

Aufgabe 1: Müllabfuhr

Teilnahme-ID: 62344

Manuel Frohn

10.04.2022

Inhalt

Lösungsansatz	1
Die Cluster Heuristik.....	2
Der Algorithmus von Edmonds und Johnson	3
Der Algorithmus von Hierholzer.....	3
Umsetzung.....	4
Die „getPostmanTour“-Methode.....	5
Beispiele	6
Muellabfuhr 0	6
Muellabfuhr 1	6
Muellabfuhr 2	6
Muellabfuhr 3	7
Muellabfuhr 4	7
Muellabfuhr 5	7
Muellabfuhr 6	9
Muellabfuhr 7	10
Muellabfuhr 8	13
Quellcode	22
Quellen	44

Lösungsansatz

Um diese Aufgabe zu lösen, sollte man, das Straßennetz, als Graphen, modellieren (Kreuzungen als Knoten und Straßen als Kanten). Tut man dies, kann man die Aufgabenstellung, daran angepasst, wie folgt, formulieren: Es gilt fünf Pfade zu finden, die alle beim selben Knoten starten und enden, und die alle zusammen, jede Kante, mindestens einmal, traversieren, und dabei, soll die Länge des längsten Pfades, möglichst gering sein. Dieses Problem ist in der Lektüre als das MinMax k-Chinese Postman Problem (MinMax k-CPP) bekannt. Es handelt sich hierbei, um eine erweiterte Form, des Chinese Postman Problems. Das Chinese Postman Problem, beschäftigt sich damit, einen Pfad, möglichst niedriger Länge, in einem Graphen, zu finden, der alle Kanten, des Graphens, mindestens einmal traversiert und bei demselben Knoten endet, bei dem der Pfad anfängt [1]. Das MinMax k-CPP wurde das erste Mal, in der Arbeit, von Frederickson, Kim und Hecht [2] erläutert. In dieser Arbeit, wurde bewiesen, dass es sich hierbei um ein NP-schweres Problem handelt. Deswegen ist die Nutzung einer Heuristik, zur Lösung dieses Problems, legitim. Ich habe mich dazu entscheiden die Cluster-Heuristik von Ahr und Reinelt [3] zu verwenden. Ich

Seite 1 von 44

Aufgabe 1: Müllabfuhr

Teilnahme-ID: 62344

habe mich für diese Heuristik entschieden, da sie, in den, in der Arbeit von Ahr und Reinelt [3] aufgeführten Beispielen, meistens, am besten abgeschnitten hat und wenn sie das nicht hat, nur geringfügig schlechter abgeschnitten hat, als die Heuristik von Frederickson, Kim und Hecht [2], und die Augment-Merge Heuristik, welche aus derselben Arbeit, wie die Cluster Heuristik, stammt. Des Weiteren erschien mir ein Cluster-First-Route-Second Ansatz besser geeignet, für das Straßennetz, einer Stadt. Bei einem solchen Ansatz, werden die Kanten, des Graphen zuerst in sogenannte „Cluster“, Gruppen von Kanten, die möglichst nah beieinander liegen, aufgeteilt, und erst danach, wird ein Pfad vom Startknoten durch das Cluster und wieder zurück berechnet. Solch ein Ansatz scheint mir passend, für eine Stadt, da auch diese in Viertel einteilbar ist, in denen es weitaus mehr Straßen gibt, an denen man Müll abholen muss, als zwischen diesen Vierteln.

Die Cluster Heuristik

In diesem Abschnitt werden wir uns damit beschäftigen, wie die Cluster Heuristik funktioniert. Gegeben sei ein Graph $G=(V,E)$, aus Knoten $v \in V$ und Kanten $e \in E$, Gewichte der Kanten $w : E \rightarrow \mathbb{R}^+$ und ein Depot Knoten v_0 . Zuerst ermitteln wir nun die „Frequency“ einer Kante. Dazu brauchen wir zuerst die Funktion $SP(v_i, v_j)$, welche den kürzesten Pfad von v_i zu v_j angibt. Danach definieren wir die „Frequency“-Funktion $\phi(e) = 0$ für alle $e \in E$. Nachdem wir die „Frequency“-Funktion definiert haben, machen wir folgendes, für jede Kante $e\{v_i, v_j\} \in E$: Erhöhe für jede Kante $g \in SP(v_0, v_i) \cup SP(v_0, v_j)$ die „Frequency“-Funktion $\phi(g)$ um eins. Dies machen wir, um sogenannte „kritische Kanten“ zu finden. Ist die „Frequency“-Funktion, einer Kante kleiner oder gleich eins, so ist eine Kante kritisch. Eine kritische Kante, ist eine Kante, die nicht auf einem Pfad, durch eine andere Kante enthalten ist. Das heißt, dass man, um diese Kante zu durchlaufen, dies explizite anstreben muss, wogegen eine nicht kritische Kante, automatisch mit irgendeiner kritischen Kante durchlaufen wird. Als nächstes bilden wir aus den kritischen Kanten Cluster. Dazu bestimmen wir zuerst fünf repräsentative Kanten, für diese Cluster. Die erste repräsentative Kante, ist die Kante, aus den kritischen Kanten $g\{v_i, v_j\}$, mit dem höchsten Abstand, berechnet durch $\min(w(SP(v_0, v_i)), w(SP(v_0, v_j)))$, wobei $w(SP)$, die Summe, der Gewichte, aller Kanten $e \in SP$, ist, zum Depot Knoten v_0 . Danach bestimmen alle anderen, repräsentativen Kanten, durch folgendes Verfahren: Finde die Kante, die noch keine repräsentative Kante ist und bei der der geringste Abstand, zu den bereits bestimmten Kanten, der sich, für die Kanten $e\{v_i, v_j\}$ und $g\{w_i, w_j\}$, durch die Funktion $d(e, g) = \max(w(SP(v_i, w_i)), w(SP(v_j, w_j)), w(SP(v_i, w_j)), w(SP(v_j, w_i)))$ berechnen lässt, möglichst groß ist. Dadurch erhalten wir fünf Kanten, die allesamt, möglichst weit, voneinander entfernt sind. Für jede repräsentative Kante legen wir nun ein neues Cluster an. Danach machen wir für alle kritischen Kanten, die nicht als repräsentative Kante ausgewählt worden sind folgendes. Finde, mithilfe der Abstandsfunktion $d(e, g)$, die repräsentative Kante, zu der, die ausgewählte Kante, den geringsten Abstand hat, sollten dies mehrere sein, dann wähle diejenige, deren Cluster am kleinsten ist (Es handelt sich hier um meine eigene Modifikation, die die Ergebnisqualität, bei dichten Graphen, signifikant verbessert), und füge die ausgewählte Kante, dem Cluster hinzu, welches die ausgewählte repräsentative Kante repräsentiert. Danach werden den Clustern alle Kanten hinzugefügt, die auf den

kürzesten Pfaden sind, die, die Endpunkte der Kanten, im Kluster, mit dem Depot Knoten v0 verbinden.

Nachdem, durch diese Methode, alle fünf, Cluster gebildet worden sind. Müssen wir nun das Chinese Postman Problem für diese Cluster lösen. Die dabei errechneten Pfade sind dann unsere Tagesrouten. Um das Chinese Postman Problem, für jedes Cluster, zu lösen, verwenden wir, denn im nächsten Abschnitt erläuterten, Algorithmus von Edmonds und Johnson [4].

Der Algorithmus von Edmonds und Johnson

Der Algorithmus von Edmonds und Jonson [4] verfolgt einen simplen Ansatz. Er erweitert den Graphen (hier unser Cluster), so, dass der Graph eulersch wird. Eulersch heißt, dass jeder Knoten, des Graphen, ein gerades Grad hat, also mit einer geraden Anzahl an Kanten verbunden ist. Danach kann man den Eulerkreis, mithilfe des Algorithmus von Hierholzer bestimmen. Der Eulerkreis ist ein Zyklus, der alle Kanten eines Graphen genau einmal durchläuft.

Um nun, einen Graphen, zu erweitern, so dass er eulersch wird, muss man ihm zusätzliche Kanten hinzufügen. Um herauszufinden, was für Kanten, dem Graphen, hinzugefügt werden sollen, bestimmen wir zuerst alle Knoten ungeraden Grades. Aus diesen Knoten formen wir einen Paarungsgraphen. Dieser Paarungsgraph ist ein vollständiger Graph, dessen Kantengewichte gleich dem Gewicht, des kürzesten Pfades, zwischen den beiden Knoten, der Kante, im Ursprungsgraphen, ist. Für dieses Graphen, wird dann das minimal-weight-perfect-matching, mithilfe des Blossom Algorithmus, aus der Arbeit von Zvi Galil[5], berechnet. Ein perfect-matching ist eine Teilmenge, der Kanten, des Paarungsgraphen, wobei alle Knoten, des Paarungsgraphen, mit genau einer dieser Kanten verbunden ist. Nachdem das Matching bestimmt wurde, werden die Kanten, der kürzesten Pfade, die durch die Kanten, im Matching, repräsentiert werden dem Ursprungsgraphen hinzugefügt. Nachdem der Graph nun so erweitert wurde, kann man, mit dem Algorithmus von Hierholzer [6], den Eulerkreis des erweiterten Graphen bestimmen.

Der Algorithmus von Hierholzer

In diesem Abschnitt werden wir uns mit dem Algorithmus von Hierholzer, zur Findung von Eulerkreisen, in eulerschen Graphen beschäftigen [6]. Der Algorithmus wählt zu beginn, erst einmal, einen beliebigen Knoten. Danach formt er, von diesem Knoten ausgehend, durch Tiefensuche, einen neuen Unterkreis K in G, sodass K keine Kante doppelt durchläuft. Entferne danach alle Kanten, die in K enthalten sind, aus G. Wiederhole folgendes, solange K kein Eulerkreis ist: Nehme einen beliebigen Knoten aus K, dessen Grad in G größer null ist. Konstruiere, von diesem Knoten ausgehend einen neuen Kreis C. Füge diesen Kreis nun in K ein. Ersetze dazu, den, zuvor, in K, ausgewählten Knoten, durch den Kreis C. Lösche alle Kanten von C aus G.

