Aufgabe 1: Informationstag

Begriffe

Vorweg kurz etwas zu den jeweiligen Begriffen, wie ich sie in meinem Programm sowie in dieser Dokumentation benutze:

Angebot – eine der 16 Angebote der Eltern

Veranstaltung – ein Angebot, das an einem bestimmten Termin stattfindet (maximal 16 * 4 = 64)

Maximum/Minimum – ist die obere und untere Schranke der Teilnehmeranzahl für ein Angebot bzw. eine Veranstaltung

Termin – einer der 4 Termine, an dem Veranstaltungen stattfinden können

Teilnehmer/Schüler – ein Schüler, der Veranstaltungen besuchen kann

Wünsche – Ein Schüler gibt 6 Wünsche mit den Angeboten ab, die er gerne besuchen möchte. Diese haben immer eine bestimmte Priorität.

Priorität von Teilnehmern/Wünschen – Die Wünsche der Teilnehmer sind nach Priorität geordnet.

Priorität von Veranstaltungen – ergibt sich aus der Summe der Prioritäten der Teilnehmer, die die Veranstaltung besuchen.

Gesamtprioritätswertung – die Summe aller Prioritäten von Veranstaltungen (somit auch die Summer aller Prioritäten der Teilnehmer). Diese gibt Auskunft über die Qualität des erstellen Plans.

1 Kriterien

1.1 Kein Zwang

Das wichtigste Kriterium, ob und wie häufig ein Angebot veranstaltet wird, ist sicherlich die Anzahl der Teilnehmer, die das Angebot als Wunsch angegeben haben. Da es sich um eine mehr oder weniger freiwillige Veranstaltung handelt, kann man wohl kaum Schüler in Veranstaltungen zwingen, wenn sie diese gar nicht besuchen wollen.

Bleibt diese Anzahl noch unter dem Minimum des Angebots, so findet die Veranstaltung mit Sicherheit nicht statt. Ist diese Anzahl größer als das Minimum, so könnte es sein, dass sie stattfindet, darüber müssen dann andere Kriterien entscheiden. Ist die Anzahl sogar größer als das doppelte Minimum des Angebots, so könnte das Angebot sogar zweimal veranstaltet werden, bei dem dreifachen Minimum dreimal und so weiter. Natürlich ist es bei so vielen Anfragen zunächst sinnvoll, eine Veranstaltung vollständig zu füllen. Doch bei einer bestimmten Konstellation kann es evtl. besser sein, Veranstaltungen gerade so bis zum Minimum zu belegen.

Somit ist ein weiteres wichtiges Kriterium, welche Priorität die Schüler dem jeweiligen Angebot gegeben haben. Dabei treten die Veranstaltungen immer in Konkurrenz gegeneinander an. Die Priorität für eine Veranstaltung ergibt sich aus der Summe der Prioritäten, die die Teilnehmer für das Angebot gegeben haben, wobei nur so viele Teilnehmer gewertet werden, die diese Veranstaltung überhaupt besuchen könnten. Das hängt natürlich vor allem direkt vom Maximum der Veranstaltung ab.

Der beste Plan ist dann am Ende der, welcher die größte Gesamtprioritätswertung besitzt. Diese ergibt sich aus der Summe aller Prioritäten der Veranstaltungen, also im Endeffekt alle Prioritäten der erfüllten Wünsche aller Teilnehmer. Dies könnte dann auch das Interesse von Herrn Lämpel sein, der das Ganze ja organisieren muss.

Das Interesse der Teilnehmer liegt natürlich darin, dass sie möglichst ihre hoch priorisierten Wünsche besuchen können. Des weiteren gehe ich davon aus, obwohl es sich um eine Schulveranstaltung handelt, dass die angemeldeten Schüler an so vielen Terminen wie möglich eine Veranstaltung besuchen möchten. Dies kann man erreichen, indem man eine recht große Differenz der Priorität zwischen „Keine Veranstaltung besuchen“ und „Wunsch 6 wird besucht“ wählt.

Die Eltern, die die Angebote anbieten, besitzen im gewissen Sinne überhaupt kein Interesse, sie bieten ihre Veranstaltung an, so oft sie

gehört werden möchte, solange das Minimum bzw. Maximum eingehalten wird.

1.2 Zwang

Um die Gesamtprioritätswertung noch zu erhöhen, kann man allerdings auch noch eine Einschränkung weglassen. Und zwar, indem man die Schüler zwingt, falls sie ihre Wunschangebote nicht besuchen können, eine andere Veranstaltung zu besuchen. Denn in dem Fall tragen sie vielleicht nichts zur Gesamtwertung bei (wenn man die Priorität „Kein Veranstaltung besuchen" mit 0 wertet), doch sie ermöglichen, dass noch mehr Veranstaltungen ihr Minimum erreichen und somit andere Teilnehmer vielleicht das Angebot ihrer Wünsche wahrnehmen können.

Das Hauptinteresse (das von Herrn Lämpel) besteht natürlich weiterhin darin, die Gesamtprioritätswertung zu maximieren.

Die Teilnehmer hätten natürlich immer noch das Interesse möglichst die Wunschangebote besuchen zu können (und natürlich auch lieber die, die oben auf der Prioritätsliste stehen), aber nun darf man von ihnen verlangen auch Angebote zu besuchen, in die sie gar nicht wollen. Des weiteren sollte man die Schüler aber nicht zwingen wollen ein Angebot zweimal zu besuchen, denn ich denke, da endet dann auch die Bereitschaft des allergrößten Strebers.

Doch hier könnte man nun auch verschiedene Interessen für die Veranstalter (Eltern) formulieren. So wollen sie vielleicht so wenig Veranstaltungen wie möglich halten oder aber natürlich eine hohe Anzahl an Teilnehmer, die ihr Angebot auch als Wunsch geäußert haben. Denn eine Veranstaltung, in der 15 Schüler sitzen, aber nur zwei, die es interessiert, ist für keinen Vortragenden angenehm. Ferner könnte man noch das Interesse einbringen, dass die Vorträge, wenn ein Elternteil mehrere davon halten soll, diese zeitlich hintereinander liegen, so dass die Eltern nicht ewig warten müssen.

1.3 Zwang mit Folgen

Eine kleine Abwandlung, die die zweite Variante vielleicht etwas mehr der Realität näher kommen lassen würde, wäre es, wenn ein Schüler eine Veranstaltung unfreiwillig, der Gemeinschaft zuliebe, besuchen muss, dass dies dann negative Prioritätspunkte gibt. Je nachdem, was für einen Wert man wählt, würden solche Unannehmlichkeiten entsprechend selten vorkommen. Wählt man ihn aber zu groß, so landet man wieder bei Variante 1.

1.4 Resultat

Obwohl die zweite Variante sehr interessant ist und viele zusätzlichen Interessen und Bedingungen enthält, halte ich sie für nicht sinnvoll. Doch auch die dritte Variante, die an sich zwischen den ersten beiden liegt, halte ich für zu kompliziert und aufwändig, einfach unter der Tatsache, dass ich ganz im Sinne der Schüler niemanden zwingen möchte.

2 Bedingungen

Die meisten Bedingungen liegen auf der Hand:

1. Jeder Schüler besucht ein Angebot einmal oder gar nicht.
2. Ein Schüler darf pro Termin nur maximal eine Veranstaltung besuchen. (Das dürfte sonst räumlich Probleme bereiten.)
3. Jedes Angebot darf maximal nur einmal pro Termin angeboten werden (Auch Eltern können sich nicht klonen.)
4. Die Anzahl der Teilnehmer jeder Veranstaltung muss das Minimum und das Maximum einhalten.

Und bei der ersten Variante kommt noch hinzu:

5. Kein Schüler besucht eine Veranstaltung, die nicht auf seiner Wunschliste steht.
6. Hierbei kommt natürlich hinzu, dass jeder Schüler alle 6 Wünsche angeben muss. (Ansonsten wird mit „Egal" aufgefüllt.)

Um nicht nur einen möglichen, sondern auch einen sinnvollen Plan zu bekommen, kann man noch folgende Bedingungen hinzunehmen:

7. Es muss eine Mindestanzahl (1 oder größer) an Veranstaltungen erreicht werden. (Ansonsten sollte Herr Lämpel noch einmal mit den Eltern über ihre angegebenen Minima reden oder noch mehr Schüler mit einbeziehen.)
8. Die Gesamtprioritätswertung soll möglichst maximal sein. (Genau dies ist natürlich die Aufgabe des Programms, dass jetzt in Teil 3 beschrieben wird).

3 Programm

3.1 Lösungsidee

Mein Ziel war es, das Programm möglichst übersichtlich zu gestalten und eine Trennung zwischen Benutzeroberfläche und dem eigentlichen Programm zum Berechnen des Planes zu haben. Das eigentliche Programm habe ich deshalb in folgenden Komponenten unterteilt: Informationstag, Angebote, Veranstaltung und Teilnehmer.

Des weiteren wollte ich ein möglichst leistungsfähiges Programm, welches in vielen unterschiedlichen Situationen eingesetzt werden kann, weshalb ich darauf geachtet habe, dass man so gut wie alle Parameter auch während der Laufzeit ändern kann. Zum Einsatz des Programms auch auf andere Situationen gibt es im Ablaufprotokoll mehr.

3.1.1 Der Informationstag

Der Informationstag muss dabei alles verwalten. Er bekommt die Angebote und die Teilnehmer geliefert und muss dann in einem Algorithmus mit Hilfe des GLPK einen möglichst optimalen Plan erstellen, das heißt, er erstellt an den entsprechenden Terminen Veranstaltungen und weist diesen dann die Schüler, die die Veranstaltung besuchen sollen, zu. Im Endeffekt hatte ich zwei Ideen um diesen Plan zu erstellen. Zunächst einen optimalen Plan, der wirklich das Maximum der Gesamtprioritätswertung ermittelt. Zum anderen einen nicht ganz optimalen Plan, dafür aber einen, der deutlich schneller ermittelt werden kann, doch dazu später mehr.

Entscheidend sind noch zwei Werte: Und zwar zum einen die Anzahl der Termine, also wie oft ein Angebot angeboten werden kann und wie viele Veranstaltungen die Schüler besuchen können, dies ist im konkreten Beispiel des Berufsinformationstages 4, zum anderen die Anzahl der Wünsche, die die Schüler angeben müssen, was natürlich auch einen großen Einfluss auf den Plan hat, im Beispiel sind das 6.

Darüber hinaus muss noch die Prioritätszahlen für die Wünsche festgelegt werden. Dabei sollten Wünsche mit hoher Priorität eine größere Zahl besitzen als Wünsche mit niedriger Priorität und alle sollten größer als 0 sein. Hier kann man nun noch relativ viel Einfluss nehmen, indem man zum Beispiel sagt, ich möchte, dass die Ersatzwünsche nur wirklich im Notfall eingesetzt werden, dann kann

man zum Beispiel den Wünschen 5 und 6 nur den Wert 1 verpassen, wohingegen Wunsch 1 bis 4 all einen Wert über 10 besitzen. Ebenso könnte man einen Hauptwunsch festlegen, der so gut wie bei jedem Schüler, wenn irgendwie möglich, erfüllt sein sollte. Dann wählt man einfach für den ersten Wunsch einen Wert von über 100.

Um ein möglichst leistungsfähiges Programm zu erhalten, sollten alle Parameter (auch die Anzahl der Termine und Wünsche) während der Laufzeit verändert werden können.

3.1.2 Die Angebote

Die Angebote sind, wie in den Begriffen schon erklärt, die verschiedenen Angebote, die von den Eltern vorgestellt werden. Sie geben also zum einen das Thema vor (also welcher Beruf vorgestellt wird). Dies ist natürlich für die Berechnung völlig egal. Zum anderen ist mit ihnen auch immer ein Minimum und ein Maximum verknüpft, welche angeben, wie viel Schüler in einer Veranstaltung sein dürfen. Dies hat natürlich sehr großen Einfluss auf die Berechnung und das Ergebnis. Dabei gilt: Je größer das Minimum, desto weniger Veranstaltungen, doch auch desto einfacher bzw. schneller kann ein Plan erstellt werden. Angebote soll es in unserem Beispiel 16 geben, doch natürlich sollte die Anzahl flexibel bleiben. Dann ist natürlich darauf zu achten, dass, wenn man den Schülern zu viele Angebote gibt, es wieder schwierig wird für die einzelnen Veranstaltungen das Minimum zu erreichen, da sich die Wünsche der Teilnehmer verteilen könnten.

Die Angebote stellen also hauptsächlich Vorlagen für Veranstaltungen dar. Des weiteren sind sie natürlich auch Vorlage für die Wünsche der Schüler, denn diese wählen nur Angebote und keine Veranstaltungen, in diese werden sie lediglich eingeteilt.

3.1.3 Die Veranstaltungen

Veranstaltungen sind Angebote, die zu einem bestimmten Termin stattfinden und denen Schüler zugeordnet sind. Sie besitzen also immer ein Angebot als Vorlage sowie einen Termin, an dem sie stattfinden und Teilnehmer die daran teilnehmen. Damit eine Veranstaltung gültig ist, muss die Teilnehmeranzahl sich zwischen dem Minimum und dem Maximum der Vorlage befinden.

Darüber hinaus darf es pro Angebot nur eine Veranstaltung pro Termin geben.

3.1.4 Die Teilnehmer

Teilnehmer besitzen neben einem Name vor allem Wünsche von Angeboten, die sie gerne besuchen würden, und zwar so viele, wie der Informationstag vorschreibt. Diese Wünsche sind geordnet nach ihrer Priorität. Teilnehmer können einer Veranstaltung zugeordnet werden, dürfen aber natürlich nur eine Veranstaltung pro Termin besuchen.

3.1.5 Die Oberfläche

Bei der Oberfläche ist es wichtig, dass man übersichtlich alle Parameter einstellen kann. So sollte man zunächst die Anzahl der Termine und der Wünsche einstellen können. Danach kann man nacheinander die Angebote hinzufügen und jeweils die Thematik sowie natürlich das Minimum und Maximum festlegen. Um das für Testzwecke etwas abzukürzen, ist es vielleicht sinnvoll direkt eine Vorlage von Angebote laden zu können.

Als Nächstes sind die Teilnehmer an der Reihe. Neben dem Namen müssen hier natürlich die Wünsche festgelegt werden, wichtig ist, dass kein Angebot doppelt gewählt werden darf. Auch hier ist es zwar vielleicht nicht für den normalen Gebrauch, aber zum Testen auf jeden Fall sinnvoll zufällig Teilnehmer erstellen zu können. Am Besten gleich mehrere auf einmal.

Als letzte Einstellung sollte man nun noch Werte den Prioritäten zu weisen. Je nachdem wie gewichtig ein Wunsch sein soll, bekommt er einen anderen Wert.

Danach muss man nur noch den Plan berechnen lassen, wobei man die Art des Algorithmus' auswählen können sollte. Da dies länger dauern könnte, soll das Ganze im Hintergrund ablaufen, so dass man, wenn man möchte, schon für den nächsten Plan neue Einstellungen vornehmen kann.

Ist der Plan berechnet, sollte dieser natürlich noch sinnvoll ausgegeben werden. Wobei eine Auflistung unterteilt in zunächst Termine, dann Veranstaltungen und zuletzt Teilnehmer sinnvoll wäre.

3.1.6 Der optimale Algorithmus

Hierbei gilt es unter den gegebenen Einstellungen wirklich das Maximum von der Gesamtprioritätswertung zu bestimmen. Da es sich um eine lineare Optimierung handelt, kommt hier das GLPK zum Einsatz. Doch dafür muss man das Problem zunächst formulieren. Das heißt, es muss in **Zielfunktionen** und Bedingungen eingeteilt werden. Dabei ergibt sich dann folgendes:

Dafür stellt man zunächst lauter binärer Variablen $\mathbf{B_{SAT}}$ auf, die darüber Auskunft geben, ob ein Teilnehmer/Schüler **S** ein bestimmtes Angebot **A** an einem bestimmten Termin **T** besucht oder nicht (Es gilt also Anzahl Teilnehmer * Anzahl Angebote * Anzahl Termine = Anzahl binärer Variablen).

Für die Zielfunktion **Z** braucht man dann noch jeweils Koeffizienten $\mathbf{P_{SA}}$ für alle $\mathbf{B_{SAT}}$, dieser entspricht dem Prioritätswert für die Priorität, die der Schüler **S** dem Angebot **A** gegeben hat.

Die Anzahl der Schüler, Angebote und Termine sind mit **Smax**, **Amax** und **Tmax** gekennzeichnet.

Die Zielfuntion gibt somit Gesamtprioritätswertung an, welche es zu maximieren gilt.

$$Z = B_{111} \cdot P_{11} + B_{112} \cdot P_{11} + \ldots + B_{121} \cdot P_{12} + \ldots + B_{((Smax)(Amax)(Tmax))} \cdot P_{((Smax)(Amax))}$$

Die Aufgabe des Algorithmus' ist es nun also die optimale Konstellation aller binären Variablen herauszubekommen, so dass Z maximal ist. Dabei gibt es natürlich allerlei Bedingungen:

Zunächst muss sichergestellt werden, dass kein Schüler ein Angebot mehrfach besucht. Dafür benutzen wir die Hilfsvariable $\mathbf{H1_{SA}}$. Es muss also für jeden Teilnehmer zu jedem Angebot überprüft werden, ob er es nur einmal besucht. Das heißt, die Anzahl der **H1** ergibt sich aus **Tmax** mal **Amax**. Es ergibt sich folgende Gleichung:

$$H1_{SA} = B_{SA1} + B_{SA2} + \ldots B_{(SA(Tmax))}$$

Da jeder Schüler jede Veranstaltung nur entweder einmal oder gar nicht besuchen darf, gilt: $0 \leq H1_{SA} \leq 1$

Als Nächstes soll getestet werden, ob die Veranstaltung das Minimum bzw. das Maximum einhalten. $\mathbf{H2_{AT}}$ ist abhängig von dem Angebot und dem Termin (also entsprechend der möglichen Veranstaltung). Dabei sollen alle Schüler zusammen summiert werden, die zu dem Termin in das Angebot gehen, also die Veranstaltung besuchen.

$$H2_{AT} = B_{1\mathrm{AT}} + B_{2\mathrm{AT}} + \ldots + B_{((Smax)BT)}$$

Hierbei gilt: $MIN_A \leq H2_{AT} \leq MAX_A$ ODER $H2_{AT} = 0$ und zwar immer dann, wenn die Veranstaltung gar nicht stattfindet. Hierbei ist nun schon inbegriffen, dass maximal eine Veranstaltung eines Angebots pro Termin stattfinden kann. Denn es werden ja alle Teilnehmer, die das Angebot an dem Termin besuchen, gezählt. Mehrere Angebote an

einem Termin würden somit das Maximum überschreiten.

Als Letztes muss noch getestet werden, ob alle Teilnehmer pro Termin nur eine Veranstaltung besuchen. Dafür müssen nun die Veranstaltungen, die ein Schüler an einem Termin besucht summiert werden. **$H3_{ST}$** ist somit von dem Schüler und dem Termin abhängig.

$$H3_{ST} = B_{S1T} + B_{S2T} + \ldots + B_{(S(Amax)T)}$$

Auch hier gilt, der Schüler kann maximal ein Angebot pro Termin wahrnehmen: $0 \leq H3_{ST} \leq 1$

Nun sind genug Bedingungen gestellt, so dass immer gültige Pläne entstehen. Lässt man nun das GPLK die Zielfunktion maximieren, kann man anhand der **B_{SAT}** auslesen, welcher Schüler zu welchem Angebot an welchem Termin hingeht.

Doch leider handelt es sich dabei schon um eine Matrix, die **Smax** * **Amax** * **Tmax** Spalten und **Smax** * **Amax** + **Amax** * **Tmax** + **Smax** * **Tmax** Zeilen besitzt.

Deshalb habe ich noch einen anderen Algorithmus entworfen, der dem GPLK deutlich weniger Arbeit gibt, dafür aber nicht die optimale Lösung ermittelt.

3.1.7 Der „schnelle" Algorithmus

Dieser Algorithmus besteht aus mehreren Teilen. Zunächst wird bestimmt, welche Veranstaltungen unabhängig vom Termin überhaupt rein theoretisch stattfinden könnten. Dafür werden die Teilnehmer gezählt, die das Angebot als Wunsch angegeben haben. Überschreitet die Zahl das Minimum, so wird schon mal eine Veranstaltung angelegt (ob sie auch stattfindet, wird erst später entschieden). Wird das Minimum sogar mehrfach erreicht, werden auch mehrere Veranstaltungen angelegt, maximal natürlich so viele, wie es Termine gibt.

Im nächsten Schritt werden den Veranstaltungen schon mal Teilnehmer lose zugeordnet, wobei immer noch nicht auf die Termine geachtet wird. Bedingung ist nur, dass ein Schüler nicht mehr Veranstaltungen besucht, als es Termine gibt. Dabei wird in diesem Schritt schon erstmals die Gesamtprioritätswertung mit Hilfe des GLPK maximiert. Die Zielfunktion sieht dabei so aus:

$$Z1 = B_{11} \cdot P_{11} + B_{12} \cdot P_{12} + \ldots + B_{21} \cdot P_{21} + \ldots + B_{((Smax)(Vmax))} \cdot P_{((Smax)(Amax))}$$

Wie man sieht, sind die binären Variablen diesmal nur von den Veranstaltungen und den Teilnehmer abhängig. Es kann zwar bei vielen Schülern mehr Veranstaltungen als Angebote geben, aber im Falle, dass ein Angebot gar nicht genügend Interessenten besitzt, als dass es angeboten werden könnte, so kommt dieses nun auch nicht bei den Veranstaltungen vor. Für die Priorität muss natürlich zu jeder Veranstaltung die Priorität des jeweiligen Angebots verwendet werden.

Doch auch die Bedingungen sind um Einiges abgespeckt. Hier muss zwar auch getestet werden, ob jeder Teilnehmer nur einmal eine Veranstaltung eines Typs (Angebot) besucht.

$$H1_{SA} = B_{S1} + B_{S2} + \ldots + B_{(S(Vmax))}$$

Wobei nur die **B**'s gezählt werden, bei der die Veranstaltung V dem Angebot A entspricht.

$H2_V$ ist nun nicht mehr von den Terminen beeinflusst. Doch auch hier muss getestet werden, ob jede Veranstaltung das Minimum bzw. Maximum einhält.

$$H2_V = B_{1V} + B_{2V} + \ldots + B_{((Smax)V)}$$

Und als allerletztes muss mit **$H3_S$** getestet werden, ob jeder Teilnehmer auch nur maximal eine Veranstaltung pro Termin besucht.

$$H3_S = B_{S1} + B_{S2} + \ldots + B_{(S(Vmax))}$$

Da keine Unterteilung in die einzelnen Termine stattfindet, gilt hier diesmal: $0 \leq H3_S \leq Tmax$

Der Schüler darf also maximal so viele Veranstaltungen besuchen, wie es Termine gibt.

Lässt man die Zielfunktion **Z1** mit dem GLPK maximieren, erhält man das Maximum der Gesamtprioritätswertung für den Fall, dass es ausreichend Termine gibt und die Schüler nur maximal 4 Veranstaltungen besuchen dürfen.

Diese Berechnung ist relativ einfach, da häufig die bestpriorisierten Wünsche erfüllt werden. Nur wenn es mal bei dem Angebot nicht für die Minimum-Hürde reicht, müssen die Teilnehmer auf andere Veranstaltungen ausweichen.

Doch nun im zweiten Schritt müssen die Veranstaltungen so auf die

Termine verteilt werden, dass auch weiterhin die Gesamtprioritätswertung maximal wird. Nur diesmal steht es schon fest, welche Schüler welche Veranstaltung besuchen.

Dabei kommen nun die Prioritäten der Veranstaltungen (also die Summe der Werte von den Prioritäten aller Teilnehmer der Veranstaltung) zum Einsatz.

Bei den binären Variablen haben wir nun nur alle Veranstaltungen an jedem Termin aufgeführt. Die Koeffizienten ergeben sich aber diesmal aus der Priorität der Veranstaltungen.

Z2 sieht also wie folgt aus:

$$Z2 = B_{11} \cdot P_1 + B_{12} \cdot P_1 + \ldots + B_{21} \cdot P_2 + \ldots + B_{((Vmax)(Tmax))} \cdot P_{((Vmax))}$$

Die Bedingung ist diesmal einfach nur, dass jeder Schüler maximal eine Veranstaltung pro Termin besucht, wobei diesmal die Veranstaltungen die Teilnehmer repräsentieren.

$$H1_{ST} = B_{1\mathrm{T}} + B_{2\mathrm{T}} + \ldots + B_{((Vmax)T)}$$

Wobei diesmal alle $\mathbf{B_{VT}}$ gezählt werden, sobald der Schüler **S** die Veranstaltung **V** besucht. Es gilt nun wieder: $0 \leq H1_{ST} \leq 1$

Die zweite Bedingung ist, dass jede Veranstaltung maximal an einem der Termine stattfindet, denn ansonsten würden genau die gleichen Schüler zweimal in die gleiche Vorstellung gehen, nur an einem anderen Termin, und das dürften sie nicht so toll finden.

$$H2_V = B_{V1} + B_{V2} + \ldots + B_{(V(Tmax))}$$

Auch hier gilt: $0 \leq H2_{ST} \leq 1$

Nachdem man nun auch Zielfunktion 2 maximiert hat, besitzt man einen gültigen Plan, doch häufig auch nicht mehr als 4 oder 5 Veranstaltungen, denn, als die Schüler den Veranstaltungen zugewiesen wurden, wurde überhaupt nicht auf die Termine geachtet, was natürlich zu extremen Einschränkungen für den zweiten Schritt führt. Man hat zwar jetzt ein paar Veranstaltungen heraus gesucht, die viele Punkte für die Gesamtprioritätswertung bringen, doch es sind halt noch zu wenig.

Außerdem gibt es sicherlich viele Schüler, die nur eine oder gar keine Veranstaltung besuchen. Es ist also noch durchaus Potenzial da. Und deshalb läuft der Algorithmus einfach nochmal von vorne durch, nur

dass diesmal alle Wünsche der Schüler herausgenommen werden, die schon erfüllt werden, alle Termine an denen Teilnehmer schon eine Veranstaltung besuchen, stehen diesen nicht mehr zur Verfügung, und auch alle Termine, an denen schon eine Veranstaltung stattfindet, ist für dieses Angebot nicht mehr möglich.

Auch die Anzahl der Veranstaltungen, die ein Teilnehmer im ersten Teil des Algorithmus' belegen kann, wird gekürzt um die Anzahl der Veranstaltungen, die er schon sicher besucht.

