

Aufgabe 1: Flohmarkt in Langdorf

Teilnahme-ID: 56727

Bearbeiter dieser Aufgabe:
Carl Friedrich Mecking

17. April 2021

Inhaltsverzeichnis

1	Betrachtung des Problems	3
1.1	Abgrenzung des Flohmarktproblems vom gewöhnlichen Rucksackproblem	3
1.2	Lösungsansätze	3
2	Lösungsidee	4
2.1	Brute-Force-Verfahren	4
2.2	Greedy Verfahren	5
3	Umsetzung	6
3.1	Bereitstellung der Daten	6
3.2	Entscheidung für ein Verfahren	6
3.3	Brute-Force-Verfahren	7
3.4	Greedy-Verfahren	8
3.5	Ausgabe	9
4	Beispiele	9
5	Laufzeitverhalten	10
5.1	“All-Fitting”	10
5.2	“Brute-Force”	11
5.3	“Greedy”	11
5.4	Best- und Worstcase-Betrachtung	12
6	Ergebnistabelle	13
7	Sourcecode	20
7.1	Main	20
7.2	allFitting	21
7.3	bruteForce	22
7.4	greedy	23

1 Betrachtung des Problems

Die Aufgabe „Flohmarkt in Langdorf“ stellt ein kombinatorisches Optimierungsproblem dar. Es ist eine spezielle Variante des 0-1 Rucksackproblems. Beim Rucksackproblem geht es darum, aus einer Menge von n Gegenständen mit einem Gewicht w und einem Wert v , die Kombination aus Gegenständen auszuwählen, welche in der Summe nicht das Gewichtslimit C des Rucksacks übersteigt und deren Wert maximal ist. Wird davon ausgegangen, dass x_i die Anzahl der Instanzen eines Gegenstandes ist, welche in den Rucksack getan werden, stellt sich dieser Sachverhalt wie folgt dar:

$$\max \sum_{i=1}^n v_i x_i$$

$$\text{mit } \sum_{i=1}^n w_i x_i \leq C \text{ und } x_i \in \{0, 1\}$$

1.1 Abgrenzung des Flohmarktproblems vom gewöhnlichen Rucksackproblem

Die Parameter des Wertes und des Gewichts im Rucksackproblem finden Ihre Entsprechung im Flohmarktproblem hinsichtlich des Ertrags der Voranmeldung und der Länge des Flohmarktstandes. Im Flohmarktproblem tritt die Zeit als weitere Restriktion hinzu, welche, neben der Länge des jeweiligen Flohmarktstandes, einschränkend für die Auswahl von Voranmeldungen ist. Gleichzeitig ergibt sich der Wert, also der Ertrag einer jeden Voranmeldung, aus dem Produkt der Länge und der Zeit, welche diese in Anspruch nimmt. Sei n_t die Anzahl der Stunden. Ausgehend von einer Startuhrzeit t_0 existiert in jeder Stunde eine individuelle, von den anderen Stunden unabhängige, Restkapazität. Daraus ergibt sich folgender Kapazitätsvektor:

$$C = (C_{t_0 \rightarrow t_1}, C_{t_1 \rightarrow t_2}, \dots, C_{t_0+t_{n_t}-1 \rightarrow t_0+t_{n_t}})$$

Jede Voranmeldung beansprucht eine Teilmenge zeitlich aufeinanderfolgender Kapazitäten. Schließlich beansprucht eine Voranmeldung immer einen fest zusammenhängenden Zeitraum und nicht Einzelstunden, welche über die Gesamtzeit verteilt sind. Hat jede Voranmeldung eine individuelle Startzeit t_S und eine Endzeit t_E , wirkt sich dies wie folgt auf die obige Darstellung aus:

$$\max \sum_{i=1}^n w_i (t_{i_E} - t_{i_S}) x_i \text{ oder } \max \sum_{i=1}^n \sum_{t=t_{i_S}}^{t_{i_E}} w_i \cdot t \cdot x_i$$

$$\text{mit } \sum_{i=1}^n w_i x_i \leq C_t, \forall t \in \{t_0, \dots, t_{n_t}\} \text{ und } x_i \in \{0, 1\}$$

Die Nebenbedingung ist so zu interpretieren, dass in jeder Stunde des Gesamtzeitraums (t) die Summe der ausgewählten Voranmeldungen, mit Belegung dieser Stunde, kleiner oder gleich der vorgegebenen Maximalkapazität sein muss. In der konkreten Aufgabenstellung wird binnen eines zehnstündigen Zeitraumes von 8 bis 18 Uhr eine Längenkapazität von 1000m angesetzt.

Durch das Hinzufügen weiterer Restriktionen zum Sachverhalt erhöht sich die Komplexitätsstufe des Problems. Schon das 0-1 Rucksackproblem ist NP-Vollständig. Durch Hinzutreten weiterer Restriktionen wird das Rucksackproblem multidimensional und damit noch erheblich schwieriger.

1.2 Lösungsansätze

Wenn davon auszugehen ist, dass nicht alle Voranmeldungen eingebracht werden können, so muss zwischen verschiedenen Voranmeldungen die Entscheidung getroffen werden, welche von diesen zu einem maximalen Ertrag führen. Im Zeitpunkt der Entscheidung fehlt die volle Informationen über die übrigen Voranmeldungen und deren Kombinationen. Somit muss man für eine optimale Lösung jede Kombinationsmöglichkeit von Voranmeldungen testen, um herauszufinden, welcher Entscheidungsweg zu einer optimalen Lösung führen wird. Als klassisches Entscheidungsproblem fällt die Aufgabe Flohmarkt in Langdorf in die Komplexitätsklasse NP-Schwer. Aufgrund der NP-Schwere ist es nicht möglich, eine exakte Lösung für das Problem in Polynomialzeit in jedem Beispiel zu finden. Auch eine Reduzierung bzw.

Teilung des Problems ist durch die Überlappung der Zeiträume nur selten möglich und nicht zielführend für eine exakte Lösung. Deshalb wird in diesem Abschnitt ein dreigleisiges Verfahren vorgestellt, welches dazu dienen soll, bestimmte Beispiele exakt zu lösen und Beispiele mit besonders vielen Voranmeldungen in einem approximativen Lösungsweg abbildet.

Zunächst wird getestet, ob jede Voranmeldung miteinbezogen werden kann, ohne die Kapazitäten auszureizen. Ist dies der Fall, ist die Aufgabe an dieser Stelle exakt gelöst. Andernfalls wird die Anzahl der Voranmeldungen überprüft, um festzustellen mit welchem der beiden übrigen Verfahren die Aufgabe exakt oder approximativ gelöst werden kann.

2 Lösungsidee

Dieser Abschnitt beschäftigt sich mit der konkreten Lösungsidee für die einzelnen Szenarien. Zunächst wird das Brute-Force-Verfahren präsentiert, was bei einer kleinen Anzahl von Voranmeldungen exakt löst. Des Weiteren wird zur Approximation ein Greedy-Verfahren beschrieben.

2.1 Brute-Force-Verfahren

In vielen Fällen ist die Anzahl der Voranmeldungen nur so groß, dass es möglich ist jede Kombinationsmöglichkeit zu testen und somit ein exaktes Ergebnis zu erhalten. Bis zu fünfzig Voranmeldungen kann das folgende Brute-Force-Verfahren angewandt werden.

So kann man sich für eine erste Voranmeldung fragen, ob diese miteinbezogen werden soll oder nicht. Ungeachtet der jeweils getroffenen Entscheidung wird ein weiterer Entscheidungspfad für jede weitere Voranmeldung eröffnet. Durch dieses Verfahren entsteht ein binärer Baum, welcher für jeden möglichen Entscheidungsweg die dabei entstandenen Kosten, die gewählten Voranmeldungen und die Kapazitätsauslastung der jeweiligen Stunden einsehen lässt. Wenn eine Voranmeldung in einer oder mehreren Stunden zu einer Auslastung der Kapazitäten führt, so wird an dieser Stelle nur der Weg ohne diese Voranmeldung gegangen. Zum Schluss erkennt man anhand des Ertrags, welcher durch die jeweiligen einbezogenen Voranmeldungen eines Weges entstanden ist, welcher Entscheidungsweg zu einem optimalen Ertrag führt.