Umsetzung

In diesem Abschnitt werde ich erläutern, wie ich, den im letzten Abschnitt beschrieben, Lösungsansatz umgesetzt habe. Beginnen wir dazu, mit der „Graph“-Klasse, welche einen Graphen repräsentiert. Ein Objekt, dieser Klasse, besitzt, zum einen die „vertexCount“- und „edgeCount“-Integer, welche angeben, wie viele Kanten respektive Knoten der Graph hat, und zum anderen das „edgeSet“- und das „vertexSet“-HashSet, welche jeweils alle Kanten, bzw. Knoten des Graphen speichern. Knoten werden dabei als Integer repräsentiert und Kanten, durch Objekte, der „Edge“-Klasse (Codeausschnitt 1). Ein Objekt, dieser Klasse, hat zwei Integer-Werte („a“ und „b“), die, die Knoten repräsentieren, die diese Kante verbindet und einen Integer-Wert, der das Gewicht („cost“), der Kantere, präsentiert. Zusätzlich besitzt jedes „Graph“-Objekt die „adjacencyList“-Karte, welche, bei Eingabe eines Knotens eine Liste aller Kanten zurückgibt, die mit diesem Knoten verbunden sind, und die „adjacencyList“-Karte, welche eine verschachtelte Karte ist, die, bei der Eingabe zweier Knoten, das „Edge“-Objekt zurückgibt, welches, die Kante, repräsentiert, die, diese Beiden, verbindet.

Kommen wir nun zur Implementierung der Cluster-Heuristik. Die Cluster-Heuristik habe ich in der „getDayRoutes“-Methode implementiert. Diese werden wir nun Stück für Stück durchgehen. Zu Beginn werden erst einmal alle kürzesten Pfade berechnet. Dazu ruft die Methode, die „getShortestPaths“-Methode (Codeausschnitt 2), des „Graph“-Objekts auf. Diese berechnet, unter Verwendung des Dijkstra-Algorithmus [7], alle kürzesten Pfade, zwischen allen Knoten. Ein Pfad wird dabei durch ein „Path“-Objekt (Codeausschnitt 3) repräsentiert. Danach bestimmt die Methode die „Frequency“ aller Kanten (Codeausschnitt 4). Dazu erstellt die Methode zuerst eine neue HashMap, in welcher die „Frequency“ der Kanten gespeichert werden. Danach wird über das Set der Kanten iteriert. Dabei wird, für jede ausgewählte Kante, der kürzeste Pfad, zu ihren beiden Endpunkten genommen, und für jede Kante, die auf diesen Pfaden liegt, wird entweder, sollte die Kante noch nicht in der Karte enthalten sein, diese, zu der Karte, mit dem Wert eins, hinzugefügt, und sonst wird der Wert, der in der Karte enthalten ist um eins erhöht. Nachdem so, die „Frequency“ der Kanten bestimmt wurde, werden die kritischen Kanten herausgesucht (Codeauschnitt 5). Dazu wird ein neues Set angelegt und über das Set aller Kanten iteriert. Es wird für jede Kante geprüft, ob ihre „Frequency“ kleiner gleich eins ist, und sollte dem so sein, so wird, die momentane betrachtete Kante, zu dem Set der kritischen Kanten hinzugefügt. Danach werden die repräsentativen Kanten, durch den, im Lösungsansatz beschriebenen Algorithmus bestimmt (Codeausschnitt 6). Dazu erstellt die Methode zuerst ein neues HashSet. Als nächstes begibt sich die Methode in eine, fünf Iterationen lange, for-Schleife. In der ersten Iteration wird die Kante gesucht, die am weitesten, vom Depot Knoten (0), entfernt ist, und zum HashSet hinzugefügt, und aus dem Set der kritischen Kanten entfernt. In allen anderen Iterationen wird die Kante gesucht, die zu den bereits berechneten repräsentativen Kanten, den höchsten Mindestabstand hat, und zum HashSet hinzugefügt. Nachdem so die repräsentativen Kanten bestimmt worden sind, werden nun die Cluster gebildet (Codeausschnitt 7). Dazu wird zuerst, für jede repräsentative Kante, ein neues HashSet erstellt, und der Kante, in der „clusters“-Karte zugeordnet. Danach wird über das Set der kritischen Kanten iteriert. Für jede Kante

wird, anhand der, im Lösungsansatz beschrieben Kriterien, bestimmt, welcher repräsentativen Kante, die ausgewählte Kante zuzuordnen ist. Nachdem diese Kante ermittelt wurde, wird die ausgewählte Kante dem, zu dieser Kante gehörendem, Set hinzugefügt. Nachdem dies für alle kritischen Kanten geschehen ist, werden nun die Kanten auf den kürzesten Pfaden, durch die kritischen Kanten, den jeweiligen Sets, hinzugefügt. Dazu erstellt die Methode, am Anfang fünf neue Sets, und fügt die Kanten, der alten Sets, den jeweiligen neuen, Sets hinzu. Danach iteriert die Methode über die alten Cluster-Sets. Für jedes, dieser Sets, iteriert die Methode dann über die darin enthaltenen Kanten. Für jede dieser Kanten werden, die kürzesten Wege vom Depot Knoten zu den beiden Knoten, die die ausgewählte Kante verbindet, ermittelt. Darauffolgend werden alle Kanten dieser Pfade, dem jeweiligen neuen Set hinzugefügt (Dadurch, dass ein HashSet ein Objekt nur einmal enthalten kann, werden hier unnötige doppelte Kanten vermieden). Am Ende werden, aus den neuen Sets, „Graph“-Objekte erstellt, und die „getPostmanTour“-Methode dieser neuen Objekte aufgerufen. Die, durch diese Aufrufe zurückgegeben Pfade, werden dann als die Tagesrouten zurückgegeben.

Die „getPostmanTour“-Methode

In diesem Abschnitt werde ich erläutern, wie ich die Berechnung, der Lösung, des Chinese Postman Problems, für die einzelnen Kluster, implementiert habe. Wie man der Überschrift schon entnehmen kann, wird dazu die „getPostmanTour“-Methode, der „Graph“-Klasse verwendet. Diese bestimmt zu beginn, erst einmal alle kürzesten Pfade, zwischen den Knoten, des Graphen. Danach werden die kritischen Knoten bestimmt (Codeausschnitt 8). Dazu iteriert die Methode über das Knotenset, des Graphen, und speichert alle Knoten, die mit einer ungeraden Anzahl an Kanten verbunden sind in einem HashSet. Danach wird, insofern das Set, der kritischen Knoten, nicht leer ist, der Graph so erweitert, dass er eulersch wird (Codeausschnitt 9). Dazu werden zuerst die Kanten für den Paarungsgraphen errechnet. Dies geschieht, indem, für jede einzigartige Paarung, der kritischen Knoten (einzigartig heißt, das, insofern, für die Paarung a,b bereits eine Kante erstellt wurde, keine mehr für die Paarung b,a erstellt wird), eine Kante erstellt wird. Das Gewicht dieser Kante ist dabei die Länge, des kürzesten Pfades, zwischen den beiden Knoten. Die neuerstellte Kante wird dann der Liste, der Kanten, für den Paarungsgraphe, hinzugefügt. Zusätzlich dazu, wird die neuerstellte Kante, in einer HashMap, dem Pfad zugeordnet, durch den, deren Gewicht, bestimmt wurde. Nachdem alle Kanten, für den Paarungsgraphen, bestimmt worden, wird aus diesen ein neues „Graph“-Objekt bestimmt. Für, diese „Graph“-Objekt, wird dann, durch die „getMinimalCostPerfectMatching“-Methode, das minimal-weight-perfect-matching erstellt. Nachdem dies geschehen ist wird folgendes für alle Kanten, die im Matching enthalten sind, getan: Nehme alle Kanten, des Pfade, von dem die Kante ihr Gewicht erhalten hat, und füge Deep Copys dieser dem Cluster-Graphen hinzu. Da der Cluster-Graph, nach dieser Modifikation eulersch ist, wird nun der Algorithmus von Hierholzer verwendet, um eine Eulertour, in ihm, zu bilden (Codeausschnitt 10). Dazu erstellt die Methode zuerst ein leeres Set („coverdEdges“), in das später die bereits, durch den bestehenden Kreis abgedeckten Kanten eingefügt werden. Zusätzlich dazu definiert die Methode das „eulerTour“-Objekt, vom Typ „Path“. Danach begibt sich die Methode in eine while-Schleife, die so lange läuft, bis alle Kanten des Graphen, im „coverdEdges“-Set, enthalten sind. Innerhalb dieser Schleife

wird zuerst abgefragt, ob das „eulerTour“-Objekt gleich null ist. Ist dem so, wird der „base“-Integer auf null gesetzt, wenn nicht, dann wird der „base“-Integer dadurch ermittelt, dass für alle Knoten, des „eulerTour“-Pfades, geprüft wird, ob dieser mit einer Kante verbunden ist, die noch nicht Teil des „coveredEdges“-Sets ist. Wird diese Bedingung erfüllt, so ist der Knoten, für die diese Bedingung erfüllt wurde, der neue „base“-Integer. Danach wird der „currentVertex“-Integer auf den Wert des „base“-Integers gesetzt. Danach wird ein neues „Path“-Objekt, mit dem „base“-Wert, mit dem Namen „subCycle“ erstellt. Nachdem dies geschehen ist, begibt sich die Methode in eine doWhile-Schleife, die solange läuft, bis der „base“-Integer gleich dem „currentVertex“-Integer ist. Innerhalb dieser Schleife wird über die Liste aller Kanten, die mit dem „currentVertex“ verbunden sind iteriert. Für die momentan ausgewählte Kante wird dann geprüft, ob sie Teil des „coveredEdges“-Set ist. Ist dem nicht so, dann wird die Kante dem „subCycle“-Pfad und dem „coveredEdges“-Set hinzugefügt. Dabei wird der „currentVertex“-Integer auf den Wert, des Knotens gesetzt, mit dem die ausgewählte Kante, den ausgewählten Knoten verbindet. Sobald die doWhile-Schleife endet, wird der „subCycle“-Pfad, in den „eulerTour“-Pfad eingefügt. Dazu nutzt die Methode die „insertCycle“-Methode (Codeausschnitt 11), des „eulerTour“-Objekts.