Da nun wesentlich mehr Einschränkungen dazugekommen sind und auch einige binäre Variablen weggefallen sind, geht der zweite Durchgang deutlich schneller als der erste und das gilt auch für die folgenden Durchgänge.

Das Ganze wird so oft rekursiv wiederholt, bis keine Veranstaltung mehr dazugekommen ist, es ein Durchgang also nichts Neues gebracht hat. Dann kann das Ganze beendet werden und die Veranstaltungen stehen fest.

3.2 Programm-Dokumentation

3.2.1 Überblick

Das Programm habe ich in C++ geschrieben und für eine schöne Benutzeroberfläche sorgt die Qt4-Bibliothek. Für die Berechnung kommt das schon mehrfach erwähnte GNU Linear Programming Kit (GLPK) zum Einsatz.

ACHTUNG: Damit das Programm einwandfrei funktioniert, wird die **libglpk.so.0.12.0** (**GLPK 4.27**) benötigt. Bei älteren Versionen kann es zu einer Fehlermeldung kommen (glp_load_matrix: ar[2] = 0; zero element not allowed).

Kompiliert ist es unter Kubuntu 7.10 Gutsy Gibbon (KDE) in Eclipse mit dem CDT-Plugin (gcc-Compiler).

Als Entwicklungsumgebung habe ich neben Eclipse auch den Qt-4-Designer für das Erstellen der Fenster verwendet.

Das Programm besteht insgesamt aus sechs Klassen. Zwei davon sind nur für die Oberfläche verantwortlich, die anderen vier bilden das Modell für den Informationstag.

Die **main** öffnet dabei nur das Hauptfenster.

QtInformationstag verwaltet dieses Fenster und reagiert auf Eingaben des Benutzers und sorgt für eine schöne Darstellung des Ergebnisses. Es handelt sich aber hauptsächlich um einen Vermittler, der die Anfragen an cInformationstag weiterleitet bzw. die Ergebnisse von dort ausliest.

cInformationstag ist nämlich für die Verwaltung der Angebote und Teilnehmer sowie dem Erstellen eines Plans in Form von Veranstaltungen zuständig.

cAngebot dient als Vorlage für Angebote, die ein Minimum und Maximum sowie einen String für den Namen besitzen.

cTeilnehmer besitzt neben dem Namen die 6 Wünsche (bzw. so viele, wie man einstellt).

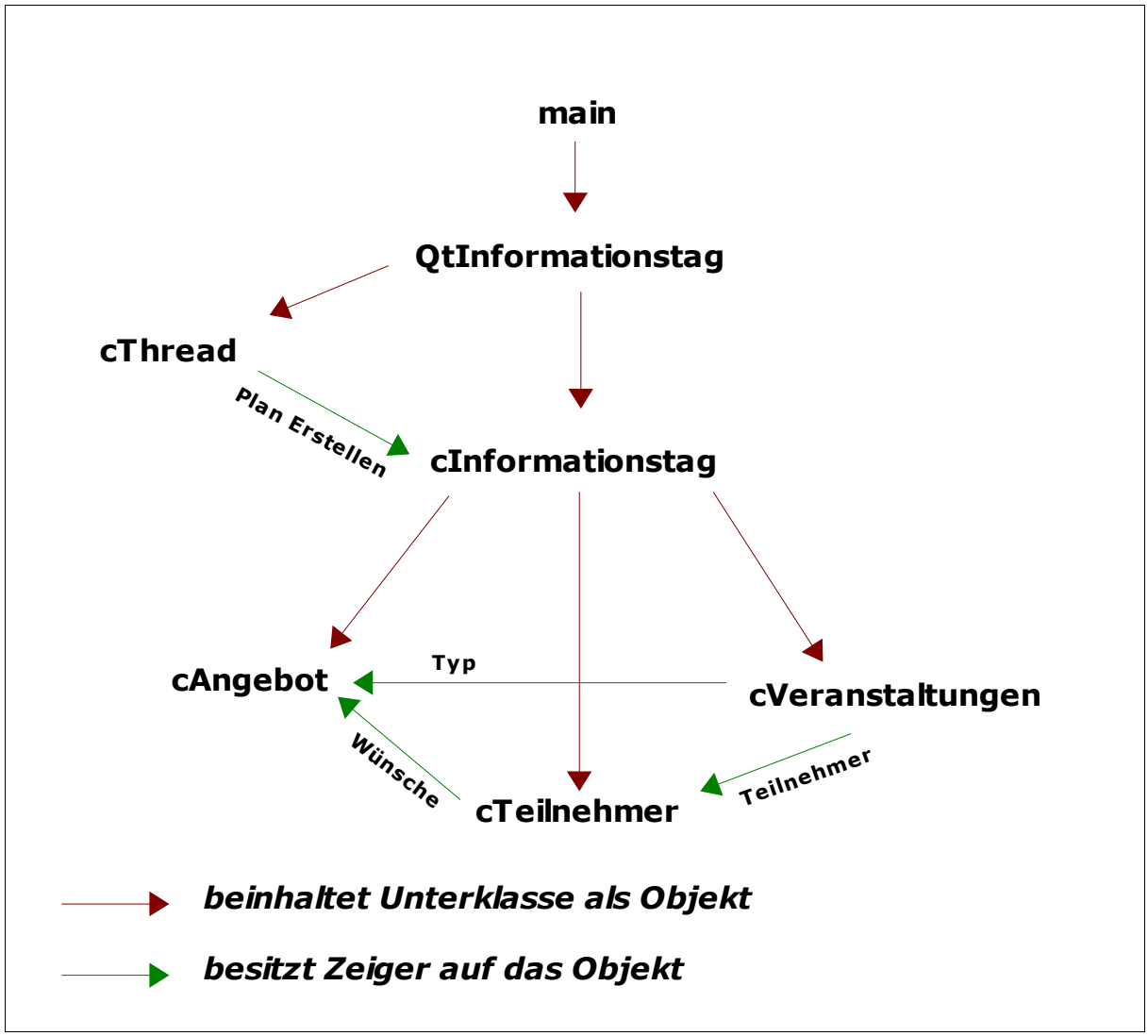
cVeranstaltung besitzt neben dem Typ (cAngebot) Zeiger auf Teilnehmer, die die Veranstaltung besuchen.

Darüber hinaus besitzen cAngebot, cTeilnehmer und cVeranstaltung eine ID, so dass die Objekte alle klar von einander unterschieden werden können. (Denn zwei Schüler mit gleichem Namen können schon mal vorkommen.)

Zusätzlich musste ich noch eine kleine Veränderung von der QtKlasse

QThread vornehmen und habe die zu **cThread** vererbt. Dies ist nur notwendig, um einen neuen Thread aufzumachen, damit der Plan im Hintergrund berechnet werden kann und dabei nicht das ganze Programm zusammenbricht. Die Klasse leitet dann die „Plan Erstellen"-Aufgabe an cInformationstag weiter.

Die Verknüpfung der einzelnen Klassen sieht dabei wie folgt aus:



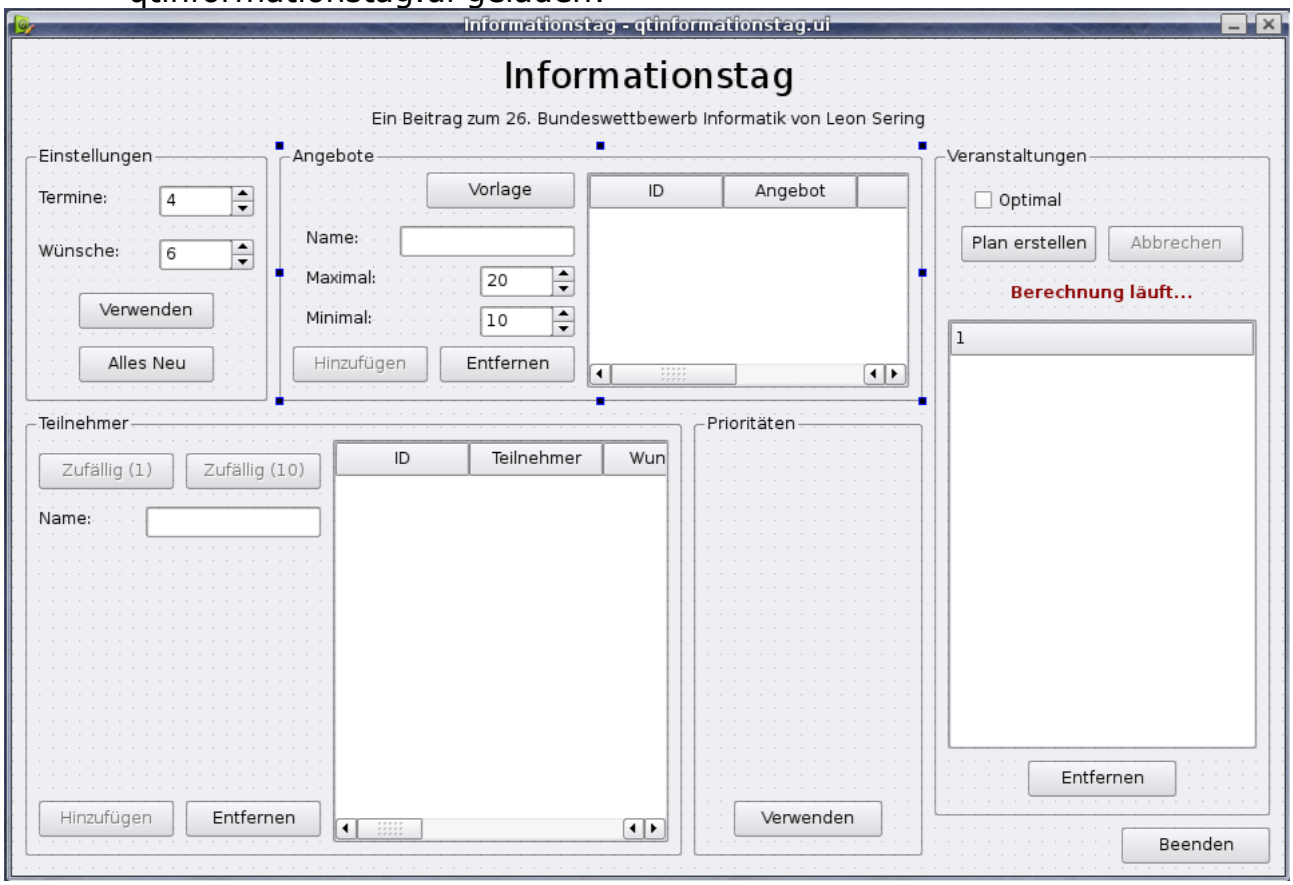
Man erkennt, dass die Struktur an sich streng hierarchisch aufgebaut ist und doch die einzelnen Komponenten untereinander kommunizieren.

3.2.2 QtInformationstag

Die Klasse wird von der main-Funktion aufgerufen und stellt die Schnittstelle zwischen Benutzer und Programm da.

Dafür wird zunächst das mit Qt-Designer erstellte Fenster aus der

qtinformationstag.ui geladen:



Neben dem Titel und dem „Beenden"-Button, ist das Fenster in 5 Bereiche eingeteilt.

Zunächst sind da die **Haupteinstellungen**, bei denen man die Anzahl der Termine und der Wünsche einstellen kann. Zweiteres ist dabei immer größer oder gleich der Anzahl der Termine. Man übernimmt die Einstellungen, wenn man auf „Verwenden" klickt. Mit „Alles Neu" setzt man alle Parameter (bis auf die Haupteinstellungen) wieder auf die Anfangswerte bzw. löscht die Listen.

Im Bereich der **Angebote** kann man neue Angebote hinzufügen. Dafür gibt man einen Namen ein und stellt die Maximal und Minimalzahl auf den gewünschten Wert. Mit „Hinzufügen" wird das Angebot in die Liste rechts daneben aufgenommen und steht nun den Teilnehmern zur Verfügung. Mit „Vorlage", kann man direkt 16 Angebote auf einmal laden. Mit „Entfernen" löscht man die gesamt Liste. Da dann auch die Liste der Teilnehmer nicht mehr aktuell ist, wird diese ebenfalls gelöscht.

Im Bereich **Teilnehmer** sieht es ähnlich aus, auch hier muss man einen Namen eingeben. Darunter wird dann je nach Anzahl der Wünsche eine Liste mit Comboboxen erstellt, mit denen man die Wünsche einstellen kann, um ihn danach mit „Hinzufügen" zur Liste tun kann.

Bei **Prioritäten** kann man den Wert für die jeweilige Priorität einstellen. Wenn man die Anzahl der Wünsche eingestellt hat, erscheint hier diese Anzahl an SpinBoxen, mit denen man ihnen den Wert geben kann.

Bei **Veranstaltungen** kommen wir nun zum entscheidenden Bereich, denn hier kann man, nachdem man mit der CheckBox „Optimal" ausgewählt hat, welchen Algorithmus man verwenden möchte, mit einem Klick auf „Plan Erstellen" die Berechnung beginnen. Ist diese fertig, wird sie in dem Feld darunter, schön übersichtlich nach Termin, Angebot und Teilnehmer sortiert, ausgegeben.

Dauert einem das Berechnen zu lange, so kann man mit „Abbrechen" diesen stoppen.

Achtung: Da dann der Vorgang einfach unterbrochen wird, kann es hier zu Fehlern kommen und manchmal schließt sich das Programm einfach.

Mit **Beenden** beendet man das Programm, wie der Name schon sagt.

Zum **Quelltext** der QtInformationstag-Klasse:

Im Konstruktor wird zunächst das Fenster noch zu Ende eingerichtet. Und ein paar Signale mit Slots verbunden. Vor allem werden ScrollAreas für die Wünsche und die Prioritäten eingerichtet, so dass der Benutzer unabhängig von der Anzahl der Wünsche an alle Einstellungen herankommt. Außerdem werden die Spalten für die Listen richtig eingestellt.

Der Rest der Klasse besteht hauptsächlich aus Slots, die auf verschiedene Ereignisse reagieren. Sie sorgen dafür, dass immer das Richtige passiert, wenn man einen Button betätigt, und kümmern sich darum, dass der Benutzer so gut wie keine falschen Eingaben machen kann. Sind zum Beispiel bei den Wünschen zweimal das gleiche Angebot eingetragen, so werden diese Rot eingefärbt und der entsprechende „Hinzufügen"-Button wird deaktiviert. Oder aber es wird Sorge dafür getragen, dass das Minimum eines Angebots auch wirklich immer kleiner ist als das Maximum bleibt.

Der Slot, der aufgerufen wird, wenn man den „Plan Erstellen"-Button klickt, leitet diese Aufgabe einfach nur an das cThread Objekt weiter, damit dieses das Problem in einem neuen Thread verarbeiten kann.

Drückt man den „Zufällig (1)"-Button, so wird per Zufall ein Teilnehmer eingestellt mit dem Namen „Schüler *X*". Man muss diesen dann nur noch mit „Hinzufügen" bestätigen. Mit „Zufällig (10)" werden direkt 10 Schüler, die der Reihe nach durchnummeriert sind, erstellt. Hier braucht man nicht mehr bestätigen.

3.2.3 cInformationstag

Diese Klasse ist relativ übersichtlich und beinhaltet nur ein paar wichtige Methoden:

Im **Konstruktor** wird alles einfach initialisiert. Dabei bekommen die Prioritäten schon mal die Standardwerte, welche einfach von 10 + Anzahl der Werte bis 11 herunter gehen.

Zusätzlich muss sich für den zweiten Plan-Erstellen-Algorithmus ja immer gemerkt werden, welcher Teilnehmer schon welchen Wunsch erfüllt wurde und an welchem Termin die Teilnehmer bzw. Angebote schon belegt sind. Diese ganzen Informationen werden in **TeilWue**, **TeilTer** und **AngTer** gespeichert und zu Beginn natürlich alles auf *false* gesetzt.

Mit den **Hinzufügen**-Methoden kann man jeweils Angebote, Teilnehmer oder auch Veranstaltungen hinzufügen, wobei letzteres nur von den Plan-Erstellen-Methoden erfolgt.

Mit den **Löschen**-Methoden kann man, wie der Name sagt, alles wieder löschen, wobei, wenn man die Angebote löscht, sowohl die Teilnehmer als auch die Veranstaltungen direkt mitgelöscht werden. Wenn man nur die Teilnehmer löschen möchte, löscht das auch die Veranstaltungen, weil sie ohne die Teilnehmer, die sie besuchen, keinen Sinn ergeben.

Ausgeben() ist für die Ausgabe aller Angebote, Teilnehmer und Veranstaltungen über die Konsole zuständig. Diese Methode wird im finalen Programm nicht mehr genutzt.

Mit **TeilnehmerZuweisen()** kann man einen Teilnehmer einer Veranstaltung zu weisen, indem man den Teilnehmer das Angebot und den Termin angibt. Existiert die Veranstaltung noch gar nicht, so wird sie erstellt. Dies ist praktisch, da die Plan-Erstellen-Methoden die Teilnehmer nacheinander auslesen.

Die **PlanErstellen**-Methode ist der Zugriff für cThread, um den Plan zu erstellen. Diese nimmt allerdings nur die Anfrage entgegen und leitet sie je nach Wert der übergebenen Bool-Variabel, an PlanErstellenOptimal() oder PlanErstellenSchnell() weiter.

PlanErstellenOptimal() liest zunächst alle relevanten Informationen aus den Angeboten und den Teilnehmern und speichert diese in Arrays ab.

Daraufhin muss nun das Problem formuliert werden, so wie es in den Lösungsideen beschrieben ist.

Zunächst werden dafür die Zeilen, also die Beschränkungen bzw. Bedingungen, formuliert. Für alle H1, H2 und H3 werden neue Zeilen

angelegt und diese jeweils von beiden Seiten begrenzt, ganz analog zur Lösungsidee. Ein kleines Problem gibt es dabei, denn für H2 sollen ja alle Werte die größer als das entsprechenden Minimum und kleiner als das entsprechende Maximum sein ODER 0. Dies kann man leider nicht so einstellen, weshalb ich auf einen kleinen Trick zurückgreifen musste. Später habe ich die HilfsBools, also zusätzliche Binäre Variablen, eingefügt, und zwar für jedes Angebot pro Termin eine. Ist diese auf *true bzw.* 1, so bedeutet das, dass das Angebot an dem Termin nicht stattfindet und die Anzahl der Teilnehmer, also die entsprechende H2 Variable, wird direkt bis zum Maximum gefüllt, so dass keine weiteren Teilnehmer mehr dort hinein können. Dann entspricht das Maximum sozusagen der 0.

Nachdem nun alle Hilfsvariablen bzw. Zeilen eingefügt wurden, werden nun die Spalten, also die binären Variablen eingefügt. Die HauptBools sind dabei, wie in der Lösungsidee beschrieben, die Variablen, die angeben, ob Teilnehmer S zu Angebot A an Termin T geht oder nicht. Die HilfsBools dienen wie oben beschrieben nur dem Umgehen des Problems mit den H2 Zeilen.

Sind die Zeilen gefüllt und alle als binär erklärt, muss nun noch die Matrix ausgefüllt werden. Denn in der Matrix steht, welche **B**s (Spalten) für welche **H**s (Zeilen) gewertet werden sollen. Das ist hier relativ einfach, denn immer, wenn die Indizes übereinstimmen, kommt eine 1 in die Matrix, ansonsten eine 0. Das GLPK benötigt die Matrix allerdings in einem eindimensionalen Array, weshalb ich den Index dazu immer ausrechnen muss und zusätzlich die Zeile in die Spalte in einem extra Array speichere. Zusätzlich werden den Spalten noch Koeffizienten für die Zielfunktion zugewiesen. Diese entsprechen immer dem Wert für die Priorität, die der Schüler dem jeweiligen Angebot bei seinen Wünschen gegeben hat.

Danach wird die Matrix an das Problem übergeben und dieses gelöst. Dies passiert in zwei Schritten, zunächst mit dem Simplex-Verfahren, welches nicht berücksichtigt, dass es sich bei den gesuchten Variablen um binäre handelt. Doch dies dient als Grundlage für die Branch-And-Cut-Methode, die danach das Problem so löst, wie ich es haben möchte, und nur binäre Werte zulässt.

Als Letztes muss dann nur noch das Ergebnis ausgelesen werden, wobei einfach die HauptBools der Reihe nach geprüft werden, und sollte es eine 1 sein, so wird der Teilnehmer an dem entsprechenden Termin zur Veranstaltung hinzugefügt.

Ganz zum Schluss wird noch das maximierte Ergebnis, also die Gesamtprioritätswertung, in die Konsole ausgegeben.

PlanErstellenSchnell() hat einen ähnlichen Aufbau, muss aber insgesamt mehr Vorarbeit leisten.

Auch hier werden zunächst die Teilnehmer und die Angebote ausgelesen, doch jetzt muss direkt berücksichtigt werden, dass ja bestimmte Termine der Teilnehmer oder der Angebote schon belegt sein könnten (im 2. Durchgang oder danach).

Als Nächstes wird erstmal durch Zählen der Wünsche ermittelt, welches Angebot wie oft veranstaltet werden kann. Dabei wird sich noch gemerkt, wie viele Veranstaltungen es pro Angebot gibt, was für spätere Schleifen sehr wichtig ist.

Danach geht es aber wie bei PlanErstellenOptimal() weiter und zwar werden nun zunächst die Bedingungen (die **H**s bzw. Zeilen) erstellt. Hier gibt es zwar auch drei verschiedene Typen, doch insgesamt sind es wesentlich weniger Zeilen. Das Problem mit der Anzahl der Teilnehmer in einer Veranstaltung wird mit dem gleichen Trick (mit HilfsBools) gelöst wie bei dem Optimal-Verfahren.

Sind die Zeilen gefüllt, kommen die Spalten an die Reihe, doch diesmal sind es nur alle Teilnehmer pro Veranstaltung und die Termine fallen erstmal raus. Doch auch die HilfsBools für den Trick sind wieder mit von der Partie. Die Koeffizienten entsprechen dabei wieder dem Wert für die Priorität des jeweiligen Wunsches.

Ist die Matrix am Ende noch gefüllt, kann es mit der ersten Berechnung los gehen.

Das Lösen des ersten Problems geht vergleichsweise ziemlich schnell, da ja nicht auf das Problem mit den Terminen eingegangen werden muss.

Nun wird als Nächstes das Ergebnis des ersten Problems ausgelesen, woraufhin dann die Prioritäten für die Veranstaltungen aus der Summe der Prioritäten der entsprechenden Teilnehmer berechnet werden kann.

Für das zweite Problem wird außerdem die Information benötigt, welche Teilnehmer welche Veranstaltungen besuchen könnten.

Danach geht es mit dem zweiten Problem weiter. Dafür werden wieder zunächst die Zeilen geladen, wobei es diesmal nur zwei Bedingungsgruppen gibt, nämlich zum Einen, dass kein Teilnehmer zwei Veranstaltungen an einem Termin belegt, und zum Anderen, dass keine Veranstaltung zweimal stattfindet.

Auch bei den Spalten nimmt es seinen gewohnten Gang, diesmal werden keine HilfsBools benötigt und die HauptBools geben an, welche Veranstaltung an welchem Termin stattfindet, dazu werden als Koeffizienten die Prioritäten der Veranstaltungen angegeben.

Bei der Matrix fließen nun die Ergebnisse aus dem ersten Problem ein, denn je nachdem, welche Teilnehmer welche Veranstaltung besuchen, müssen andere Spalten Einfluss auf die Zeilen haben.

Ist die Matrix geladen, wird das Problem wieder von GLPK gelöst und die Zielfunktion maximiert.

Nun wird noch ausgelesen, welche Veranstaltung an welchem Termin stattfinden soll, und mit Hilfe der Ergebnisse aus dem ersten Problem kann auch bestimmt werden, welcher Teilnehmer welcher Veranstaltung angehört. Diese werden dann direkt hinzugefügt und zusätzlich werden die Arrays, die sich merken, welche Teilnehmer an welchen Terminen schon verhindert sind usw., aktualisiert.

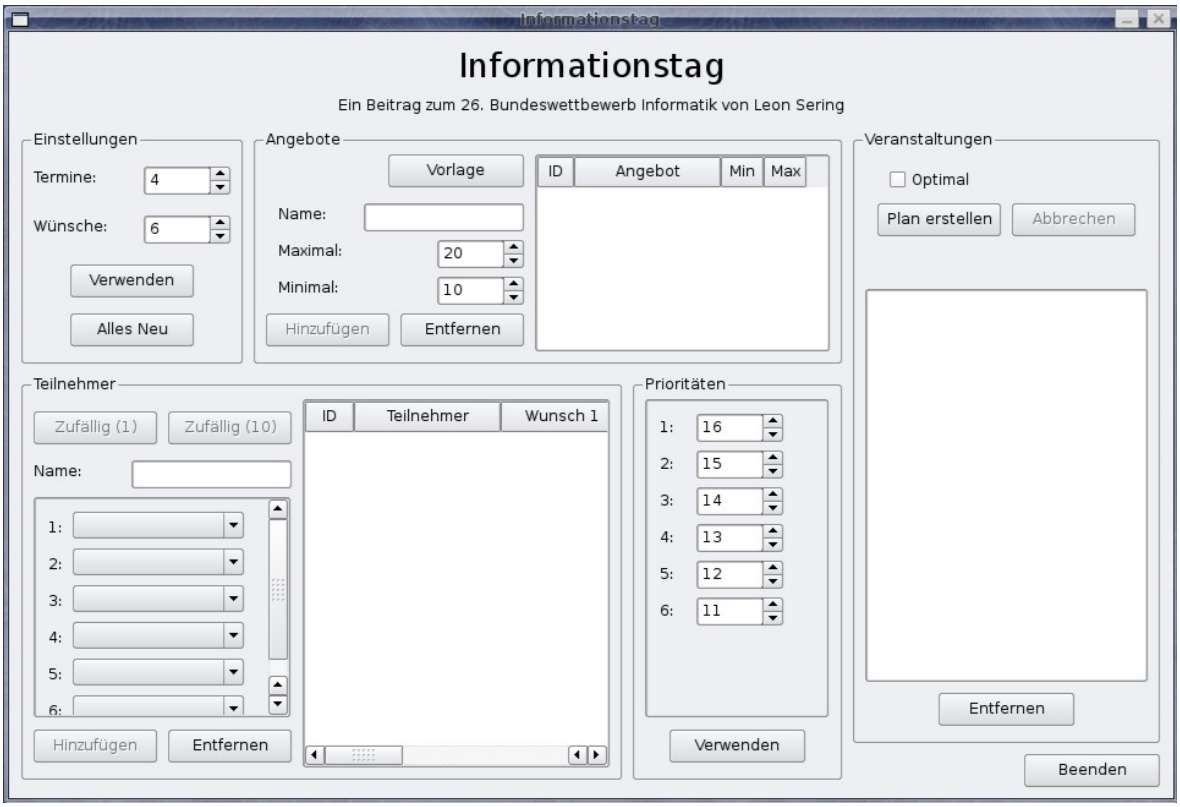
Danach ruft sich die Methode selber auf, aber nur, wenn in dem Durchgang eine neue Veranstaltung entstanden ist. Ist dies nicht der Fall, kann keine weitere mehr hinzukommen und der Plan ist fertig.