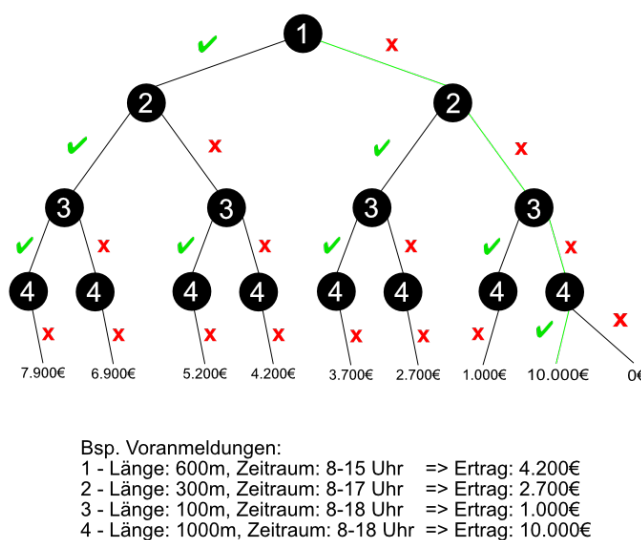


Abbildung 1: Darstellung des Entscheidungsbaums der Brute-Force-Methode

Abbildung 1 zeigt anhand eines einfachen Beispiels einen Entscheidungsbaum. Das Beispiel wurde so gewählt, dass die zuletzt betrachtete Voranmeldung den Platz über alle Stunden maximal belegt. Jeder Entscheidungsweg, welcher eine oder mehrere andere Voranmeldungen einbezieht führt zu einem nicht optimalen Ergebnis. Eine exakte Lösung fordert folglich stets vollständige Informationen über sämtliche Voranmeldungen. Entscheidungen, welche, bei Betrachtung einer Teilmenge der Voranmeldungen, zu einem lokalen Optimum führen, blockieren die optimale Lösung bei späterer Berücksichtigung der

Gesamtheit aller übrigen Voranmeldungen. Es ist deshalb nicht möglich, das Problem zu reduzieren. Zudem ist es nicht möglich, bei Betrachtung einer Teilmenge der Voranmeldungen, eine Ober- oder Untergrenze für das Ertragsverhalten eines Entscheidungsweges im Vorfeld abzuschätzen, um so die Anzahl der Entscheidungswege zu reduzieren. Jeder voreilige Entschluss könnte eine potentielle optimale Lösung verbauen. Somit muss tatsächlich jede einzelne Kombinationsmöglichkeit in Betracht gezogen werden.

2.2 Greedy Verfahren

Wie in Abschnitt 5 später beschrieben wird, ist das Testen aller Kombinationsmöglichkeiten von Voranmeldungen sehr laufzeitaufwendig. Je nach Rechenleistung können zwar unterschiedlich große Beispieldateien exakt gelöst werden, jedoch betrachtet die hiesige Lösung eine Limitierung auf fünfzig Voranmeldungen, welche mit standardmäßigen Rechnern in absehbarer Zeit zu lösen ist. Beispieldateien mit mehr als fünfzig Voranmeldungen werden mit einem Greedy Verfahren gelöst, das Voranmeldungen vorsortiert und dann solche selektiert, die einen möglichst hohen Ertrag versprechen.

Dazu wird das Problem zunächst als Instanz eines Rectangle-Packing-Problems betrachtet, in dem Rechtecke mit fester Größe und x-Position in einem Container mit fester Höhe und Breite platziert werden, ohne zu überlappen. Ziel ist, eine möglichst große Fläche zu belegen, die den Ertrag widerspiegelt. Abbildung 2 zeigt beispielsweise die beiden ertragsreichsten Entscheidungswege aus dem vorherigen Beispiel als Rectangle-Packing.

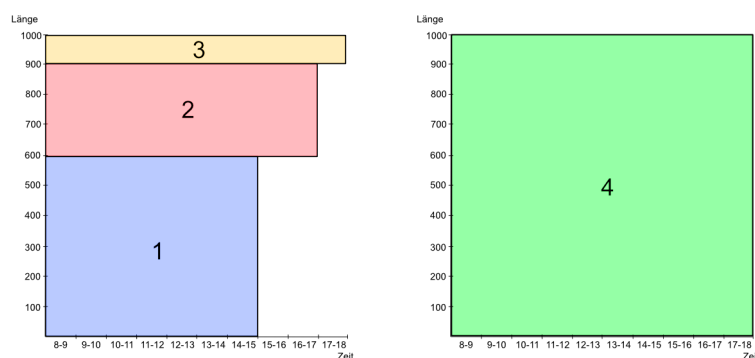


Abbildung 2: Darstellung der ertragsreichsten Entscheidungswege von Abbildung 1 als Rectangle-Packing

Die Voranmeldungen werden zunächst nach ihrer Startzeit sortiert. Unter mehreren Voranmeldungen identischer Startzeit wird die Voranmeldung mit der größeren Laufzeit vorgezogen. Letztlich wird die Sortierung unter Voranmeldungen gleicher Start- und Laufzeit durch den Ertrag bestimmt. Aus dieser Vorsortierung heraus werden nun nacheinander Voranmeldungen miteinbezogen. Wenn das Hinzufügen einer Voranmeldung die Kapazitätsgrenze in einem oder mehreren Zeiträumen übersteigt, wird diese Voranmeldung ausgelassen und mit der Nächsten fortgefahren.

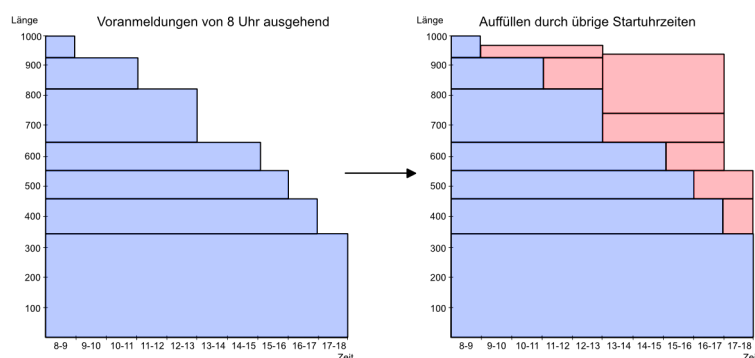


Abbildung 3: Greedy-Algorithmus mit Vorsortierung zur approximativen Lösung

Abbildung 3 zeigt den Prozess, in dem der Greedy Algorithmus versucht die Fläche möglichst optimal zu

füllen. Das erste Diagramm zeigt dabei eine beispielhafte Belegung der Fläche, wenn man Voranmeldungen beginnend ab 8 Uhr nach ihrem Ertragswert geordnet einreicht, soweit die Gesamtkapazität nicht ausgereizt wird. Im Anschluss werden nach dem selben Schema Voranmeldungen, mit späteren Starturzeiten, miteinbezogen. So kann der Algorithmus stückweise und möglichst ertragsreich die Kapazitäten in den einzelnen Stunden auslasten.

3 Umsetzung

In der Implementierung bestehen vier Klassen. Einer Instanz der Klasse “Data” wird ein Dateipfad als Parameter übergeben, welcher durch das Konsoleninterface gewählt wurde. Sie liest dann bei der Instanziierung die Informationen über die Voranmeldungen aus der angegebenen Datei aus und erstellt für jede Voranmeldung ein Objekt vom Typ “PreRegistration”. Durch die erfolgte Datenbereitstellung kann nun die Hauptklasse, nach der bereits beschriebenen Lösungsidee, das Problem lösen. Für die verschiedenen Verfahren wird eine Klasse “DecisionPathData” verwendet, welche Daten eines momentanen Entscheidungsweges der Voranmeldungen speichert. Ausgegeben wird sowohl ein Ertrag, als auch die Auswahl aus den Voranmeldungen der gewählten Testdatei, welche zu diesem Ertrag führt.