Beispiele

Muellabfuhr 0

Tag 1: 0 -> 2 -> 3 -> 4 -> 0 , Gesamtlänge: 4

Tag 2: 0 -> 4 -> 5 -> 6 -> 0 , Gesamtlänge: 4

Tag 3: 0 -> 2 -> 1 -> 8 -> 0 , Gesamtlänge: 4

Tag 4: 0 -> 8 -> 7 -> 6 -> 0 , Gesamtlänge: 4

Tag 5: 0 -> 8 -> 9 -> 8 -> 0 , Gesamtlänge: 4

Maximale Länge einer Tagestour: 4

Muellabfuhr 1

Tag 1: 0 -> 4 -> 7 -> 6 -> 0 , Gesamtlänge: 16

Tag 2: 0 -> 5 -> 3 -> 5 -> 7 -> 6 -> 3 -> 1 -> 6 -> 0 , Gesamtlänge: 27

Tag 3: 0 -> 4 -> 3 -> 6 -> 0 , Gesamtlänge: 15

Tag 4: 0 -> 6 -> 3 -> 2 -> 3 -> 6 -> 0 , Gesamtlänge: 18

Tag 5: 0 -> 4 -> 5 -> 7 -> 6 -> 0 , Gesamtlänge: 15

Maximale Länge einer Tagestour: 27

Muellabfuhr 2

Tag 1: 0 -> 6 -> 1 -> 13 -> 9 -> 13 -> 14 -> 5 -> 11 -> 7 -> 9 -> 0 -> 5 -> 14 -> 6 -> 0 ,
Gesamtlänge: 15

Tag 2: 0 -> 6 -> 4 -> 10 -> 2 -> 11 -> 3 -> 13 -> 9 -> 5 -> 0 -> 5 -> 11 -> 2 -> 10 -> 9 -> 0 ,
Gesamtlänge: 16

Tag 3: 0 -> 9 -> 7 -> 9 -> 12 -> 8 -> 14 -> 7 -> 8 -> 2 -> 14 -> 5 -> 11 -> 2 -> 14 -> 5 -> 0 ,
Gesamtlänge: 16

Tag 4: 0 -> 9 -> 13 -> 4 -> 3 -> 13 -> 4 -> 6 -> 0 -> 6 -> 9 -> 10 -> 14 -> 5 -> 0 ,
Gesamtlänge: 14

Tag 5: 0 -> 6 -> 1 -> 7 -> 9 -> 12 -> 1 -> 12 -> 8 -> 11 -> 5 -> 0 -> 9 -> 0 , Gesamtlänge: 13

Maximale Länge einer Tagestour: 16

Muellabfuhr 3

Tag 1: 0 -> 8 -> 11 -> 12 -> 5 -> 13 -> 2 -> 13 -> 0 -> 9 -> 4 -> 14 -> 7 -> 0 -> 14 -> 3 -> 0 -> 2 -> 5 -> 6 -> 0 -> 5 -> 11 -> 0 -> 4 -> 10 -> 6 -> 8 -> 3 -> 10 -> 0 -> 8 -> 7 -> 5 -> 12 -> 0 , Gesamtlänge: 35

Tag 2: 0 -> 8 -> 4 -> 8 -> 12 -> 9 -> 13 -> 11 -> 0 -> 3 -> 11 -> 10 -> 12 -> 3 -> 6 -> 0 -> 14 -> 5 -> 14 -> 2 -> 0 -> 5 -> 4 -> 0 -> 10 -> 1 -> 0 -> 7 -> 12 -> 10 -> 9 -> 3 -> 6 -> 12 -> 0 -> 9 -> 13 -> 0 , Gesamtlänge: 37

Tag 3: 0 -> 8 -> 10 -> 0 -> 12 -> 13 -> 0 -> 14 -> 10 -> 0 -> 11 -> 4 -> 12 -> 2 -> 4 -> 6 -> 14 -> 11 -> 4 -> 0 -> 2 -> 7 -> 3 -> 4 -> 13 -> 1 -> 0 -> 6 -> 4 -> 3 -> 0 -> 7 -> 13 -> 10 -> 5 -> 0 , Gesamtlänge: 35

Tag 4: 0 -> 8 -> 13 -> 3 -> 1 -> 0 -> 12 -> 1 -> 7 -> 6 -> 11 -> 9 -> 11 -> 0 -> 9 -> 2 -> 10 -> 0 -> 14 -> 9 -> 5 -> 0 -> 2 -> 1 -> 8 -> 14 -> 12 -> 1 -> 6 -> 0 -> 4 -> 1 -> 7 -> 0 -> 3 -> 13 -> 0 , Gesamtlänge: 36

Tag 5: 0 -> 8 -> 9 -> 7 -> 9 -> 6 -> 13 -> 14 -> 1 -> 11 -> 2 -> 3 -> 5 -> 0 -> 2 -> 3 -> 0 -> 4 -> 7 -> 10 -> 0 -> 11 -> 7 -> 0 -> 9 -> 1 -> 5 -> 8 -> 2 -> 6 -> 0 -> 14 -> 1 -> 0 -> 13 -> 0 ,
Gesamtlänge: 35

Maximale Länge einer Tagestour: 37

Muellabfuhr 4

Tag 1: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 0 , Gesamtlänge: 10

Maximale Länge einer Tagestour: 10

Muellabfuhr 5

Tag 1: 0 -> 13 -> 14 -> 4 -> 49 -> 27 -> 12 -> 24 -> 20 -> 25 -> 5 -> 3 -> 5 -> 1 -> 48 -> 10 -> 6 -> 47 -> 48 -> 6 -> 38 -> 14 -> 4 -> 27 -> 49 -> 31 -> 26 -> 0 -> 27 -> 38 -> 23 -> 31 -> 1 -> 5 -> 43 -> 31 -> 13 -> 21 -> 8 -> 9 -> 37 -> 27 -> 44 -> 37 -> 10 -> 31 -> 30 -> 42 -> 19 -> 37 -> 14 -> 17 -> 44 -> 9 -> 31 -> 7 -> 21 -> 12 -> 31 -> 32 -> 23 -> 48 -> 27 -> 1 -> 10 -> 3 -> 31 -> 20 -> 24 -> 10 -> 44 -> 7 -> 23 -> 30 -> 47 -> 36 -> 31 -> 19 -> 42 -> 38 -> 48 -> 43 -> 21 -> 30 -> 36 -> 27 -> 31 -> 14 -> 13 -> 47 -> 31 -> 36 -> 22 -> 31 -> 41 -> 20 -> 9 -> 12 -> 9 -> 0 -> 18 -> 20 -> 16 -> 6 -> 8 -> 41 -> 14 -> 42 -> 36 -> 10 -> 21 -> 36 -> 49 -> 30 -> 42 -> 22 -> 10 -> 38 -> 16 -> 10 -> 21 -> 1 -> 44 -> 37 -> 46 -> 23 -> 15 -> 4 -> 6 -> 31 ->

18 -> 27 -> 6 -> 30 -> 38 -> 31 -> 15 -> 43 -> 37 -> 31 -> 16 -> 27 -> 46 -> 31 -> 45 -> 26 -> 0 -> 17 -> 45 -> 23 -> 13 -> 7 -> 27 -> 32 -> 22 -> 27 -> 21 -> 41 -> 8 -> 31 -> 24 -> 43 -> 0 , Gesamtlänge: 921