3.2.4 Anderes

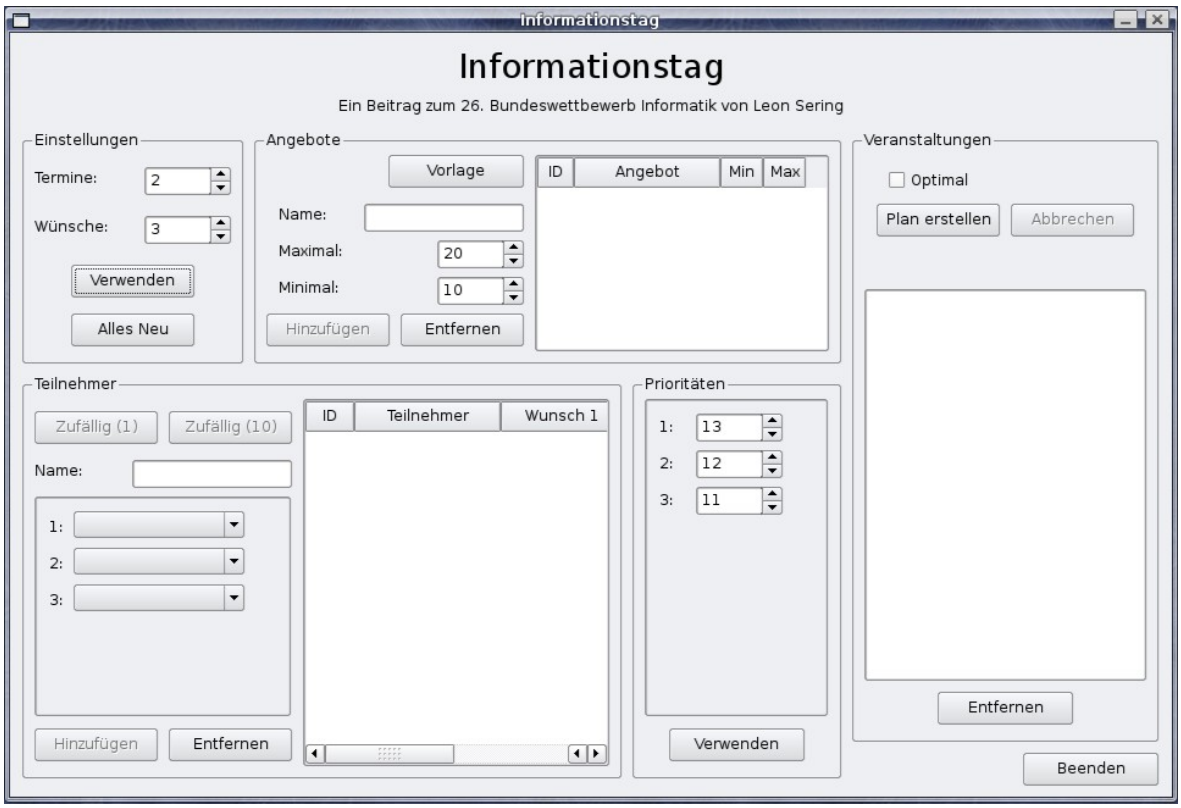
Zu den anderen Klassen gibt es eigentlich nicht mehr zu sagen, als es in der Übersicht schon beschrieben ist. Dem entsprechend fallen die Quelltexte auch sehr kurz aus.

3.3 Programm-Ablaufsprotokoll

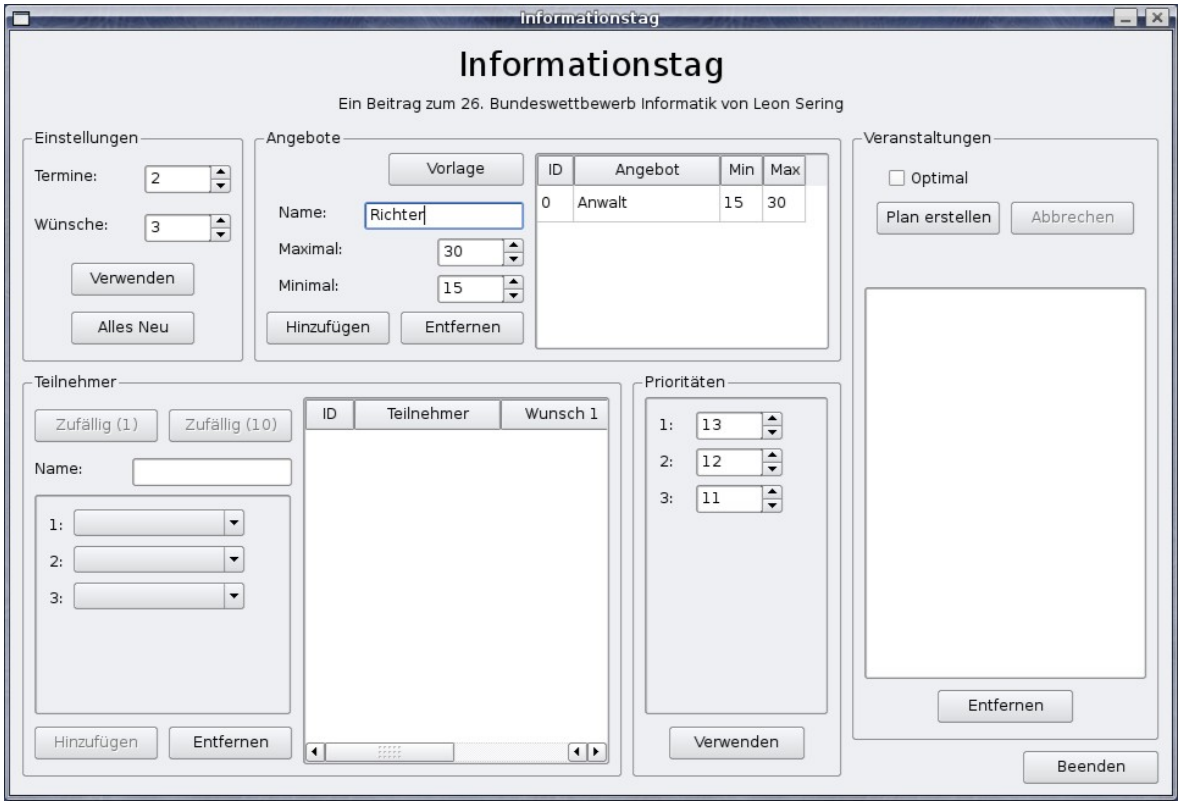
Sobald man das Programm startet, erscheint das Fenster, welches mit 4 Terminen und 6 Wünschen geladen wird. Dabei wurde zu der Vorlage, die aus der qtinformationstag.ui geladen wird, noch die Wünsche für den Teilnehmer sowie die Prioritäten erstellt. Die Listen sind zu Beginn alle leer:



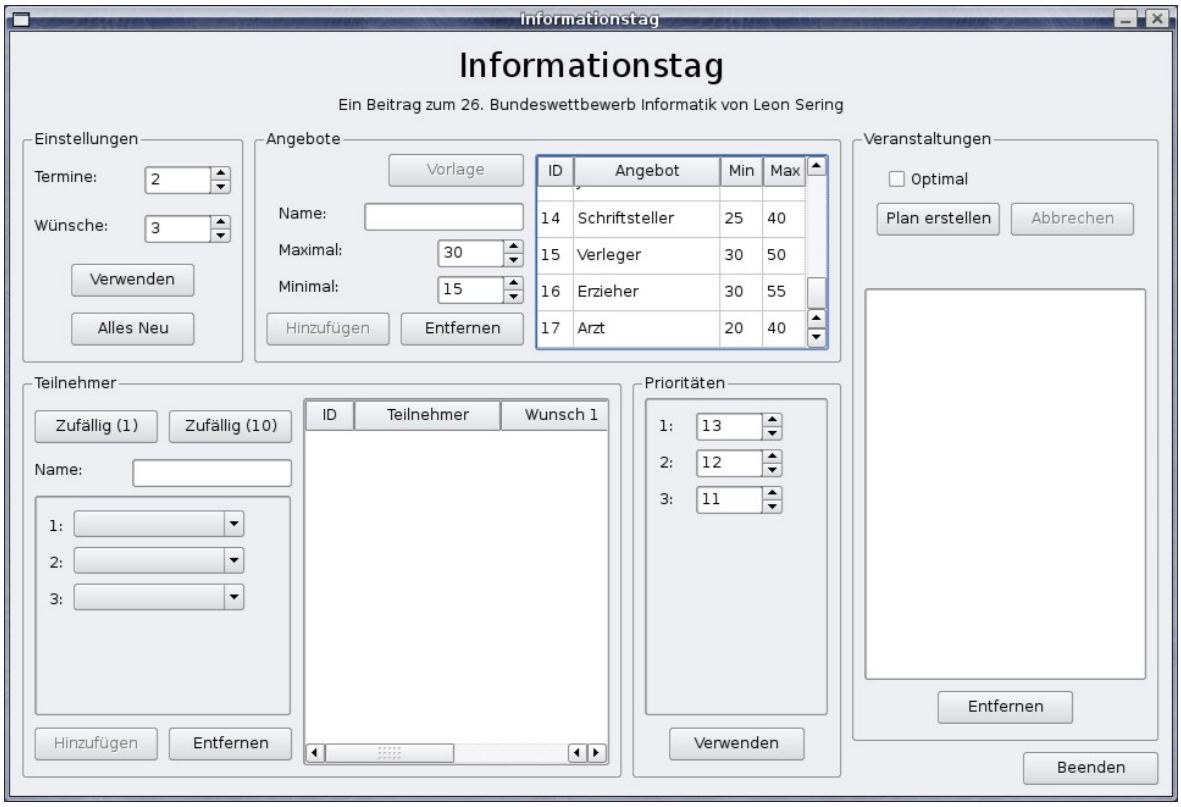
Ändert man bei den Einstellungen die Anzahl der Termine oder der Wünsche, so passt sich das Fenster automatisch an, sobald man auf „Verwenden" klickt. Hier habe ich zwei Termine und drei Wünsche eingestellt. Man kann erkennen, dass man nun für den Teilnehmer auch nur drei Wünsche auswählen und ebenfalls nur drei Prioritätswerte einstellen kann:



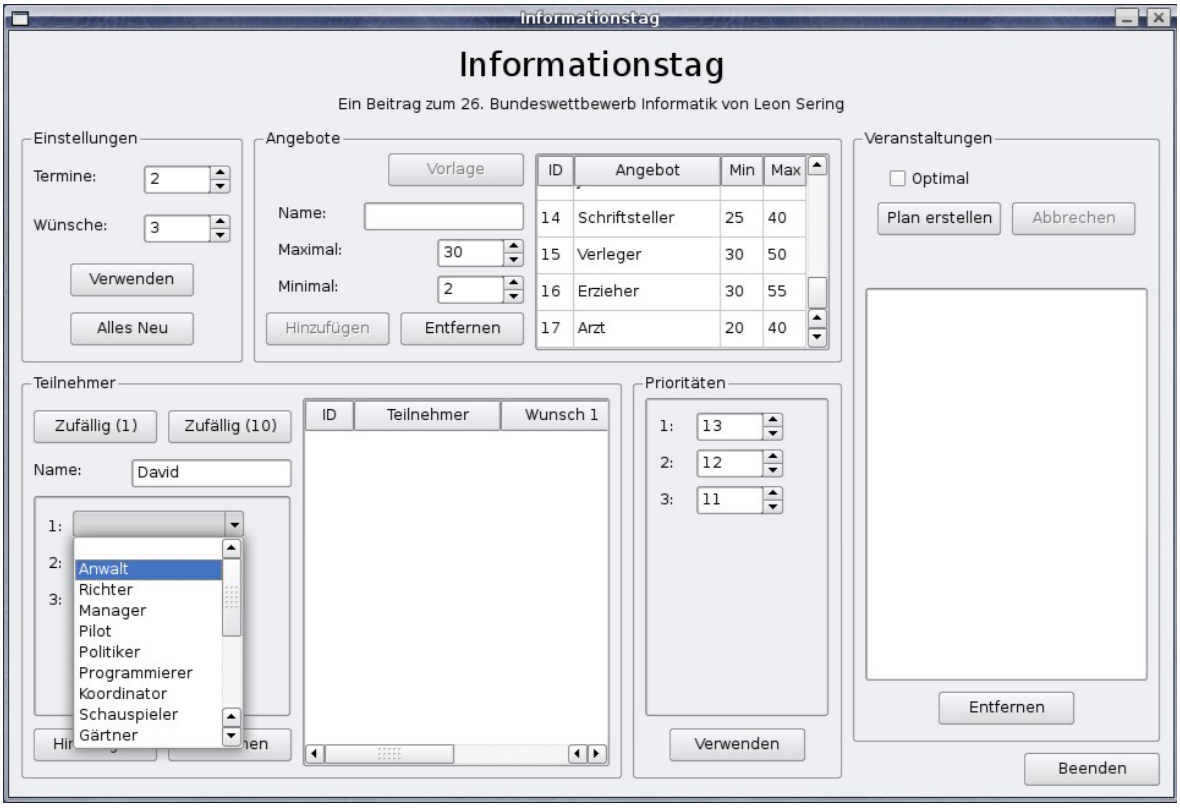
Nun wollen wir ein paar Angebote hinzufügen. Dafür wählt man einfach einen Namen (ein Thema) und stellt die gewünschte Minimal- und Maximalzahl ein. Sobald man auf „Hinzufügen" klickt, wird das Angebot in die rechte Liste geschrieben:



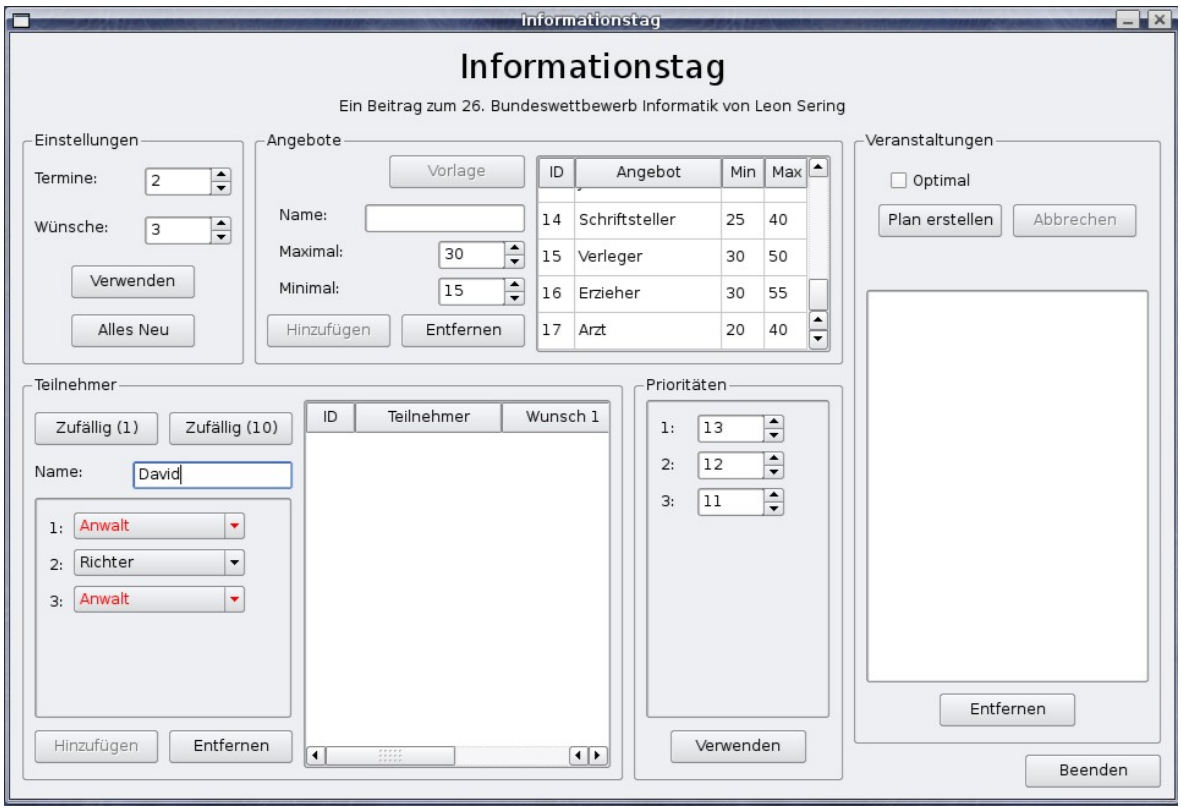
Klickt man auf „Vorlage“, werden zusätzlich 16 Angebote hinzugefügt, so dass in unserem Fall jetzt 18 Angebote in der Liste sind. (Zu beachten: Die IDs fangen bei 0 an):



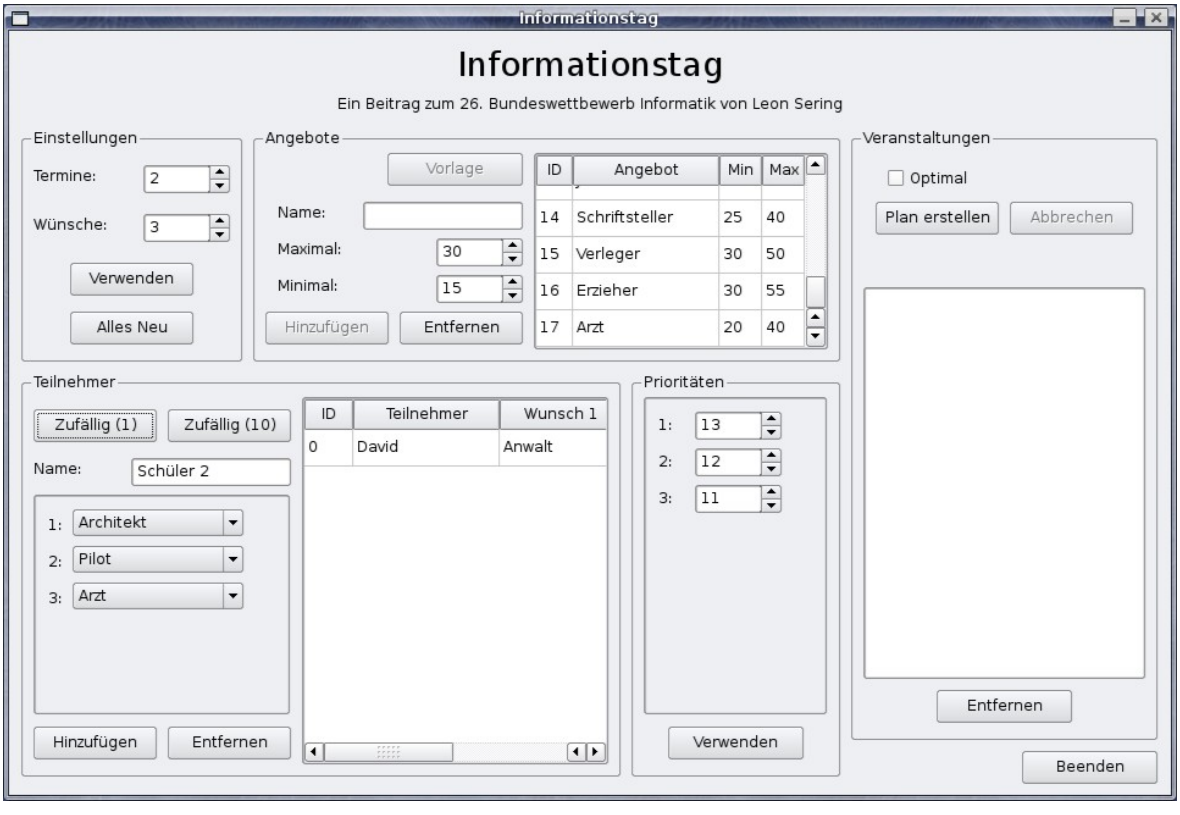
Als Nächstes sollte man ein paar Teilnehmer hinzufügen. Dafür wählt man einen Namen sowie die Wünsche aus:



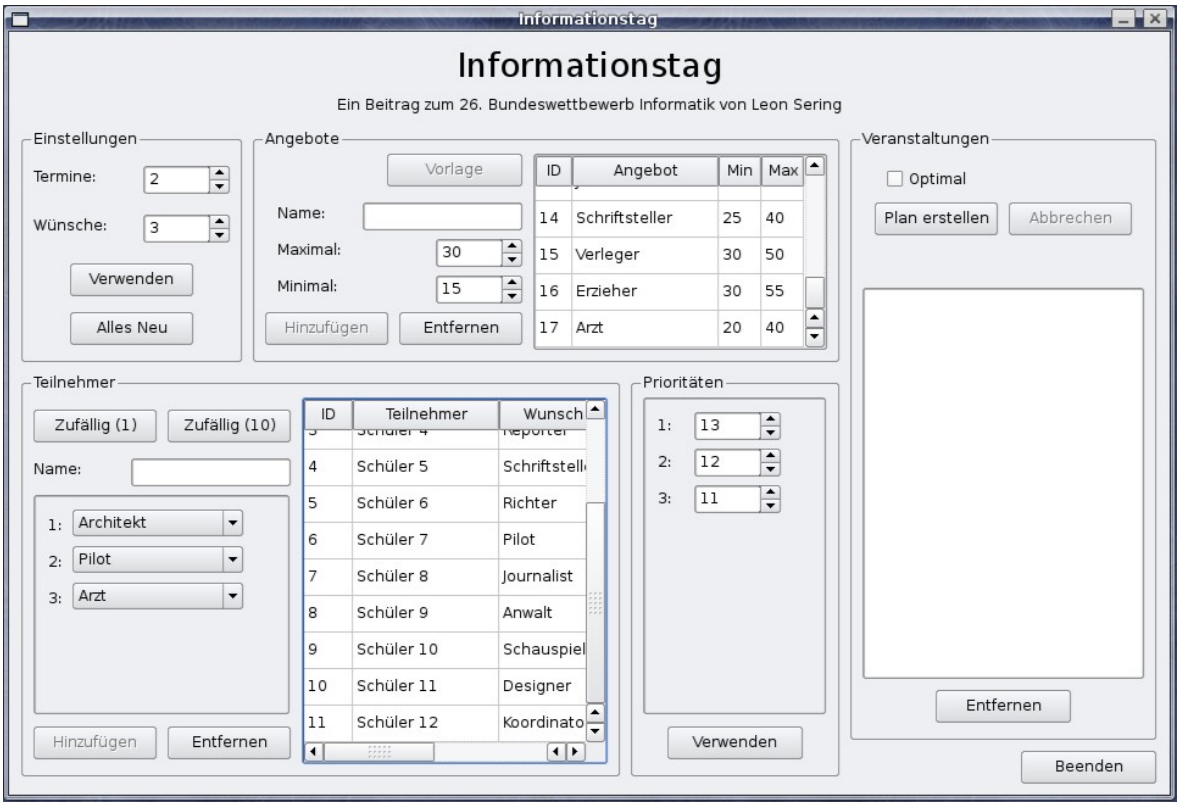
Wählt man verbotenerweise für zwei Wünsche das gleiche Angebot aus, so werden die entsprechenden Comboboxen rot eingefärbt und der „Hinzufügen“-Button deaktiviert:



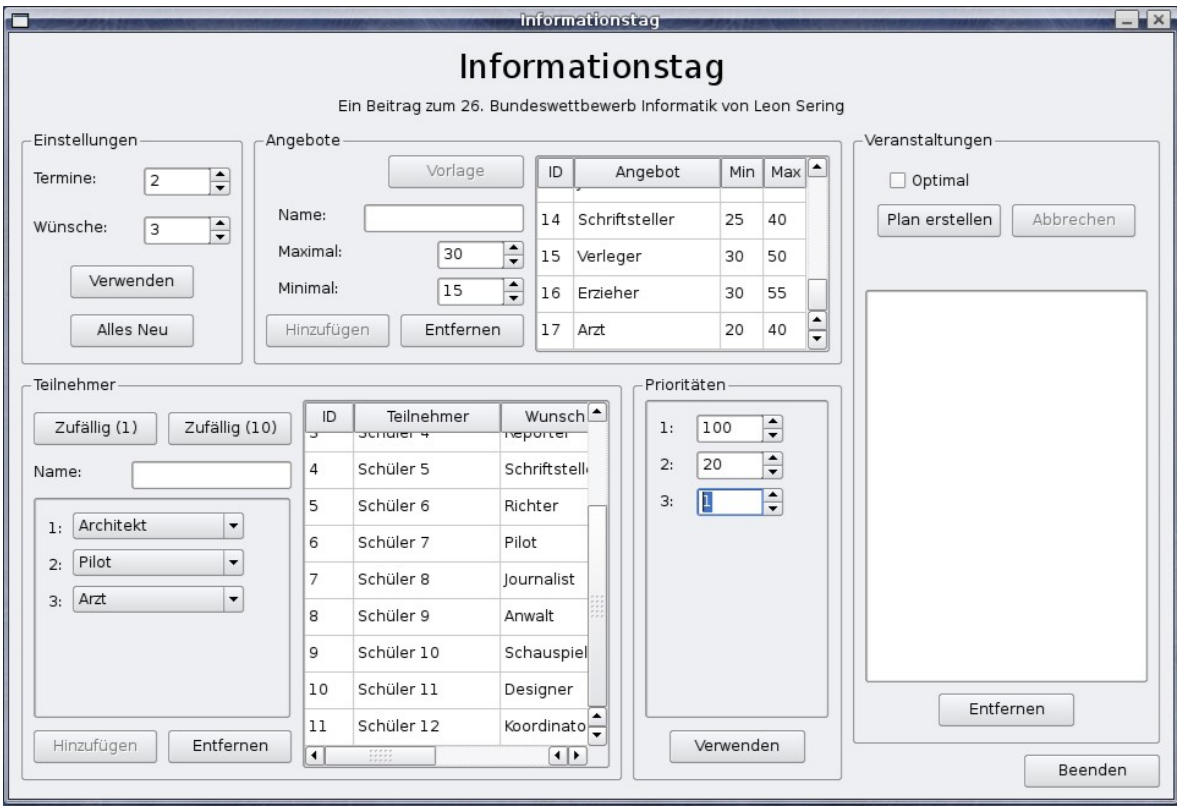
Auch hier erscheint der Teilnehmer in der Liste, sobald man ihn hinzufügt. Mit „Zufällig (1)“ kann man die Felder zufällig füllen lassen und es wird automatisch ein Name generiert („Schüler X“):