3.1 Bereitstellung der Daten

Wie bereits erwähnt wird der Klasse “Data” ein Dateipfad über das Konsoleninterface übergeben. Die Zeilen der Testdatei werden mit einem Scanner durchlaufen, wobei die drei Werte für jede Voranmeldung: Startzeit, Endezeit und Länge, ausgelesen werden. Ein Laufindex zählt dabei, wie viele Voranmeldungen bereits durchlaufen wurden. Für jeden Datensatz werden Objekte der Klasse “PreRegistration” erstellt. Dem Konstruktor dieser Klasse wird der Laufindex als ID der Voranmeldung, die Start- und Endezeit und die Länge des jeweiligen Flohmarktstandes, übergeben. Im Konstruktor werden für die Voranmeldungsobjekte zunächst noch die Anzahl der abgedeckten Stunden, als Differenz von Start- und Endezeit und der Ertrag der jeweiligen Voranmeldung als Produkt aus der Anzahl der abgedeckten Stunden und der Länge der Voranmeldung berechnet. Jedes Voranmeldungsobjekt wird dann in einer ArrayList gespeichert, welche von der Klasse “Data” zurückgegeben wird.

3.2 Entscheidung für ein Verfahren

In der Hauptmethode wird mittels der zuvor beschriebenen Datenbereitstellung eine ArrayList mit Voranmeldungsobjekten erstellt. Dann wird der Methode “allFitting” die Liste der Voranmeldungsobjekte übergeben. Sie prüft, ob alle Voranmeldungen miteinbezogen werden können, ohne das Kapazitätslimit in den einzelnen Stunden zu übersteigen. Dazu wird eine Ertragsvariable als Integer und ein Zeitplan in Form eines Integer-Arrays erstellt, welcher für jede Stunde die Kapazitätsauslastung speichert. Ein Boolean-Array, das für jede Voranmeldung speichert, ob diese miteinbezogen wurde oder nicht, wird zudem initialisiert. Das Boolean-Array und die Liste der Voranmeldungsobjekte sind dabei Index-gleich. Infolgedessen werden alle Voranmeldungen einmal durchlaufen.

Für jede Voranmeldung wird nun die Methode “isPRFitting” angewandt, die eine Voranmeldung auf Korrespondenz mit dem aktuellen Zeitplan testet. Ihr wird ein Voranmeldungsobjekt und ein Zeitplan als Parameter übergeben. Sie durchläuft zunächst die Stunden des Zeitplans, welche durch die übergebene Voranmeldung beansprucht werden. Übersteigt die Länge der übergebenen Voranmeldung in einer oder mehreren Stunden des Zeitplans das Kapazitätslimit von 1000m, wird false zurückgegeben. Andernfalls wird die Länge der Voranmeldung auf die bereits belegten Längen im Zeitplan, an den Indizes der entsprechenden Stunden, aufaddiert.

Passt die aktuelle Voranmeldung noch hinein, wird die Ertragsvariable um den Ertrag dieser Voranmeldung erhöht und der boolsche Wert im Selektierungsarray auf true gesetzt, also selektiert. Wenn die betrachtete Voranmeldung nicht mehr hinzugenommen werden kann, so wird ein entsprechendes Entscheidungswegobjekt zurückgegeben. Ein Entscheidungswegobjekt wird mit einem Zeitplan als Integer-Array, einem Boolean-Array für die Selektierungen und einem Ertrag als Integer, welcher als Summe der selektierten Voranmeldungen errechnet wurde, initialisiert. In dem Fall, dass die momentane Voranmeldung nicht mehr hinzugenommen werden kann, wird der Zeitplan und das Selektierungsarray bis zu diesem Punkt übergeben. Zudem wird der Ertrag -1 als Parameter überreicht, um zu symbolisieren, dass bei diesem Entscheidungsweg nicht alle Voranmeldungen ohne Überlappung Verwendung finden. Das entsprechende

Entscheidungswegobjekt wird dann zurückgegeben. Sollten alle Voranmeldungen miteinbezogen werden können, wird nach Durchlauf aller Voranmeldungen ein Entscheidungswegobjekt mit dem tatsächlichen Ertrag, dem Zeitplan und den selektierten Voranmeldungen zurückgegeben.

3.3 Brute-Force-Verfahren

Wenn die Methode “allFitting” true zurückgibt, also alle Voranmeldungen miteinbezogen werden können, dann erfolgt, wie in Abschnitt 3.5 beschrieben die Ausgabe. Andernfalls wird geprüft, ob die Länge der Liste aller Voranmeldungsobjekte kleiner oder gleich fünfzig ist. Ist dies der Fall wird das folgende Brute-Force-Verfahren angewandt.

Die rekursive Methode “BruteForce” nimmt als Parameter eine ArrayList von Voranmeldungsobjekten, einen unbelegten Zeitplan als Integer-Array, ein Boolean-Array für die selektierten Voranmeldungen, einen Ertragswert als Integer und einen Laufindex entgegen. Der Ertrag und der Laufindex werden zu Anfang entsprechend mit null initialisiert. Zugleich stehen in dem Boolean-Array nur false-Werte. Zurückgegeben wird dann von der Methode ein Entscheidungswegobjekt, welches die Daten für einen optimal gewählten Entscheidungsweg speichert.

Ein Rekursionsanker testet zunächst, ob der Laufindex gleich der Länge der Voranmeldungsliste ist. Ist dies der Fall, wurden bereits alle Voranmeldungen für die jeweiligen Entscheidungswege betrachtet. Dann werden die Daten des entstandenen Entscheidungsweges in Form einer Instanz der Klasse “DecisionPathData” zurückgegeben. Wenn noch nicht alle Voranmeldungsobjekte betrachtet wurden, muss zunächst eine Kopie vom momentanen Zeitplan und dem Selektierungsarray angelegt werden. Schließlich gehen, wie in der Lösungsidee beschrieben, von jedem unvollständigen Entscheidungsweg zwei weitere Entscheidungswege aus, welche das momentane Voranmeldungsobjekt entweder ausschließen oder miteinbeziehen. Um die Rekursion mit unterschiedlichen Entscheidungswegen fortzusetzen, muss von den Daten des momentanen Entscheidungsweges eine tatsächliche Kopie angelegt werden. Eine Speicherreferenz auf die Variablen ist nicht ausreichend. Um den Zeitplan und das Selektierungsarray zu kopieren, werden zwei neue Arrays angelegt und die Daten des ursprünglichen Entscheidungsweges in die neuen Arrays kopiert. Die Methode “isPRFitting” dient erneut der Prüfung, ob eine Voranmeldung noch dem Zeitplan des vorangegangenen Entscheidungsweges hinzuzufügen ist. Der Methode wird dazu als Parameter die momentane Voranmeldung und die Kopie des Zeitplanes übergeben, der, solange die entsprechenden Kapazitäten durch die momentane Voranmeldung nicht ausgereizt werden, mit der Länge der momentanen Voranmeldung ergänzt wird. Kann die Voranmeldung in den momentanen Entscheidungsweg noch miteinbezogen werden, dann wird sie in der Kopie des Selektierungsarrays mit true selektiert.

Nun starten zwei Rekursionsaufrufe:

(1) Wenn die momentane Voranmeldung noch in den Zeitplan passt, speichert die Variable “included” vom Typ “DecisionPathData” den Entscheidungsweg, inklusive der momentanen Voranmeldung. Dazu wird dieser Voranmeldung der Rückgabewert der Methode “BruteForce” zugeordnet. Der Brute-Force-Methode wird wieder die Liste der Voranmeldungen, dann jedoch die ergänzte Kopie des Zeitplans, das ergänzte Selektierungsarray, die Summe des Ertrags des vorherigen Entscheidungsweges und des Ertrags der miteinzubeziehenden Voranmeldung und den um eins erhöhten Index, übergeben. Wenn die momentane Voranmeldung nicht miteinbezogen werden kann, bleibt die Variable “included” ein leeres Entscheidungswegobjekt.

(2) Die Variable “excluded” speichert den Entscheidungsweg exklusive der momentanen Voranmeldung. Dazu wird wieder der Rückgabewert des rekursiven Aufrufs der Methode “BruteForce” zugeordnet. Dieses Mal werden der Brute-Force-Methode jedoch nur die Daten des vorherigen Entscheidungsweges übergeben, da die derzeit betrachtete Voranmeldung nicht miteinbezogen wird. Nur der Index wird für den nächsten Rekursionsaufruf ebenfalls um eins erhöht.

