellcode 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Xgraphics.h"

#define xMax 30
#define yMax 20
#define rMax 10 // max. Anzahl Rechtecke

/* Variablen */
Window mywindow;
World myworld;
XEvent myevent;
int i,j,k,l,m,done=0, numR, numGefundeneRechtecke;
// Schnittpunkte Array
int sp[xMax+1][yMax+1]; // Null mitzaehlen!
// Counter fuer highlighted Rechteck
int hlight = -1;
// temporaerer String
char str[20];
char *str2;
// linke untere Ecke (x1|y1), rechte obere Ecke (x2|y2)
struct rechteck { int x1,y1, x2,y2; };
// Rechtecke Array
struct rechteck *re[rMax];

/* Zeichne Grafik */
void redraw (int hlightnr, struct rechteck *hrec) {
    // Loesche vorheriges
```

```
30      ClearWorld(myworld);
        // Zeichne Raster
        for (i = 1; i <= 30; i++)
            WDrawLine(myworld, i,0, i,20, 5);
        for (i = 1; i <= 20; i++)
35          WDrawLine(myworld, 0,i, 30,i, 5);
        WDrawLine(myworld, 0,0, 30,0, 1);
        WDrawLine(myworld, 0,0, 0,20, 1);
        WDrawString(myworld, -0.5,-0.5, "0", 1);
        WDrawString(myworld, 29.7,-0.8, "30", 1);
40      WDrawString(myworld, -1,19.7, "20", 1);
        // Highlighte hlight-tes Rechteck
        if (hlightnr > -1) {
            WFillRectangle(myworld, hrec->x1, hrec->y2, hrec->x2, hrec->y1,
                3);
            sprintf(str, "Rechteck %i", hlightnr+1);
45          WDrawString(myworld, 26,1, str, 3);
        }
        // Zeichne alle Rechtecke
        for (i = 0; i < numR; i++)
            WDrawRectangle(myworld, re[i]->x1, re[i]->y2, re[i]->x2, re[i
               ]->y1, 4);
50       // Zeichne Schnittpunkte */
        for (i = 0; i < xMax+1; i++)
            for (j = 0; j < yMax+1; j++)
                if (sp[i][j] == 1)
                    WDrawCircle(myworld, i,j, 0.2, 1);
55  }


    int main (int argc, char **argv) {
        if (argc != 2) {
60          printf("Keine Eingabedatei angegeben.\nAufruf: zapp input-file.
                in\n");
            exit(0);
        }
        /* Variablen */
        FILE *in, *out;
65
        /* Initialisierung der Grafik */
        InitX();
        // Erzeuge neues Fenster mywindow
        mywindow = CreateWindow(700,400,"Zapp");
70      // Erzeuge neue Welt myworld
        myworld = CreateWorld(mywindow,0,0,600,400,-1,21,31,-1,0,0);
        // Buttons
        InitButtons(mywindow, "t,Kommandos:,,b,ende,e,b,weiter,w", 100);
        // Zeige Fenster
75      ShowWindow(mywindow);

        /* Lese Input ein */
        in = fopen(argv[1], "r");
        // Fuelle x1,y1,x2,y2 Arrays mit Koordinaten
80      for (i = 0; !feof(in) && i < rMax; i++) {
```

```
        re[i] = (struct rechteck *) malloc(sizeof re);
        fscanf(in, "%i %i %i %i", &re[i]->x1, &re[i]->y1, &re[i]->x2, &
            re[i]->y2);
    }
    // Anzahl eingelesene Rechtecke
85  numR = i;

    // initialisiere sp Schnittpunkte-Array
    for (i = 0; i < xMax+1; i++)
        for (j = 0; j < yMax+1; j++)
90          sp[i][j] = 0;

    // Finde alle Schnittpunkte
    for (i = 0; i < numR; i++)
        for (j = 0; j < numR; j++) {
95          // A
            if (re[j]->x1 >= re[i]->x1 && re[j]->x1<=re[i]->x2) {
                // A1
                if (re[i]->y1>=re[j]->y1 && re[i]->y1<=re[j]->y2)
                    sp[re[j]->x1][re[i]->y1] = 1;
100             // A2
                if (re[i]->y2>=re[j]->y1 && re[i]->y2<=re[j]->y2)
                    sp[re[j]->x1][re[i]->y2] = 1;
            }
            // B
105         if (re[j]->x2>=re[i]->x1 && re[j]->x2<=re[i]->x2) {
                // B1
                if (re[i]->y1>=re[j]->y1 && re[i]->y1<=re[j]->y2)
                    sp[re[j]->x2][re[i]->y1] = 1;
                // B2
110             if (re[i]->y2>=re[j]->y1 && re[i]->y2<=re[j]->y2)
                    sp[re[j]->x2][re[i]->y2] = 1;
            }
        }