Mit „Zufällig (10)“ werden direkt 10 zufällig erstellte Teilnehmer in die Liste eingetragen. Dies ist für Testzwecke sehr angenehm:



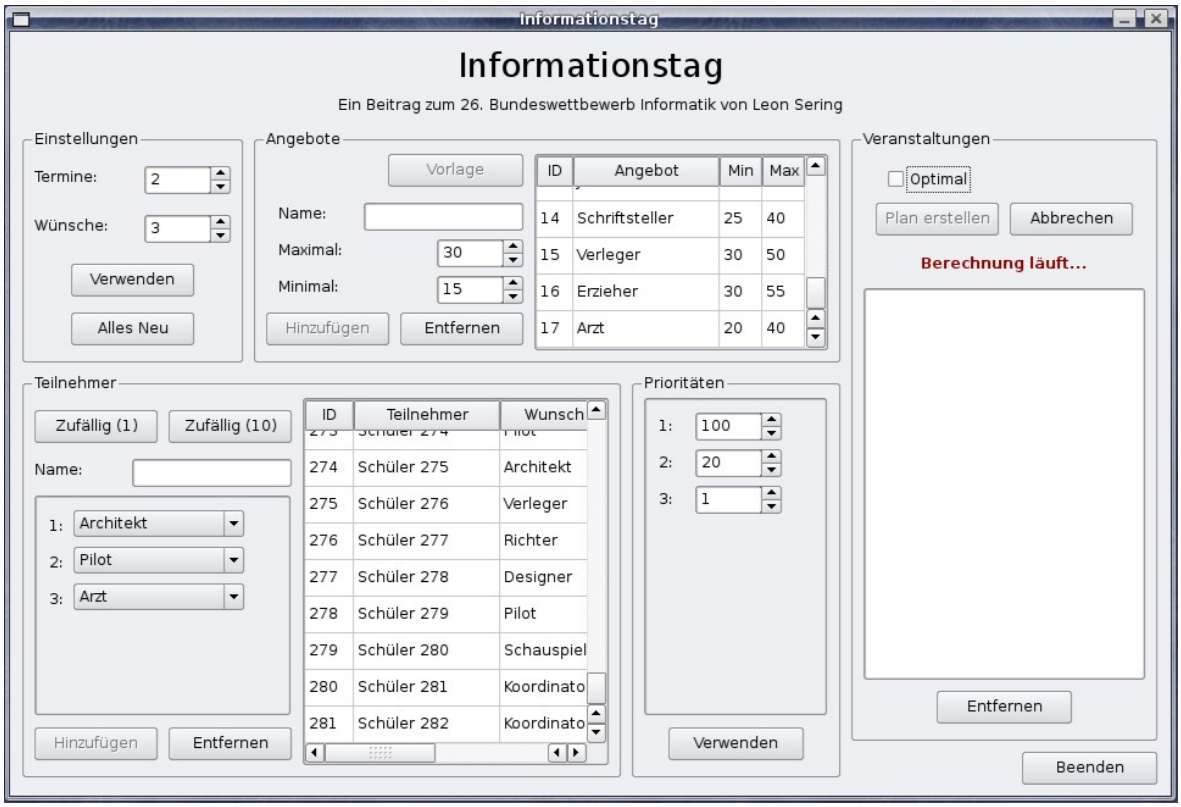
Bei Prioritäten kann man den Wünschen der Teilnehmer bestimmte Prioritätswerte zuordnen. Wenn man zum Beispiel möchte, dass eigentlich jeder seinen ersten Wunsch erfüllt bekommt, der zweite schon deutlich schlechter bewertet wird und der dritte eigentlich nur für den Notfall ist, kann man die Priorität wie folgt einstellen:



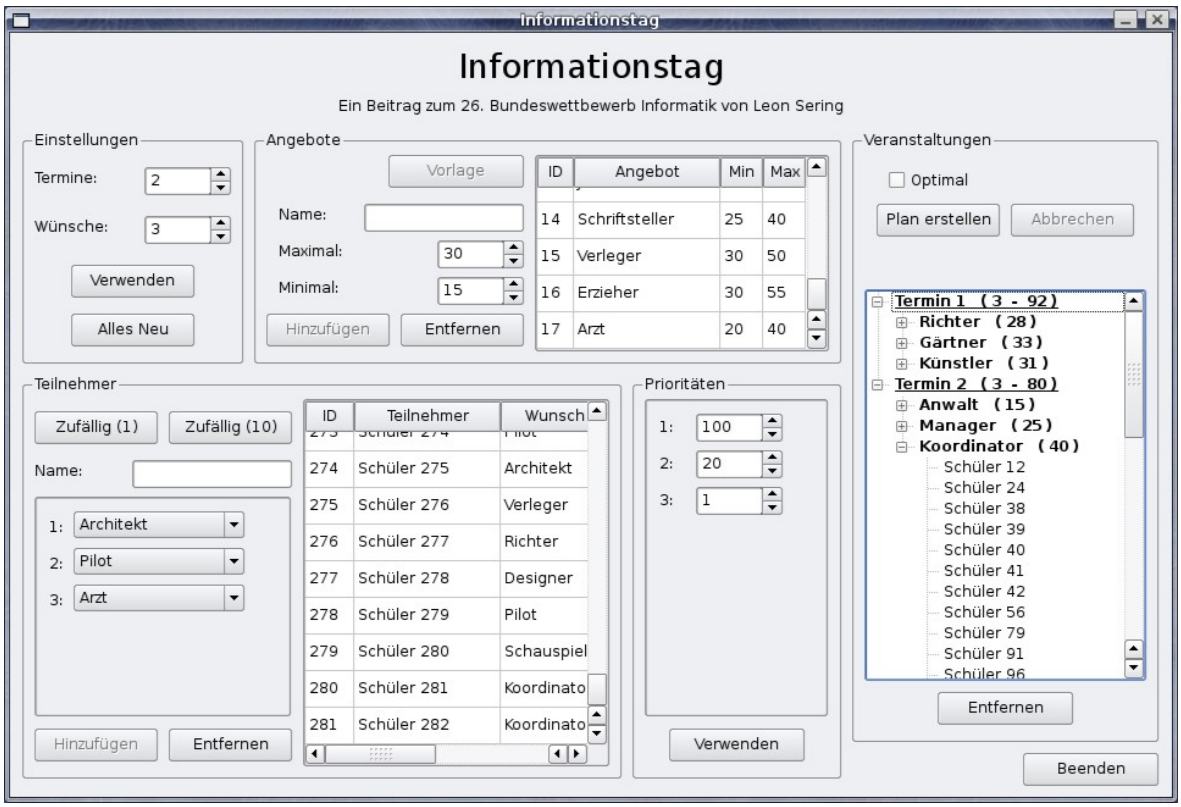
Hat man die Prioritäten mit „Verwenden“ bestätigt, kann man nun den Plan erstellen lassen.

Dafür wählt man zunächst aus, ob man die optimale oder die schnelle Methode verwenden möchte, danach klickt man auf „Plan erstellen“.

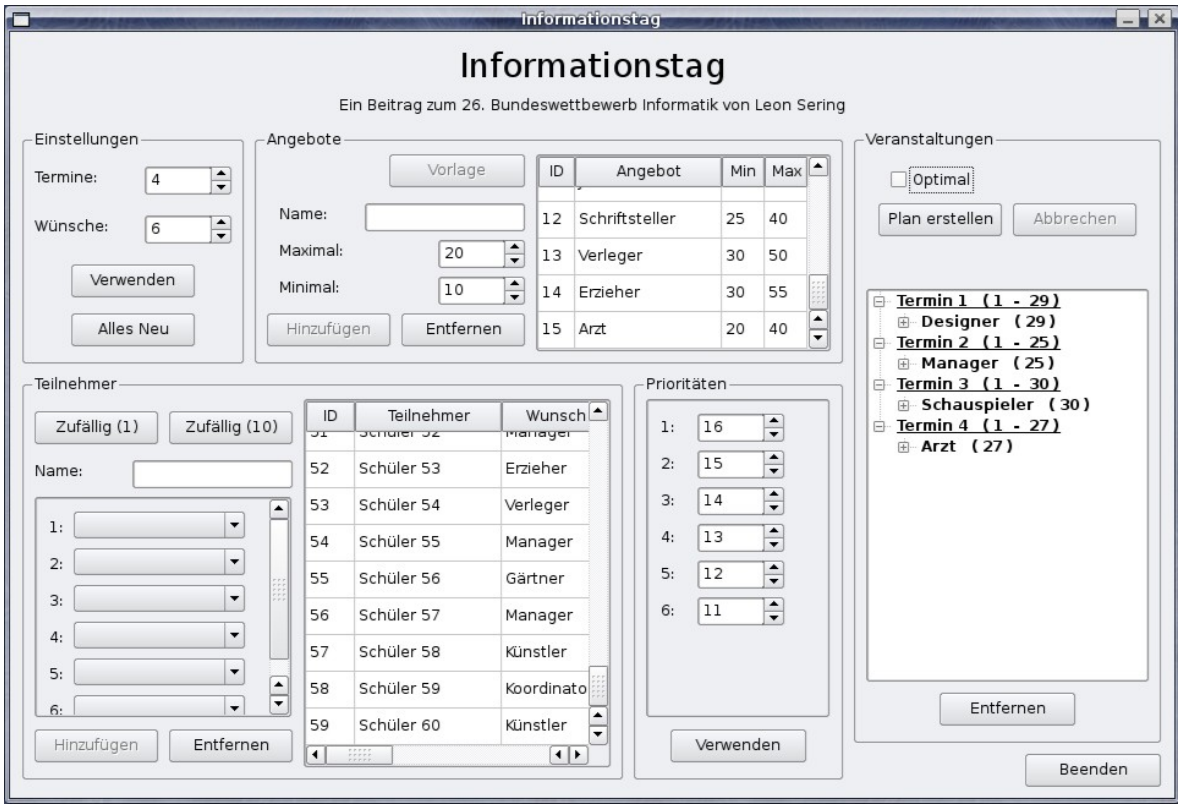
Hier habe ich nun noch eine Menge Teilnehmer hinzugefügt und benutze die schnelle Methode. Da es etwas dauert, wird angezeigt, dass das Programm gerade den Plan berechnet. Mit „Abbrechen“ könnte man den Berechnungsvorgang abbrechen, was leider nicht ganz fehlerfrei funktioniert:



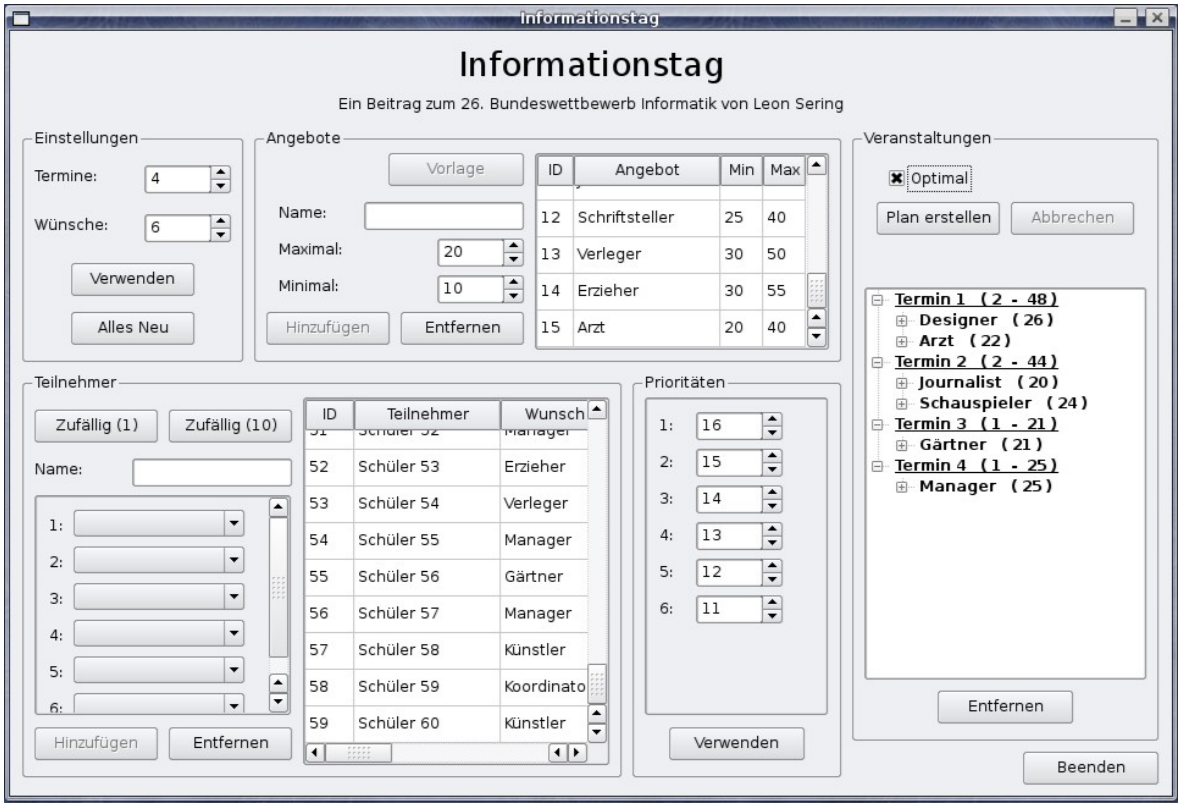
Ist die Liste berechnet, wird diese in dem TreeWidget angezeigt, sortiert nach Terminen, Veranstaltungen und Teilnehmern. In Klammern dahinter steht bei den Veranstaltungen, wie viele Teilnehmer sie besitzen, und bei den Terminen sowohl die Anzahl der Veranstaltungen als auch die Summe der Teilnehmer:



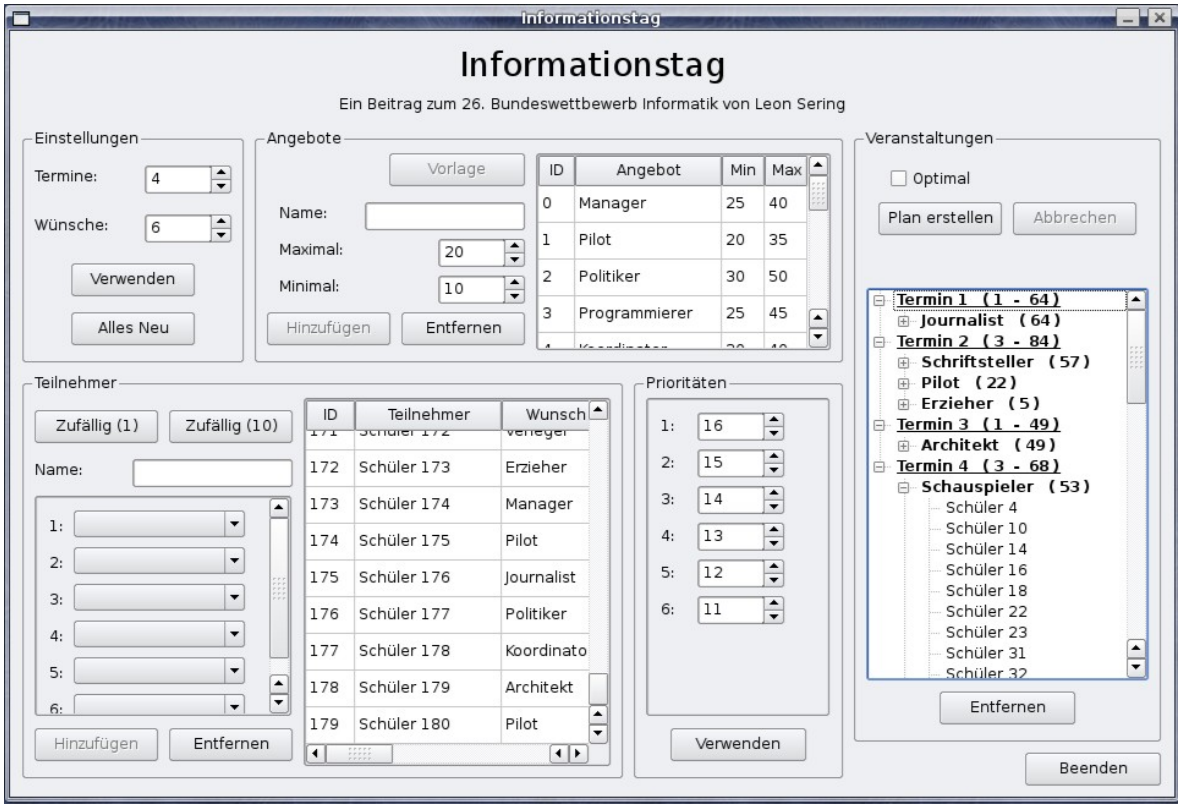
Nun wollen wir die beiden Berechnungsmethoden vergleichen. Dafür habe ich 4 Termine und 6 Wünsche eingestellt sowie die Vorlage für die Angebote und 60 Teilnehmer geladen. Die Prioritäten bleiben so, wie sie standardmäßig sind. Die schnelle Methode gibt folgenden Plan heraus:



Die optimale Methode schneidet, wie erwartet, besser ab und teilt den Plan so ein, dass mehr Veranstaltungen stattfinden können. Man sieht, dass alle Veranstaltungen, die schon die schnelle Methode ermittelt hat, auch hier wieder vorkommen, allerdings häufig nicht von so vielen Schülern besucht werden:



Nun wollen wir uns dem Problem von Herrn Lämpel widmen. Dafür benutze ich wieder 4 Termine und 6 Wünsche sowie die Vorlage. Doch diesmal erstelle ich zufällig ganze 180 Teilnehmer. Die optimale Methode würde nun zu lange dauern, weshalb ich nur die „schnelle" benutze. Doch auch sie benötigt auf meinem Laptop volle acht Minuten. Das Ergebnis sieht dann so aus:



Nun möchte ich auch noch eine andere Anwendungsmöglichkeit des Programms zeigen. Und zwar nehme ich dafür ein aktuelles Problem von meiner Schule, der Katholischen Schule Liebfrauen (KSL).

Hier können, wie an jeder Schule, alle Schüler der Oberstufe in jedem Semester ihre Sportkurse wählen. Allerdings verläuft die Wahl immer nach der Regel: „Wer zuerst kommt malt zuerst". Alle anderen müssen sich nach anderen Möglichkeiten umschauen.

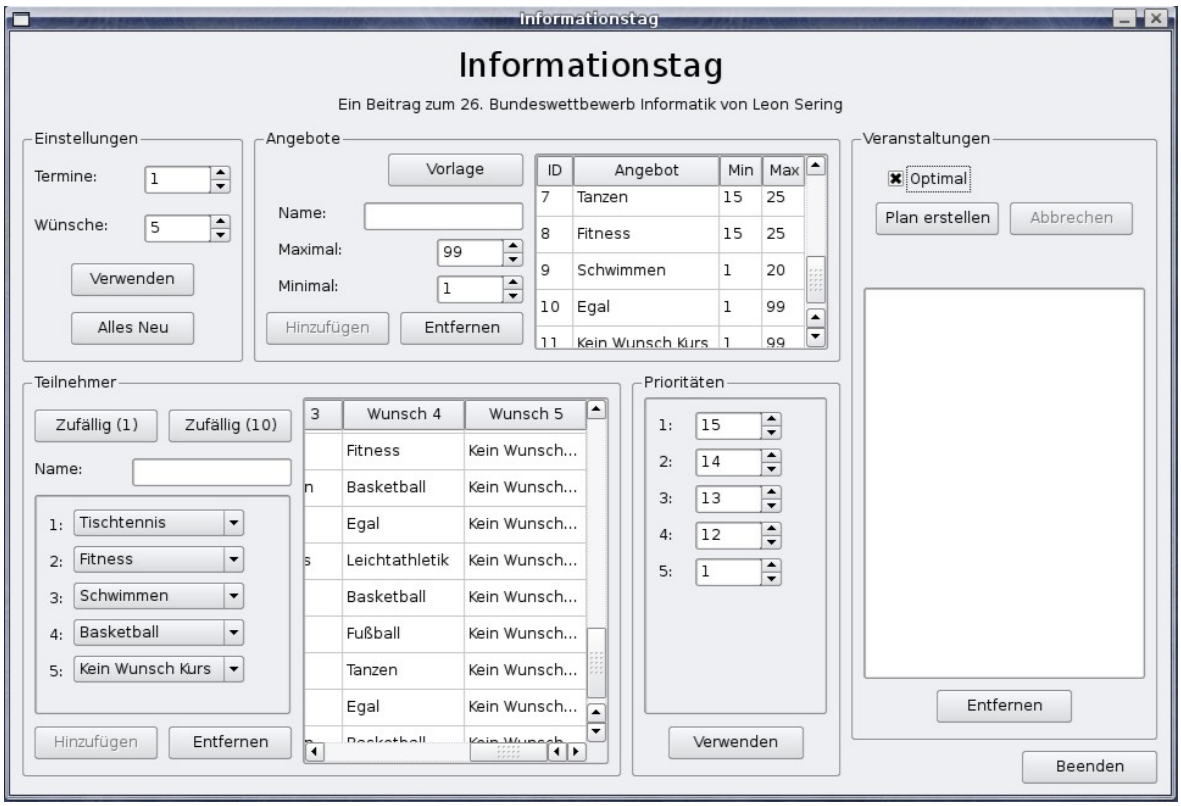
Dies führt natürlich immer zu großen Turbulenzen, weshalb unser Oberstufenkoordinator sich für nächstes Semester überlegt hat, Wünsche entgegen zu nehmen, um dann eine möglichst faire Einteilung vornehmen zu können. Mit ein paar kleinen Tricks könnte er dafür mein Programm benutzen.

Sagen wir mal, jeder Schüler darf 4 Wünsche nach Priorität sortiert angeben. Da jeder nur einen Sportkurs am Ende belegt, gibt es sozusagen nur einen Termin. Probleme wie Überschneidungen mit anderen Kursen oder dem Belegungsplan der Sporthalle können dabei nicht berücksichtigt werden.

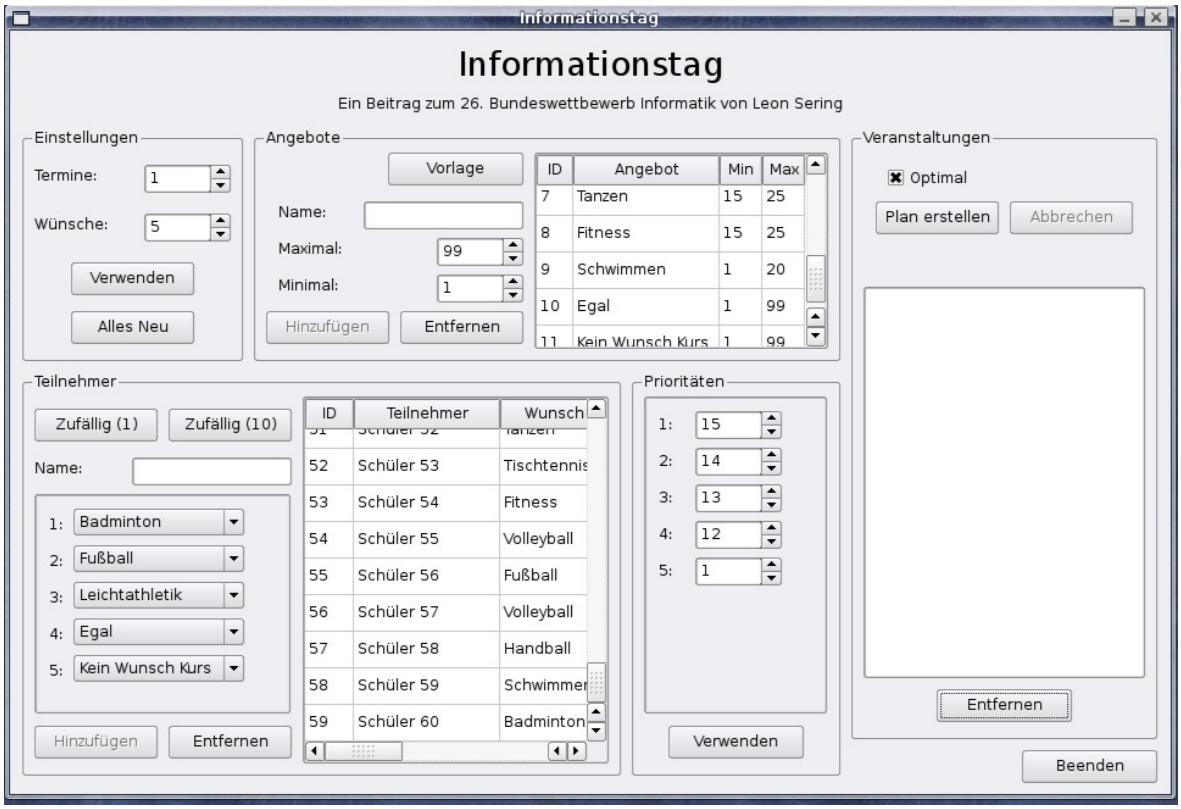
Da ich noch eine Auflistung aller Schüler haben möchte, die keinen Kurs ihrer Wünsche bekommen haben, führe ich noch einen Wunsch zusätzlich ein.

Danach erstelle ich bei den Angeboten die unterschiedlichen Sportkurse, wobei natürlich auch hier die minimale und maximale Anzahl beachtet werden sollte. Dazu kommt noch eine Rubrik „Egal", falls ein Schüler nur drei Wünsche angegeben hat (man kann auch mehrere „Egal"s einführen), und das Angebot „Kein Wunsch Kurs". Diese beiden Angebote müssen minimal ein Schüler und maximal 99 Schüler haben, sie sind also quasi völlig frei.