Um den bestmöglichen Entscheidungsweg zurückzugeben wird nun noch getestet, welcher von den beiden Wegen, “included” oder “excluded”, zum höheren Ertrag führt. Nachdem also die Rekursion durchlaufen ist, werden dazu die beiden Ertragsattribute der Entscheidungswegobjekte verglichen. Je nach dem, welcher Entscheidungsweg zu einem höheren Ertrag geführt hat, wird einer von beiden Entscheidungswegobjekten von der Methode “BruteForce” zurückgegeben. Abbildung 4 zeigt einige Rückgabewerte anhand des in der Lösungsidee verwendeten Beispiels. Wenn am Ende des Baumes der Rekursionsanker greift, wird der jeweilige Entscheidungsweg, der zu dem entsprechenden Ertrag führte, zurückgegeben.

Anschließend werden die Entscheidungswege verglichen. Erneut wird der Entscheidungsweg des größeren Ertrags zurückgegeben, bis methodisch das optimale Ergebnis erreicht wird. In Abbildung 4 sind zur besseren Übersicht nur die Ertragswerte als Rückgabewerte dargestellt. Im tatsächlichen Programm wird allerdings, ein Objekt vom Typ “DecisionPathData” zurückgegeben, welches neben dem Ertrag des Entscheidungsweges zusätzlich die Selektion der Voranmeldungen und einen Zeitplan mit den Kapazitätsauslastungen der jeweiligen Stunden speichert.

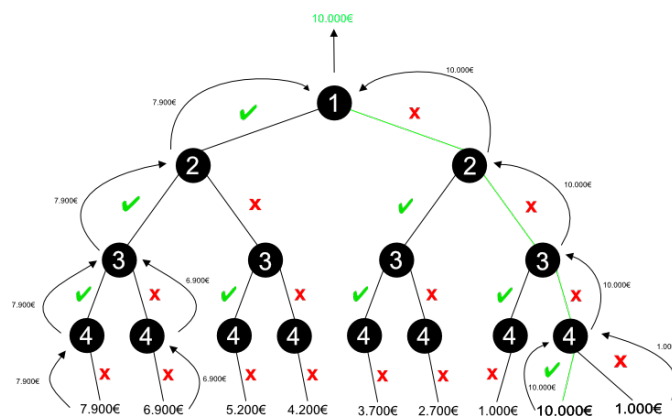


Abbildung 4: Beispiel für Rückgabewerte einiger Entscheidungswege bei der Brute-Force-Methode

Nach der Ermittlung des optimalen Entscheidungsweges wird dieser mit all seinen Daten ausgegeben. Die Beschreibung der Ausgabe erfolgt in Abschnitt 3.5.

3.4 Greedy-Verfahren

Wenn die Anzahl der Voranmeldungen größer ist als fünfzig, wird das folgende Greedy-Verfahren angewandt. Wie in der Lösungsidee beschrieben, müssen die Voranmeldungen vorsortiert werden. Dazu wird die Klasse “PRComparator” verwendet, welche den Java-Comparator implementiert und das Sortieren in Array-Lists vereinfacht. Der “sort”-Methode für ArrayLists wird hier der “PRComparator” übergeben, welcher die Voranmeldungen nach dem Schema sortiert, das im “PRComparator” durch die “compare”-Methode festgelegt wird. Die “compare”-Methode nimmt jeweils zwei Parameter, in diesem Fall Voranmeldungsobjekte entgegen und gibt dann einen Integer zurück, der definiert, welche Voranmeldung in der Sortierung weiter vorne bzw. hinten auftauchen soll.

Die Methode “greedy” nimmt nach erfolgreicher Sortierung eine Liste von Voranmeldungen entgegen und gibt zur Selektion einen approximativen Entscheidungsweg zurück. Dafür wird zunächst wieder ein leerer Zeitplan in Form eines Integerarrays, ein Booleanarray zur Selektierung der Voranmeldungen und ein Ertrag als Integer initialisiert. Die Voranmeldungen werden konsequent durchlaufen. Für jede Voranmeldung wird geprüft, ob diese die Kapazitäten des Zeitplans in einer oder mehreren Stunden ausreizt. Die Methode “isPRFitting” kann an dieser Stelle nicht verwendet werden. Die Längen der ihr übergebenen Voranmeldungen werden schließlich so lange in einem Zeitplan, in den jeweils betreffenden Stunden der Voranmeldungen, aufaddiert, bis in einer der Stunden die Kapazität überschritten wird. Der Zeitplan soll jedoch hier nicht verändert werden, wenn die Voranmeldung nicht tatsächlich miteinbezogen werden kann. Zunächst werden die Stunden des Zeitplans durchlaufen, welche durch die momentane Voranmeldung beansprucht werden. An diesen Stellen wird geprüft, ob das Addieren der Länge des Flohmarktstandes der derzeitigen Voranmeldung die Kapazitätsgrenze von 1000m in den jeweiligen Stunden übersteigt. Ist dies der Fall, wird mit der nächsten Voranmeldung fortgefahren. Kann die Voranmeldung jedoch noch hinzugefügt werden, so wird sie am entsprechenden Index im Selektierungsarray markiert. Der Gesamtertrag wird um den Ertrag der hinzuzufügenden Voranmeldung erweitert und schlussendlich auch der Zeitplan ergänzt. Die betreffenden Stunden werden auf dem Zeitplan also ein weiteres Mal durchlaufen. Dabei wird die Länge des Flohmarktstandes der betreffenden Voranmeldung auf die bereits belegten Längen in den Stunden aufaddiert. Dann wird mit der nächsten Voranmeldung fortgefahren.

Der Prozess wird final durch Ausgabe eines Entscheidungsobjektes. Dem Konstruktor der Klasse “DecisionPathData” wird dazu der Zeitplan, das Selektierungsarray und der finale Ertrag übergeben. Im

Anschluss gibt die folgende Methode den Entscheidungsweg auf die Konsole aus.

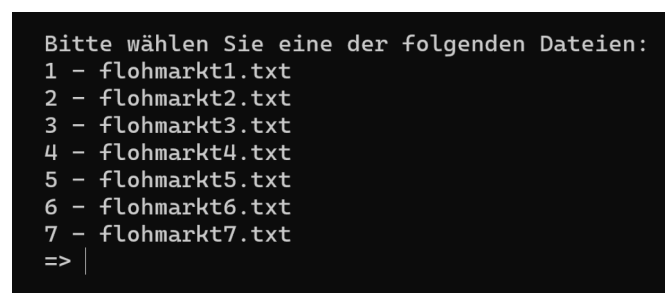
3.5 Ausgabe

Für die Ausgabe besteht die Methode “printRes”. Diese nimmt die Liste von Voranmeldungen, einen Entscheidungsweg, also ein Objekt vom Typ “DecisionPathData” und das verwendete Verfahren als String entgegen und generiert daraus eine aussagekräftige Ausgabe. Zuerst wird der Ertrag des Entscheidungsweges ausgegeben. Die Voranmeldungen werden dann mit ID, Zeit, Länge und individuellem Ertrag der Ausgabe hinzugefügt. Wurde die jeweilige Voranmeldung für den übergebenen Entscheidungsweg selektiert (Selektierungsarray gibt true aus), wird auch diese Information in der Ausgabe berücksichtigt. Dabei wird gespeichert, ob jede Voranmeldung miteinbezogen wurde oder nicht, da am Ende dazu noch eine Nachricht ausgegeben wird. Nun wird noch ein weiterer Zeitplan angelegt, welcher alle Voranmeldungen miteinbezieht, um zu sehen, wie in diesem Fall die Kapazitätsauslastung der einzelnen Stunden aussieht. Zunächst wird die Zeitplanbelegung des Zeitplans des übergebenen Entscheidungsweges ausgegeben. Dann wird der Zeitplan, unter Berücksichtigung aller Voranmeldungen, ausgegeben. Eine boolsche Variable speichert dabei, ob in einer der zehn Stunden im Zeitplan aller Voranmeldungen die Kapazitätsgrenze von 1000m überschritten wird, die Kapazitätsauslastung des Zeitplans des Entscheidungsweges in dieser Stunde unter der Kapazitätsgrenze liegt und die Kapazitätsauslastungen in der Stunde bei beiden Zeitplänen ungleich sind. Ist dies nämlich der Fall, dann konnte die Aufgabe nur approximativ gelöst werden. Andernfalls wurde in jeder Stunde die Kapazität maximal ausgelastet ohne diese jedoch zu überschreiten. Je nach dem, ob die Aufgabe approximativ oder exakt gelöst werden konnte, wird dann noch eine Aussage über diese Optimalität der Lösung ausgegeben. Zudem wird noch die Nachricht ausgegeben, ob alle Voranmeldungen miteinbezogen werden konnten oder nicht.