Tag 2: 0 -> 30 -> 47 -> 11 -> 39 -> 20 -> 5 -> 26 -> 19 -> 35 -> 14 -> 13 -> 21 -> 40 -> 22 -> 39 -> 49 -> 2 -> 3 -> 22 -> 32 -> 13 -> 25 -> 38 -> 11 -> 29 -> 30 -> 11 -> 41 -> 5 -> 48 -> 29 -> 49 -> 0 -> 39 -> 4 -> 35 -> 36 -> 5 -> 3 -> 36 -> 28 -> 48 -> 35 -> 10 -> 34 -> 6 -> 41 -> 42 -> 32 -> 40 -> 4 -> 47 -> 17 -> 1 -> 46 -> 26 -> 7 -> 35 -> 37 -> 11 -> 36 -> 27 -> 5 -> 39 -> 9 -> 43 -> 22 -> 17 -> 13 -> 34 -> 49 -> 27 -> 11 -> 7 -> 9 -> 38 -> 16 -> 40 -> 25 -> 47 -> 5 -> 38 -> 43 -> 15 -> 28 -> 27 -> 39 -> 2 -> 0 -> 23 -> 5 -> 4 -> 30 -> 1 -> 3 -> 47 -> 40 -> 41 -> 43 -> 35 -> 22 -> 12 -> 17 -> 25 -> 42 -> 14 -> 4 -> 26 -> 35 -> 44 -> 2 -> 12 -> 39 -> 42 -> 29 -> 45 -> 0 -> 20 -> 2 -> 16 -> 7 -> 5 -> 25 -> 6 -> 43 -> 19 -> 4 -> 42 -> 46 -> 25 -> 28 -> 23 -> 2 -> 21 -> 35 -> 0 -> 33 -> 43 -> 12 -> 28 -> 43 -> 15 -> 0 -> 34 -> 29 -> 0 -> 13 -> 47 -> 41 -> 30 -> 9 -> 16 -> 46 -> 29 -> 1 -> 19 -> 25 -> 36 -> 13 -> 3 -> 5 -> 42 -> 43 -> 3 -> 0 -> 41 -> 2 -> 36 -> 12 -> 42 -> 19 -> 2 -> 24 -> 34 -> 30 -> 43 -> 37 -> 0 -> 18 -> 19 -> 9 -> 36 -> 1 -> 25 -> 26 -> 42 -> 3 -> 19 -> 29 -> 10 -> 0 -> 11 -> 24 -> 5 -> 17 -> 32 -> 47 -> 9 -> 11 -> 32 -> 22 -> 9 -> 13 -> 16 -> 39 -> 47 -> 29 -> 32 -> 6 -> 13 -> 30 -> 18 -> 47 -> 34 -> 36 -> 4 -> 28 -> 2 -> 18 -> 7 -> 17 -> 2 -> 37 -> 5 -> 35 -> 25 -> 32 -> 1 -> 39 -> 18 -> 41 -> 35 -> 3 -> 26 -> 34 -> 32 -> 9 -> 0 -> 28 -> 17 -> 26 -> 39 -> 34 -> 40 -> 30 -> 47 -> 12 -> 7 -> 42 -> 2 -> 6 -> 12 -> 1 -> 13 -> 5 -> 40 -> 28 -> 21 -> 10 -> 2 -> 43 -> 32 -> 46 -> 47 -> 43 -> 39 -> 40 -> 27 -> 29 -> 7 -> 36 -> 27 -> 35 -> 34 -> 16 -> 39 -> 11 -> 49 -> 27 -> 34 -> 33 -> 2 -> 45 -> 34 -> 9 -> 42 -> 18 -> 16 -> 26 -> 38 -> 17 -> 30 -> 25 -> 9 -> 3 -> 34 -> 4 -> 38 -> 2 -> 8 -> 9 -> 17 -> 40 -> 7 -> 22 -> 41 -> 14 -> 38 -> 35 -> 17 -> 29 -> 23 -> 11 -> 1 -> 16 -> 25 -> 18 -> 29 -> 8 -> 28 -> 19 -> 40 -> 49 -> 5 -> 1 -> 41 -> 16 -> 0 -> 7 -> 38 -> 29 -> 35 -> 33 -> 5 -> 45 -> 35 -> 46 -> 7 -> 30 -> 46 -> 13 -> 22 -> 5 -> 14 -> 11 -> 10 -> 28 -> 30 -> 26 -> 42 -> 11 -> 13 -> 41 -> 26 -> 43 -> 5 -> 43 -> 0 -> 46 -> 41 -> 25 -> 4 -> 32 -> 19 -> 17 -> 18 -> 20 -> 40 -> 15 -> 11 -> 4 -> 1 -> 35 -> 13 -> 18 -> 38 -> 39 -> 28 -> 6 -> 17 -> 3 -> 7 -> 35 -> 32 -> 0 -> 5 -> 12 -> 40 -> 18 -> 35 -> 40 -> 21 -> 39 -> 3 -> 28 -> 1 -> 7 -> 34 -> 28 -> 32 -> 26 -> 28 -> 18 -> 32 -> 23 -> 45 -> 11 -> 25 -> 7 -> 13 -> 23 -> 39 -> 13 -> 2 -> 22 -> 18 -> 26 -> 9 -> 2 -> 27 -> 0 -> 44 -> 11 -> 20 -> 29 -> 24 -> 28 -> 46 -> 22 -> 29 -> 40 -> 3 -> 4 -> 12 -> 35 -> 6 -> 29 -> 8 -> 5 -> 18 -> 43 -> 4 -> 6 -> 26 -> 2 -> 32 -> 41 -> 39 -> 34 -> 46 -> 17 -> 0 -> 26 -> 12 -> 32 -> 5 -> 19 -> 46 -> 43 -> 48 -> 40 -> 45 -> 28 -> 49 -> 35 -> 24 -> 43 -> 7 -> 2 -> 25 -> 3 -> 18 -> 46 -> 9 -> 1 -> 38 -> 0 -> 24 -> 39 -> 17 -> 41 -> 34 -> 37 -> 46 -> 4 -> 2 -> 5 -> 29 -> 37 -> 44 -> 39 -> 10 -> 5 -> 25 -> 22 -> 11 -> 26 -> 22 -> 4 -> 6 -> 0 -> 25 -> 34 -> 42 -> 13 -> 43 -> 1 -> 42 -> 40 -> 14 -> 29 -> 15 -> 35 -> 8 -> 34 -> 44 -> 29 -> 43 -> 16 -> 17 -> 34 -> 38 -> 12 -> 9 -> 6 -> 46 -> 11 -> 48 -> 0 -> 8 -> 40 -> 0 -> 36 -> 18 -> 12 -> 3 -> 38 -> 40 -> 26 -> 36 -> 29 -> 41 -> 12 -> 46 -> 39 -> 15 -> 34 -> 11 -> 33 -> 39 -> 19 -> 12 -> 30 -> 32 -> 7 -> 6 -> 5 -> 28 -> 7 -> 19 -> 41 -> 9 -> 4 -> 16 -> 29 -> 2 -> 1 -> 34 -> 20 -> 35 -> 23 -> 40 -> 33 -> 29 -> 28 -> 12 -> 13 -> 19 -> 0 , Gesamtlänge: 4007

Tag 3: 0 -> 13 -> 14 -> 4 -> 23 -> 45 -> 30 -> 14 -> 38 -> 16 -> 38 -> 31 -> 40 -> 21 -> 13 -> 15 -> 18 -> 44 -> 37 -> 49 -> 1 -> 33 -> 37 -> 47 -> 10 -> 17 -> 24 -> 43 -> 20 -> 48 -> 46 -> 37 -> 3 -> 5 -> 1 -> 24 -> 33 -> 17 -> 0 -> 43 -> 33 -> 32 -> 22 -> 15 -> 3 -> 48 -> 43 -> 45 ->

> 38 -> 20 -> 49 -> 8 -> 18 -> 49 -> 14 -> 41 -> 27 -> 19 -> 42 -> 27 -> 49 -> 47 -> 20 -> 19
> 33 -> 4 -> 8 -> 26 -> 14 -> 15 -> 43 -> 37 -> 44 -> 46 -> 14 -> 10 -> 45 -> 37 -> 20 -> 14 -
> 3 -> 24 -> 23 -> 13 -> 27 -> 30 -> 10 -> 23 -> 41 -> 14 -> 1 -> 23 -> 32 -> 23 -> 26 -> 49 -
> 46 -> 10 -> 21 -> 14 -> 42 -> 45 -> 27 -> 24 -> 43 -> 5 -> 43 -> 48 -> 33 -> 22 -> 37 -> 17
> 21 -> 25 -> 5 -> 25 -> 24 -> 44 -> 26 -> 48 -> 15 -> 36 -> 14 -> 38 -> 8 -> 9 -> 27 -> 10 ->
21 -> 3 -> 21 -> 48 -> 32 -> 15 -> 16 -> 49 -> 27 -> 36 -> 45 -> 14 -> 23 -> 25 -> 37 -> 26 ->
0 -> 27 -> 23 -> 47 -> 33 -> 30 -> 47 -> 13 -> 20 -> 18 -> 0 -> 9 -> 12 -> 39 -> 31 -> 29 -> 0
, Gesamtlänge: 916

Tag 4: 0 -> 13 -> 14 -> 32 -> 8 -> 23 -> 44 -> 32 -> 22 -> 45 -> 25 -> 33 -> 45 -> 23 -> 13 ->
7 -> 49 -> 3 -> 5 -> 25 -> 44 -> 22 -> 23 -> 9 -> 14 -> 4 -> 6 -> 33 -> 43 -> 49 -> 38 -> 16 ->
23 -> 32 -> 23 -> 37 -> 34 -> 31 -> 38 -> 14 -> 24 -> 6 -> 37 -> 12 -> 23 -> 3 -> 44 -> 37 ->
46 -> 37 -> 16 -> 47 -> 15 -> 49 -> 25 -> 8 -> 42 -> 49 -> 32 -> 10 -> 18 -> 24 -> 43 -> 5 ->
43 -> 24 -> 16 -> 8 -> 33 -> 13 -> 47 -> 24 -> 13 -> 21 -> 10 -> 13 -> 48 -> 7 -> 28 -> 31 ->
11 -> 24 -> 37 -> 43 -> 15 -> 42 -> 19 -> 49 -> 48 -> 43 -> 23 -> 45 -> 49 -> 6 -> 45 -> 12 ->
45 -> 41 -> 24 -> 48 -> 9 -> 12 -> 15 -> 46 -> 8 -> 12 -> 14 -> 42 -> 33 -> 41 -> 14 -> 18 ->
0 -> 27 -> 49 -> 41 -> 44 -> 33 -> 38 -> 37 -> 42 -> 14 -> 8 -> 36 -> 27 -> 0 -> 9 -> 15 -> 37
> 41 -> 14 -> 43 -> 37 -> 13 -> 47 -> 30 -> 47 -> 19 -> 16 -> 22 -> 49 -> 10 -> 12 -> 48 ->
18 -> 24 -> 9 -> 8 -> 42 -> 19 -> 45 -> 8 -> 47 -> 45 -> 6 -> 15 -> 30 -> 37 -> 7 -> 45 -> 44 ->
43 -> 0 , Gesamtlänge: 888