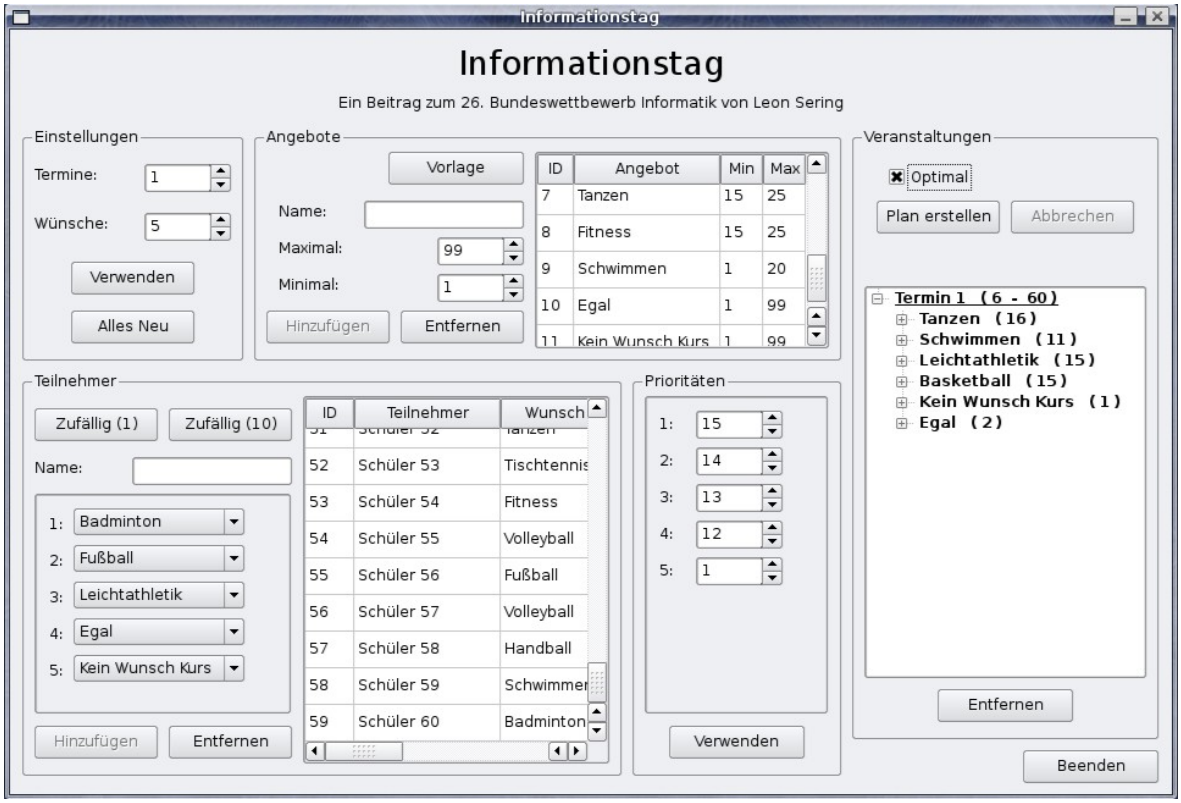
Danach müssen noch die Schüler erstellt werden, wobei darauf zu achten ist, dass, wenn einer nur drei Wünsche angegeben hat, das „Egal" an Position 4 ist und bei jedem Schüler an Position 5 das Angebot „Kein Wunsch Kurs" steht.



Als Letztes muss man noch die Prioritäten festlegen. Dabei ist wichtig, dass Wunsch 5 nur einen sehr niedrigen Wert besitzt, denn eigentlich soll es ja nicht zur Gesamtprioritätswertung beitragen, wenn einem Schüler kein Wunsch erfüllt wird, und er dann in die „Kein Wunsch Kurs"-Gruppe kommt. Ich habe jetzt mal 60 Schüler hinzugefügt:



Bei einem Termin ist sogar die optimale Methode äußerst schnell, weshalb diese ausgewählt wurde und dann folgendes Ergebnis ausgibt:



Somit sieht unser Oberstufenkoordinator nun, welche Kurse angeboten werden sollten und welche Schüler diese besuchen. Einem Schüler konnte leider kein Wunsch erfüllt werden, dieser muss dann einen anderen wählen. Zwei Leuten wurde der Egal-Wunsch ausgewählt, weshalb auch diese sich andere Kurse suchen müssen.

4 Quellcode

4.1 main.cpp

```
1.  #include <iostream> // fuer Konsolenausgaben
2.  using namespace std;
3.
4.  #include <time.h>  // fuer Zufallswerte
5.
6.  // Qt
7.  #include <QtGui>
8.  #include <QApplication>
9.  #include <QString>
10.
11.  #include "QtInformationstag.h"
12.
13.  int main(int argc, char *argv[]) {
14.
15.      srand( (unsigned)time( NULL ) ); // neue Zufallswerte erstellen
16.
17.      QApplication a(argc, argv);
18.
19.      a.connect(&a, SIGNAL(lastWindowClosed()), &a, SLOT(quit()));
20.
21.      QtInformationstag fenster;
22.
23.      fenster.show();
24.
25.      return a.exec();
26.  }
27.
```

4.2 QtInformationstag.h

```
1.  #ifndef QTINFORMATIONSTAG_H
2.  #define QTINFORMATIONSTAG_H
3.
4.  #include <iostream>
5.  using namespace std;
6.
7.  // Qt
8.  #include <QString>
9.  #include <QScrollArea>
10. #include <QComboBox>
11. #include <QHeaderView>
12. #include <QList>
13. #include <QPalette>
14.
15.
16. #include "ui_qtinformationstag.h"
17.
18. // eigene
19. #include "cInformationstag.h"
20. #include "cThread.h"
21.
22. class QtInformationstag : public QWidget
23. {
24.     Q_OBJECT
25.
26. public:
27.     QtInformationstag(QWidget *parent = 0);
28.     ~QtInformationstag();
29.
30. private:
31.      cInformationstag itsInformationstag;
32.
33.     Ui::QtInformationstagClass ui;
34.
35.     // WunschBox
36.     QScrollArea scrollWunschBox;
37.     QWidget widgetWunschBox;
38.     QComboBox * cBoxWunschBox;
39.     QLabel * labelWunschBox;
40.
41.     // Prioritaeten
42.     QScrollArea scrollPrioritaeten;
43.     QWidget widgetPrioritaeten;
44.     QSpinBox * sBoxPrioritaeten;
45.     QLabel * labelPrioritaeten;
46.
47.     // Thread
48.     cThread itsThread; // um Berechnungen im Hintergrund laufen lassen
                          zu koennen
49.     bool abgebrochen; // merkt sich, ob abgebrochen wurde (da das
                          Abbrechen-Klicken und "fertig abgebrochen" zeitlich
                          auseinander liegen )
50.
51.
52.     bool IndexDoppelt(int index);
53.
54. public slots:
```

```
55.
56.         // Button-Slots
57.         void on_btEinstellungenVerwenden_clicked();
58.         void on_btAllesNeu_clicked();
59.
60.         void on_btAngebotHinzufuegen_clicked();
61.         void on_btAngebotEntfernen_clicked();
62.
63.         void on_btTeilnehmerHinzufuegen_clicked();
64.         void on_btTeilnehmerEntfernen_clicked();
65.
66.         void on_btPlanErstellen_clicked();
67.         void on_btVeranstaltungenEntfernen_clicked();
68.         void on_btAbbrechen_clicked();
69.
70.         void on_btVorlageHinzufuegen_clicked();
71.         void on_btZufaellig1_clicked();
72.         void on_btZufaellig10_clicked();
73.
74.         void on_btPrioritaetenVerwenden_clicked();
75.
76.         // Groesser-Kleiner-Slots
77.         void when_sBoxMax_changed(int value);
78.         void when_sBoxMin_changed(int value);
79.
80.         void when_sBoxAnzWuensche_changed(int value);
81.         void when_sBoxAnzTermine_changed(int value);
82.
83.         // Buttondeaktivier-Slots
84.         void btAngebotHinzufuegen_deaktivieren(QString text);
85.         void btTeilnehmerHinzufuegen_deaktivieren(QString text);
86.         void btTeilnehmerHinzufuegen_deaktivieren(int nix);  // Parameter
                    macht nichts, ist nur fuer die Verknuepfung mit dem
                    Signal
87.
88.         // WuenschBox modifizieren
89.         void when_AnzAngebote_changed();
90.         void wunschBox_index_changed(int index);
91.
92.         // Planerstellen
93.         void plan_erstellt();  // wird aufgerufen, sobald itsThread den
                    Plan fertig erstellt hat. Plan wird hier ausgegeben
94.
95.
96.   };
97.
98.   #endif // QTINFORMATIONSTAG_H
99.
```

4.3 QtInformationstag.cpp

```
1.  #include "QtInformationstag.h"
2.
3.  QtInformationstag::QtInformationstag(QWidget *parent)
4.      : QWidget(parent)
5.  {
6.       ui.setupUi(this);
7.
8.       ui.labelStatus->hide();
9.
10.      // itsTread wird eingerichtet
11.      itsThread.setInformationstag(&itsInformationstag);
12.      connect(&itsThread, SIGNAL (finished()), this,
                    SLOT(plan_erstellt()));
13.      abgebrochen = false;
14.
15.
16.      // Tabellen fuer die Listen werden eingerichtet (Spaltbreite
                    veraendert)
17.      ui.tableAngebote->setColumnWidth(0, 30);
18.      ui.tableAngebote->setColumnWidth(1, 120);
19.      ui.tableAngebote->setColumnWidth(2, 35);
20.      ui.tableAngebote->setColumnWidth(3, 35);
21.
22.      ui.tableAngebote->verticalHeader()->hide(); // Headers sollen alle
                    nicht angezeigt werden
23.      ui.tableTeilnehmer->verticalHeader()->hide();
24.      ui.treeVeranstaltungen->header()->hide();
25.
26.      // ScrollMode aendern, so dass kontinuierliches Scrollen moeglich
                    ist
27.      ui.tableAngebote-
                    >setHorizontalScrollMode(QAbstractItemView::ScrollPer
                    Pixel);
28.      ui.tableAngebote-
                    >setVerticalScrollMode(QAbstractItemView::ScrollPerPi
                    xel);
29.      ui.tableTeilnehmer-
                    >setHorizontalScrollMode(QAbstractItemView::ScrollPer
                    Pixel);
30.      ui.tableTeilnehmer-
                    >setVerticalScrollMode(QAbstractItemView::ScrollPerPi
                    xel);
31.      ui.treeVeranstaltungen-
                    >setVerticalScrollMode(QAbstractItemView::ScrollPerPi
                    xel);
32.
33.      // Die Scrollarea fuer die WunschBox:
34.      scrollWunschBox.setParent(ui.gBoxTeilnehmer);
35.      scrollWunschBox.setGeometry(10, 100, 210, 180);
36.
37.      widgetWunschBox.setParent(&scrollWunschBox);
38.
39.      scrollWunschBox.setWidget(&widgetWunschBox);
40.
41.      cBoxWunschBox = new QComboBox[1];
42.      labelWunschBox = new QLabel[0];
43.
```

```
44.         cBoxWunschBox[0].insertItem(0, "");
45.
46.         // Die Scrollarea fuer die Prioritaeten
47.         scrollPrioritaeten.setParent(ui.gBoxPrioritaeten);
48.         scrollPrioritaeten.setGeometry(10, 20, 150, 260);
49.
50.         widgetPrioritaeten.setParent(&scrollPrioritaeten);
51.
52.         scrollPrioritaeten.setWidget(&widgetPrioritaeten);
53.
54.         sBoxPrioritaeten = new QSpinBox[0];
55.         labelPrioritaeten = new QLabel[0];
56.
57.         when_sBoxAnzWuensche_changed(6); // Wuensche zeichnen
58.
59.         // Wenn neues Angebot hinzugefuegt wurde, soll evtl., die
                       Zufaells-Buttons deaktiviert werden
60.         connect(ui.btAngebotHinzufuegen, SIGNAL(clicked()), this,
                       SLOT(when_AnzAngebote_changed()));
61.         connect(ui.btAngebotEntfernen, SIGNAL(clicked()), this,
                       SLOT(when_AnzAngebote_changed()));
62.         connect(ui.btVorlageHinzufuegen, SIGNAL(clicked()), this,
                       SLOT(when_AnzAngebote_changed()));
63.         connect(ui.btAllesNeu, SIGNAL(clicked()), this,
                       SLOT(when_AnzAngebote_changed()));
64.
65.         // wenn beim Nameneingaben Enter gedrueckt wird, soll der
                       entsprechende Button geklickt werden.
66.         connect(ui.nameAngebot, SIGNAL (returnPressed ()),
                       ui.btAngebotHinzufuegen, SLOT (click()));
67.         connect(ui.nameTeilnehmer, SIGNAL (returnPressed ()),
                       ui.btTeilnehmerHinzufuegen, SLOT (click()));
68.
69.         // wenn die Min bzw. Max Spinboxen veraendert werden, soll der
                       jeweilige andere evtl. angepasst werden
70.         connect(ui.sBoxMax, SIGNAL(valueChanged(int)), this,
                       SLOT(when_sBoxMax_changed(int)));
71.         connect(ui.sBoxMin, SIGNAL(valueChanged(int)), this,
                       SLOT(when_sBoxMin_changed(int)));
72.
73.         // das gleiche bei den Spinboxen AnzTermine und AnzWuensche
74.         connect(ui.sBoxAnzTermine, SIGNAL(valueChanged(int)), this,
                       SLOT(when_sBoxAnzTermine_changed(int)));
75.         connect(ui.sBoxAnzWuensche, SIGNAL(valueChanged(int)), this,
                       SLOT(when_sBoxAnzWuensche_changed(int)));
76.
77.         // Wenn der Name veraendert wird, soll ggf. der entsprechende
                       Button aktiviert oder deaktiviert werden
78.         connect(ui.nameAngebot, SIGNAL(textChanged (QString) ), this,
                       SLOT(btAngebotHinzufuegen_deaktivieren(QString)));
79.         connect(ui.nameTeilnehmer, SIGNAL(textChanged (QString) ), this,
                       SLOT(btTeilnehmerHinzufuegen_deaktivieren(QString)));
80.
81.         on_btEinstellungenVerwenden_clicked();
82.
83.   }
84.
85.   QtInformationstag::~QtInformationstag() {
86.         // Alle Arrays sollen freigegeben werden
87.         delete [] cBoxWunschBox;
88.         delete [] labelWunschBox;
89.
```

```
90.         delete [] sBoxPrioritaeten;
91.         delete [] labelPrioritaeten;
92.     }
93.
94.
95.     ////////////////////////////////////////////////////////////////////
96.     ////////////////////////////// SLOTS ///////////////////////////////
97.     ////////////////////////////////////////////////////////////////////
98.
99.     void QtInformationstag::on_btEinstellungenVerwenden_clicked() {
100.
101.        itsInformationstag.TeilnehmerLoeschen();
102.
103.        int anzTermine = ui.sBoxAnzTermine->value();
104.        int anzWuensche = ui.sBoxAnzWuensche->value();
105.
106.        // itsInformationstag aktualisieren
107.        itsInformationstag.setAnzTermine(anzTermine);
108.        itsInformationstag.setAnzWuensche(anzWuensche);
109.
110.        // tableTeilnehmer aktualisieren
111.        ui.tableTeilnehmer->clear();
112.        ui.tableTeilnehmer->setRowCount(0);
113.
114.        ui.tableTeilnehmer->setColumnCount(anzWuensche + 2);  // + ID +
                    Name
115.        ui.tableTeilnehmer->setColumnWidth(0, 40);
116.        ui.tableTeilnehmer->setColumnWidth(1, 120);
117.
118.        QStringList headerString;
119.        headerString << "ID" << "Teilnehmer";
120.
121.        for (int i = 0; i < anzWuensche; i++) {
122.            ui.tableTeilnehmer->setColumnWidth(2 + i, 100);
123.            headerString << ("Wunsch " + QString::number(i+1));
124.        }
125.        ui.tableTeilnehmer->setHorizontalHeaderLabels(headerString);
126.
127.        // WunschBox erstellen
128.        widgetWunschBox.setGeometry(0, 0, 180, 15 + 30 * anzWuensche);
129.        widgetWunschBox.show();
130.
131.        QComboBox cBoxTemp; // einer wird sich gemerkt, um die Items
                    daraus zu kopieren
132.        int anzItems = cBoxWunschBox[0].count();
133.        for (int i = 0; i < anzItems; i++)
134.            cBoxTemp.insertItem(i, cBoxWunschBox[0].itemText(i));
135.
136.        delete [] cBoxWunschBox;
137.        delete [] labelWunschBox;
138.
139.        cBoxWunschBox = new QComboBox[anzWuensche];
140.        labelWunschBox = new QLabel[anzWuensche];
141.
142.        for (int i = 0 ; i < anzWuensche; i++) {
143.            cBoxWunschBox[i].setParent(&widgetWunschBox);
144.            cBoxWunschBox[i].setGeometry(30, 10 + 30 * i, 140, 22);
145.            for (int j = 0; j < anzItems; j++)
146.                cBoxWunschBox[i].insertItem(j, cBoxTemp.itemText(j));
147.            cBoxWunschBox[i].show();
148.            connect(&cBoxWunschBox[i], SIGNAL(currentIndexChanged(int)),
                        this, SLOT(wunschBox_index_changed(int)));
```

```
149.              connect(&cBoxWunschBox[i], SIGNAL(currentIndexChanged(int)),
                          this,
                          SLOT(btTeilnehmerHinzufuegen_deaktivieren(int)));
150.
151.              labelWunschBox[i].setParent(&widgetWunschBox);
152.              labelWunschBox[i].setGeometry(10, 14 + 30 * i, 20, 18);
153.              labelWunschBox[i].setText( QString::number(i+1) + ": ");
154.              labelWunschBox[i].show();
155.        }
156.
157.        // prioritaeten erstellen
158.
159.        widgetPrioritaeten.setGeometry(0, 0, 130, 15 + 30 * anzWuensche);
160.        widgetPrioritaeten.show();
161.
162.        delete [] sBoxPrioritaeten;
163.        delete [] labelPrioritaeten;
164.
165.        sBoxPrioritaeten = new QSpinBox[anzWuensche];
166.        labelPrioritaeten = new QLabel[anzWuensche];
167.
168.        for (int i = 0; i < anzWuensche; i++) {
169.              sBoxPrioritaeten[i].setParent(&widgetPrioritaeten);
170.              sBoxPrioritaeten[i].setGeometry(40, 10 + 30 * i, 71, 23);
171.              sBoxPrioritaeten[i].setMaximum(9999);
172.              sBoxPrioritaeten[i].setMinimum(1);
173.              sBoxPrioritaeten[i].setValue(10 + anzWuensche - i);
174.              sBoxPrioritaeten[i].show();
175.
176.              labelPrioritaeten[i].setParent(&widgetPrioritaeten);
177.              labelPrioritaeten[i].setGeometry(10, 12 + 30 * i, 20, 18);
178.              labelPrioritaeten[i].setText(QString::number(i+1) + ": ");
179.              labelPrioritaeten[i].show();
180.        }
181.  }
182.
183.  void QtInformationstag::on_btAllesNeu_clicked() {
184.
185.        // einfach alles leeren
186.        on_btEinstellungenVerwenden_clicked();
187.        on_btAngebotEntfernen_clicked();
188.        on_btVeranstaltungenEntfernen_clicked();
189.
190.        ui.nameTeilnehmer->setText("");
191.        ui.nameAngebot->setText("");
192.        ui.sBoxMax->setValue(20);
193.        ui.sBoxMin->setValue(10);
194.
195.
196.  }
197.
198.
199.  void QtInformationstag::on_btAngebotHinzufuegen_clicked() {
200.        int min = ui.sBoxMin->value();
201.        int max = ui.sBoxMax->value();
202.        QString name = ui.nameAngebot->text();
203.
204.        // zunaechst das Angebot bei itsInformationstag hinzufuegen
205.        int id = itsInformationstag.AngebotHinzufuegen(min, max, name);
                      // angebot zurueckgegeben und id gespeichert
206.
207.
```

```
208.        if (id == -1) {// wenn ein Fehler zurueckgegeben wurde
209.             cout << "Fehler beim Hinzufuegen eines Angebots!" << endl;
210.             return;  // soll abgebrochen werden
211.        }
212.
213.        // Dann das Fenster aktualsieren (in die Liste eintragen)
214.        ui.nameAngebot->setText("");
215.        ui.nameAngebot->setFocus();
216.
217.        ui.tableAngebote->insertRow(id);
218.        ui.tableAngebote->setItem(id, 0, new
                      QTableWidgetItem(QString::number(id)));
219.        ui.tableAngebote->setItem(id, 1, new QTableWidgetItem(name));
220.        ui.tableAngebote->setItem(id, 2, new
                      QTableWidgetItem(QString::number(min)));
221.        ui.tableAngebote->setItem(id, 3, new
                      QTableWidgetItem(QString::number(max)));
222.
223.        int anzWuensche = itsInformationstag.getAnzWuensche();
224.
225.        for (int i = 0; i < anzWuensche; i++)
226.             cBoxWunschBox[i].insertItem(id+1, name);
227.   }
228.
229.   void QtInformationstag::on_btAngebotEntfernen_clicked() {
230.
231.        // wenn die Angebote entfernt werden sollen, muessen auch alle
                      Teilnehmer entfernt werden
232.        ui.tableAngebote->clearContents();
233.        ui.tableAngebote->setRowCount(0);
234.        ui.tableTeilnehmer->clearContents();
235.        ui.tableTeilnehmer->setRowCount(0);
236.
237.        itsInformationstag.AllesLoeschen();
238.
239.        int anzWuensche = itsInformationstag.getAnzWuensche();
240.
241.        for (int i = 0; i < anzWuensche; i++) {
242.             cBoxWunschBox[i].clear();
243.             cBoxWunschBox[i].insertItem(0, "");
244.        }
245.
246.        ui.btVorlageHinzufuegen->setEnabled(true);
247.   }
248.
249.
250.   void QtInformationstag::on_btTeilnehmerHinzufuegen_clicked() {
251.        QString name = ui.nameTeilnehmer->text();
252.
253.        // erstmal auslesen
254.        int anzWuensche = itsInformationstag.getAnzWuensche();
255.
256.        int wuensche[anzWuensche];
257.        for (int i = 0; i < anzWuensche; i++)
258.             wuensche[i] = cBoxWunschBox[i].currentIndex() - 1;
259.
260.        // dann itsInformationstag aktualisieren
261.        int id = itsInformationstag.TeilnehmerHinzufuegen(wuensche, name);
262.
263.
264.        if (id == -1) { // Wenn Fehler zuruckegegeben wurde
265.             cout << "Fehler beim Hinzufuegen eines Teilnehmers!" << endl;
```

```
266.            return; // soll abgebrochen werden
267.        }
268.
269.        // und in die Liste eintragen
270.        ui.nameTeilnehmer->setText("");
271.        ui.nameTeilnehmer->setFocus();
272.
273.        ui.tableTeilnehmer->insertRow(id);
274.
275.        ui.tableTeilnehmer->setItem(id, 0, new
                    QTableWidgetItem(QString::number(id)));
276.        ui.tableTeilnehmer->setItem(id, 1, new QTableWidgetItem(name));
277.
278.        for (int i = 0; i < anzWuensche; i++)
279.            ui.tableTeilnehmer->setItem(id, i + 2, new
                    QTableWidgetItem(cBoxWunschBox[i].currentText()));
280.
281.    }
282.
283.    void QtInformationstag::on_btTeilnehmerEntfernen_clicked() {
284.        ui.tableTeilnehmer->clearContents();
285.        ui.tableTeilnehmer->setRowCount(0);
286.
287.        itsInformationstag.TeilnehmerLoeschen();
288.    }
289.
290.
291.    void QtInformationstag::on_btPlanErstellen_clicked(){
292.        on_btVeranstaltungenEntfernen_clicked(); // zunaechst das TreeFeld
                    wieder loeschen
293.
294.        // Das Fenster aktualisieren
295.        ui.labelStatus->show();
296.        ui.btPlanErstellen->setEnabled(false);
297.        ui.btAbbrechen->setEnabled(true);
298.
299.        // und dann die Aufgabe an itsThread weiterleiten
300.        if (ui.cBoxOptimal->checkState() == Qt::Checked)
301.            itsThread.setOptimal(true);
302.        else
303.            itsThread.setOptimal(false);
304.        itsThread.start();
305.    }
306.
307.    void QtInformationstag::plan_erstellt() {
308.
309.        // ist der Thread mit seiner Aufgabe fertig, wurde der Plan
                    erstellt
310.
311.        ui.treeVeranstaltungen->clear();  // Zunächst wird das Feld
                    nochmal gelöscht
312.
313.        ui.labelStatus->hide();
314.        ui.btPlanErstellen->setEnabled(true);
315.        ui.btAbbrechen->setEnabled(false);
316.
317.        // Falls er abgebrochen wurde, soll kein Plan angezeigt werden
318.        if (abgebrochen == true) {
319.            ui.labelStatus->setText(QString::fromUtf8("Berechnung
                    läuft..."));
320.            abgebrochen = false;
321.            return;
```

```
322.        }
323.
324.        // itsInformationstag auslesen
325.        int anzTermine = itsInformationstag.getAnzTermine();
326.        int anzVeranstaltungen =
                        itsInformationstag.getAnzVeranstaltungen();
327.        cVeranstaltung * veranstaltungen =
                        itsInformationstag.getVeranstaltungen(); // die
                        Veranstaltungen
328.
329.        int anzTeilnehmer;  // Die Teilnehmer pro Veranstaltunge
330.
331.        // dann den Plan im TreeWidget ausgeben
332.        QTreeWidgetItem * itemTermin;
333.        QTreeWidgetItem * itemVeranstaltung;
334.        QTreeWidgetItem * itemTeilnehmer;
335.
336.        QFont font;
337.        int gesTeilnehmer;
338.
339.        for (int i = 0; i < anzTermine; i++) {
340.             itemTermin = new QTreeWidgetItem(1);  // neues Item mit nur
                        einer Spalte
341.
342.             gesTeilnehmer = 0;
343.
344.             for (int j = 0; j < anzVeranstaltungen; j++)
345.                 if (veranstaltungen[j].getTermin() == i) {
346.                     itemVeranstaltung = new QTreeWidgetItem(1);
347.
348.                     anzTeilnehmer =
                        veranstaltungen[j].getAnzTeilnehmer();
349.                     for (int k = 0; k < anzTeilnehmer; k++) {
350.                         itemTeilnehmer = new QTreeWidgetItem(1);
351.                         itemTeilnehmer->setText(0,
                        veranstaltungen[j].getTeilnehmer(k)->getName());
352.
353.                         itemVeranstaltung->addChild(itemTeilnehmer);
354.                     }
355.
356.                     itemVeranstaltung->setText(0,
                        veranstaltungen[j].getTyp()->getName() + "    ( " +
                        QString::number(veranstaltungen[j].getAnzTeilnehmer()
                        ) + " )");
357.                     font.setBold(true);
358.                     font.setUnderline(false);
359.                     itemVeranstaltung->setFont(0, font);
360.
361.
362.                     itemTermin->addChild(itemVeranstaltung);
363.                     gesTeilnehmer +=
                        veranstaltungen[j].getAnzTeilnehmer();
364.                 }
365.
366.             itemTermin->setText(0, "Termin " + QString::number(i+1) + "
                        ( " + QString::number(itemTermin->childCount()) + "
                        -  " + QString::number(gesTeilnehmer) + " )");
367.             font.setBold(true);
368.             font.setUnderline(true);
369.             itemTermin->setFont(0,font);
370.
371.             ui.treeVeranstaltungen->addTopLevelItem(itemTermin);
```

```
372.            ui.treeVeranstaltungen->topLevelItem(i)->setExpanded(true);
373.
374.      }
375.  }
376.
377.  void QtInformationstag::on_btAbbrechen_clicked() {
378.
379.      // da es etwas dauert, bis itsThread abgebrochen ist, soll er sich
                  erstmal merken, dass abgebrochen wurde
380.      abgebrochen = true;
381.      itsThread.terminate();
382.
383.      on_btVeranstaltungenEntfernen_clicked(); // und das Feld wieder
                  geleert
384.
385.      // solange das Abbrechen dauert, wird dies angezeigt
386.      ui.labelStatus->setText("Berechnung wird abgebrochen!");
387.
388.
389.      // in der Zeit kann man keinen Button klicken
390.      ui.btPlanErstellen->setEnabled(false);
391.      ui.btAbbrechen->setEnabled(false);
392.  }
393.
394.  void QtInformationstag::on_btVeranstaltungenEntfernen_clicked() {
395.      ui.treeVeranstaltungen->clear();
396.      itsInformationstag.VeranstaltungenLoeschen();
397.  }
398.
399.
400.
401.  void QtInformationstag::on_btPrioritaetenVerwenden_clicked() {
402.
403.      int anzWuensche = itsInformationstag.getAnzWuensche();
404.
405.      int prioritaeten[anzWuensche];
406.
407.      // Auslesen
408.      for (int i = 0; i < anzWuensche; i++)
409.          prioritaeten[i] = sBoxPrioritaeten[i].value();
410.
411.      //  an itsInformationstag weiterleiten
412.      itsInformationstag.setPrioritaeten(prioritaeten);
413.  }
414.
415.  void QtInformationstag::on_btVorlageHinzufuegen_clicked() {
416.
417.      ui.btVorlageHinzufuegen->setEnabled(false);
418.
419.      // aktuelle Werte merken
420.
421.      int min = ui.sBoxMin->value();
422.      int max = ui.sBoxMax->value();
423.      QString name = ui.nameAngebot->text();
424.
425.      ////// Vorgefertige Angebote: ////////
426.
427.      ui.sBoxMin->setValue(25);
428.      ui.sBoxMax->setValue(40);
429.      ui.nameAngebot->setText("Manager");
430.      on_btAngebotHinzufuegen_clicked();
431.
```

```
432.        ui.sBoxMin->setValue(20);
433.        ui.sBoxMax->setValue(35);
434.        ui.nameAngebot->setText("Pilot");
435.        on_btAngebotHinzufuegen_clicked();
436.
437.        ui.sBoxMin->setValue(30);
438.        ui.sBoxMax->setValue(50);
439.        ui.nameAngebot->setText("Politiker");
440.        on_btAngebotHinzufuegen_clicked();
441.
442.        ui.sBoxMin->setValue(25);
443.        ui.sBoxMax->setValue(45);
444.        ui.nameAngebot->setText("Programmierer");
445.        on_btAngebotHinzufuegen_clicked();
446.
447.        ui.sBoxMin->setValue(30);
448.        ui.sBoxMax->setValue(40);
449.        ui.nameAngebot->setText("Koordinator");
450.        on_btAngebotHinzufuegen_clicked();
451.
452.        ui.sBoxMin->setValue(20);
453.        ui.sBoxMax->setValue(30);
454.        ui.nameAngebot->setText("Schauspieler");
455.        on_btAngebotHinzufuegen_clicked();
456.
457.        ui.sBoxMin->setValue(20);
458.        ui.sBoxMax->setValue(35);
459.        ui.nameAngebot->setText(QString::fromUtf8("Gärtner"));
460.        on_btAngebotHinzufuegen_clicked();
461.
462.        ui.sBoxMin->setValue(20);
463.        ui.sBoxMax->setValue(40);
464.        ui.nameAngebot->setText("Architekt");
465.        on_btAngebotHinzufuegen_clicked();
466.
467.        ui.sBoxMin->setValue(25);
468.        ui.sBoxMax->setValue(45);
469.        ui.nameAngebot->setText("Designer");
470.        on_btAngebotHinzufuegen_clicked();
471.
472.        ui.sBoxMin->setValue(30);
473.        ui.sBoxMax->setValue(45);
474.        ui.nameAngebot->setText(QString::fromUtf8("Künstler"));
475.        on_btAngebotHinzufuegen_clicked();
476.
477.        ui.sBoxMin->setValue(30);
478.        ui.sBoxMax->setValue(60);
479.        ui.nameAngebot->setText("Reporter");
480.        on_btAngebotHinzufuegen_clicked();
481.
482.        ui.sBoxMin->setValue(20);
483.        ui.sBoxMax->setValue(40);
484.        ui.nameAngebot->setText("Journalist");
485.        on_btAngebotHinzufuegen_clicked();
486.
487.        ui.sBoxMin->setValue(25);
488.        ui.sBoxMax->setValue(40);
489.        ui.nameAngebot->setText("Schriftsteller");
490.        on_btAngebotHinzufuegen_clicked();
491.
492.        ui.sBoxMin->setValue(30);
493.        ui.sBoxMax->setValue(50);
```

```
494.        ui.nameAngebot->setText("Verleger");
495.        on_btAngebotHinzufuegen_clicked();
496.
497.        ui.sBoxMin->setValue(30);
498.        ui.sBoxMax->setValue(55);
499.        ui.nameAngebot->setText("Erzieher");
500.        on_btAngebotHinzufuegen_clicked();
501.
502.        ui.sBoxMin->setValue(20);
503.        ui.sBoxMax->setValue(40);
504.        ui.nameAngebot->setText("Arzt");
505.        on_btAngebotHinzufuegen_clicked();
506.
507.        // alten Werte wieder laden
508.        ui.sBoxMin->setValue(min);
509.        ui.sBoxMax->setValue(max);
510.        ui.nameAngebot->setText(name);
511.   }
512.
513.   void QtInformationstag::on_btZufaellig1_clicked() {
514.
515.        int anzWuensche = itsInformationstag.getAnzWuensche();
516.
517.        int anzahlAngebote = ui.tableAngebote->rowCount();
518.        if (anzahlAngebote < anzWuensche)  // wenns zu wenige sind, soll
                    abgebrochen werden
519.             return;
520.
521.        ui.nameTeilnehmer->setText( QString::fromUtf8("Schüler ") +
                    QString::number(ui.tableTeilnehmer->rowCount() + 1));
522.
523.        int wunsch[anzWuensche];
524.
525.        for (int i = 0; i < anzWuensche; i++) { // fuer jeden Wunsch
526.             wunsch[i] = (rand() % anzahlAngebote) + 1; // soll solange
                    zufaellig ein Wunsch ausgewaehlt werden
527.
528.             for (int j = 0; j < i; j++)
529.                  if (wunsch[j] == wunsch[i]) { // bis er sich von allen
                         vorigen Wuensche unterscheidet
530.                       i--;  // ansonsten muss ein neuer Wunsch zufaellig
                         ermittelt werden
531.                       break;
532.                  }
533.        }
534.
535.        for (int i = 0; i < anzWuensche; i++)  // die Wuensche in die
                    ComboBox eintragen
536.             cBoxWunschBox[i].setCurrentIndex(wunsch[i]);
537.
538.   }
539.
540.   void QtInformationstag::on_btZufaellig10_clicked() {
541.
542.        int anzWuensche = itsInformationstag.getAnzWuensche();
543.
544.        // alte Werte merken
545.        QString currentName = ui.nameTeilnehmer->text();
546.        int currentIndex[anzWuensche];
547.
548.        for (int i = 0; i < anzWuensche; i++)
549.             currentIndex[i] = cBoxWunschBox[i].currentIndex();
```

```
550.
551.        if( ui.tableAngebote->rowCount() < anzWuensche)
552.             return;
553.
554.        // 10-mal Zufaellig einen Teilnehmer erstellen
555.        for (int i = 0; i < 10; i++) {
556.             on_btZufaellig1_clicked();
557.             on_btTeilnehmerHinzufuegen_clicked();
558.        }
559.
560.
561.        // Alten Werte wieder erstellen
562.        ui.nameTeilnehmer->setText(currentName);
563.
564.        for (int i = 0; i < anzWuensche; i++)
565.             cBoxWunschBox[i].setCurrentIndex(currentIndex[i]);
566.
567.   }
568.
569.
570.
571.   void QtInformationstag::when_sBoxMax_changed(int value) {
572.        // Min immer kleiner als Max
573.        if (ui.sBoxMin->value() >= value)
574.             ui.sBoxMin->setValue(value - 1);
575.   }
576.
577.   void QtInformationstag::when_sBoxMin_changed(int value) {
578.        // Max immer groesser als Min
579.        if (ui.sBoxMax->value() <= value)
580.             ui.sBoxMax->setValue(value + 1);
581.   }
582.
583.
584.   void QtInformationstag::when_sBoxAnzTermine_changed(int value) {
585.        // AnzWuensche immer groesser gleich AnzTermine
586.        if (ui.sBoxAnzWuensche->value() < value)
587.             ui.sBoxAnzWuensche->setValue(value);
588.   }
589.
590.   void QtInformationstag::when_sBoxAnzWuensche_changed(int value) {
591.        // anz Termine immer kleiner gleich AnzWuensche
592.        if (ui.sBoxAnzTermine->value() > value)
593.             ui.sBoxAnzTermine->setValue(value);
594.   }
595.
596.
597.
598.   void QtInformationstag::btAngebotHinzufuegen_deaktivieren(QString text)
                        {
599.        // wenn kein Name eingetragen wurde, soll der Button deaktiviert
                        werden
600.        if (text != "")
601.             ui.btAngebotHinzufuegen->setEnabled(true);
602.        else
603.             ui.btAngebotHinzufuegen->setEnabled(false);
604.   }
605.
606.   void QtInformationstag::btTeilnehmerHinzufuegen_deaktivieren(QString
                       text) {
607.        // wenn kein Name eingetragen wurde, oder ein Wunsch doppelt oder
                        gar nicht ausgewaehlt wurde, soll der Button
```

```
                    deaktivert werden
608.
609.        int anzWuensche = itsInformationstag.getAnzWuensche();
610.        bool indexdoppelt = false;
611.
612.        for (int i = 0; i < anzWuensche; i++)
613.             if (IndexDoppelt(cBoxWunschBox[i].currentIndex()) == true ||
                        cBoxWunschBox[i].currentIndex() == 0) // wenn ein
                        Wunsch doppelt oder gar nicht ausgewaehlt wurde
614.                 indexdoppelt = true;
615.
616.        if (text != "" && indexdoppelt == false)
617.             ui.btTeilnehmerHinzufuegen->setEnabled(true);
618.        else
619.             ui.btTeilnehmerHinzufuegen->setEnabled(false);
620.
621.   }
622.
623.   void QtInformationstag::btTeilnehmerHinzufuegen_deaktivieren(int nix) {
624.        // Damit der Slot aufgerufen werden kann, wenn ein Index von einer
                        der WunschBoxen geaendert wurde
625.        if (nix != -1) // bedingung stimmt immmer! nur zum Benutzern von
                        nix und umgehen eines Errors!
626.             btTeilnehmerHinzufuegen_deaktivieren(ui.nameTeilnehmer-
                        >text()); // leitet die Aufgabe an den anderen Slot
                        weiter
627.   }
628.
629.   void QtInformationstag::when_AnzAngebote_changed() {
630.        // wenn noch zu wenig Angebote bestehen, soll man nicht zufaellig
                        Teilnehmer erstellen duerfen
631.        if( ui.tableAngebote->rowCount() >=
                        itsInformationstag.getAnzWuensche()) {
632.             ui.btZufaellig1->setEnabled(true);
633.             ui.btZufaellig10->setEnabled(true);
634.        }
635.        else {
636.             ui.btZufaellig1->setEnabled(false);
637.             ui.btZufaellig10->setEnabled(false);
638.        }
639.   }
640.
641.   void QtInformationstag::wunschBox_index_changed(int index) {
642.        // Wenn ein Index einer wunschBox geaendert wurde, soll geguckt
                        werden, ob ein Index doppelt existiert und dann rot
                        einfaerben
643.        int anzWuensche = itsInformationstag.getAnzWuensche();
644.        QPalette standardPalette;
645.        QPalette rotePalette;
646.        rotePalette.setColor(QPalette::ButtonText, Qt::red);
647.        for (int i = 0; i < anzWuensche; i++) {
648.             index = cBoxWunschBox[i].currentIndex();
649.             if (IndexDoppelt(index) == true) {  // wenn es den Index
                        doppelt gibt, sollen alle Rot eingefaerbt werden
650.                 for (int j = 0; j < anzWuensche; j++)
651.                      if (cBoxWunschBox[j].currentIndex() == index)
652.                           cBoxWunschBox[j].setPalette(rotePalette);
653.             }
654.             else
655.                 cBoxWunschBox[i].setPalette(standardPalette);
656.        }
657.   }
```

```
658.
659.
660.  bool QtInformationstag::IndexDoppelt(int index) {
661.       // Die Hilfsmethode prueft, ob der uebergebene Index doppelt
                  existiert
662.       if (index == 0) // 0 ist eine ausnahme
663.            return false;
664.
665.       int anzahl = 0;
666.
667.       int anzWuensche = itsInformationstag.getAnzWuensche();
668.
669.       for (int i = 0; i < anzWuensche; i++)
670.            if (cBoxWunschBox[i].currentIndex() == index)
671.                anzahl++;
672.
673.       if (anzahl > 1) // wenn er haeufiger als einmal vorgekommen ist,
                  so gibt es ihn doppelt
674.            return true;
675.       else
676.            return false;
677.  }
678.
```