Schlussendlich ist noch anzumerken, dass den Methoden neben allen genannten Parametern auch eine Kapazitätsgrenze und eine Startzeit oder gegebenenfalls auch eine Endezeit des gesamten Flohmarktes mitgegeben wird. Dies war während des Testens notwendig, da Versuche zum Erproben der Verfahren verwendet wurden, in denen die Zeiträume und Kapazitätsgrenzen von den Vorgaben der Aufgabenstellung abwichen.

4 Beispiele

In diesem Abschnitt werden noch einige Beispiele vorgestellt, welche durch die verschiedenen Verfahren unterschiedlich gelöst werden konnten. In der Lösung zu Aufgabe 2, gibt es ein Konsoleninterface, über das, durch das Eingeben einer gegebenen Nummer, eine Datei gewählt werden kann. Abbildung 5 zeigt dieses Konsolenmenü beim Aufrufen des Programmes.



```
Bitte wählen Sie eine der folgenden Dateien:
1 - flohmarkt1.txt
2 - flohmarkt2.txt
3 - flohmarkt3.txt
4 - flohmarkt4.txt
5 - flohmarkt5.txt
6 - flohmarkt6.txt
7 - flohmarkt7.txt
=> |
```

Abbildung 5: Konsolenmenü nach Aufrufen des Programmes

Die Ausgaben sind dann von den verwendeten Beispieldateien und den Verfahren abhängig, mit denen diese zu lösen sind. Abbildung 6 zeigt die Ausgabe des Ergebnisses bei Anwendung des dreigleisigen Verfahrens auf die erste Testdatei. Hier ist es möglich, alle Voranmeldungen einzubeziehen. Zur besseren Übersicht wurden einige Voranmeldungen in der Darstellung ausgelassen. Bei Ausführung des Programms in der Konsole werden jedoch alle Voranmeldungen gelistet.

Abbildung 7 zeigt die Ergebnisausgabe für die Testdatei 2. Diese ist besonders interessant, da sie als

```

Ertrag: 8028

Voranmeldungen:
ID: 0, Uhrzeit: 8-18, Länge: 2, Ertrag: 20 - Diese Voranmeldung wurde miteinbezogen!
ID: 1, Uhrzeit: 13-18, Länge: 3, Ertrag: 15 - Diese Voranmeldung wurde miteinbezogen!
ID: 2, Uhrzeit: 8-18, Länge: 3, Ertrag: 30 - Diese Voranmeldung wurde miteinbezogen!
ID: 3, Uhrzeit: 8-13, Länge: 3, Ertrag: 15 - Diese Voranmeldung wurde miteinbezogen!
ID: 4, Uhrzeit: 8-13, Länge: 2, Ertrag: 10 - Diese Voranmeldung wurde miteinbezogen!
ID: 5, Uhrzeit: 8-15, Länge: 1, Ertrag: 7 - Diese Voranmeldung wurde miteinbezogen!
ID: 6, Uhrzeit: 13-18, Länge: 2, Ertrag: 10 - Diese Voranmeldung wurde miteinbezogen!
ID: 7, Uhrzeit: 13-17, Länge: 6, Ertrag: 24 - Diese Voranmeldung wurde miteinbezogen!
ID: 8, Uhrzeit: 8-13, Länge: 1, Ertrag: 5 - Diese Voranmeldung wurde miteinbezogen!
:
ID: 489, Uhrzeit: 13-18, Länge: 8, Ertrag: 40 - Diese Voranmeldung wurde miteinbezogen!

Die Stunden konnten somit folgendermaßen belegt werden:
8-9 Uhr: 718m
9-10 Uhr: 754m
10-11 Uhr: 755m
11-12 Uhr: 782m
12-13 Uhr: 780m
13-14 Uhr: 995m
14-15 Uhr: 954m
15-16 Uhr: 869m
16-17 Uhr: 752m
17-18 Uhr: 669m

Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:
8-9 Uhr: 718m
9-10 Uhr: 754m
10-11 Uhr: 755m
11-12 Uhr: 782m
12-13 Uhr: 780m
13-14 Uhr: 995m
14-15 Uhr: 954m
15-16 Uhr: 869m
16-17 Uhr: 752m
17-18 Uhr: 669m
Alle Voranmeldungen konnten miteinbezogen werden.
Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.

```

Abbildung 6: Ausgabe des Ergebnisses für Testdatei 1

einzigste Testdatei nur approximativ gelöst werden kann. In den kritischen Stunden kommt die Kapazitätsauslastung zwar nah an das Kapazitätslimit, bei Einbeziehung aller Voranmeldungen, heran, erreicht dieses jedoch von 14 bis 18 Uhr nicht. Somit kann nicht eindeutig gesagt werden, ob ein anderer Entscheidungsweg zu einem optimaleren Ergebnis geführt hätte. Im Zeitraum von 8 - 14 Uhr wird die maximale Kapazitätsauslastung, durch Einbeziehung aller Voranmeldungen, erreicht. Von 13 - 14 Uhr, würde das Kapazitätslimit bei Einbeziehung aller Voranmeldungen überschritten, jedoch liegt die Kapazitätsauslastung hier genau bei 1000m. Somit wurde der Zeitraum von 8 - 14 Uhr optimal belegt. Zusätzlich kann man erkennen, dass im Gegensatz zu Abbildung 6 die Voranmeldungen nicht fortlaufend geordnet sind. Dies ist dadurch bedingt, dass das Greedy-Verfahren zur Anwendung kam und beschriebene Umsortierungen der Voranmeldungen stattgefunden haben. Bei der Testdatei 1 hingegen konnten alle Voranmeldungen miteingebracht werden, ohne zu überlappen. Somit sind die ID's hier noch in der ursprünglichen Reihenfolge.

Um auch noch ein Beispiel für das Brute-Force-Verfahren zu präsentieren, wird in Abbildung 8 die Ausgabe der Testdatei 4 gezeigt. Hier können nicht alle Voranmeldungen miteinbezogen werden und ihre Anzahl ist gleichzeitig so gering, dass man per Brute-Force-Verfahren den optimalen Entscheidungsweg finden kann.

5 Laufzeitverhalten

In diesem Abschnitt findet eine Laufzeitanalyse der einzelnen Verfahren statt. Nachdem diese im Einzelnen betrachtet wurden, wird anschließend eine Betrachtung im Best- und Worstcase stattfinden.

5.1 "All-Fitting"

Bevor zwischen Greedy- und Brute-Force-Methode entschieden wird, testet das Programm, ob allen Voranmeldungen entsprochen werden kann, ohne das Kapazitätslimit von 1000m in den einzelnen Stunden

```

Ertrag: 9056

Voranmeldungen:
ID: 548, Uhrzeit: 8-18, Länge: 8, Ertrag: 80 - Diese Voranmeldung wurde miteinbezogen!
ID: 587, Uhrzeit: 8-18, Länge: 8, Ertrag: 80 - Diese Voranmeldung wurde miteinbezogen!
ID: 362, Uhrzeit: 8-18, Länge: 7, Ertrag: 70 - Diese Voranmeldung wurde miteinbezogen!
ID: 530, Uhrzeit: 8-18, Länge: 7, Ertrag: 70 - Diese Voranmeldung wurde miteinbezogen!
ID: 47, Uhrzeit: 8-18, Länge: 6, Ertrag: 60 - Diese Voranmeldung wurde miteinbezogen!
ID: 186, Uhrzeit: 8-18, Länge: 6, Ertrag: 60 - Diese Voranmeldung wurde miteinbezogen!
ID: 282, Uhrzeit: 8-18, Länge: 6, Ertrag: 60 - Diese Voranmeldung wurde miteinbezogen!
ID: 462, Uhrzeit: 8-18, Länge: 6, Ertrag: 60 - Diese Voranmeldung wurde miteinbezogen!

:

ID: 452, Uhrzeit: 17-18, Länge: 2, Ertrag: 2 - Diese Voranmeldung wurde miteinbezogen!

Die Stunden konnten somit folgendermaßen belegt werden:
8-9 Uhr: 878m
9-10 Uhr: 921m
10-11 Uhr: 928m
11-12 Uhr: 947m
12-13 Uhr: 952m
13-14 Uhr: 1000m
14-15 Uhr: 977m
15-16 Uhr: 910m
16-17 Uhr: 811m
17-18 Uhr: 732m

Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:
8-9 Uhr: 878m
9-10 Uhr: 921m
10-11 Uhr: 928m
11-12 Uhr: 947m
12-13 Uhr: 952m
13-14 Uhr: 1262m
14-15 Uhr: 1215m
15-16 Uhr: 1102m
16-17 Uhr: 959m
17-18 Uhr: 838m
Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.
Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens approximativ gelöst werden.