Tag 5: 0 -> 13 -> 14 -> 48 -> 45 -> 21 -> 13 -> 7 -> 10 -> 8 -> 24 -> 43 -> 35 -> 31 -> 38 ->
16 -> 14 -> 47 -> 42 -> 19 -> 10 -> 20 -> 30 -> 24 -> 15 -> 41 -> 14 -> 4 -> 37 -> 46 -> 45 ->
23 -> 6 -> 44 -> 38 -> 19 -> 23 -> 13 -> 21 -> 44 -> 37 -> 44 -> 48 -> 43 -> 0 -> 17 -> 27 ->
25 -> 5 -> 25 -> 10 -> 4 -> 6 -> 36 -> 27 -> 15 -> 43 -> 37 -> 21 -> 19 -> 14 -> 38 -> 14 -> 6
> 19 -> 36 -> 48 -> 19 -> 15 -> 20 -> 8 -> 9 -> 33 -> 31 -> 2 -> 37 -> 43 -> 10 -> 21 -> 15 ->
7 -> 20 -> 22 -> 14 -> 42 -> 14 -> 38 -> 44 -> 30 -> 19 -> 22 -> 30 -> 47 -> 13 -> 49 -> 27 ->
0 -> 26 -> 20 -> 4 -> 24 -> 26 -> 27 -> 3 -> 20 -> 1 -> 45 -> 32 -> 24 -> 49 -> 21 -> 10 -> 9 ->
12 -> 9 -> 0 -> 18 -> 21 -> 33 -> 18 -> 20 -> 17 -> 23 -> 20 -> 33 -> 49 -> 12 -> 33 -> 43 ->
5 -> 31 -> 38 -> 22 -> 30 -> 8 -> 7 -> 13 -> 47 -> 21 -> 42 -> 23 -> 32 -> 37 -> 46 -> 20 ->
17 -> 48 -> 4 -> 44 -> 42 -> 48 -> 8 -> 1 -> 5 -> 43 -> 33 -> 3 -> 5 -> 3 -> 8 -> 44 -> 36 -> 16
> 21 -> 26 -> 20 -> 32 -> 22 -> 21 -> 9 -> 0 , Gesamtlänge: 895

Maximale Länge einer Tagestour: 4007

Muellabfuhr 6

Tag 1: 0 -> 44 -> 60 -> 61 -> 24 -> 59 -> 60 -> 33 -> 60 -> 59 -> 62 -> 54 -> 17 -> 56 -> 26 ->
57 -> 29 -> 53 -> 95 -> 53 -> 57 -> 29 -> 26 -> 9 -> 56 -> 47 -> 9 -> 22 -> 24 -> 82 -> 28 ->
30 -> 33 -> 83 -> 28 -> 31 -> 70 -> 28 -> 70 -> 82 -> 31 -> 24 -> 22 -> 54 -> 9 -> 17 -> 22
> 47 -> 26 -> 6 -> 62 -> 59 -> 44 -> 4 -> 0 , Gesamtlänge: 492107

Tag 2: 0 -> 93 -> 45 -> 42 -> 5 -> 77 -> 18 -> 23 -> 78 -> 69 -> 38 -> 23 -> 69 -> 75 -> 23 ->
97 -> 55 -> 75 -> 69 -> 97 -> 38 -> 75 -> 78 -> 55 -> 52 -> 73 -> 88 -> 35 -> 98 -> 58 -> 0 ,
Gesamtlänge: 343178

Tag 3: 0 -> 58 -> 98 -> 35 -> 88 -> 73 -> 52 -> 55 -> 36 -> 34 -> 8 -> 20 -> 52 -> 55 -> 78 -> 37 -> 68 -> 14 -> 36 -> 11 -> 37 -> 68 -> 3 -> 1 -> 3 -> 49 -> 21 -> 49 -> 1 -> 11 -> 3 -> 14 -> 68 -> 91 -> 68 -> 49 -> 91 -> 50 -> 15 -> 12 -> 21 -> 50 -> 12 -> 85 -> 15 -> 94 -> 27 -> 99 -> 89 -> 40 -> 99 -> 43 -> 66 -> 40 -> 99 -> 27 -> 43 -> 99 -> 96 -> 89 -> 99 -> 27 -> 87 -> 27 -> 85 -> 50 -> 7 -> 12 -> 15 -> 7 -> 12 -> 85 -> 87 -> 8 -> 34 -> 20 -> 73 -> 88 -> 35 -> 98 -> 58 -> 0 , Gesamtlänge: 673300

Tag 4: 0 -> 44 -> 60 -> 33 -> 83 -> 30 -> 33 -> 60 -> 44 -> 0 -> 93 -> 98 -> 4 -> 93 -> 45 -> 41 -> 42 -> 86 -> 72 -> 13 -> 77 -> 67 -> 5 -> 2 -> 84 -> 19 -> 2 -> 13 -> 84 -> 77 -> 86 -> 41 -> 5 -> 86 -> 77 -> 5 -> 13 -> 2 -> 72 -> 86 -> 5 -> 42 -> 45 -> 10 -> 98 -> 35 -> 32 -> 90 -> 41 -> 67 -> 42 -> 45 -> 98 -> 35 -> 46 -> 32 -> 88 -> 81 -> 32 -> 98 -> 58 -> 0 -> 58 -> 45 -> 98 -> 58 -> 93 -> 35 -> 88 -> 90 -> 32 -> 46 -> 71 -> 0 -> 4 -> 10 -> 48 -> 19 -> 39 -> 83 -> 25 -> 30 -> 39 -> 25 -> 48 -> 25 -> 39 -> 60 -> 44 -> 0 , Gesamtlänge: 889397

Tag 5: 0 -> 58 -> 98 -> 32 -> 81 -> 51 -> 64 -> 80 -> 76 -> 65 -> 80 -> 63 -> 76 -> 81 -> 76 -> 63 -> 64 -> 51 -> 65 -> 81 -> 32 -> 98 -> 35 -> 88 -> 73 -> 20 -> 8 -> 87 -> 27 -> 99 -> 89 -> 16 -> 80 -> 53 -> 29 -> 26 -> 57 -> 79 -> 92 -> 74 -> 16 -> 92 -> 57 -> 26 -> 6 -> 62 -> 59 -> 44 -> 0 , Gesamtlänge: 558741

Maximale Lange einer Tagestour: 889397

Muellabfuhr 7

Tag 1: 0 -> 1 -> 2 -> 217 -> 68 -> 256 -> 14 -> 324 -> 386 -> 96 -> 261 -> 64 -> 330 -> 86 -> 261 -> 330 -> 96 -> 429 -> 96 -> 86 -> 261 -> 386 -> 421 -> 2 -> 126 -> 174 -> 372 -> 297 -> 64 -> 386 -> 297 -> 96 -> 324 -> 386 -> 372 -> 324 -> 174 -> 297 -> 330 -> 334 -> 386 -> 256 -> 243 -> 68 -> 218 -> 243 -> 211 -> 2 -> 218 -> 256 -> 217 -> 126 -> 68 -> 211 -> 256 -> 218 -> 386 -> 243 -> 174 -> 297 -> 261 -> 334 -> 324 -> 64 -> 86 -> 14 -> 324 -> 96 -> 330 -> 324 -> 261 -> 334 -> 64 -> 297 -> 86 -> 386 -> 126 -> 211 -> 68 -> 421 -> 211 -> 217 -> 421 -> 243 -> 372 -> 126 -> 421 -> 372 -> 2 -> 19 -> 272 -> 485 -> 491 -> 105 -> 413 -> 383 -> 481 -> 379 -> 124 -> 220 -> 19 -> 27 -> 279 -> 90 -> 497 -> 359 -> 237 -> 90 -> 359 -> 433 -> 346 -> 460 -> 438 -> 65 -> 104 -> 149 -> 460 -> 318 -> 184 -> 118 -> 204 -> 240 -> 79 -> 207 -> 43 -> 113 -> 118 -> 198 -> 184 -> 65 -> 87 -> 438 -> 346 -> 359 -> 279 -> 335 -> 465 -> 346 -> 465 -> 460 -> 87 -> 165 -> 335 -> 346 -> 165 -> 433 -> 198 -> 207 -> 204 -> 43 -> 118 -> 79 -> 237 -> 207 -> 184 -> 149 -> 65 -> 318 -> 149 -> 445 -> 184 -> 445 -> 104 -> 438 -> 318 -> 198 -> 149 -> 318 -> 104 -> 445 -> 87 -> 433 -> 165 -> 198 -> 87 -> 465 -> 433 -> 90 -> 79 -> 279 -> 497 -> 237 -> 279 -> 335 -> 90 -> 27 -> 204 -> 79 -> 207 -> 240 -> 113 -> 207 -> 118 -> 240 -> 27 -> 19 -> 485 -> 122 -> 383 -> 272 -> 252 -> 124 -> 485 -> 379 -> 485 -> 252 -> 105 -> 124 -> 272 -> 481 -> 19 -> 124 -> 122 -> 252 -> 481 -> 220 -> 272 -> 413 -> 383 -> 491 -> 379 -> 252 -> 220 -> 481 -> 491 -> 383 -> 19 -> 2 -> 1 -> 0 , Gesamtlänge: 328459

Tag 2: 0 -> 317 -> 52 -> 127 -> 354 -> 167 -> 52 -> 163 -> 127 -> 0 -> 1 -> 250 -> 316 -> 177 -> 189 -> 50 -> 177 -> 50 -> 35 -> 280 -> 477 -> 348 -> 374 -> 134 -> 189 -> 50 -> 225 -> 50 -> 250 -> 316 -> 225 -> 1 -> 25 -> 327 -> 219 -> 342 -> 313 -> 311 -> 405 -> 350 -> 397 -> 405 -> 29 -> 313 -> 25 -> 419 -> 219 -> 67 -> 313 -> 200 -> 89 -> 245 -> 435 -> 366 -> 283 -> 245 -> 327 -> 342 -> 28 -> 327 -> 435 -> 154 -> 141 -> 142 -> 57 -> 132 -> 311 ->