4.4 cThread.h

```
1.  #ifndef CTHREAD_H_
2.  #define CTHREAD_H
3.
4.  #include <QThread>
5.
6.  #include "cInformationstag.h"
7.
8.  class cThread : public QThread
9.  {
10. public:
11.       // zum initialisieren
12.       void setInformationstag(cInformationstag * informationstag)
                        { itsInformationstag = informationstag; }
13.       void setOptimal(bool optimal) { itsOptimal = optimal; };
14.
15.       void run();  // wird ausgefuhert wenn der Thread gestartet wird
16.
17. private:
18.
19.       cInformationstag * itsInformationstag;
20.       bool itsOptimal;
21.
22. };
23.
24.
25. #endif /*CTHREAD_H_*/
26.
```

4.5 cThread.cpp

```
1.  #include "cThread.h"
2.
3.  void cThread::run()
4.  {
5.       itsInformationstag->PlanErstellen(itsOptimal);
6.
7.        cout << "BERECHNUNGEN FERTIG!!!" << endl;
8.  }
9.
```

4.6 cInformationstag.h

```
1.  #ifndef CINFORMATIONSTAG_H_
2.  #define CINFORMATIONSTAG_H_
3.
4.  #include <glpk.h> // glpk wird hier eingebunden
5.  #include <QString>
6.
7.  // eigene
8.  #include "cAngebot.h"
9.  #include "cVeranstaltung.h"
10. #include "cTeilnehmer.h"
11.
12. class cInformationstag
13. {
14. public:
15.       cInformationstag(int anzTermine = 4, int anzWuensche = 6, int *
                  prioritaeten = 0);
16.       ~cInformationstag();
17.
18.        // Hinzufuege-Methoden geben jeweils die ID zurueck! Beim Fehler
                  -1!!
19.       int AngebotHinzufuegen(int min, int max, QString name =
                  "Angebot");
20.       int TeilnehmerHinzufuegen(int * wuensche, QString name =
                  QString::fromUtf8("Schüler"));
21.
22.       // Loeschenmethoden
23.       void AllesLoeschen();
24.       void TeilnehmerLoeschen();
25.       void VeranstaltungenLoeschen();
26.
27.
28.       void Ausgeben(); // fuer die Konsole
29.
30.       void PlanErstellen(bool optimal = false); // Schnittstelle fuer
                  cThread, leitet die Aufgabe an den richtigen
                  Algorithmus weiter standardmaessig wird der schnelle
                  Algorithmus verwendet
31.
32.
33.       void setAnzTermine(int anzTermine);
34.       void setAnzWuensche(int anzWuensche);
35.       void setPrioritaeten(int * prioritaeten = 0);  // bei 0 werden die
                  Prioritaeten standardmaeÃŸig eingestellt
36.             // Achtung: Prioritaeten muessen in einem Array
                  gespeichert sein, der die Laenge von itsAnzWuensche
                  hat
37.
38.       int getAnzTermine() {return itsAnzTermine; }
39.       int getAnzWuensche() {return itsAnzWuensche; }
40.       int * getPrioritaeten() {return itsPrioritaeten; }
41.
42.       cAngebot * getAngebote() {return itsAngebote; }
43.       int getAnzAngeobte() {return itsAnzAngebote; }
44.
45.       cVeranstaltung * getVeranstaltungen() {return
                  itsVeranstaltungen; }
46.       int getAnzVeranstaltungen() {return itsAnzVeranstaltungen; }
47.
```

```
48.         cTeilnehmer * getTeilnehmer() {return itsTeilnehmer; }
49.         int getAnzTeilnehmer() {return itsAnzTeilnehmer; }
50.
51.   private:
52.
53.         int VeranstaltungHinzufuegen(cAngebot * typ, int termin);
54.         void TeilnehmerZuweisen(int teilnehmer, int angebot, int termin);
                        // Weist einen Teilnehmer einer Veranstaltung zu,
                        wenn diese schon existiert, sonsten wird diese
                        erstellt
55.
56.
57.         void PlanErstellenSchnell();
58.         void PlanErstellenOptimal();
59.
60.         int itsAnzTermine;
61.         int itsAnzWuensche;
62.         int * itsPrioritaeten;  // Prioritaet[0] ist fuer den obersten
                        Wunsch
63.
64.
65.         cAngebot * itsAngebote;
66.         int itsAnzAngebote;
67.
68.         cVeranstaltung * itsVeranstaltungen;
69.         int itsAnzVeranstaltungen;
70.
71.         cTeilnehmer * itsTeilnehmer;
72.         int itsAnzTeilnehmer;
73.
74.
75.         // ein paar Arrays zum speichern, welche Sachen schon mit dem
                        PlanErstellen() zugeordnet wurden
76.         bool * TeilWue;  // speichert in einem Array welche Wueschne von
                        welchen Teilnehmern schon erfuellt wurden
77.         bool * TeilTer;  // speichert in einem Array welche Termine schon
                        von welchen Teilnehmer belegt wurden
78.         bool * AngTer; // speichert in einem Array welche Termine schon
                        von welchen Angeboten belegt wurden
79.
80.         void GespeichertesLoeschen();  // setzt alle oberen Arrays wieder
                        auf die gewuenschte laenge und auf false
81.   };
82.
83.   #endif /*CINFORMATIONSTAG_H_*/
84.
```