```

Abbildung 7: Ausgabe des Ergebnisses für Testdatei 2

zu übersteigen. Dazu werden die Voranmeldungen einmal ganz durchlaufen. Für jede Voranmeldung, wird zunächst getestet, ob ihr Hinzufügen zu dem bisherigen Entscheidungsweg noch möglich ist, also die Kapazitäten in den einzelnen Stunden dadurch nicht ausgereizt werden. Dazu wird der Zeitraum durchlaufen, welcher von der jeweiligen Voranmeldung beansprucht wird. Wenn die Voranmeldung nicht mehr hinzugefügt werden kann, können nicht alle Voranmeldungen miteinbezogen werden. Dann bricht die Methode an dieser Stelle ab und es wird mit einem der beiden anderen Verfahren fortgesetzt. Diese Methode ist also bezüglich der Laufzeit am intensivsten, wenn alle Voranmeldungen miteinbezogen werden können. Sei n die Anzahl der Voranmeldungen und n_t die Anzahl der Stunden, welche von der jeweiligen Voranmeldungen beansprucht werden, ergibt sich hier folgendes Laufzeitverhalten:

$$O\left(\sum_{i=1}^n n_{t_i}\right) \leq O(10n)$$

5.2 “Brute-Force”

Die Brute-Force-Methode hat in der Einzelbetrachtung die längste Laufzeit. Da für jede Voranmeldung zwei Entscheidungswege, existieren und vor allem jeder dieser Entscheidungswege bis zum Ende getestet werden muss, ergibt sich hier etwa eine Laufzeit von $O(2^n)$. Wenn nicht alle Voranmeldungen miteinbezogen werden können, kann eine exakte Lösung nur unter Verwendung eines Algorithmuses mit exponentieller Laufzeit gefunden werden. Da approximative Ansätze, wie das Greedy-Verfahren, kleinere Eingaben nur sehr ungenau lösen können, werden diese hier mit dem Brute-Force-Verfahren gelöst. Beispiele, welche mehr als fünfzig Voranmeldungen enthalten, sollten besser approximativ gelöst werden.

5.3 “Greedy”

Das Greedy-Verfahren beginnt mit einer Vorsortierung der Voranmeldungen nach dem bereits genannten Schema. Die Programmiersprache Java verwendet dazu den “TimSort”, welcher auf dem InsertionSort und dem MergeSort basiert. Beim Sortieren entsteht hier dementsprechend eine Laufzeit von $O(n \log(n))$.

```

Ertrag: 7370

Voranmeldungen:
ID: 0, Uhrzeit: 8-15, Länge: 249, Ertrag: 1743 - Diese Voranmeldung wurde miteinbezogen!
ID: 1, Uhrzeit: 15-18, Länge: 526, Ertrag: 1578 - Diese Voranmeldung wurde miteinbezogen!
ID: 2, Uhrzeit: 12-15, Länge: 520, Ertrag: 1560
ID: 3, Uhrzeit: 8-9, Länge: 503, Ertrag: 503 - Diese Voranmeldung wurde miteinbezogen!
ID: 4, Uhrzeit: 9-15, Länge: 477, Ertrag: 2862 - Diese Voranmeldung wurde miteinbezogen!
ID: 5, Uhrzeit: 8-12, Länge: 171, Ertrag: 684 - Diese Voranmeldung wurde miteinbezogen!
ID: 6, Uhrzeit: 14-18, Länge: 401, Ertrag: 1604

Die Stunden konnten somit folgendermaßen belegt werden:
8-9 Uhr: 923m
9-10 Uhr: 897m
10-11 Uhr: 897m
11-12 Uhr: 897m
12-13 Uhr: 726m
13-14 Uhr: 726m
14-15 Uhr: 726m
15-16 Uhr: 526m
16-17 Uhr: 526m
17-18 Uhr: 526m

Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:
8-9 Uhr: 923m
9-10 Uhr: 897m
10-11 Uhr: 897m
11-12 Uhr: 897m
12-13 Uhr: 1246m
13-14 Uhr: 1246m
14-15 Uhr: 1647m
15-16 Uhr: 927m
16-17 Uhr: 927m
17-18 Uhr: 927m
Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.
Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.

```

Abbildung 8: Ausgabe des Ergebnisses für Testdatei 3

Darauffolgend werden alle Voranmeldungen durchlaufen. Die Stunden, welche durch die jeweilige Voranmeldung belegt werden, werden dabei durchlaufen, um zu testen, ob die Voranmeldung noch in den Entscheidungsweg passt. Wenn nicht, dann wird mit der nächsten Voranmeldung fortgesetzt. Wenn die Voranmeldung jedoch noch miteinzubeziehen ist, dann werden die betreffenden Stunden nochmal durchlaufen, um den Zeitplan durch die Länge der Voranmeldung in den betreffenden Stunden zu ergänzen. Im Worst-Case können hier also alle Voranmeldungen bis auf eine einzige miteinbezogen werden. In diesem Fall ergibt sich eine Laufzeit von $O\left(2 \sum_{i=1}^{n-1} n_{t_i} + n_{t_n}\right)$. Für große Eingaben ist jedoch das einfache Durchlaufen der Voranmeldungen und deren Stunden unbedeutend. Die Sortierung vorab ist am bezüglich der Laufzeit am aufwendigsten.

5.4 Best- und Worstcase-Betrachtung

Im besten Fall können alle Voranmeldungen miteinbezogen werden, ohne die Kapazitätsgrenze in den einzelnen Stunden zu überschreiten. In diesem Fall müssen die Voranmeldungen, wie bereits beschrieben, einmal zusammen mit ihren Stunden durchlaufen werden. Wenn alle Voranmeldungen den gesamten Zeitraum des Flohmarktes beanspruchen, dann ergibt sich hier maximal eine Laufzeit von $O(10n)$, da sich der Flohmarkt nur über zehn Stunden erstreckt.

Im schlechtesten Fall liegt die Laufzeit bei $O(2^n)$ unter Anwendung des Brute-Force-Verfahrens. Bei Begrenzung dieses Verfahrens auf fünfzig Voranmeldungen liegt die Laufzeit also bei 2^{50} . Somit bleibt das Laufzeitwachstum des Brute-Force-Verfahrens begrenzt und würde für sehr große n bei Anwendung des Greedy-Verfahrens überstiegen werden. Die Ungleichung $2^{50} < n \log(n)$ ist erfüllt für $n > 8 \cdot 10^{13}$, was als Anzahl für Voranmeldungen nicht realistisch ist. Daher bleibt das Laufzeitverhalten im Worstcase bei 2^{50} . An dieser Stelle wird ein Kompromiss zwischen Effizienz und Optimalität getroffen. Je größer die Anzahl der Voranmeldungen ist, für die eine exakte Lösung gefunden werden soll, desto größer ist die Anzahl der Voranmeldungen auf die das Brute-Force-Verfahren mit der Laufzeit $O(2^n)$ angewandt wird. Bei Verzicht auf die exakte Lösung kann die Laufzeit auf $O(n \log(n))$ reduziert werden. Je größer die Anzahl der Voranmeldungen, desto optimaler löst die Greedy-Variante.

6 Ergebnistabelle

Die Ergebnistabelle umfasst nur die Auswahlen aus Voranmeldungen in Beispielen, die überschaubar sind. Die Auswahl aus Voranmeldungen in den großen Testdateien wird einerseits auch aus den Belegungsplänen erkennbar, kann jedoch auch durch Anwendung des Programms eingesehen werden.