340 -> 57 -> 164 -> 340 -> 350 -> 57 -> 132 -> 142 -> 397 -> 164 -> 154 -> 142 -> 164 -> 350 -> 142 -> 340 -> 154 -> 350 -> 132 -> 313 -> 405 -> 132 -> 397 -> 350 -> 141 -> 154 -> 57 -> 397 -> 311 -> 132 -> 313 -> 25 -> 89 -> 342 -> 67 -> 327 -> 283 -> 366 -> 200 -> 25 -> 219 -> 293 -> 219 -> 25 -> 342 -> 419 -> 200 -> 28 -> 89 -> 327 -> 419 -> 28 -> 283 -> 141 -> 435 -> 283 -> 200 -> 245 -> 25 -> 419 -> 245 -> 327 -> 200 -> 25 -> 1 -> 10 -> 360 -> 269 -> 449 -> 468 -> 254 -> 423 -> 277 -> 254 -> 290 -> 468 -> 138 -> 282 -> 449 -> 277 -> 290 -> 360 -> 100 -> 449 -> 290 -> 138 -> 468 -> 10 -> 449 -> 62 -> 423 -> 282 -> 277 -> 10 -> 377 -> 360 -> 112 -> 343 -> 269 -> 447 -> 331 -> 195 -> 447 -> 58 -> 343 -> 331 -> 269 -> 227 -> 62 -> 227 -> 269 -> 112 -> 447 -> 343 -> 195 -> 10 -> 138 -> 423 -> 468 -> 390 -> 343 -> 195 -> 269 -> 58 -> 195 -> 449 -> 254 -> 282 -> 10 -> 447 -> 227 -> 331 -> 112 -> 449 -> 58 -> 331 -> 449 -> 377 -> 100 -> 290 -> 62 -> 360 -> 227 -> 112 -> 227 -> 377 -> 62 -> 10 -> 1 -> 215 -> 4 -> 7 -> 188 -> 213 -> 315 -> 498 -> 147 -> 353 -> 301 -> 498 -> 97 -> 363 -> 323 -> 234 -> 294 -> 235 -> 471 -> 242 -> 42 -> 242 -> 210 -> 294 -> 108 -> 474 -> 213 -> 284 -> 188 -> 474 -> 7 -> 9 -> 135 -> 329 -> 22 -> 205 -> 94 -> 310 -> 191 -> 246 -> 482 -> 120 -> 246 -> 95 -> 246 -> 175 -> 228 -> 130 -> 416 -> 344 -> 40 -> 411 -> 287 -> 22 -> 456 -> 144 -> 95 -> 329 -> 319 -> 22 -> 21 -> 329 -> 94 -> 9 -> 482 -> 94 -> 191 -> 175 -> 310 -> 130 -> 382 -> 228 -> 411 -> 130 -> 232 -> 228 -> 40 -> 130 -> 228 -> 175 -> 246 -> 94 -> 144 -> 22 -> 9 -> 95 -> 94 -> 175 -> 232 -> 120 -> 191 -> 95 -> 482 -> 246 -> 310 -> 482 -> 135 -> 329 -> 205 -> 9 -> 21 -> 456 -> 21 -> 382 -> 287 -> 319 -> 21 -> 205 -> 319 -> 144 -> 329 -> 144 -> 9 -> 329 -> 456 -> 287 -> 21 -> 40 -> 416 -> 411 -> 344 -> 310 -> 232 -> 344 -> 382 -> 416 -> 287 -> 9 -> 7 -> 242 -> 294 -> 7 -> 235 -> 242 -> 108 -> 42 -> 235 -> 474 -> 289 -> 363 -> 147 -> 363 -> 309 -> 289 -> 301 -> 284 -> 301 -> 498 -> 289 -> 147 -> 97 -> 234 -> 210 -> 42 -> 294 -> 474 -> 7 -> 108 -> 289 -> 284 -> 353 -> 147 -> 378 -> 309 -> 323 -> 234 -> 378 -> 147 -> 309 -> 323 -> 97 -> 309 -> 498 -> 353 -> 289 -> 315 -> 353 -> 213 -> 301 -> 315 -> 284 -> 474 -> 42 -> 188 -> 294 -> 471 -> 108 -> 235 -> 7 -> 4 -> 427 -> 215 -> 260 -> 151 -> 193 -> 477 -> 189 -> 280 -> 348 -> 151 -> 375 -> 348 -> 134 -> 189 -> 63 -> 193 -> 374 -> 35 -> 407 -> 280 -> 374 -> 63 -> 375 -> 260 -> 427 -> 225 -> 304 -> 36 -> 170 -> 260 -> 4 -> 36 -> 215 -> 304 -> 1 -> 4 -> 225 -> 1 -> 4 -> 225 -> 260 -> 36 -> 260 -> 1 -> 3 -> 183 -> 450 -> 47 -> 466 -> 255 -> 258 -> 16 -> 312 -> 466 -> 255 -> 116 -> 356 -> 214 -> 183 -> 371 -> 255 -> 197 -> 305 -> 46 -> 312 -> 255 -> 183 -> 128 -> 3 -> 128 -> 466 -> 197 -> 459 -> 20 -> 46 -> 459 -> 214 -> 371 -> 183 -> 258 -> 312 -> 46 -> 20 -> 255 -> 3 -> 47 -> 368 -> 336 -> 258 -> 197 -> 312 -> 459 -> 16 -> 305 -> 459 -> 258 -> 20 -> 336 -> 47 -> 255 -> 173 -> 356 -> 450 -> 128 -> 116 -> 214 -> 173 -> 128 -> 450 -> 255 -> 356 -> 173 -> 450 -> 368 -> 450 -> 116 -> 371 -> 356 -> 3 -> 1 -> 304 -> 170 -> 375 -> 134 -> 477 -> 151 -> 170 -> 375 -> 477 -> 63 -> 151 -> 63 -> 348 -> 193 -> 134 -> 35 -> 189 -> 407 -> 316 -> 35 -> 250 -> 304 -> 427 -> 1 -> 0 -> 194 -> 12 -> 354 -> 66 -> 409 -> 484 -> 66 -> 167 -> 484 -> 352 -> 409 -> 194 -> 249 -> 72 -> 467 -> 464 -> 467 -> 66 -> 352 -> 167 -> 5 -> 317 -> 12 -> 0 -> 30 -> 209 -> 247 -> 431 -> 394 -> 136 -> 172 -> 103 -> 41 -> 244 -> 308 -> 41 -> 395 -> 321 -> 54 -> 41 -> 308 -> 140 -> 395 -> 244 -> 410 -> 347 -> 420 -> 224 -> 395 -> 172 -> 431 -> 209 -> 394 -> 489 -> 493 -> 321 -> 172 -> 103 -> 140 -> 244 -> 420 -> 436 -> 431 -> 403 -> 136 -> 247 -> 30 -> 436 -> 410 -> 347 -> 172 -> 80 -> 431 -> 209 -> 80 -> 402 -> 403 -> 489 -> 136 -> 80 -> 436 -> 172 -> 431 -> 402 -> 209 -> 30 -> 402 -> 436 -> 172 -> 209 -> 136 -> 209 -> 493 -> 54 -> 395 -> 420 ->

308 -> 103 -> 321 -> 395 -> 308 -> 54 -> 103 -> 224 -> 140 -> 420 -> 410 -> 80 -> 244 -> 30 -> 247 -> 403 -> 30 -> 0 -> 409 -> 317 -> 358 -> 495 -> 72 -> 169 -> 317 -> 0 -> 401 -> 72 -> 409 -> 495 -> 249 -> 358 -> 495 -> 169 -> 72 -> 194 -> 358 -> 169 -> 409 -> 467 -> 484 -> 354 -> 163 -> 354 -> 464 -> 163 -> 5 -> 52 -> 12 -> 127 -> 5 -> 66 -> 464 -> 484 -> 167 -> 464 -> 409 -> 249 -> 72 -> 358 -> 401 -> 169 -> 194 -> 401 -> 495 -> 0 , Gesamtlänge:
678064