4.7 cInformationstag.cpp

```
1.  #include "cInformationstag.h"
2.
3.  cInformationstag::cInformationstag(int anzTermine, int anzWuensche, int
                    * prioritaeten) {
4.        if (anzTermine > anzWuensche)
5.             cout << "ERROR! Es darf nicht mehr Termine als Wuensche
                    geben)" << endl;
6.
7.        itsAnzTermine = anzTermine;
8.        itsAnzWuensche = anzWuensche;
9.
10.       itsPrioritaeten = new int[itsAnzWuensche];
11.
12.       if (prioritaeten == 0)
13.            for (int i = 0; i < itsAnzWuensche; i++)
14.                 itsPrioritaeten[i] = 10 + itsAnzWuensche -i;  //
                        Proritaeten auf Standardwerte
15.       else
16.            for (int i = 0; i < itsAnzWuensche; i++)
17.                 itsPrioritaeten[i] = prioritaeten[i];
18.
19.       itsAngebote = new cAngebot[0];
20.       itsVeranstaltungen = new cVeranstaltung[0];
21.       itsTeilnehmer = new cTeilnehmer[0];
22.
23.
24.       itsAnzAngebote = 0;
25.       itsAnzVeranstaltungen = 0;
26.       itsAnzTeilnehmer = 0;
27.
28.       // hier werden die Arrays zum merken fuer den schnellen
                    Algorithmus erstellt und mit false gefuellt
29.       TeilWue = new bool[itsAnzTeilnehmer*itsAnzWuensche];
30.       TeilTer = new bool[itsAnzTeilnehmer*itsAnzTermine];
31.       AngTer = new bool[itsAnzAngebote*itsAnzTermine];
32.
33.       for (int i = 0 ; i < itsAnzTeilnehmer; i++) {
34.            for (int j = 0; j < itsAnzWuensche; j++)
35.                 TeilWue[i*itsAnzWuensche + j] = false;
36.            for (int j = 0; j < itsAnzTermine; j++)
37.                 TeilTer[i*itsAnzTermine + j] = false;
38.       }
39.       for (int i = 0; i < itsAnzAngebote; i++)
40.            for (int j = 0; j < itsAnzTermine; j++)
41.                 AngTer[i*itsAnzTermine + j] = false;
42. }
43.
44. cInformationstag::~cInformationstag() {
45.
46.       // Alle Arrays wieder freigeben
47.       delete [] itsPrioritaeten;
48.       delete [] itsAngebote;
49.       delete [] itsVeranstaltungen;
50.       delete [] itsTeilnehmer;
51.
52.       delete [] TeilWue;
53.       delete [] TeilTer;
54.       delete [] AngTer;
```

```
55.  }
56.
57.  void cInformationstag::GespeichertesLoeschen() {
58.       // hier werden die Merk-Arrays erneut erstellt und mit false
                  beschrieben
59.       delete [] TeilWue;
60.       delete [] TeilTer;
61.       delete [] AngTer;
62.
63.
64.       TeilWue = new bool[itsAnzTeilnehmer*itsAnzWuensche];
65.       TeilTer = new bool[itsAnzTeilnehmer*itsAnzTermine];
66.       AngTer = new bool[itsAnzAngebote*itsAnzTermine];
67.
68.       for (int i = 0 ; i < itsAnzTeilnehmer; i++) {
69.            for (int j = 0; j < itsAnzWuensche; j++)
70.                 TeilWue[i*itsAnzWuensche + j] = false;
71.            for (int j = 0; j < itsAnzTermine; j++)
72.                 TeilTer[i*itsAnzTermine + j] = false;
73.       }
74.       for (int i = 0; i < itsAnzAngebote; i++)
75.            for (int j = 0; j < itsAnzTermine; j++)
76.                 AngTer[i*itsAnzTermine + j] = false;
77.  }
78.
79.  int cInformationstag::AngebotHinzufuegen(int min, int max, QString name)
                  {
80.       if (min >= max) // min muss immer echt kleiner als max sein
81.            return -1; // Dann fehler zurueckgeben
82.
83.       itsAnzAngebote++;
84.       cAngebot * tempAngebote = new cAngebot[itsAnzAngebote];
85.       for (int i = 0; i < itsAnzAngebote -1; i++)
86.            tempAngebote[i] = itsAngebote[i];
87.       delete [] itsAngebote;
88.       tempAngebote[itsAnzAngebote -1] = cAngebot(itsAnzAngebote-1, min,
                  max, name);
89.       itsAngebote = tempAngebote;
90.
91.       GespeichertesLoeschen();
92.
93.       return itsAnzAngebote-1; // id wird zuueckgegeben
94.
95.
96.  }
97.
98.  int cInformationstag::TeilnehmerHinzufuegen(int * wuensche, QString
                  name) {
99.
100.      if (itsAnzAngebote < itsAnzWuensche) // Wenn es gar nicht
                  genuegend Angebote gibt
101.           return -1; // Fehler zurueckgeben
102.
103.      cAngebot * tempWuensche = new cAngebot[itsAnzWuensche];
104.      for (int i = 0; i < itsAnzWuensche; i++)
105.           if (wuensche[i] <= itsAnzAngebote)
106.                tempWuensche[i] = itsAngebote[wuensche[i]];
107.           else
108.                return -1; // sollte ein Wunsch nicht guteltig sein, wird
                       -1 zurueckgegeben.
109.
110.      itsAnzTeilnehmer++;
```

```
111.        cTeilnehmer * tempTeilnehmer = new cTeilnehmer[itsAnzTeilnehmer
                    -1];
112.        for (int i = 0; i < itsAnzTeilnehmer - 1; i++)
113.             tempTeilnehmer[i] = itsTeilnehmer[i];   // alte Teilnehmer
                        kopieren
114.
115.        delete [] itsTeilnehmer;
116.
117.
118.        itsTeilnehmer = new cTeilnehmer[itsAnzTeilnehmer];
119.        for (int i = 0; i < itsAnzTeilnehmer - 1; i++)
120.             itsTeilnehmer[i] = tempTeilnehmer[i];   // alte Teilnehmer
                        kopieren
121.        delete [] tempTeilnehmer;
122.
123.        itsTeilnehmer[itsAnzTeilnehmer - 1] =
                        cTeilnehmer(itsAnzTeilnehmer-1, itsAnzWuensche,
                        tempWuensche, name); // neuen Teilnehmer hinzufuegen
124.
125.        GespeichertesLoeschen();
126.
127.        return itsAnzTeilnehmer -1; // id wird zurueckgegeben
128.   }
129.
130.   int cInformationstag::VeranstaltungHinzufuegen(cAngebot * typ, int
                        termin) {
131.        itsAnzVeranstaltungen++;
132.        cVeranstaltung * tempVeranstaltungen = new
                        cVeranstaltung[itsAnzVeranstaltungen];
133.        for (int i = 0; i < itsAnzVeranstaltungen - 1; i++)
134.             tempVeranstaltungen[i] = itsVeranstaltungen[i];
135.        delete [] itsVeranstaltungen;
136.        tempVeranstaltungen[itsAnzVeranstaltungen - 1] =
                        cVeranstaltung(itsAnzVeranstaltungen-1, typ, termin);
                        // neue Veranstaltung wird erstellt
137.        itsVeranstaltungen = tempVeranstaltungen;
138.        return itsAnzVeranstaltungen -1; // id wird zurueckgegeben
139.   }
140.
141.   void cInformationstag::TeilnehmerZuweisen(int teilnehmer, int angebot,
                        int termin) {
142.        // nunaechst pruefen, ob die Veranstaltung schon existiert:
143.        bool VeranstaltungSchonDa = false;  // ob die Veranstaltung schon
                        existiert
144.        bool TeilnehmerSchonDa = false;// ob der Teilnehmer in der
                        Veranstaltung schon existiert
145.        int id = 0;
146.
147.        for (int i = 0; i < itsAnzVeranstaltungen; i++)
148.             if (itsVeranstaltungen[i].getTyp()->getId() == angebot &&
                             itsVeranstaltungen[i].getTermin() == termin) {
149.                  VeranstaltungSchonDa = true;
150.                  id = i;
151.             }
152.
153.        int anzahlTeilnehmer;
154.
155.
156.        if (VeranstaltungSchonDa == false)  // wenn sie noch nicht
                        existiert, soll sie hinzugefuegt werden
157.             id = VeranstaltungHinzufuegen(&itsAngebote[angebot], termin);
158.        else {  // wenn sie schon existiert hat, soll geguckt werden, ob
```

```
                    der Teilnehmer dadrin schon vorhanden ist
159.            anzahlTeilnehmer = itsVeranstaltungen[id].getAnzTeilnehmer();
160.            for (int i = 0; i < anzahlTeilnehmer; i++)
161.                if (itsVeranstaltungen[id].getTeilnehmer(i)->getId() ==
                        teilnehmer)
162.                    TeilnehmerSchonDa = true;
163.        }
164.
165.
166.        if (TeilnehmerSchonDa == false) // Wenn der Teilnehmer in der
                    Veranstaltung noch nicht aufgefuert wird, soll er
                    dort hinzugefuegt werden
167.
                    itsVeranstaltungen[id].TeilnehmerHinzufuegen(&itsTeil
                    nehmer[teilnehmer]);
168.
169.    }
170.
171.
172.    void cInformationstag::AllesLoeschen() {
173.
174.        // zunaechst die Angebote loeschen
175.        delete [] itsAngebote;
176.        itsAngebote = new cAngebot[0];
177.        itsAnzAngebote = 0;
178.
179.        // danach auch alles andere
180.        TeilnehmerLoeschen();
181.        GespeichertesLoeschen();
182.    }
183.
184.    void cInformationstag::TeilnehmerLoeschen() {
185.        // zunaechst die Teilnehmer loeschen
186.        delete [] itsTeilnehmer;
187.        itsTeilnehmer = new cTeilnehmer[0];
188.        itsAnzTeilnehmer = 0;
189.
190.        // danach auf die Veranstaltungen loeschen
191.        VeranstaltungenLoeschen();
192.        GespeichertesLoeschen();
193.    }
194.
195.    void cInformationstag::VeranstaltungenLoeschen() {
196.        delete [] itsVeranstaltungen;
197.
198.        itsVeranstaltungen = new cVeranstaltung[0];
199.
200.        itsAnzVeranstaltungen = 0;
201.        GespeichertesLoeschen();
202.    }
203.
204.    void cInformationstag::Ausgeben() { // in die Konsole ausgeben
205.        for (int i = 0; i < itsAnzAngebote; i++)
206.            itsAngebote[i].Ausgeben();
207.        cout << "itsAnzTeilnehmer: " << itsAnzTeilnehmer << endl;
208.        for (int i = 0; i < itsAnzTeilnehmer; i++)
209.            itsTeilnehmer[i].Ausgeben();
210.
211.        cout << "itsAnzVeranstaltungen: " << itsAnzVeranstaltungen <<
                    endl;
212.        for (int i = 0; i < itsAnzVeranstaltungen; i++)
```

```
213.            itsVeranstaltungen[i].Ausgeben();
214.    }
215.
216.    void cInformationstag::setAnzTermine(int anzTermine) {
217.          itsAnzTermine = anzTermine;
218.
219.          // wurde die Anzahl der Termine geaendert muessen leider die
                        Teilnehmer und die Mark-Arrays geloescht werden
220.          TeilnehmerLoeschen();
221.          GespeichertesLoeschen();
222.    }
223.    void cInformationstag::setAnzWuensche(int anzWuensche) {
224.          itsAnzWuensche = anzWuensche;
225.          TeilnehmerLoeschen();  // bei einer neuen Anzahl an Wuenschen,
                        sind die alten Teilnehmer hinfaellig
226.
227.          // neuen Prioritaeten-Array erstellen
228.          delete [] itsPrioritaeten;
229.          itsPrioritaeten = new int[itsAnzWuensche];
230.
231.          setPrioritaeten(); // Prioritaeten neu beschreiben (mit
                        Standardwerten)
232.
233.          GespeichertesLoeschen();
234.
235.    }
236.
237.    void cInformationstag::setPrioritaeten(int * prioritaeten) {
238.
239.          if (prioritaeten == 0)  // bei 0 sollen Standardwerte benutzt
                        werden
240.               for (int i = 0; i < itsAnzWuensche; i++)
241.                    itsPrioritaeten[i] = 10 + itsAnzWuensche -i;
242.          else  // Ansonsten wird der Parameter ausgelesen
243.               for (int i = 0; i < itsAnzWuensche; i++)
244.                    itsPrioritaeten[i] = prioritaeten[i];
245.
246.          GespeichertesLoeschen();
247.    }
248.
249.    void cInformationstag::PlanErstellen(bool optimal) {
250.
251.          // Die alte Ausgabe loeschen
252.          VeranstaltungenLoeschen();
253.          GespeichertesLoeschen();
254.          // Fehler vorbeugen
255.          if (itsAnzTeilnehmer <= 0) {
256.               cout << "Fehler bei der Anzahl der Teilnehmer!" << endl;
257.               return;
258.          }
259.          if (itsAnzAngebote <= 0) {
260.               cout << "Fehler bei der Anzahl der Angebote!" << endl;
261.               return;
262.          }
263.          if (itsAnzTermine <= 0) {
264.               cout << "Fehler bei der Anzahl der Termine!" << endl;
265.               return;
266.          }
267.          if (itsAnzWuensche <= 0) {
268.               cout << "Fehler bei der Anzahl der Wuensche!" << endl;
269.               return;
270.          }
```

```
271.
272.        if (optimal == true)
273.             PlanErstellenOptimal();
274.        else
275.             PlanErstellenSchnell();
276.
277.   }
278.
279.
280.   void cInformationstag::PlanErstellenOptimal() {
281.
282.        cout << "Optimal-Plan-Erstellen wird geladen!" << endl;
283.
284.        // Angebote auslesen:
285.
286.        double min[itsAnzAngebote];
287.        double max[itsAnzAngebote];
288.
289.        for (int i = 0; i < itsAnzAngebote; i++) {
290.             min[i] = itsAngebote[i].getMin();
291.             max[i] = itsAngebote[i].getMax();
292.        }
293.
294.        // Teilnehmer auslesen:
295.        cAngebot * wuensche;
296.        int priori[itsAnzTeilnehmer][itsAnzAngebote];
297.
298.        // zunaechst mit 0en fuellen:
299.        for (int i = 0; i < itsAnzTeilnehmer; i++)
300.             for (int j = 0; j < itsAnzAngebote; j++)
301.                  priori[i][j] = 0;
302.
303.
304.        for(int i = 0; i < itsAnzTeilnehmer; i++) { // Alle Teilnehmer der
                       Reihe nach durchgehen
305.             wuensche = itsTeilnehmer[i].getWuensche();
306.             for(int j = 0; j < itsAnzAngebote; j++)  // Alle Angebote
                       durchgehen
307.                  for(int k = 0; k < itsAnzWuensche; k++)  // Alle Wuensche
                       des Teilnehmer durchgehen
308.
309.                       if (wuensche[k].getId() == j)  // Wenn beim akt
                       Schueler der akt Wunsch mit dem akt Angebot
                       uebereinstimmt
310.                            priori[i][j] = itsPrioritaeten[k];  // dann
                       soll in die Matrix die entsprechende Prioritaetszahl
                       geschrieben werden
311.        }
312.
313.
314.        // Das Problem formulieren
315.        int AnzH1 = itsAnzAngebote * itsAnzTeilnehmer;
316.        int AnzH2 = itsAnzAngebote * itsAnzTermine;
317.        int AnzH3 = itsAnzTeilnehmer * itsAnzTermine;
318.
319.        int AnzZeilen = AnzH1 + AnzH2 + AnzH3;
320.
321.
322.
323.        glp_prob * meinProblem;
324.        meinProblem = glp_create_prob();
325.        glp_set_prob_name(meinProblem, "Informationstag");
```

```
326.        glp_set_obj_dir(meinProblem, GLP_MAX);
327.
328.        QString name;  // dadrin wird der name der Zeile bzw. Spalte
                           gespeichert.
329.        int nummer;  // hier wird die jeweilige Nummer der Zeile bzw.
                           Spalte gespeichert. Dient beides nur der Uebersicht
330.
331.
332.        // Zeilen (Beschrenkungen/Hilfsvariablen Hs)
333.        glp_add_rows(meinProblem, AnzZeilen);
334.
335.
336.        for (int i = 0; i < itsAnzTeilnehmer; i++)
337.             for (int j = 0; j < itsAnzAngebote; j++) {
338.                  name = "H1_" + QString::number(i) + "_" +
                         QString::number(j);
339.                  nummer = i * itsAnzAngebote + j + 1;
340.                  glp_set_row_name(meinProblem, nummer, name.toLatin1());
341.                  glp_set_row_bnds(meinProblem,nummer, GLP_DB, 0, 1);
342.             }
343.
344.        for (int j = 0; j < itsAnzAngebote; j++)
345.             for (int k = 0; k < itsAnzTermine; k++) {
346.                  name = "H2_" + QString::number(j) + "_" +
                         QString::number(k);
347.                  nummer = AnzH1 + j * itsAnzTermine + k + 1;
348.                  glp_set_row_name(meinProblem, nummer, name.toLatin1());
349.                  glp_set_row_bnds(meinProblem, nummer, GLP_DB, min[j],
                         max[j]);
350.             }
351.
352.        for (int i = 0; i < itsAnzTeilnehmer; i++)
353.             for (int k = 0; k < itsAnzTermine; k++) {
354.                  name = "H3_" + QString::number(i) + "_" +
                         QString::number(k);
355.                  nummer = AnzH1 + AnzH2 + i * itsAnzTermine + k + 1;
356.                  glp_set_row_name(meinProblem, nummer, name.toLatin1());
357.                  glp_set_row_bnds(meinProblem, nummer, GLP_DB, 0, 1);
358.             }
359.
360.        // Spalten (structural variables) Die, die vom Programm bestimmt
                   werden muessen (binaere Varibalen
361.        // Bools: Sind die binaeren Variablen die das Programm festlegt
                   Bool_1_2_3 == 1 bedeutet, das Schüler 1 Angebot 2 zu
                   Termin 3 besucht
362.
363.        int AnzHauptBools = itsAnzTeilnehmer * itsAnzWuensche *
                   itsAnzTermine; // Da Schueler keine Veranstaltung die
                   Besuchen, die sie nicht auf gewaelt haben, wird hier
                   nicht AnzAngebote benutzt, sondern nur die Anzahl der
                   Wuensche
364.        int AnzHilfsBools = itsAnzAngebote * itsAnzTermine;  // ein paar
                   Hilfs-Strukturale-Varibalen mit denen eine leere
                   Veranstaltung direkt als voll behandelt werden kann
365.        int AnzSpalten = AnzHauptBools + AnzHilfsBools;
366.
367.        glp_add_cols(meinProblem, AnzSpalten);
368.
369.
370.        // nun werden alle Bools hinzugefuegt und mit dem entsprechenden
                   Koeffizenten fuer die Zielfunktion verknuepft
371.        for (int i = 0; i < itsAnzTeilnehmer; i++)
```

```
372.            for (int j = 0, jwunsch = 0; j < itsAnzAngebote; j++)
373.                 if (priori[i][j] != 0)  // wenn der Schueler i das
                         Angebot j gewaehlt hat
374.                 {
375.                      for (int k = 0; k < itsAnzTermine; k++) {
376.                           name = "Bool_" + QString::number(i) + "_" +
                         QString::number(j) + "_" + QString::number(k);
377.                           nummer = i * itsAnzWuensche * itsAnzTermine +
                         jwunsch * itsAnzTermine + k + 1;
378.                           glp_set_col_name(meinProblem, nummer,
                         name.toLatin1());
379.                           glp_set_col_kind(meinProblem, nummer, GLP_BV);
                         // mach die Variable binaer
380.                           glp_set_obj_coef(meinProblem, nummer, priori[i]
                         [j]); // der Koeffizient fuer die Zielfunktion
381.                      }
382.                      jwunsch++;
383.                 }
384.
385.        // Hilfsbools
386.        for (int j = 0; j < itsAnzAngebote; j++)
387.             for (int k = 0; k < itsAnzTermine; k++) {
388.                 name = "NichtStatt_" + QString::number(j) + "_" +
                         QString::number(k);
389.                 nummer = AnzHauptBools + j * itsAnzTermine + k + 1;
390.                 glp_set_col_name(meinProblem, nummer, name.toLatin1());
391.                 glp_set_col_kind(meinProblem, nummer, GLP_BV); // mach
                         die Variable binaer
392.                 glp_set_obj_coef(meinProblem, nummer, 0); // diese haben
                         keinen Einfluss auf die Zielfunktion
393.             }
394.
395.
396.
397.
398.        // Als letztes muss die Matrix geladen werden, diese muss dafuer
                     in ein Eindemensionales Feld gepresst werden:
399.        int AnzFelder = AnzZeilen * AnzSpalten;
400.
401.        int *zeile = new int[AnzFelder + 1];
402.        int *spalte = new int[AnzFelder + 1];
403.        double *matrix = new double[AnzFelder + 1];
404.
405.        zeile[0] = 0;
406.        spalte[0] = 0;
407.        matrix[0] = 0;
408.
409.        int aktSpalte;
410.        int aktZeile;
411.
412.        int index;
413.        // alle Spalten durchgehen
414.        for (int i1 = 0; i1 < itsAnzTeilnehmer; i1++)
415.             for (int j1 = 0, jwunsch = 0; j1 < itsAnzAngebote; j1++)
416.                 if (priori[i1][j1] != 0) {  // wenn der Schueler i das
                         Angebot j gewaehlt hat
417.                      for (int k1 = 0; k1 < itsAnzTermine; k1++) {
418.                           aktSpalte = i1 * itsAnzWuensche * itsAnzTermine
                         + jwunsch * itsAnzTermine + k1;
419.
420.
421.
```

```
422.                         // nun alle Zeilen durchgehen
423.                         for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
424.                             for (int j2 = 0; j2 < itsAnzAngebote; j2++)
                        {
425.                                 aktZeile = i2 * itsAnzAngebote + j2;
426.
427.                                 index = aktSpalte * AnzZeilen +
                        aktZeile + 1;
428.                                 zeile[index] = aktZeile + 1;
429.                                 spalte[index] = aktSpalte + 1;
430.                                 if (i1 == i2 && j1 == j2)  // wenn es
                        sich bei der Spalte und der Reihe um die gleichen
                        Indizes handelt
431.                                     matrix[index] = 1;
432.                                 else
433.                                     matrix[index] = 0;
434.                             }
435.
436.                         for (int j2 = 0; j2 < itsAnzAngebote; j2++)
437.                             for (int k2 = 0; k2 < itsAnzTermine; k2++)
                        {
438.                                 aktZeile = AnzH1 + j2 * itsAnzTermine +
                        k2;
439.                                 index = aktSpalte * AnzZeilen +
                        aktZeile + 1;
440.                                 zeile[index] = aktZeile + 1;
441.                                 spalte[index] = aktSpalte + 1;
442.                                 if (j1 == j2 && k1 == k2)  // wenn es
                        sich bei der Spalte und der Reihe um die gleichen
                        Indizes handelt
443.                                     matrix[index] = 1;
444.                                 else
445.                                     matrix[index] = 0;
446.                             }
447.
448.                         for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
449.                             for (int k2 = 0; k2 < itsAnzTermine; k2++)
                        {
450.                                 aktZeile = AnzH1 + AnzH2 + i2 *
                        itsAnzTermine + k2;
451.                                 index = aktSpalte * AnzZeilen +
                        aktZeile + 1;
452.                                 zeile[index] = aktZeile + 1;
453.                                 spalte[index] = aktSpalte + 1;
454.                                 if (i1 == i2 && k1 == k2)  // wenn es
                        sich bei der Spalte und der Reihe um die gleichen
                        Indizes handelt
455.                                     matrix[index] = 1;
456.                                 else
457.                                     matrix[index] = 0;
458.                             }
459.                      }
460.                 jwunsch++;
461.                 }
462.
463.         // hilfsBool spalten
464.         for (int j1 = 0; j1 < itsAnzAngebote; j1++)
465.              for (int k1 = 0; k1 < itsAnzTermine; k1++) {
466.                  aktSpalte = AnzHauptBools + j1 * itsAnzTermine + k1;
467.
468.                  // nun alle Zeilen durchgehen
469.                  for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
```

```
470.                     for (int j2 = 0; j2 < itsAnzAngebote; j2++) {
471.                         aktZeile = i2 * itsAnzAngebote + j2;
472.
473.                         index = aktSpalte * AnzZeilen + aktZeile + 1;
474.                         zeile[index] = aktZeile + 1;
475.                         spalte[index] = aktSpalte + 1;
476.                         matrix[index] = 0;  // hier soll das Programm
                        nichts veraendern duerfen, deshalb alles 0
477.
478.                     }
479.
480.                 for (int j2 = 0; j2 < itsAnzAngebote; j2++)
481.                     for (int k2 = 0; k2 < itsAnzTermine; k2++) {
482.                         aktZeile = AnzH1 + j2 * itsAnzTermine + k2;
483.                         index = aktSpalte * AnzZeilen + aktZeile + 1;
484.                         zeile[index] = aktZeile + 1;
485.                         spalte[index] = aktSpalte + 1;
486.                         if (j1 == j2 && k1 == k2)  // hier soll das
                        Programm direkt die volle Anzahl an moeglichen
                        Teilnehmern fuer die Veranstaltunge setzen koennen
                        (immer dann, wenn die Veranstaltung nicht
                        stattfindet)
487.                             matrix[index] = max[j1];
488.                         else
489.                             matrix[index] = 0;
490.                     }
491.
492.                 for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
493.                     for (int k2 = 0; k2 < itsAnzTermine; k2++) {
494.                         aktZeile = AnzH1 + AnzH2 + i2 * itsAnzTermine +
                        k2;
495.                         index = aktSpalte * AnzZeilen + aktZeile + 1;
496.                         zeile[index] = aktZeile + 1;
497.                         spalte[index] = aktSpalte + 1;
498.                         matrix[index] = 0;  // das hat dabei keinen
                        Einfluss auf andere Sachen
499.                     }
500.             }
501.
502.
503.         glp_load_matrix(meinProblem, AnzFelder, zeile, spalte, matrix);
504.
505.
506.
507.         // Problem loesen
508.         cout << endl << endl << "Problem wird geloest: " << endl;
509.         glp_simplex(meinProblem, NULL);          // Problem wird geliest
                     zuerst mit dem Simplex-Verfahren
510.         glp_intopt(meinProblem, NULL);     // danach richtig mit dem
                    Branch-and-Cut-Verfahren
511.
512.
513.         // Ergebnis auslesen
514.         double prioritaetenMax = glp_mip_obj_val(meinProblem);
515.
516.
517.         for (int i = 0; i < itsAnzTeilnehmer; i++)
518.             for (int j = 0, jwunsch = 0; j < itsAnzAngebote; j++)
519.                 if (priori[i][j] != 0)  // wenn der Schueler i das
                    Angebot j gewaehlt hat
520.                 {
521.                     for (int k = 0; k < itsAnzTermine; k++) {
```

```
522.                         nummer = i * itsAnzWuensche * itsAnzTermine +
                         jwunsch * itsAnzTermine + k + 1;
523.                         if (glp_mip_col_val(meinProblem, nummer) != 0)
524.                             TeilnehmerZuweisen(i,j,k);
525.                          }
526.                          jwunsch++;
527.                 }
528.
529.
530.
531.
532.         cout << "Das Prioritaeten Maximum liegt bei: " << prioritaetenMax
                         << "!" << endl;
533.
534.         delete [] zeile;
535.         delete [] spalte;
536.         delete [] matrix;
537.   }
538.
539.   void cInformationstag::PlanErstellenSchnell() {
540.
541.         cout << "Schnelles-Plan-Erstellen wird geladen!" << endl;
542.
543.         // Angebote auslesen:
544.
545.         double min[itsAnzAngebote];
546.         double max[itsAnzAngebote];
547.
548.         for (int j = 0; j < itsAnzAngebote; j++) {
549.              min[j] = itsAngebote[j].getMin();
550.              max[j] = itsAngebote[j].getMax();
551.         }
552.
553.         // Teilnehmer auslesen:
554.         cAngebot * wuensche;
555.         int prioriAngebot[itsAnzTeilnehmer][itsAnzAngebote];
556.
557.         // zunaechst mit 0en fuellen:
558.         for (int i = 0; i < itsAnzTeilnehmer; i++)
559.              for (int j = 0; j < itsAnzAngebote; j++)
560.                  prioriAngebot[i][j] = 0;
561.
562.
563.         for(int i = 0; i < itsAnzTeilnehmer; i++) { // Alle Teilnehmer der
                         Reihe nach durchgehen
564.              wuensche = itsTeilnehmer[i].getWuensche();
565.              for(int j = 0; j < itsAnzAngebote; j++)  // Alle Angebote
                         durchgehen
566.                  for(int k = 0; k < itsAnzWuensche; k++)  // Alle Wuensche
                         des Teilnehmer durchgehen
567.
568.                       if (wuensche[k].getId() == j &&
                         TeilWue[i*itsAnzWuensche + k] == false)  // Wenn beim
                         akt Schueler der akt Wunsch mit dem akt Angebot
                         uebereinstimmt
569.                          prioriAngebot[i][j] = itsPrioritaeten[k];  //
                         dann soll in die Matrix die entsprechende
                         Prioritaetszahl geschrieben werden
570.         }
571.
572.
573.         // Das erste Problem formulieren
```

```
574.
575.        // welche Veranstaltungen, koennten stattfinden? Dies soll
                    gezaehlt werden
576.        int AnzVeranstaltungen = 0;
577.        int AnzBenuzterAngebote = 0;  // ist die Anzahl der Angebote, bei
                    der mindestens eine Veranstaltung zustande kommen
                    kann
578.
579.        int anzVerProAngebot[itsAnzAngebote];  // hier drin wird
                    gespeichert, wie oft ein Angebot evtl. stattfindet
580.
581.        for (int j = 0; j < itsAnzAngebote; j++)
582.             anzVerProAngebot[j] = 0;  // erstmal mit 0en fuellen
583.
584.        int TeilnehmerInVeranstaltung;
585.
586.        int prioriVeranstaltung[itsAnzTeilnehmer][itsAnzAngebote *
                    itsAnzTermine];  // gibt an welche Veranstaltung
                    welche Prioritaet beim Teilnehmer hat.
587.
588.        for (int j = 0; j < itsAnzAngebote; j++) { // alle Angebote
                    durchgehen
589.             TeilnehmerInVeranstaltung = 0;  // noch gibt es keine
                    Teilnehmer die evtl in die Veranstaltung wollen
590.             for (int i = 0; i < itsAnzTeilnehmer; i++) // nun soll
                    geprueft werden, welcher Teilnehmer das Angebot
                    gewaehlt hat
591.                 if (prioriAngebot[i][j] > 0) {
592.                      TeilnehmerInVeranstaltung++;
593.                      if (TeilnehmerInVeranstaltung >= min[j]) {
594.                          if (anzVerProAngebot[j] >= itsAnzTermine)
595.                              break;
596.                          else {
597.                              TeilnehmerInVeranstaltung = 0;
598.                              AnzVeranstaltungen++;
599.                              anzVerProAngebot[j]++;
600.                          }
601.                      }
602.
603.
604.                 }
605.
606.             if (anzVerProAngebot[j] > 0)
607.                 AnzBenuzterAngebote++;
608.
609.             // nun muss noch prioriVeranstaltung gefuellt werden
610.             for (int k = 0; k < anzVerProAngebot[j]; k++) {
611.                 for (int i = 0; i < itsAnzTeilnehmer; i++) {
612.                      prioriVeranstaltung[i][AnzVeranstaltungen - 1 - k] =
                         prioriAngebot[i][j];
613.                 }
614.             }
615.        }
616.
617.
618.
619.
620.
621.        int AnzH1 = AnzBenuzterAngebote * itsAnzTeilnehmer;
622.        int AnzH2 = AnzVeranstaltungen;
623.        int AnzH3 = itsAnzTeilnehmer;
624.
```

```
625.        int AnzZeilen = AnzH1 + AnzH2 + AnzH3;
626.
627.        if (AnzVeranstaltungen == 0) {
628.             cout << "Es kann kein Angebot besucht werden!" << endl;
629.             return;
630.        }
631.
632.
633.        glp_prob * meinProblem;
634.        meinProblem = glp_create_prob();
635.        glp_set_prob_name(meinProblem, "Informationstag");
636.        glp_set_obj_dir(meinProblem, GLP_MAX);
637.
638.        QString name;  // dadrin wird der Name der Zeile bzw. Spalte
                           gespeichert.
639.        int nummer = 0;  // hier wird die jeweilige Nummer der Zeile bzw.
                           Spalte gespeichert. Dient beides nur der Uebersicht
640.
641.
642.        // Zeilen (Beschrenkungen)
643.        glp_add_rows(meinProblem, AnzZeilen);
644.
645.
646.        for (int i = 0; i < itsAnzTeilnehmer; i++)
647.             for (int j = 0; j < AnzBenuzterAngebote; j++) {
648.                  name = "H1_" + QString::number(i) + "_" +
                          QString::number(j);
649.                  nummer = i * AnzBenuzterAngebote + j + 1;
650.                  glp_set_row_name(meinProblem, nummer, name.toLatin1());
651.                  glp_set_row_bnds(meinProblem,nummer, GLP_DB, 0, 1);
652.             }
653.
654.        for (int j = 0, aktVer = 0; j < itsAnzAngebote; j++)
655.             if(anzVerProAngebot[j] > 0)
656.                  for (int k = 0; k < anzVerProAngebot[j]; k++) {
657.
658.                       name = "checkVeranstaltung_" +
                          QString::number(aktVer);
659.                       nummer = AnzH1 + aktVer + 1;
660.                       glp_set_row_name(meinProblem, nummer,
                          name.toLatin1());
661.                       glp_set_row_bnds(meinProblem, nummer, GLP_DB,
                          min[j], max[j]);
662.                       aktVer++;
663.                  }
664.
665.        int erlaubteAnz;
666.
667.        for (int i = 0; i < itsAnzTeilnehmer; i++) {
668.             erlaubteAnz = itsAnzTermine;
669.             for (int k = 0; k < itsAnzWuensche; k++)
670.                  if (TeilWue[i * itsAnzWuensche + k] == true)  // Besucht
                          der Teilnehmer schon 1 oder mehr Veranstaltungen,
                          darf er nur noch in entsprechend weniger
671.                       erlaubteAnz--;
672.
673.             name = "H3_" + QString::number(i);
674.             nummer = AnzH1 + AnzH2 + i + 1;
675.             glp_set_row_name(meinProblem, nummer, name.toLatin1());
676.             glp_set_row_bnds(meinProblem, nummer, GLP_DB, 0,
                          erlaubteAnz);   // Ein Schueler darf mindests so
                          viele Veranstaltungen besuchen, wie es ihm noch
```

```
                    erlaubt ist
677.        }
678.
679.        // Spalten/binaere Varibalen (structural variables) Die, die vom
                    Programm bestimmt werden muessen.
680.        // Bools: Sind die binaeren Variablen die das Programm festlegt
                    Bool_1_2_3 == 1 bedeutet, das Schüler 1 Angebot 2 zu
                    Termin 3 besucht
681.
682.
683.        int AnzHauptBools = itsAnzTeilnehmer * AnzVeranstaltungen; // Da
                    Schueler keine Veranstaltung die Besuchen, die sie
                    nicht auf gewaelt haben, wird hier nicht AnzAngebote
                    benutzt, sondern nur die Anzahl der Wuensche
684.        int AnzHilfsBools = AnzVeranstaltungen;  // ein paar Hilfs-
                    Strukturale-Varibalen mit denen eine leere
                    Veranstaltung direkt als voll behandelt werden kann
685.
686.        int AnzSpalten = AnzHauptBools + AnzHilfsBools;
687.
688.        glp_add_cols(meinProblem, AnzSpalten);
689.
690.
691.        // nun werden alle Bools hinzugefuegt und mit dem entsprechenden
                    Koeffizienten fuer die Zielfunktion verknuepft
692.        for (int i = 0; i < itsAnzTeilnehmer; i++)
693.             for (int j = 0; j < AnzVeranstaltungen; j++) {
694.                  name = "Bool_" + QString::number(i) + "_" +
                         QString::number(j);
695.                  nummer = i * AnzVeranstaltungen + j + 1;
696.                  glp_set_col_name(meinProblem, nummer, name.toLatin1());
697.                  glp_set_col_kind(meinProblem, nummer, GLP_BV); // mach
                         die Variable binaer
698.                  glp_set_obj_coef(meinProblem, nummer,
                         prioriVeranstaltung[i][j]); // der Koeffizient fuer
                         die Zielfunktion
699.
700.             }
701.
702.        // Hilfsbools
703.        for (int j = 0; j < AnzVeranstaltungen; j++) {
704.             name = "NichtStatt_" + QString::number(j);
705.             nummer = AnzHauptBools + j + 1;
706.             glp_set_col_name(meinProblem, nummer, name.toLatin1());
707.             glp_set_col_kind(meinProblem, nummer, GLP_BV); // mach die
                         Variable binaer
708.             glp_set_obj_coef(meinProblem, nummer, 0); // diese haben
                         keinen Einfluss auf die Zielfunktion
709.
710.        }
711.
712.
713.        // Als letztes muss die Matrix geladen werden, diese muss dafuer
                    in ein eindemensionales Feld gepresst werden:
714.        int AnzFelder = AnzZeilen * AnzSpalten;
715.
716.        int *zeile = new int[AnzFelder + 1];
717.        int *spalte = new int[AnzFelder + 1];
718.        double *matrix = new double[AnzFelder + 1];
719.
720.        zeile[0] = 0;
721.        spalte[0] = 0;
```

```
722.        matrix[0] = 0;
723.
724.        int aktSpalte;
725.        int aktZeile;
726.
727.        int index;
728.        // alle Spalten durchgehen
729.        for (int i1 = 0; i1 < itsAnzTeilnehmer; i1++)
730.             for (int j1 = 0, aktVer = 0, aktAng = 0; j1 < itsAnzAngebote;
                         j1++)
731.                  if (anzVerProAngebot[j1] > 0) {
732.                       for (int k1 = 0; k1 < anzVerProAngebot[j1]; k1++) {
733.                            aktSpalte = i1 * AnzVeranstaltungen + aktVer;
734.
735.
736.                            // nun alle Zeilen durchgehen
737.                            for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
738.                                 for (int j2 = 0; j2 < AnzBenuzterAngebote;
                                 j2++) {
739.                                      aktZeile = i2 * AnzBenuzterAngebote +
                                 j2;
740.                                      index = aktSpalte * AnzZeilen + aktZeile
                                 + 1;
741.                                      zeile[index] = aktZeile + 1;
742.                                      spalte[index] = aktSpalte + 1;
743.                                      if (i1 == i2 && aktAng == j2)  // wenn
                                 es sich bei der Spalte und der Reihe um die gleichen
                                 Indizes handelt
744.                                           matrix[index] = 1;
745.                                      else
746.                                           matrix[index] = 0;
747.                                 }
748.
749.
750.                            for (int j2 = 0; j2 < AnzVeranstaltungen; j2++)
                                 {
751.                                 aktZeile = AnzH1 + j2;
752.                                 index = aktSpalte * AnzZeilen + aktZeile +
                                 1;
753.                                 zeile[index] = aktZeile + 1;
754.                                 spalte[index] = aktSpalte + 1;
755.                                 if (aktVer == j2)  // wenn es sich bei der
                                 Spalte und der Reihe um die gleichen Indizes handelt
756.                                      matrix[index] = 1;
757.                                 else
758.                                      matrix[index] = 0;
759.                            }
760.
761.
762.
763.                            for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++) {
764.                                 aktZeile = AnzH1 + AnzH2 + i2;
765.                                 index = aktSpalte * AnzZeilen + aktZeile +
                                 1;
766.                                 zeile[index] = aktZeile + 1;
767.                                 spalte[index] = aktSpalte + 1;
768.                                 if (i1 == i2)  // wenn es sich bei der
                                 Spalte und der Reihe um die gleichen Indizes handelt
769.                                      matrix[index] = 1;
770.                                 else
771.                                      matrix[index] = 0;
772.
```

```
773.                         }
774.
775.                         aktVer++;
776.                     }
777.
778.                     aktAng++;
779.                 }
780.
781.
782.         // hilfsBool Spalten
783.         for (int j1 = 0, aktVer = 0, aktAng = 0; j1 < itsAnzAngebote; j1+
                    +)
784.             if (anzVerProAngebot[j1] > 0) {
785.                 for (int k1 = 0; k1 < anzVerProAngebot[j1]; k1++) {
786.                     aktSpalte = AnzHauptBools + aktVer;
787.
788.
789.                     // nun alle Zeilen durchgehen
790.                     for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
791.                         for (int j2 = 0; j2 < AnzBenuzterAngebote; j2++)
                        {
792.                             aktZeile = i2 * AnzBenuzterAngebote + j2;
793.                             index = aktSpalte * AnzZeilen + aktZeile +
                        1;
794.                             zeile[index] = aktZeile + 1;
795.                             spalte[index] = aktSpalte + 1;
796.                             matrix[index] = 0;
797.                         }
798.
799.
800.                     for (int j2 = 0; j2 < AnzVeranstaltungen; j2++) {
801.                         aktZeile = AnzH1 + j2;
802.                         index = aktSpalte * AnzZeilen + aktZeile + 1;
803.                         zeile[index] = aktZeile + 1;
804.                         spalte[index] = aktSpalte + 1;
805.                         if (aktVer == j2)  // wenn es sich bei der
                        Spalte und der Reihe um die gleichen Indizes handelt
806.                             matrix[index] = max[aktAng];
807.                         else
808.                             matrix[index] = 0;
809.                     }
810.
811.
812.
813.                     for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++) {
814.                         aktZeile = AnzH1 + AnzH2 + i2;
815.                         index = aktSpalte * AnzZeilen + aktZeile + 1;
816.                         zeile[index] = aktZeile + 1;
817.                         spalte[index] = aktSpalte + 1;
818.                         matrix[index] = 0;
819.                     }
820.
821.                     aktVer++;
822.                 }
823.
824.                 aktAng++;
825.
826.
827.             }
828.
829.
830.         glp_load_matrix(meinProblem, AnzFelder, zeile, spalte, matrix);
```

```
831.
832.
833.
834.        // Problem loesen
835.        cout << endl << endl << "Problem wird geloest: " << endl;
836.        glp_simplex(meinProblem, NULL);          // Problem wird geloest
837.        glp_intopt(meinProblem, NULL);
838.
839.
840.
841.
842.
843.
844.        // Ergebnis auslesen
845.        double prioritaetenMax = glp_mip_obj_val(meinProblem);
846.
847.
848.        double loesung[itsAnzTeilnehmer][AnzVeranstaltungen]; // besucht
                    der Teilnehmer die Veranstaltung??
849.
850.        cout << "Das Prioritaeten Maximum liegt bei: " << prioritaetenMax
                    << "!" << endl;
851.
852.        for (int i = 0; i < itsAnzTeilnehmer; i++)
853.             for (int j = 0, aktVer = 0; j < itsAnzAngebote; j++)
854.                 if (anzVerProAngebot[j] > 0) {
855.                      for (int k = 0; k < anzVerProAngebot[j]; k++) {
856.                          nummer = i * AnzVeranstaltungen + aktVer + 1;
857.                          loesung[i][aktVer] =
                         glp_mip_col_val(meinProblem, nummer);
858.                          aktVer++;
859.                      }
860.                 }
861.
862.
863.
864.        double prioriVeranstaltungen[AnzVeranstaltungen];
865.
866.
867.        for (int j = 0; j < AnzVeranstaltungen; j++) {
868.             prioriVeranstaltungen[j] = 0;
869.
870.             for (int i = 0; i < itsAnzTeilnehmer; i++)
871.                 if (loesung[i][j] != 0) {
872.                      nummer = i * AnzVeranstaltungen + j + 1;
873.                      prioriVeranstaltungen[j] +=
                         glp_get_obj_coef(meinProblem, nummer);
874.                 }
875.        }
876.
877.        double prioriGesamt = 0;
878.        for (int i = 0; i < AnzVeranstaltungen; i++)
879.             prioriGesamt += prioriVeranstaltungen[i];
880.        cout << "prioriGesamt: " << prioriGesamt << endl;
881.
882.
883.        // Das zweite Problem
884.        glp_prob * meinProblem2;
885.        meinProblem2 = glp_create_prob();
886.        glp_set_prob_name(meinProblem2, "Veranstaltungen ordnen");
887.        glp_set_obj_dir(meinProblem2, GLP_MAX);
888.
```

```
889.
890.
891.        // Zunaechst wieder die Zeilen
892.        AnzH1 = itsAnzTeilnehmer * itsAnzTermine; //Dass kein Schueler
                zwei Angebote parallel besuchen muss
893.        AnzH2 = AnzVeranstaltungen;  // Dass keine Veranstaltung mehrfach
                angeboten wird
894.        AnzZeilen = AnzH1 + AnzH2;
895.        glp_add_rows(meinProblem2, AnzZeilen);
896.
897.        for (int i = 0; i < itsAnzTeilnehmer; i++)
898.             for (int k = 0; k < itsAnzTermine; k++) {
899.                  name = "H1_" + QString::number(i) + "_" +
                         QString::number(k);
900.                  nummer = i * itsAnzTermine + k + 1;
901.                  glp_set_row_name(meinProblem2, nummer, name.toLatin1());
902.                  if (TeilTer[i*itsAnzTermine + k] == false)   // wenn der
                         Teilnehmer an dem noch kein Termin besucht
903.                       glp_set_row_bnds(meinProblem2,nummer, GLP_DB, 0, 1);
904.                  else                              // falls er doch schon
                         eine Veranstaltung an dem Termin besucht, soll er
                         hier eingeschrenkt werden
905.                       glp_set_row_bnds(meinProblem2,nummer, GLP_FX, 0, 0);
906.             }
907.
908.        for (int j = 0; j < AnzVeranstaltungen; j++) {
909.             name = "H2_" + QString::number(j);
910.             nummer = AnzH1 + j + 1;
911.             glp_set_row_name(meinProblem2, nummer, name.toLatin1());
912.             glp_set_row_bnds(meinProblem2, nummer, GLP_DB, 0, 1);
913.        }
914.
915.        // Als naechstes die Spalten
916.        AnzSpalten = AnzHauptBools = AnzVeranstaltungen * itsAnzTermine;
917.
918.        glp_add_cols(meinProblem2, AnzSpalten);
919.
920.
921.        for (int j = 0, aktVer = 0; j < itsAnzAngebote; j++)
922.             if (anzVerProAngebot[j] > 0)
923.                  for (int k = 0; k < anzVerProAngebot[j]; k++) {
924.                       for (int t = 0; t < itsAnzTermine; t++) {
925.                            name = "Bool_" + QString::number(aktVer) + "_" +
                             QString::number(t);
926.                            nummer = aktVer * itsAnzTermine + t + 1;
927.                            glp_set_col_name(meinProblem2, nummer,
                             name.toLatin1());
928.                            if (prioriVeranstaltungen[aktVer] == 0 ||
                             AngTer[j*itsAnzTermine + t] == true)
929.                                 glp_set_col_bnds(meinProblem2, nummer,
                             GLP_FX, 0, 0);  // wenn die Veranstaltung nicht
                             stattfindet, weil zu wenig Interessenten oder sie
                             schon existiert, soll sie fixiert werden
930.                            else
931.                                 glp_set_col_kind(meinProblem2, nummer,
                             GLP_BV); // mach die Variable binaer
932.                            glp_set_obj_coef(meinProblem2, nummer,
                             prioriVeranstaltungen[aktVer]); // der Koeffizient
                             fuer die Zielfunktion
933.                       }
934.                  aktVer++;
935.                  }
```

```
936.
937.
938.
939.
940.
941.        // nun muss noch die Matrix geladen werden:
942.        delete [] zeile;
943.        delete [] spalte;
944.        delete [] matrix;
945.
946.        AnzFelder = AnzZeilen * AnzSpalten;
947.
948.        zeile = new int[AnzFelder + 1];
949.        spalte = new int[AnzFelder + 1];
950.        matrix = new double[AnzFelder + 1];
951.
952.        zeile[0] = 0;
953.        spalte[0] = 0;
954.        matrix[0] = 0;
955.
956.        // alle Spalten durchgehen
957.        for (int j1 = 0; j1 < AnzVeranstaltungen; j1++)
958.             for (int k1 = 0; k1 < itsAnzTermine; k1++) {
959.                  aktSpalte = j1 * itsAnzTermine + k1;
960.                  // alle Zeilen durchgehen
961.                  for (int i2 = 0; i2 < itsAnzTeilnehmer; i2++)
962.                       for (int k2 = 0; k2 < itsAnzTermine; k2++) {
963.                            aktZeile =  i2 * itsAnzTermine + k2;
964.                            index = aktSpalte * AnzZeilen + aktZeile + 1;
965.                            zeile[index] = aktZeile + 1;
966.                            spalte[index] = aktSpalte + 1;
967.                            if (k1 == k2 && loesung[i2][j1] != 0)  // wenn
                           es sich um den gleichen Termin handelt und der
                           Teilnehmer die Veranstaltung besucht
968.                                 matrix[index] = 1;
969.                            else
970.                                 matrix[index] = 0;
971.                       }
972.
973.                  for (int j2 = 0; j2 < AnzVeranstaltungen; j2++) {
974.                       aktZeile = AnzH1 + j2;
975.                       index = aktSpalte * AnzZeilen + aktZeile + 1;
976.                       zeile[index] = aktZeile + 1;
977.                       spalte[index] = aktSpalte + 1;
978.                       if (j1 == j2)  // wenn es sich um die gleiche
                           Veranstaltung handelt
979.                            matrix[index] = 1;
980.                       else
981.                            matrix[index] = 0;
982.                  }
983.                  cout << endl;
984.             }
985.
986.        glp_load_matrix(meinProblem2, AnzFelder, zeile, spalte, matrix);
987.
988.
989.
990.        // Problem loesen
991.        cout << endl << endl << "Problem2 wird geloest: " << endl;
992.
993.        glp_simplex(meinProblem2, NULL);         // Problem wird geliest
994.        glp_intopt(meinProblem2, NULL);
```

```
995.
996.
997.
998.        // Ergebnis auslesen und Veranstaltungen erstellen
999.        cAngebot*wunsch;
1000.
1001.       bool veraenderung = false;  // hat sich was bei dem durchgang
                    veraendert?
1002.
1003.       double loesung2[AnzVeranstaltungen][itsAnzTermine];
1004.
1005.       for (int i = 0; i < itsAnzTeilnehmer; i++)
1006.            for (int j = 0, aktVer = 0; j < itsAnzAngebote; j++)
1007.                if (anzVerProAngebot[j] > 0) {
1008.                     for (int k = 0; k < anzVerProAngebot[j]; k++) {
1009.                         for(int t = 0; t < itsAnzTermine; t++) {
1010.                              nummer = aktVer * itsAnzTermine + t + 1;
1011.                              loesung2[aktVer][t] =
                        glp_mip_col_val(meinProblem2, nummer);
1012.                              if(loesung[i][aktVer] != 0 &&
                        glp_mip_col_val(meinProblem2, nummer) != 0) {
1013.
1014.                                  TeilnehmerZuweisen(i, j, t); // der
                        Teilnehmer wird zugewiesen
1015.                                  veraenderung = true; // es hat sich was
                        veraendert
1016.                                  TeilTer[i*itsAnzTermine + t] = true; //
                        und es soll sich alles gemerkt werden
1017.                                  AngTer[j*itsAnzTermine + t] = true;
1018.
1019.                                  wunsch = itsTeilnehmer[i].getWuensche();
1020.                                  for (int w = 0; w < itsAnzWuensche; w++)
1021.                                      if (wunsch[w].getId() ==
                        itsAngebote[j].getId())
1022.                                          TeilWue[i*itsAnzWuensche + w] =
                        true;
1023.
1024.
1025.
1026.
1027.                              }
1028.                         }
1029.
1030.                         aktVer++;
1031.                     }
1032.                }
1033.
1034.       delete [] zeile;
1035.       delete [] spalte;
1036.       delete [] matrix;
1037.
1038.       if (veraenderung == true) // es hat sich was veraendert, das ganze
                    soll also nochmal durchlaufen
1039.            PlanErstellenSchnell();
1040.
1041.  }
1042.
```