115 // Finde alle Rechtecke
    struct rechteck *rechteck[1000];
    m = 0;
    for (i = 0; i < xMax+1; i++)
        for (j = 0; j < yMax+1; j++)
120         if (sp[i][j] == 1)
                for (k = i+1; k < xMax+1; k++)
                    if (sp[k][j] == 1)
                        for (l = j+1; l < yMax+1; l++)
                            if (sp[k][l] == 1)
125                             if (sp[i][l] == 1) {
                                    printf("Rechteck%i (%i|%i),(%i|%i)\
                                        n",m,i,j,k,l);
                                    rechteck[m] = (struct rechteck *)
                                        malloc(sizeof rechteck);
                                    rechteck[m]->x1 = i; rechteck[m]->
                                        y1 = j;
                                    rechteck[m]->x2 = k; rechteck[m]->
                                        y2 = l;
```

```
130                                                  m++;
                                                 }

          numGefundeneRechtecke = m;
          printf("Es gibt %i Rechtecke.\n", numGefundeneRechtecke);
135
          while(!done) {
               if (GetEvent(&myevent,1)) {
                    switch(myevent.type) {
                    case KeyPress:
140                      switch (ExtractChar(myevent) ) {
                         case 'e': done = 1; break;
                         case 'w':
                              hlight++;
                              hlight = hlight % (numGefundeneRechtecke);
145                           redraw(hlight, rechteck[hlight]);
                              break;
                         default: break;
                         }
                         break;
150                 case Expose: redraw(hlight, rechteck[hlight]); break;
                    default: break;
                    }
               }
          }
155
          while(!done) {
               GetEvent(&myevent,1);
               if (myevent.type == ButtonPress) done = 1;
          }
160
          ExitX();
          exit(0);
     }
```

6 Wörterketten

6.1 Lösungsidee

Aus der Aufgabenstellung leitet sich Folgendes ab: Zunächst müssen alle Wörter eingelesen werden, die länger als drei Buchstaben sind. Dazu verwenden wir bequemerweise eine Textdatei. Im Prinzip müssten für n Wörter $n!$ Permutationen betrachtet werden. Eine Permutation ist aber nur dann brauchbar, wenn eine Teilfolge der Wörter innerhalb dieser Permutation auch so dem Kriterium entspricht, dass die letzten und ersten drei Buchstaben jedes Wortes in der Folge übereinstimmen. Die Anzahl an brauchbaren Wörtern für das erste durchzuführende Beispiel beträgt 328, es müssten also

$$328! = 0,26 * 10^{685}$$

Wortfolgen überprüft werden. Wenn man in einer Sekunde eine Million Permutationen prüfen könnte, so bräuchte man immer noch 10^{12} Mal so lange, um ein Ergebnis zu erhalten, wie seit heutiger Erkenntnis vom Urknall bis jetzt an Zeit verstrich. Offensichtlich ist dies grober Unfug. Es muss einen Algorithmus geben, der dieses in wesentlich kürzerer Zeit zu berechnen imstande ist. Dazu geben wir entsprechende Abbruchbedingungen vor und lassen nicht die Permutationen im Voraus statisch berechnen, sondern lenken bei der Berechnung dynamisch ein. Dieser Algorithmus funktioniert wie folgt:

Nimm jedes brauchbare Wort, sieh nach, ob in den übrigen, das heißt der Menge der brauchbaren Wörter abzüglich des gerade zu betrachtenden Wortes und aller vorher betrachteten, sich passende Gegenstücke finden und tue für alle Fundstellen dasselbe. Im Klartext heißt dies: Zunächst werden alle brauchbaren Wörter w_i durchlaufen. Jedes w_i wird mit allen anderen Wörtern v_j abgeglichen: Das heißt, die letzten und die ersten drei Buchstaben beider Wörter müssen kongruent sein. v_j sind dabei alle Wörter, die in der bisherigen Wortzusammensetzung nicht bereits verwendet wurden. Sollte es eine oder mehrere Fundstellen geben, so wird für diese $w_i = v_j$ gesetzt und der Algorithmus rekursiv von neuem durchlaufen. Hierbei werden folgenden Definition getroffen: Die längste Wörterreihung ist die mit den meisten Buchstaben. Da in einer Datei mehrmals vorkommende Wörter wie „ich“ nur einmal durchlaufen werden sollen, wird folgende Defintion getroffen:

$$ich = ich$$

$$Ich\ != ich$$

Aber:

$$tratsCH = tratsch = trATsch$$

6.2 Programmdokumentation

Der Programmaufruf erwartet als ersten Parameter den Pfad zu einer einzulesenden Datei. Im Normalfall sollte der gewählte Standardwert von 20000 eizulesenden Zeichen vollaufgenügen. Um aber auch größere Dateien zu ermöglichen, können vom Benutzer als zweites Argumente aber auch beliebige andere numerische Werte eingegeben werden.

Die Arbeit wurde der Struktur halber in Objektorientierung verrichtet. Dazu wird zuerst in der Main-Funktion ein **wortkette**-Objekt generiert, wovon aus sich der Code verselbstständig: Es wird der Konstruktor aufgerufen.

In ihm wird zuerst die Datei, die als erstes Argument angegeben wurde, geöffnet und in das **char**-Array eingelesen. Darauffolgend wird überprüft, wie viele Wörter sich in der Datei befinden, das heißt, wir gehen zeichenweise durch das ganze Array wie folgt vor:

> Wenn das aktuelle Zeichen ein Wortrenner ist, setze **neueswort** auf true, was den Beginn eines neuen Wortes ab dem nächsten Buchstaben signalisieren soll, und **vorgaenger** = 0.
>
> Wenn **neueswort** true ist, so ist dies der Beginn eines neuen Wortes, aber nur dann, wenn der aktuelle Buchstaben auch tatsächlich ein Buchstabe ist.
>
> ```
> if(isalpha(buf[i]) || buf[i] == 'ä' || buf[i] == 'Ä' ||
>     buf[i] ==
> 'ö' || buf[i] == 'Ö' || buf[i] == 'ü' || buf[i] == 'Ü' ||
>     buf[i]
> == 'ß' || buf[i] == '\'')
> ```
>
> Wenn **vorgaenger** gleich null, handelt es sich um ein völlig neues Wort. Erhöhe **elemente** um 1.
>
> Erhöhe **vorgaenger** um 1.

Nun benötigen wir der besseren Handelbarkeit wegen ein temporäres Stringarray, das wir dynamisch mit **new** im *free store* erzeugen. Es soll die Größe **elemente** erhalten, die wir vorher ermittelt haben. Es wird genau **elemente** Elemente enthalten, also genau nachdenselben Regeln wie oben mit Wörtern initialisiert.

Schließlich muss dieser temporäre String nochmals in seiner Gesamtheit durchgangen werden und redundante (d.h. doppelt oder sogar noch öfter vorkommende Wörter) gelöscht werden. Dazu zählen auch Wörter, welche kleiner als 3 Zeichen sind.