Testdatei	Ertrag	Auswahl der Voranmeldungen	Belegungspläne	Aussage
flohmarkt1.txt	8028€	Alle Voranmeldungen wurden selektiert (siehe Programmausgabe)	<p>Die Stunden konnten somit folgendermaßen belegt werden:</p> <p>8-9 Uhr: 718m</p> <p>9-10 Uhr: 754m</p> <p>10-11 Uhr: 755m</p> <p>11-12 Uhr: 782m</p> <p>12-13 Uhr: 780m</p> <p>13-14 Uhr: 995m</p> <p>14-15 Uhr: 954m</p> <p>15-16 Uhr: 869m</p> <p>16-17 Uhr: 752m</p> <p>17-18 Uhr: 669m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:</p> <p>8-9 Uhr: 718m</p> <p>9-10 Uhr: 754m</p> <p>10-11 Uhr: 755m</p> <p>11-12 Uhr: 782m</p> <p>12-13 Uhr: 780m</p> <p>13-14 Uhr: 995m</p> <p>14-15 Uhr: 954m</p> <p>15-16 Uhr: 869m</p> <p>16-17 Uhr: 752m</p> <p>17-18 Uhr: 669m</p>	<p>Alle Voranmeldungen konnten miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>

flohmarkt2.txt	9056€	Siehe Programm- ausgabe	<p>Die Stunden konnten somit folgendermaßen belegt werden:</p> <p>8-9 Uhr: 878m 9-10 Uhr: 921m 10-11 Uhr: 928m 11-12 Uhr: 947m 12-13 Uhr: 952m 13-14 Uhr: 1000m 14-15 Uhr: 977m 15-16 Uhr: 910m 16-17 Uhr: 811m 17-18 Uhr: 732m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:</p> <p>8-9 Uhr: 878m 9-10 Uhr: 921m 10-11 Uhr: 928m 11-12 Uhr: 947m 12-13 Uhr: 952m 13-14 Uhr: 1262m 14-15 Uhr: 1215m 15-16 Uhr: 1102m 16-17 Uhr: 959m 17-18 Uhr: 838m</p>	<p>Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens approximativ gelöst werden.</p>
----------------	-------	----------------------------	---	---

flohmarkt3.txt	9056€	Siehe Programm- ausgabe	<p>Die Stunden konnten somit folgendermaßen belegt werden:</p> <p>8-9 Uhr: 616m 9-10 Uhr: 823m 10-11 Uhr: 976m 11-12 Uhr: 1000m 12-13 Uhr: 1000m 13-14 Uhr: 1000m 14-15 Uhr: 1000m 15-16 Uhr: 1000m 16-17 Uhr: 1000m 17-18 Uhr: 363m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:</p> <p>8-9 Uhr: 616m 9-10 Uhr: 823m 10-11 Uhr: 976m 11-12 Uhr: 1093m 12-13 Uhr: 1249m 13-14 Uhr: 1348m 14-15 Uhr: 1332m 15-16 Uhr: 1199m 16-17 Uhr: 1011m 17-18 Uhr: 363m</p>	<p>Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>
----------------	-------	----------------------------	---	--

flohmarkt4.txt	7370€	<p>ID: 0, Uhrzeit: 8-15, Länge: 249, Ertrag: 1743 - Diese Voranmeldung wurde miteinbezogen!</p> <p>ID: 1, Uhrzeit: 15-18, Länge: 526, Ertrag: 1578 - Diese Voranmeldung wurde miteinbezogen!</p> <p>ID: 2, Uhrzeit: 12-15, Länge: 520, Ertrag: 1560</p> <p>ID: 3, Uhrzeit: 8-9, Länge: 503, Ertrag: 503 - Diese Voranmeldung wurde miteinbezogen!</p> <p>ID: 4, Uhrzeit: 9-15, Länge: 477, Ertrag: 2862 - Diese Voranmeldung wurde miteinbezogen!</p> <p>ID: 5, Uhrzeit: 8-12, Länge: 171, Ertrag: 684 - Diese Voranmeldung wurde miteinbezogen!</p> <p>ID: 6, Uhrzeit: 14-18, Länge: 401, Ertrag: 1604</p>	<p>Die Stunden konnten somit folgendermaßen belegt werden:</p> <p>8-9 Uhr: 923m 9-10 Uhr: 897m 10-11 Uhr: 897m 11-12 Uhr: 897m 12-13 Uhr: 726m 13-14 Uhr: 726m 14-15 Uhr: 726m 15-16 Uhr: 526m 16-17 Uhr: 526m 17-18 Uhr: 526m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:</p> <p>8-9 Uhr: 923m 9-10 Uhr: 897m 10-11 Uhr: 897m 11-12 Uhr: 897m 12-13 Uhr: 1246m 13-14 Uhr: 1246m 14-15 Uhr: 1647m 15-16 Uhr: 927m 16-17 Uhr: 927m 17-18 Uhr: 927m</p>	<p>Nur ein Teil der Voranmeldungen konnte miteinbezogen werden. Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>
----------------	-------	---	--	---

flohmarkt5.txt	8705€	Siehe Programm- ausgabe.	<p>Die Stunden konnten somit folgendermaßen belegt werden:</p> <p>8-9 Uhr: 955m 9-10 Uhr: 986m 10-11 Uhr: 982m 11-12 Uhr: 916m 12-13 Uhr: 551m 13-14 Uhr: 551m 14-15 Uhr: 952m 15-16 Uhr: 958m 16-17 Uhr: 927m 17-18 Uhr: 927m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden:</p> <p>8-9 Uhr: 2629m 9-10 Uhr: 3348m 10-11 Uhr: 1902m 11-12 Uhr: 3916m 12-13 Uhr: 3791m 13-14 Uhr: 3650m 14-15 Uhr: 4501m 15-16 Uhr: 4222m 16-17 Uhr: 2054m 17-18 Uhr: 927m</p>	<p>Nur ein Teil der Voranmeldungen konnte miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>
----------------	-------	-----------------------------	--	--

flohmarkt6.txt	10000€	Alle Voranmeldungen wurde miteinbezogen (siehe Programmausgabe)	<p>Die Stunden konnten somit folgendermaßen belegt werden: 8-9 Uhr: 1000m 9-10 Uhr: 1000m 10-11 Uhr: 1000m 11-12 Uhr: 1000m 12-13 Uhr: 1000m 13-14 Uhr: 1000m 14-15 Uhr: 1000m 15-16 Uhr: 1000m 16-17 Uhr: 1000m 17-18 Uhr: 1000m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden: 8-9 Uhr: 1000m 9-10 Uhr: 1000m 10-11 Uhr: 1000m 11-12 Uhr: 1000m 12-13 Uhr: 1000m 13-14 Uhr: 1000m 14-15 Uhr: 1000m 15-16 Uhr: 1000m 16-17 Uhr: 1000m 17-18 Uhr: 1000m</p>	<p>Alle Voranmeldungen konnten miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>
----------------	--------	---	---	---

flohmarkt7.txt	10000€	Alle Voranmeldungen wurden miteinbezogen (siehe Programmausgabe)	<p>Die Stunden konnten somit folgendermaßen belegt werden: 8-9 Uhr: 1000m 9-10 Uhr: 1000m 10-11 Uhr: 1000m 11-12 Uhr: 1000m 12-13 Uhr: 1000m 13-14 Uhr: 1000m 14-15 Uhr: 1000m 15-16 Uhr: 1000m 16-17 Uhr: 1000m 17-18 Uhr: 1000m</p> <p>Bei Einbeziehung aller Voranmeldungen wären in den einzelnen Stunden folgende Kapazitätsauslastungen vorzufinden: 8-9 Uhr: 1000m 9-10 Uhr: 1000m 10-11 Uhr: 1000m 11-12 Uhr: 1000m 12-13 Uhr: 1000m 13-14 Uhr: 1000m 14-15 Uhr: 1000m 15-16 Uhr: 1000m 16-17 Uhr: 1000m 17-18 Uhr: 1000m</p>	<p>Alle Voranmeldungen konnten miteinbezogen werden.</p> <p>Das Beispiel konnte durch Anwendung des dreigleisigen Verfahrens exakt gelöst werden.</p>
----------------	--------	--	---	---

7 Sourcecode

In diesem Abschnitt wird die Implementierung der einzelnen Verfahren, sowie deren Zusammenführung in der Hauptmethode dargestellt.