4.8 cAngebot.h

```
1.  #ifndef CANGEBOT_H_
2.  #define CANGEBOT_H_
3.
4.  #include <iostream>
5.  using namespace std;
6.
7.  #include <QString>
8.
9.  class cAngebot
10. {
11. public:
12.       cAngebot(int id = 0, int min = 0, int max = 0, QString name =
                    "Angebot");
13.       ~cAngebot();
14.
15.       void Ausgeben();  // fuer die Konsole
16.
17.       QString getName() {return itsName; }
18.       int getId() {return itsId; }
19.
20.       int getMin() {return itsMin; }
21.       int getMax() {return itsMax; }
22.
23. private:
24.       int itsId; // werden durchnummeriert
25.       QString itsName;
26.
27.       int itsMin;
28.       int itsMax;
29. };
30.
31. #endif /*CANGEBOT_H_*/
32.
```

4.9 cAngebot.cpp

```
1.  #include "cAngebot.h"
2.
3.  cAngebot::cAngebot(int id, int min, int max, QString name):
4.            itsId(id),
5.            itsMin(min),
6.            itsMax(max) {
7.       if (name == "Angebot")
8.            itsName = "Angebot " + QString::number(id);
9.       else
10.           itsName = name;
11. }
12.
13. cAngebot::~cAngebot()
14. {
15. }
16.
17. void cAngebot::Ausgeben() {
18.       cout << "Angebot " << itsId << ": \"" << itsName.toStdString() <<
                    "\" | min: " << itsMin << " | max: " << itsMax <<
                    endl;
19. }
20.
```

4.10 cTeilnehmer.h

```
1.  #ifndef CTEILNEHMER_H_
2.  #define CTEILNEHMER_H_
3.
4.  #include <iostream>
5.  using namespace std;
6.
7.  #include <QString>
8.
9.  #include "cAngebot.h"
10.
11. class cTeilnehmer
12. {
13. public:
14.       cTeilnehmer();
15.       cTeilnehmer(int id, int anzWuensche, cAngebot * wuensche, QString
                      name = QString::fromUtf8("Schüler"));
16.       ~cTeilnehmer();
17.
18.       cAngebot * getWuensche();
19.
20.       void Ausgeben();  // fuer die Konsole
21.
22.       int getId() { return itsId; }
23.       QString getName() { return itsName; }
24.
25.
26.
27. private:
28.       int itsId; // Teilnehmer werden auch durchnummeriert
29.       QString itsName;
30.
31.       int itsAnzWuensche;
32.       cAngebot * itsWuensche;
33.
34. };
35.
36. #endif /*CTEILNEHMER_H_*/
37.
```

4.11 cTeilnehmer.cpp

```
1.  #include "cTeilnehmer.h"
2.
3.  cTeilnehmer::cTeilnehmer() {
4.        itsId = -2;  // objekt besitzt noch keinen inhalt
5.        itsAnzWuensche = 0;
6.        itsName = "leeres Objekt";
7.        itsWuensche = new cAngebot[0];
8.  }
9.
10. cTeilnehmer::cTeilnehmer(int id, int anzWuensche, cAngebot * wuensche,
                    QString name) {
11.       if (name == QString::fromUtf8("Schüler"))
12.             itsName = QString::fromUtf8("Schüler ") +
                    QString::number(id);
13.       else
14.             itsName = name;
15.       itsId = id;
16.       itsAnzWuensche = anzWuensche;
17.
18.       itsWuensche = new cAngebot[itsAnzWuensche];
19.
20.       for (int i = 0; i < itsAnzWuensche; i++)
21.             itsWuensche[i] = wuensche[i];
22. }
23.
24. cTeilnehmer::~cTeilnehmer() {
25. }
26.
27. cAngebot * cTeilnehmer::getWuensche() {
28.       return itsWuensche;
29. }
30.
31. void cTeilnehmer::Ausgeben() {
32.       cout << "Teilnehmer " << itsId << ": \"" << itsName.toStdString()
                    << "\" Wünsche: ";
33.       for (int i = 0; i< itsAnzWuensche; i++)
34.             cout << itsWuensche[i].getName().toStdString() << " (" <<
                    itsWuensche[i].getId() << ") | ";
35.       cout << endl;
36. }
37.
```

4.12 cVeranstaltung.h

```
1.  #ifndef CVERANSTALTUNG_H_
2.  #define CVERANSTALTUNG_H_
3.
4.  #include "cAngebot.h"
5.  #include "cTeilnehmer.h"
6.
7.  class cVeranstaltung
8.  {
9.  public:
10.       cVeranstaltung(int id = 0, cAngebot * typ = 0, int termin = 0);
11.       ~cVeranstaltung();
12.
13.       void TeilnehmerHinzufuegen(cTeilnehmer * teilnehmer);
14.
15.       int getId() { return itsId; }
16.       int getTermin() { return itsTermin; }
17.       cAngebot * getTyp() { return itsTyp; }
18.
19.
20.       int getAnzTeilnehmer() {return itsAnzTeilnehmer; }
21.       cTeilnehmer * getTeilnehmer(int i);
22.       void Ausgeben();  // fuer die Konsole
23.
24.  private:
25.       int itsId; // auch Veranstaltungen werden seperat durchnummeriert
26.       cAngebot * itsTyp;  // Das Angebot, von der Sorte die
                    Veranstaltung ist
27.
28.       int itsTermin; // 0: unbelegt; 1 - 4: Termin 1 bis 4
29.
30.       int itsAnzTeilnehmer;
31.       cTeilnehmer ** itsTeilnehmer; // Die Teilnehmer, die bei dieser
                    Veranstaltung teilnehmen.
32.            // Ein Zeiger auf ein Array von Zeigern!!!
33. };
34.
35. #endif /*CVERANSTALTUNG_H_*/
36.
```

4.13 cVeranstaltung.cpp

```
37.  #include "cVeranstaltung.h"
38.
39.  cVeranstaltung::cVeranstaltung(int id, cAngebot * typ, int termin):
40.        itsId (id),
41.        itsTyp (typ),
42.        itsTermin (termin)
43.  {
44.        itsAnzTeilnehmer = 0;
45.        itsTeilnehmer = new cTeilnehmer*[itsAnzTeilnehmer]; // Denklariert
                    eine neues Array von Zeigern auf cTeilnehmer-Objekte
                    und laesst itsTeilnehmer darauf zeigen
46.  }
47.
48.  cVeranstaltung::~cVeranstaltung()
49.  {
50.  }
51.
52.  void cVeranstaltung::TeilnehmerHinzufuegen(cTeilnehmer * teilnehmer) {
53.        itsAnzTeilnehmer++;
54.        cTeilnehmer ** tempTeilnehmer = new
                    cTeilnehmer*[itsAnzTeilnehmer];
55.
56.        for (int i = 0; i < itsAnzTeilnehmer -1; i++)
57.             tempTeilnehmer[i] = itsTeilnehmer[i];
58.        delete [] itsTeilnehmer;
59.
60.        tempTeilnehmer[itsAnzTeilnehmer -1] = teilnehmer;
61.        itsTeilnehmer = tempTeilnehmer;
62.  }
63.
64.  cTeilnehmer * cVeranstaltung::getTeilnehmer(int i) {  // gibt einen
                    Zeiger auf den i-ten Teilnehmer zurueck
65.        if (i < itsAnzTeilnehmer)
66.             return itsTeilnehmer[i];
67.        else
68.             return 0;
69.  }
70.
71.
72.  void cVeranstaltung::Ausgeben() {
73.        cout << "Veranstaltung " << itsId << ": Typ: " << itsTyp-
                    >getName().toStdString() << " (" << itsTyp->getId()
                    << ") | " << itsAnzTeilnehmer << " Teilnehmer: ";
74.        for (int i = 0; i< itsAnzTeilnehmer; i++)
75.             cout << itsTeilnehmer[i]->getName().toStdString() << " (" <<
                    itsTeilnehmer[i]->getId() << ") | ";
76.        cout << endl;
77.  }
78.
```