```
for(int j=0;j<reali;j++) {
  if(pTempWortliste[i].compare(pTempWortliste[j]) == 0)
```

Prüft, ob das Wort, das momentan eingefügt werden soll nicht schon in den bereitseingefügten Wörtern vorhanden ist. Nur wenn dies nicht der Fall war (d.h. **vorhanden** false ist), darf das Wort dem finalen Stringarray hinzugefügt werden. Auch hierfür werden wir **new** benötigen, da ja im Vorhinein noch nicht feststeht, wie viele Wörter wegfallen werden. Deshalb wie schon zuvor, auch hier der Aufruf zweier fast kongruenter Codeblöcke (Zeilen 85-95; 99-111): Zuerst wird die Größe des finalen Arrays **pWortliste** festgestellt und dieses dann nach den bereits zur Größenfestellung benutzten Regeln mit Werten versehen.

Im Anschluss wird **pTempWortliste** redundant und folgerichtig aus dem Speicher entfernt:

```
delete [] pTempWortliste;
```

Die größte Schwierigkeit besteht im Erfinden des richtigen Algorithmuses. Dieser muss rekursiv sein, und ist damit, das weiß jeder Informatikstudent, besonders anfällig für Fehler aufgrund seiner schweren Überblickbarkeit. Die Memberfunktion **erzeugeElementreihe** ruft sich selbst rekursiv auf. Wir müssen ihr lediglich das Wort vorgeben, das sie anfangen soll zu überprüfen und ein bool-Array, das die Wörter, die bereits überprüft und gegebenenfalls an **wortreihe** angehängt wurden, mit „true“ markiert (**pVisited**).

erzeugeElementreihe(bool* pVisited, string aktwortreihe)

Starte bei $i = 0$. Gehe alle Wörter in **pWortliste** durch, die noch nicht besucht wurden (dazu zählt per Defintion nicht das Anfangswort, es wird ja als zweites Argument an **erzeugeElementreihe()** übergeben!

Substrahiere aus **aktwortreihe** die letzten drei Buchstaben, wandle sie in Kleinbuchstaben um[19] und tue Selbiges mit dem i-ten Element des Arrays **pWortliste**-Arrays. Um das Originalarray nicht zu verfälschen, wird dabei eine Kopie in **pString** gesichert, die von nun an weiterbearbeitet wird. Selbiges geschieht mit dem 2. Argument, welches die Funktion erwartet, nämlich **aktwortreihe**, allerdings wird es in **tmpwort** zwischengespeichert.

Sind die letzten drei Buchstaben von **pString** und **tmpwort** gleich (und nur dann!), so setze das i-te Element des Booleanarrays **pVisited** auf true (was einem Entfernen dieses Wortes = Verhindern einer Endlosreihung wie „Kerker...“ einen Riegel vorschiebt). Füge zu **aktwortreihe** das i-te Wort aus **pWortliste** ohne die ersten drei Buchstaben (da diese ja bereits vorhanden sind!) hinzu.

```
if(pString.compare(0,3,tmpwort) == 0) {
                pVisited[i] = true;
                aktwortreihe += pWortliste[i].substr(3);
```

Rufe die Memberfunktion der Klasse, **erzeugeElementreihe** mit dem so veränderten **pVisited** und dem neuen **aktwortreihe** auf.

Ist man dann bei einer gewissen Rekursionstiefe (und zwar $i = elemente$) angelangt, so muss überprüft werden, ob die gefundene Lösung länger ist, als die bereits vorhandene:

```
if(aktwortreihe.length() > wortreihe.length())
wortreihe = aktwortreihe;
```