Tag 3: 0 -> 1 -> 4 -> 17 -> 38 -> 26 -> 437 -> 160 -> 78 -> 299 -> 369 -> 385 -> 307 -> 295 -> 158 -> 437 -> 137 -> 396 -> 437 -> 398 -> 307 -> 137 -> 398 -> 408 -> 160 -> 307 -> 158 -> 398 -> 396 -> 408 -> 298 -> 143 -> 196 -> 55 -> 298 -> 55 -> 119 -> 143 -> 434 -> 119 -> 196 -> 295 -> 434 -> 17 -> 33 -> 34 -> 44 -> 345 -> 133 -> 182 -> 404 -> 473 -> 422 -> 253 -> 422 -> 93 -> 490 -> 473 -> 39 -> 345 -> 380 -> 426 -> 181 -> 380 -> 133 -> 181 -> 292 -> 349 -> 222 -> 230 -> 473 -> 114 -> 221 -> 424 -> 444 -> 292 -> 182 -> 349 -> 230 -> 291 -> 93 -> 114 -> 221 -> 483 -> 444 -> 349 -> 424 -> 291 -> 426 -> 444 -> 380 -> 182 -> 44 -> 473 -> 114 -> 291 -> 483 -> 114 -> 230 -> 222 -> 291 -> 181 -> 444 -> 426 -> 292 -> 345 -> 34 -> 39 -> 44 -> 270 -> 404 -> 39 -> 34 -> 380 -> 34 -> 270 -> 430 -> 39 -> 186 -> 39 -> 93 -> 473 -> 221 -> 422 -> 186 -> 93 -> 44 -> 490 -> 270 -> 39 -> 253 -> 186 -> 490 -> 39 -> 404 -> 93 -> 221 -> 222 -> 483 -> 349 -> 181 -> 182 -> 345 -> 270 -> 93 -> 253 -> 490 -> 430 -> 34 -> 33 -> 376 -> 71 -> 451 -> 303 -> 49 -> 33 -> 451 -> 376 -> 82 -> 357 -> 268 -> 384 -> 399 -> 322 -> 399 -> 223 -> 446 -> 376 -> 357 -> 370 -> 376 -> 121 -> 303 -> 223 -> 446 -> 303 -> 33 -> 370 -> 325 -> 139 -> 231 -> 82 -> 451 -> 361 -> 451 -> 121 -> 370 -> 389 -> 357 -> 139 -> 376 -> 325 -> 82 -> 268 -> 399 -> 231 -> 384 -> 322 -> 49 -> 399 -> 82 -> 326 -> 322 -> 139 -> 268 -> 82 -> 33 -> 121 -> 33 -> 71 -> 303 -> 376 -> 268 -> 322 -> 82 -> 384 -> 326 -> 325 -> 322 -> 231 -> 325 -> 357 -> 384 -> 139 -> 389 -> 370 -> 361 -> 303 -> 71 -> 446 -> 33 -> 17 -> 299 -> 24 -> 385 -> 299 -> 26 -> 408 -> 385 -> 196 -> 434 -> 38 -> 369 -> 24 -> 26 -> 78 -> 437 -> 408 -> 158 -> 137 -> 295 -> 385 -> 78 -> 437 -> 396 -> 160 -> 158 -> 55 -> 385 -> 26 -> 38 -> 24 -> 434 -> 369 -> 119 -> 143 -> 369 -> 17 -> 4 -> 1 -> 0 , Gesamtlänge: 383376

Tag 4: 0 -> 1 -> 3 -> 6 -> 11 -> 457 -> 475 -> 60 -> 101 -> 373 -> 488 -> 60 -> 76 -> 92 -> 262 -> 156 -> 83 -> 278 -> 328 -> 208 -> 91 -> 59 -> 208 -> 31 -> 76 -> 208 -> 455 -> 488 -> 475 -> 92 -> 110 -> 288 -> 216 -> 476 -> 425 -> 32 -> 216 -> 425 -> 51 -> 267 -> 69 -> 476 -> 69 -> 32 -> 61 -> 267 -> 425 -> 156 -> 475 -> 262 -> 76 -> 475 -> 156 -> 60 -> 92 -> 457 -> 11 -> 488 -> 457 -> 60 -> 11 -> 13 -> 123 -> 85 -> 364 -> 472 -> 152 -> 339 -> 152 -> 492 -> 320 -> 145 -> 152 -> 201 -> 148 -> 123 -> 99 -> 226 -> 286 -> 281 -> 264 -> 199 -> 286 -> 496 -> 275 -> 461 -> 146 -> 453 -> 190 -> 146 -> 226 -> 102 -> 190 -> 275 -> 226 -> 496 -> 281 -> 453 -> 275 -> 281 -> 226 -> 13 -> 102 -> 123 -> 148 -> 499 -> 125 -> 462 -> 99 -> 13 -> 462 -> 129 -> 364 -> 339 -> 472 -> 417 -> 152 -> 320 -> 178 -> 152 -> 417 -> 364 -> 123 -> 462 -> 365 -> 15 -> 462 -> 499 -> 129 -> 125 -> 365 -> 148 -> 417 -> 492 -> 178 -> 461 -> 259 -> 199 -> 461 -> 286 -> 190 -> 226 -> 264 -> 259 -> 492 -> 339 -> 201 -> 364 -> 472 -> 85 -> 15 -> 123 -> 365 -> 102 -> 462 -> 226 -> 99 -> 472 -> 123 -> 129 -> 99 -> 102 -> 499 -> 15 -> 365 -> 499 -> 85 -> 148 -> 15 -> 129 -> 13 -> 45 -> 238 -> 45 -> 77 -> 241 -> 88 -> 479 -> 355 -> 155 -> 300 -> 241 -> 248 -> 157 -> 241 -> 428 -> 185 -> 306 -> 236 -> 98 -> 367 -> 300 -> 88 -> 77 -> 88 -> 300 -> 45 -> 179 -> 98 -> 351 -> 428 -> 362 ->

478 -> 428 -> 248 -> 362 -> 241 -> 479 -> 155 -> 479 -> 351 -> 229 -> 238 -> 179 -> 236 ->
 428 -> 157 -> 155 -> 248 -> 478 -> 351 -> 241 -> 478 -> 185 -> 98 -> 238 -> 306 -> 179 ->
 367 -> 351 -> 306 -> 98 -> 238 -> 351 -> 300 -> 367 -> 478 -> 362 -> 351 -> 236 -> 238 ->
 179 -> 229 -> 98 -> 185 -> 229 -> 236 -> 300 -> 355 -> 45 -> 13 -> 125 -> 85 -> 417 -> 129 ->
 226 -> 453 -> 461 -> 145 -> 264 -> 146 -> 281 -> 190 -> 496 -> 281 -> 146 -> 286 -> 320 ->
 145 -> 178 -> 199 -> 320 -> 461 -> 264 -> 286 -> 264 -> 13 -> 11 -> 101 -> 455 -> 76 ->
 110 -> 51 -> 216 -> 328 -> 400 -> 208 -> 457 -> 76 -> 60 -> 373 -> 208 -> 59 -> 400 -> 83 ->
 288 -> 69 -> 216 -> 23 -> 267 -> 425 -> 69 -> 32 -> 288 -> 156 -> 61 -> 23 -> 288 -> 51 ->
 328 -> 83 -> 278 -> 59 -> 31 -> 91 -> 11 -> 455 -> 101 -> 373 -> 76 -> 11 -> 6 -> 3 -> 1 -> 0
 , Gesamtlänge: 823118

Tag 5: 0 -> 1 -> 3 -> 6 -> 448 -> 251 -> 412 -> 81 -> 406 -> 131 -> 470 -> 285 -> 75 -> 8 ->
 115 -> 84 -> 162 -> 271 -> 487 -> 171 -> 162 -> 239 -> 271 -> 203 -> 271 -> 171 -> 111 ->
 487 -> 162 -> 84 -> 388 -> 274 -> 8 -> 53 -> 233 -> 168 -> 486 -> 432 -> 263 -> 161 -> 56 ->
 153 -> 265 -> 233 -> 166 -> 393 -> 266 -> 381 -> 106 -> 70 -> 212 -> 117 -> 212 -> 494 ->
 314 -> 296 -> 117 -> 70 -> 314 -> 150 -> 414 -> 486 -> 443 -> 153 -> 53 -> 153 -> 168 ->
 265 -> 443 -> 415 -> 265 -> 153 -> 166 -> 443 -> 381 -> 168 -> 494 -> 106 -> 168 -> 443 ->
 53 -> 56 -> 432 -> 56 -> 494 -> 70 -> 150 -> 393 -> 414 -> 168 -> 166 -> 494 -> 432 -> 161 ->
 296 -> 106 -> 117 -> 314 -> 266 -> 314 -> 212 -> 486 -> 381 -> 414 -> 393 -> 381 -> 494 ->
 117 -> 150 -> 266 -> 296 -> 70 -> 296 -> 212 -> 106 -> 486 -> 153 -> 233 -> 415 -> 153 ->
 263 -> 53 -> 8 -> 463 -> 84 -> 171 -> 203 -> 448 -> 187 -> 412 -> 406 -> 159 -> 338 -> 115 ->
 341 -> 176 -> 37 -> 159 -> 333 -> 274 -> 341 -> 463 -> 338 -> 37 -> 81 -> 6 -> 18 -> 441 ->
 109 -> 454 -> 332 -> 273 -> 392 -> 480 -> 180 -> 337 -> 18 -> 73 -> 337 -> 276 -> 387 ->
 439 -> 337 -> 391 -> 109 -> 458 -> 441 -> 439 -> 206 -> 180 -> 107 -> 480 -> 454 -> 273 ->
 202 -> 107 -> 74 -> 392 -> 440 -> 469 -> 192 -> 442 -> 441 -> 442 -> 452 -> 206 -> 337 ->
 276 -> 391 -> 73 -> 439 -> 442 -> 73 -> 458 -> 439 -> 18 -> 439 -> 391 -> 18 -> 441 -> 391 ->
 458 -> 441 -> 387 -> 18 -> 442 -> 107 -> 332 -> 302 -> 454 -> 107 -> 469 -> 442 -> 454 ->
 469 -> 452 -> 180 -> 440 -> 107 -> 273 -> 480 -> 332 -> 392 -> 454 -> 202 -> 302 -> 480 ->
 332 -> 202 -> 392 -> 302 -> 454 -> 442 -> 469 -> 180 -> 73 -> 387 -> 276 -> 206 -> 442 ->
 180 -> 480 -> 192 -> 452 -> 107 -> 192 -> 180 -> 276 -> 18 -> 6 -> 159 -> 333 -> 341 -> 463 ->
 115 -> 341 -> 388 -> 111 -> 162 -> 257 -> 187 -> 251 -> 271 -> 187 -> 406 -> 37 -> 285 ->
 131 -> 81 -> 406 -> 159 -> 6 -> 406 -> 285 -> 159 -> 338 -> 8 -> 48 -> 418 -> 341 -> 48 ->
 388 -> 418 -> 338 -> 8 -> 418 -> 341 -> 8 -> 333 -> 75 -> 176 -> 75 -> 37 -> 412 -> 251 -> 6 ->
 412 -> 81 -> 131 -> 6 -> 448 -> 162 -> 448 -> 257 -> 111 -> 84 -> 487 -> 257 -> 239 ->
 487 -> 171 -> 257 -> 203 -> 187 -> 6 -> 3 -> 1 -> 0 , Gesamtlänge: 725422