7.1 Main

```
1  /**
2   * Hauptmethode fuer die Loesung des Problems.
3   *
4   */
5  public static void main(String[] args) {
6      // Nutzereingabe => Bestimmung des Dateipfades.
7      String filePath = consoleInterface();
8
9      // Generiert eine Liste von Voranmeldungen aus den Daten der Datei.
10     ArrayList<PreRegistration> pRs = new Data(filePath).getPRs();
11
12     // Entscheidungsweg fuer das Miteinbeziehen aller Voranmeldungen.
13     DecisionPathData allFitting = allFitting(pRs, 1000, 8);
14
15     if (allFitting.getCosts() > -1) {
16         // Gibt den Entscheidungsweg aus, wenn alle Voranmeldungen passen.
17         printRes(pRs, allFitting, "all_fitting", 8, 1000);
18     } else if (pRs.size() <= 50) {
19         // Brute-Force-Verfahren, wenn die Anzahl der Voranmeldungen <= 50 ist.
20         DecisionPathData res = bruteForce(pRs, new int[10],
21             new boolean[pRs.size()], 0, 0, 8, 1000);
22
23         // Gibt das Ergebnis der Brute-Force-Methode aus.
24         printRes(pRs, res, "brute_force", 8, 1000);
25     } else {
26         // Greedy-Verfahren bei Voranmeldungen
27
28         // Vorsortierung der Voranmeldungen.
29         pRs.sort(new PRComparator());
30
31         // Erstellt den Entscheidungsweg nach dem Greedy-Verfahren.
32         DecisionPathData res = greedy(pRs, 1000, 8);
33
34         // Gibt das Ergebnis des Greedy-Verfahrens aus.
35         printRes(pRs, res, "greedy", 8, 1000);
36     }
37 }
```

7.2 allFitting

```
1  /**
2   * Testet, ob alle Voranmeldungen passen.
3   *
4   * @param pRs      Liste der Voranmeldungen
5   * @param maxCap    Kapazitaetslimit
6   * @param startTime Startuhrzeit des Flohmarktes
7   * @return         Einen optimalen Entscheidungsweg, wenn alle Voranmeldungen passen
8   *                Einen Entscheidungsweg mit dem Ertrag -1, um zu symbolisieren,
9   *                dass nicht alle Voranmeldungen passen
10  */
11 private static DecisionPathData allFitting(ArrayList<PreRegistration> pRs, int maxCap,
12     int startTime) {
13     // Wenn alle Voranmeldungen passen:
14     // Der dabei entstehende Ertrag
15     int costs = 0;
16
17     // Die Belegung des Zeitplanes
18     int[] tP = new int[10];
19
20     // Die ausgewaehlten Voranmeldungen (alle)
21     boolean[] selected = new boolean[pRs.size()];
22
23     // Durchlaeuft die Voranmeldungen.
24     for (int i = 0; i < pRs.size(); i++) {
25
26         // Erhoeht den Ertrag und selektiert die Voranmeldung, falls sie noch passt.
27         if (isPRFitting(pRs.get(i), tP, maxCap, startTime)) {
28             costs += pRs.get(i).getCosts();
29             selected[i] = true;
30         } else {
31             // Gibt einen Entscheidungsweg mit Ertrag -1 zurueck,
32             // wenn nicht alle Voranmeldungen passen.
33             return new DecisionPathData(tP, selected, -1);
34         }
35     }
36     return new DecisionPathData(tP, selected, costs);
37 }
```

7.3 bruteForce

```

1  /**
2   * Wendet das Brute-Force-Verfahren an, um die optimale Wahl aus
3   *   Voranmeldungen zu treffen.
4   *
5   * @param pRs      Liste der Voranmeldungen
6   * @param tP      Zeitplan des momentanen Entscheidungsweges
7   * @param selectedPRs  ausgewählte Voranmeldungen des momentanen Entscheidungsweges
8   * @param costs    Ertrag des momentanen Entscheidungsweges
9   * @param i        Laufindex ueber die Voranmeldungen
10  * @param startTime Startzeit des Flohmarktes
11  * @param maxCap    maximale Kapazitaet
12  * @return          Daten der endgueltigen Entscheidungswege
13  */
14  private static DecisionPathData bruteForce(ArrayList<PreRegistration> pRs,
15      int[] tP, boolean[] selectedPRs, int costs, int i,
16      int startTime, int maxCap) {
17
18      // Rekursionsanker: Wenn jede Voranmeldung in Betracht gezogen wurde,
19      // wird der endgueltige Entscheidungsweg zurueckgegeben.
20      if (i == pRs.size()) {
21          return new DecisionPathData(tP, selectedPRs, costs);
22      }
23
24      // "copy of time plan": Eine Kopie des Zeitplanes,
25      // die fuer diesen Entscheidungsweg ergaenzt wird
26      int[] cTP = new int[tP.length];
27
28      // "current selected pre registrations": Eine zu ergaenzende Kopie der ausgewaehlten
29      // Voranmeldungen fuer diesen Entscheidungsweg.
30      boolean[] cSPRs = new boolean[pRs.size()];
31
32      //Kopiert die Auswahlmarkierungen der Voranmeldungen
33      for (int j = 0; j < selectedPRs.length; j++) {
34          if (selectedPRs[j]) {
35              cSPRs[j] = true;
36          }
37      }
38
39      //Kopiert den uebergebenen Zeitplan
40      for (int j = 0; j < cTP.length; j++) {
41          cTP[j] = tP[j];
42      }
43
44      // Entscheidungsweg mit der Voranmeldung am momentanen Index miteinbezogen
45      DecisionPathData included = new DecisionPathData();
46
47      // Der Entscheidungsweg wird gegangen, wenn die momentane Voranmeldung noch passt.
48      if (isPRFitting(pRs.get(i), cTP, maxCap, startTime)) {
49          // Markiert die Voranmeldung als ausgewaehlt.
50          cSPRs[i] = true;
51
52          // Rekursionsaufruf mit den veraenderten Daten des Entscheidungsweges
53          included = bruteForce(pRs, cTP, cSPRs, costs+pRs.get(i).getCosts(),
54              i+1, startTime, maxCap);
55      }
56
57      // Rekursionsaufruf fuer den Entscheidungsweg exklusive
58      // der Voranmeldung am momentanen Index
59      DecisionPathData excluded = bruteForce(pRs, tP, selectedPRs, costs,
60          i+1, startTime, maxCap);
61
62      // Gibt den Entscheidungsweg zurueck, welcher einen hoeheren Ertrag bringt.
63      if (included.getCosts() > excluded.getCosts()) {
64          return included;
65      } else {
66          return excluded;
67      }
68  }

```

7.4 greedy

```

/**
 * Wendet das Greedy-Verfahren auf die vorsortierte Liste von Voranmeldungen an.
 *
 * @param pRs      Vorsortierte Liste der Voranmeldungen
 * @param maxCap    maximale Kapazitaet
 * @param startTime Startzeit des Flohmarktes
 * @return          Entscheidungsweg durch Greedy-Verfahren
 */
private static DecisionPathData greedy(ArrayList<PreRegistration> pRs, int maxCap,
    int startTime) {
    // Zeitplan des Entscheidungsweges
    int[] tP = new int[10];

    // Array fuer die Selektierung der Voranmeldungen.
    boolean selected[] = new boolean[pRs.size()];

    // Ertrag des Entscheidungsweges
    int costs = 0;

    // Durchlaeuft die Voranmeldungen
    for (int i = 0; i < pRs.size(); i++) {

        // Aktuelle Voranmeldung
        PreRegistration pR = pRs.get(i);

        // true, wenn die momentane Voranmeldung nicht mehr passt
        boolean isFull = false;

        // Durchlaeuft die Zeiten, welche durch die momentane Voranmeldung beansprucht werden
        for (int j = pR.getStart() - startTime; j < pR.getEnd() - startTime; j++) {

            // Testet, ob das Kapazitaetsmaximum erreicht wird.
            if (tP[j] + pR.getLength() > 1000) {
                isFull = true;
                break;
            }
        }

        // Wenn die Voranmeldung noch passt:
        if(!isFull) {
            // Selektiert die Voranmeldung.
            selected[i] = true;

            // Erhoeht den Gesamtertrag um den Ertrag der momentanen Voranmeldung.
            costs += pR.getCosts();

            // Durchlaeuft die entsprechenden Zeiten und belegt den Zeitplan.
            for (int j = pR.getStart() - 8; j < pR.getEnd() - 8; j++) {
                tP[j] = tP[j] + pR.getLength();
            }
        }
    }

    // Gibt den approximativen Entscheidungsweg zurueck.
    return new DecisionPathData(tP, selected, costs);
}

```