wortreihe ist eine Membervariable der Klasse Wortkette, die wir danach auch ansprechend ausgeben. Im schlechtesten Fall enthält sie zumindest das längste im Text vorkommende Wort.

[19]Nur, um sicherzugehen

6.3 Programm-Ablaufprotokoll

junior1.in

Zunächst wird das Beispiel aus der Aufgabenstellung getestet:

Verlassen hab ich Feld und Auen,
Die eine tiefe Nacht bedeckt,
Mit ahnungsvollem, heil'gem Grauen
In uns die beßre Seele weckt.
Entschlafen sind nun wilde Triebe
Mit jedem ungestümen Tun;
Es reget sich die Menschenliebe,
Die Liebe Gottes regt sich nun. Sei ruhig, Pudel! renne nicht hin und wider!
An der Schwelle was schnoperst du hier?
Lege dich hinter den Ofen nieder,
Mein bestes Kissen geb ich dir.
Wie du draußen auf dem bergigen Wege
Durch Rennen und Springen ergetzt uns hast,
So nimm nun auch von mir die Pflege,
Als ein willkommner stiller Gast. Ach wenn in unsrer engen Zelle
Die Lampe freundlich wieder brennt,
Dann wird's in unserm Busen helle,
Im Herzen, das sich selber kennt.
Vernunft fängt wieder an zu sprechen,
Und Hoffnung wieder an zu blühn,
Man sehnt sich nach des Lebens Bächen,
Ach! nach des Lebens Quelle hin. Knurre nicht, Pudel! Zu den heiligen Tönen,
Die jetzt meine ganze Seel umfassen,
Will der tierische Laut nicht passen.
Wir sind gewohnt, daß die Menschen verhöhnen,
Was sie nicht verstehn,
Daß sie vor dem Guten und Schönen,
Das ihnen oft beschwerlich ist, murren;
Will es der Hund, wie sie, beknurren?

Aber ach! schon fühl ich, bei dem besten Willen,
Befriedigung nicht mehr aus dem Busen quillen.
Aber warum muß der Strom so bald versiegen,
Und wir wieder im Durste liegen?
Davon hab ich so viel Erfahrung.
Doch dieser Mangel läßt sich ersetzen,
Wir lernen das Überirdische schätzen,
Wir sehnen uns nach Offenbarung,
Die nirgends würd'ger und schöner brennt
Als in dem Neuen Testament.

Mich drängt's, den Grundtext aufzuschlagen,
Mit redlichem Gefühl einmal
Das heilige Original
In mein geliebtes Deutsch zu übertragen,
Hier stock ich schon! Wer hilft mir weiter fort?
Ich kann das Wort so hoch unmöglich schätzen,
Ich muß es anders übersetzen,
Wenn ich vom Geiste recht erleuchtet bin.
Geschrieben steht: Im Anfang war der Sinn.
Bedenke wohl die erste Zeile,
Daß deine Feder sich nicht übereile!
Ist es der Sinn, der alles wirkt und schafft?
Es sollte stehn: Im Anfang war die Kraft!
Doch, auch indem ich dieses niederschreibe,
Schon warnt mich was, daß ich dabei nicht bleibe.
Mir hilft der Geist! Auf einmal seh ich Rat
Und schreibe getrost: Im Anfang war die Tat!

Soll ich mit dir das Zimmer teilen,
Pudel, so laß das Heulen,
So laß das Bellen!
Solch einen störenden Gesellen
Mag ich nicht in der Nähe leiden.
Einer von uns beiden
Muß die Zelle meiden.
Ungern heb ich das Gastrecht auf,
Die Tür ist offen, hast freien Lauf.
Aber was muß ich sehen!
Kann das natürlich geschehen?
Ist es Schatten? ist's Wirklichkeit?
Wie wird mein Pudel lang und breit!
Er hebt sich mit Gewalt,
Das ist nicht eines Hundes Gestalt!
Welch ein Gespenst bracht ich ins Haus!
Schon sieht er wie ein Nilpferd aus,
Mit feurigen Augen, schrecklichem Gebiß.
Oh! du bist mir gewiß!
Für solche halbe Höllenbrut
Ist Salomonis Schlüssel gut.

Die Ausgabe des Programms ist:

```
$$ junior junior1.in
Es sind 328 nutzbare Wörter in der Datei festzustellen.

Das längste aneinandergereihte Wort lautet
```

```
    Befriedigungestümenschenliebe.
Es besitzt eine Länge von 29 Zeichen.
```

Das längste aneinandergereihte Wort is also Befriedigungestümenschenliebe.

junior2.in

Als nächstes soll die Bibel für ein Beispiel herhalten (Ruth ['root], Kap. 4, Die Lösung):

1Boas ging hinauf ins Tor und setzte sich daselbst. Und siehe, als der Löser vorüberging, von dem er geredet hatte, sprach Boas: Komm, mein Lieber, und setze dich hierher! Und er kam herüber und setzte sich dort hin. 2Und Boas nahm zehn Männer von den Ältesten der Stadt und sprach: Setzt euch hierher! Und sie setzten sich. 3Da sprach er zu dem Löser: Noomi, die aus dem Lande der Moabiter zurückgekommen ist, bietet feil den Anteil an dem Feld, der unserm Bruder Elimelech gehörte. 4Darum gedachte ich's vor deine Ohren zu bringen und zu sagen: Willst du es lösen, so kaufe es vor den Bürgern und vor den Ältesten meines Volks; willst du es aber nicht lösen, so sage mir's, daß ich's wisse; denn es ist kein anderer Löser da als du, und ich nach dir. Er sprach: Ich will's lösen. a 5Boas sprach: An dem Tage, da du von Noomi das Feld kaufst, mußt du auch Rut, die Moabiterin, die Frau des Verstorbenen, nehmen, um den Namen des Verstorbenen zu erhalten auf seinem Erbteil. b 6Da antwortete er: Ich vermag es nicht zu lösen, sonst würde ich mein Erbteil schädigen. Löse dir zugut, was ich hätte lösen sollen; denn ich vermag es nicht zu lösen. 7Es war aber von alters her ein Brauch in Israel: Wenn einer eine Sache bekräftigen wollte, die eine Lösung oder einen Tausch betraf, so czog er seinen Schuh aus und gab ihn dem andern; das diente zur Bezeugung in Israel. 8Und der Löser sprach zu Boas: Kaufe du es! und zog seinen Schuh aus. 9Und Boas sprach zu den Ältesten und zu allem Volk: Ihr seid heute Zeugen, daß ich von Noomi alles gekauft habe, was Elimelech, und alles, was Kiljon und Machlon gehört hat. 10Dazu habe ich mir auch Rut, die Moabiterin, die Frau Machlons, zum Weibe genommen, daß ich den Namen des Verstorbenen erhalte auf seinem Erbteil und sein Name nicht ausgerottet werde unter seinen Brüdern und aus dem Tor seiner Stadt; dessen seid ihr heute Zeugen. 11Und alles Volk, das im Tor war, samt den Ältesten sprach: Wir sind Zeugen. Der HERR mache die Frau, die in dein Haus kommt, wie Rahel und Lea, die beide das Haus Israel gebaut haben; sei stark in Efrata, und dein Name werde gepriesen zu Bethlehem. 12Und dein Haus werde wie das Haus des dPerez, den Tamar dem Juda gebar, durch die Nachkommen, die dir der HERR geben wird von dieser jungen Frau. a: 3. Mose 25,25; Jer 32,6-10 b: 5. Mose 25,5-6 c: 5. Mose 25,7-10 d: 1. Mose 38,29

Boas heiratet Rut, die Stammutter Davids 13So nahm Boas die Rut, daß sie seine Frau wurde. Und als er zu ihr einging, gab ihr der HERR, daß sie schwanger ward, und sie gebar einen Sohn. 14Da sprachen die Frauen zu Noomi: Gelobt sei der HERR, der dir zu dieser Zeit einen Löser nicht versagt hat! Dessen Name werde gerühmt in Israel! 15Der wird dich erquicken und dein Alter versorgen. Denn deine Schwiegertochter, die dich geliebt hat, hat ihn geboren, die dir mehr wert ist als sieben Söhne. 16Und Noomi nahm das Kind und legte es auf ihren Schoß und ward seine Wärterin. 17Und ihre Nachbarinnen gaben ihm einen Namen und sprachen: Noomi ist ein Sohn geboren; und sie nannten ihn aObed. Der ist der Vater Isais, welcher Davids Vater ist. a: Mt 1,5-6; Lk 3,32

18Dies ist das Geschlecht des aPerez: Perez zeugte Hezron; 19Hezron zeugte Ram; Ram zeugte Amminadab; b 20Amminadab zeugte cNachschon; Nachschon zeugte Salmon; 21Salmon zeugte Boas; Boas zeugte Obed; 22Obed zeugte Isai; Isai zeugte David. d Die Ausgabe des Programms ist:

```
$$ junior junior2.in
Es sind 342 nutzbare Wörter in der Datei festzustellen.

Das längste aneinandergereihte Wort lautet Tauschädigenommen.
Es besitzt eine Länge von 17 Zeichen.
```

junior3.in

Von selbigem Beispiel soll auch die lateinische Fassung überprüft werden:

ascendit ergo Booz ad portam et sedit ibi cumque vidisset propinquum praeterire de quo prius sermo habitus est dixit ad eum declina paulisper et sede hic vocans eum nomine suo qui devertit et sedit tollens autem Booz decem viros de senioribus civitatis dixit ad eos sedete hic quibus residentibus locutus est ad propinquum partem agri fratris nostri Helimelech vendit Noemi quae reversa est de regione moabitide quod audire te volui et tibi dicere coram cunctis sedentibus et maioribus natu de populo meo si vis possidere iure propinquitatis eme et posside sin autem tibi displicet hoc ipsum indica mihi ut sciam quid facere debeam nullus est enim propinquus excepto te qui prior es et me qui secundus sum at ille respondit ego agrum emam cui dixit Booz quando emeris agrum de manu mulieris Ruth quoque Moabitidem quae uxor defuncti fuit debes accipere ut suscites nomen propinqui tui in hereditate sua qui respondit cedo iure propinquitatis neque enim posteritatem familiae meae delere debeo tu meo utere privilegio quo me libenter carere profiteor hic autem erat mos antiquitus in Israhel inter propinquos et si quando alter alteri suo iure cedebat ut esset firma concessio solvebat homo calciamentum suum et dabat proximo suo hoc erat testimonium cessionis in Israhel dixit ergo propinquus Booz tolle calciamentum quod statim solvit de pede suo at ille maioribus natu et universo populo testes inquit vos estis hodie quod possederim omnia quae fuerunt Helimelech et Chellion et Maalon tradente Noemi et Ruth Moabitidem uxorem Maalon in coniugium sumpserim ut suscitem nomen defuncti in hereditate sua ne vocabulum eius de familia sua ac fratribus et populo deleatur vos inquam huius rei testes estis respondit omnis populus qui erat in porta et maiores natu nos testes sumus faciat Dominus hanc mulierem quae ingreditur domum tuam sicut Rachel et Liam quae aedificaverunt domum Israhel ut sit exemplum virtutis in Ephrata et habeat celebre nomen in Bethleem fiatque domus tua sicut domus Phares quem Thamar peperit Iudae de semine quod dederit Dominus tibi ex hac puella tulit itaque Booz Ruth et accepit uxorem ingressusque est ad eam et dedit illi Dominus ut conciperet et pareret filium dixeruntque mulieres ad Noemi benedictus Dominus qui non est passus ut deficeret successor familiae tuae et vocaretur nomen eius in Israhel et habeas qui consoletur animam tuam et enutriat senectutem de nuru enim tua natus est quae te diligit et multo tibi est melior quam si septem haberes filios susceptumque Noemi puerum posuit in sinu suo et nutricis ac gerulae officio fungebatur vicinae autem mulieres congratulantes ei et dicentes natus est filius Noemi vocaverunt nomen eius Obed hic est pater Isai patris David hae sunt generationes Phares Phares genuit Esrom Esrom genuit Aram Aram genuit Aminadab Aminadab genuit Naasson Naasson genuit Salma Salma genuit Booz Booz genuit Obed Obed genuit Isai Isai genuit David

Die Ausgabe des Programms ist:

```
$$ junior junior3.in
Es sind 306 nutzbare Wörter in der Datei festzustellen.

Das längste aneinandergereihte Wort lautet congratulantestimonium.
Es besitzt eine Länge von 22 Zeichen.
```

junior4.in

Zuletzt ein sehr einleuchtende Beispiel:

schulen gerichtsvollzieher zu HervoRRagenden nerviger KEnner denkanstößen. lenken

Die Ausgabe des Programms ist:

```
$$ junior junior4.in
Es sind 7 nutzbare Wörter in der Datei festzustellen.

Das längste aneinandergereihte Wort lautet
    schulenkennervigerichtsvollziehervoRRagendenkanstößen.
Es besitzt eine Länge von 53 Zeichen.
```

6.4 Code

junior.h

```
#ifndef JUNIOR_H
#define JUNIOR_H

#include <string>
using namespace std;

class wortkette {

  public:
   wortkette(const char*,int groesse=20000);           // constructor
   ~wortkette();                                       // destructor

   int wortreihenlaenge(void);
   void erzeugeElementreihe(bool*,string);

  private:
   int elemente;

   string wortreihe;
   string* pWortliste;
   bool* pVisited;
};

#endif
```

junior.cpp

```
#include "junior.h"
#include <iostream>
#include <cctype>
#include <stdio.h>
using namespace std;

/* Konstruktor für char*-Argumente (old-style C) */

wortkette::wortkette(const char* datei, int groesse) {
   elemente = 0;

   // Dateieinlesungsprozess
```

```
    int tmpelement = -1;

15  char buf[groesse];
    for(int i=0;i<groesse;i++)
      buf[i] = ' ';

    FILE * stream;
20  stream=fopen(datei,"r");

    int numRead=fread(buf, 1, groesse, stream);
    buf[numRead]='\0'; // String beenden, damit er als char * umgewandelt
        werden kann
    fclose(stream);
25
    /* Finde heraus, wie viele Wörter aus der Datei extrahiert werden
       können
    */
    bool neueswort = true;
30  int vorgaenger = 0;
    for(int i=0;i<groesse;i++) {
      if(buf[i] == ' ' || buf[i] == '\n') {
        neueswort = true;
        vorgaenger = 0;
35    }
      else if(neueswort == true) {
        if(isalpha(buf[i]) || buf[i] == 'ä' || buf[i] == 'Ä' || buf[i]
            == 'ö' || buf[i] == 'Ö' || buf[i] == 'ü' || buf[i] == 'Ü' ||
            buf[i] == 'ß' || buf[i] == '\'') {
          if(vorgaenger == 0) {
            elemente++;
40        }
          vorgaenger++;
        }
      }
    }
45
    /* Durchgehe Datei ein zweites Mal, um die Wörter in dem dynamisch,
       der Anzahl der oben festgestellten verwendbaren, temporären
       Stringarray abzuspeichern
    */
50  string* pTempWortliste = new string[elemente];

    for(int i=0; i < elemente; i++) {
      pTempWortliste[i] = " ";
    }
55
    neueswort = true;
    vorgaenger = 0;

    for(int i=0;i<groesse;i++) {
60    if(buf[i] == ' ' || buf[i] == '\n') {
        neueswort = true;
        vorgaenger = 0;
```

```
      }

65    else if(neueswort == true) {
        if(isalpha(buf[i])  || buf[i] == 'ä' || buf[i] == 'Ä' || buf[i]
            == 'ö' || buf[i] == 'Ö' || buf[i] == 'ü' || buf[i] == 'Ü' ||
            buf[i] == 'ß' || buf[i] == '\'') {
          if(vorgaenger == 0) {
            tmpelement++;
          }
70
          pTempWortliste[tmpelement] += buf[i];
          vorgaenger++;
        }
      }
75  }

    /* Durch gehe den im letzten Schritt erstellten String wiederum, um
       Wörter < 3 Buchstaben herauszufiltern, speichere sie in einem
       endgültigen Arraystring und lösche den temporären
80  */
    int tmpelemente = 0;
    int reali = 0;
    bool vorhanden = false;

85  for(int i=0;i<elemente;i++) {
      vorhanden = false;
      if(pTempWortliste[i].length() > 3) { // Da das Leerzeichen noch
          enthalten ist
        for(int j=0; j<tmpelemente; j++) {
          if(pTempWortliste[i].compare(pTempWortliste[j]) == 0)
90          vorhanden = true;
        }
        if(vorhanden == false)
          tmpelemente++;
      }
95  }

    pWortliste = new string[tmpelemente];

    for(int i=0;i<elemente;i++) {
100   vorhanden = false;
      if(pTempWortliste[i].length() > 3) {
        for(int j=0;j<reali;j++) {
          if(pTempWortliste[i].compare(pTempWortliste[j]) == 0)
            vorhanden = true;
105     }
        if(vorhanden == false) {
          pWortliste[reali] = pTempWortliste[i].substr(1); // Entferne
              das Leerzeichen am Anfang des Wortes
          reali++;
        }
110   }
    }
```

```
    delete [] pTempWortliste;

115   elemente = tmpelemente; // Fehlerbereinigte, zur Berechnung
        tatsächlich verwenbare Anzahl an Elementen

    cout << "Es sind " << elemente << " nutzbare Wörter in der Datei
        festzustellen."
         << endl
         << endl;
120
    pVisited = new bool[elemente];
    for(int i=0;i<elemente;i++) {
      for(int j=0; j < elemente; j++)
        pVisited[j] = false;
125   pVisited[i] = true;

      erzeugeElementreihe(pVisited,pWortliste[i]);
    }

130 cout << "Das längste aneinandergereihte Wort lautet " << wortreihe <<
        ". "
         << endl
         << "Es besitzt eine Länge von " << wortreihenlaenge() << "
            Zeichen."
         << endl;
  }
135

  wortkette::~wortkette() {
    delete [] pWortliste;
    delete [] pVisited;
140 }


  int wortkette::wortreihenlaenge(void) {
    return  wortreihe.length();
145 }


  void wortkette::erzeugeElementreihe(bool* pVisited,string aktwortreihe)
      {
      for(int i=0; i < elemente; i++) {
150     if(pVisited[i] == false) {
          string tmpwort = aktwortreihe.substr(aktwortreihe.length()-3,3)
              ;
          if(pWortliste[i].length() > 2) {
              string pString = pWortliste[i];
              for(int j=0;j<3;j++) {
155             pString[j] = tolower(pString[j]);
                tmpwort[j] = tolower(tmpwort[j]);
              }
              if(pString.compare(0,3,tmpwort) == 0) {
                pVisited[i] = true;
```

```
160                 aktwortreihe += pWortliste[i].substr(3);
                    erzeugeElementreihe(pVisited,aktwortreihe);
                  }
                }
              }
165         }

        if(aktwortreihe.length() > wortreihe.length())
          wortreihe = aktwortreihe;
170   }
```

main.cpp

```
  #include <iostream>
  #include <cstdio>
  #include "junior.h"
  using namespace std;
5
  int main(int argc, char** argv) {
    if(argv[1] && argv[2]) {
      long groesse;
      sscanf(argv[2], "%i", &groesse);
10    wortkette test(argv[1],groesse);
    }
    else if(argv[1])
      wortkette test(argv[1]);
    else
15    cout << "Dieses Programm erwartet "
           << endl
           << "    -  als erstes Argument eine Pfadangabe zu einer
               lesbaren Datei"
           << endl
           << "    -  als zweites Argument die Zahl der einzulesenden
               Buchstaben in der Datei (Standardwert: 20000)"
20         << endl;

    return 0;
  }
```