Maximale Länge einer Tagestour: 823118

Muellabfuhr 8

Tag 1: 0 -> 294 -> 1 -> 460 -> 451 -> 582 -> 563 -> 772 -> 440 -> 658 -> 457 -> 440 -> 713 ->
 414 -> 691 -> 772 -> 56 -> 437 -> 489 -> 658 -> 691 -> 745 -> 442 -> 460 -> 451 -> 582 ->
 440 -> 563 -> 713 -> 457 -> 745 -> 440 -> 691 -> 676 -> 787 -> 860 -> 389 -> 3 -> 483 ->
 571 -> 755 -> 483 -> 573 -> 728 -> 374 -> 571 -> 860 -> 9 -> 744 -> 396 -> 491 -> 632 ->
 772 -> 414 -> 676 -> 56 -> 24 -> 106 -> 855 -> 374 -> 830 -> 131 -> 66 -> 320 -> 447 -> 147 ->
 372 -> 905 -> 119 -> 485 -> 20 -> 583 -> 192 -> 464 -> 906 -> 886 -> 702 -> 464 -> 886 ->

> 395 -> 951 -> 119 -> 545 -> 515 -> 699 -> 688 -> 387 -> 702 -> 975 -> 19 -> 951 -> 8 ->
 629 -> 397 -> 550 -> 624 -> 429 -> 373 -> 436 -> 521 -> 382 -> 487 -> 328 -> 429 -> 382 ->
 505 -> 544 -> 429 -> 907 -> 397 -> 624 -> 750 -> 328 -> 373 -> 988 -> 629 -> 550 -> 907 ->
 505 -> 23 -> 486 -> 207 -> 327 -> 433 -> 183 -> 309 -> 324 -> 625 -> 443 -> 779 -> 427 ->
 45 -> 110 -> 548 -> 80 -> 817 -> 110 -> 804 -> 817 -> 42 -> 625 -> 779 -> 780 -> 279 -> 360
 -> 698 -> 239 -> 329 -> 762 -> 592 -> 360 -> 780 -> 244 -> 779 -> 511 -> 244 -> 436 -> 988
 -> 550 -> 395 -> 697 -> 886 -> 975 -> 464 -> 656 -> 192 -> 699 -> 192 -> 387 -> 837 -> 682
 -> 170 -> 319 -> 473 -> 844 -> 84 -> 223 -> 473 -> 193 -> 237 -> 223 -> 844 -> 237 -> 665 -
 -> 282 -> 337 -> 381 -> 801 -> 458 -> 337 -> 237 -> 74 -> 84 -> 282 -> 381 -> 681 -> 801 ->
 681 -> 665 -> 223 -> 204 -> 844 -> 319 -> 212 -> 837 -> 455 -> 473 -> 212 -> 455 -> 97 ->
 968 -> 486 -> 494 -> 82 -> 947 -> 640 -> 97 -> 444 -> 169 -> 640 -> 94 -> 486 -> 82 -> 327 -
 -> 544 -> 328 -> 988 -> 397 -> 697 -> 550 -> 683 -> 629 -> 852 -> 599 -> 320 -> 905 -> 66 ->
 438 -> 454 -> 66 -> 485 -> 372 -> 8 -> 683 -> 395 -> 620 -> 329 -> 52 -> 106 -> 755 -> 728 -
 -> 830 -> 329 -> 855 -> 52 -> 762 -> 830 -> 573 -> 755 -> 743 -> 437 -> 604 -> 56 -> 638 ->
 52 -> 24 -> 638 -> 161 -> 106 -> 329 -> 698 -> 279 -> 511 -> 427 -> 110 -> 443 -> 511 ->
 625 -> 244 -> 324 -> 487 -> 436 -> 429 -> 487 -> 639 -> 382 -> 183 -> 521 -> 639 -> 183 ->
 80 -> 324 -> 511 -> 780 -> 620 -> 360 -> 329 -> 592 -> 239 -> 780 -> 147 -> 373 -> 629 ->
 624 -> 683 -> 697 -> 8 -> 978 -> 852 -> 187 -> 750 -> 988 -> 907 -> 629 -> 978 -> 372 ->
 187 -> 147 -> 599 -> 882 -> 239 -> 279 -> 620 -> 239 -> 360 -> 363 -> 279 -> 592 -> 882 ->
 447 -> 599 -> 131 -> 425 -> 320 -> 131 -> 882 -> 762 -> 698 -> 882 -> 830 -> 361 -> 374 ->
 573 -> 571 -> 743 -> 604 -> 24 -> 161 -> 52 -> 499 -> 638 -> 106 -> 363 -> 620 -> 698 ->
 363 -> 161 -> 499 -> 24 -> 437 -> 638 -> 604 -> 787 -> 489 -> 676 -> 658 -> 414 -> 440 ->
 772 -> 713 -> 489 -> 604 -> 860 -> 743 -> 787 -> 632 -> 743 -> 483 -> 389 -> 744 -> 389 ->
 483 -> 361 -> 3 -> 765 -> 861 -> 448 -> 438 -> 302 -> 872 -> 448 -> 913 -> 26 -> 454 -> 302
 -> 861 -> 224 -> 448 -> 765 -> 680 -> 861 -> 224 -> 438 -> 913 -> 66 -> 448 -> 26 -> 302 ->
 224 -> 913 -> 302 -> 872 -> 985 -> 902 -> 545 -> 583 -> 699 -> 656 -> 387 -> 212 -> 170 ->
 193 -> 665 -> 422 -> 143 -> 74 -> 143 -> 282 -> 681 -> 337 -> 665 -> 223 -> 204 -> 74 ->
 319 -> 682 -> 844 -> 665 -> 681 -> 458 -> 381 -> 455 -> 193 -> 223 -> 319 -> 837 -> 364 ->
 486 -> 947 -> 879 -> 771 -> 494 -> 23 -> 929 -> 494 -> 364 -> 771 -> 23 -> 82 -> 929 -> 771
 -> 947 -> 494 -> 94 -> 444 -> 82 -> 640 -> 748 -> 968 -> 444 -> 640 -> 968 -> 94 -> 169 ->
 748 -> 207 -> 444 -> 748 -> 97 -> 82 -> 748 -> 947 -> 23 -> 433 -> 639 -> 544 -> 521 -> 80 -
 -> 461 -> 324 -> 427 -> 42 -> 80 -> 309 -> 42 -> 324 -> 443 -> 427 -> 309 -> 548 -> 45 ->
 461 -> 817 -> 45 -> 324 -> 110 -> 461 -> 804 -> 548 -> 183 -> 804 -> 309 -> 487 -> 373 ->
 750 -> 8 -> 852 -> 147 -> 425 -> 485 -> 905 -> 20 -> 951 -> 975 -> 119 -> 886 -> 19 -> 906 -
 -> 879 -> 82 -> 968 -> 169 -> 97 -> 94 -> 947 -> 929 -> 879 -> 212 -> 682 -> 387 -> 719 ->
 515 -> 432 -> 833 -> 746 -> 765 -> 224 -> 454 -> 872 -> 985 -> 746 -> 445 -> 680 -> 445 ->
 572 -> 767 -> 820 -> 684 -> 554 -> 974 -> 976 -> 570 -> 396 -> 570 -> 976 -> 974 -> 200 ->
 420 -> 0 , Gesamtlänge: 2157699

Tag 2: 0 -> 922 -> 784 -> 492 -> 349 -> 859 -> 731 -> 25 -> 510 -> 848 -> 462 -> 130 -> 176
 -> 218 -> 202 -> 300 -> 47 -> 300 -> 835 -> 127 -> 39 -> 47 -> 218 -> 981 -> 896 -> 747 ->
 186 -> 109 -> 61 -> 816 -> 495 -> 346 -> 176 -> 47 -> 981 -> 896 -> 747 -> 186 -> 986 ->
 186 -> 747 -> 145 -> 964 -> 557 -> 218 -> 127 -> 202 -> 981 -> 300 -> 145 -> 300 -> 145 ->
 964 -> 47 -> 557 -> 964 -> 145 -> 747 -> 145 -> 747 -> 186 -> 986 -> 868 -> 986 -> 868 ->

125 -> 85 -> 120 -> 602 -> 788 -> 952 -> 0 -> 952 -> 788 -> 602 -> 120 -> 85 -> 125 -> 868 -> 125 -> 371 -> 909 -> 880 -> 673 -> 203 -> 987 -> 39 -> 127 -> 47 -> 176 -> 219 -> 230 -> 716 -> 77 -> 785 -> 230 -> 76 -> 575 -> 219 -> 130 -> 575 -> 495 -> 575 -> 219 -> 346 -> 176 -> 76 -> 230 -> 716 -> 785 -> 191 -> 474 -> 805 -> 842 -> 164 -> 196 -> 805 -> 516 -> 613 -> 842 -> 482 -> 613 -> 540 -> 775 -> 560 -> 266 -> 297 -> 288 -> 925 -> 715 -> 910 -> 315 -> 543 -> 135 -> 118 -> 562 -> 401 -> 793 -> 705 -> 791 -> 376 -> 390 -> 477 -> 342 -> 565 -> 877 -> 618 -> 206 -> 16 -> 663 -> 206 -> 270 -> 618 -> 75 -> 124 -> 541 -> 720 -> 717 -> 828 -> 829 -> 153 -> 304 -> 828 -> 850 -> 431 -> 452 -> 407 -> 995 -> 431 -> 828 -> 908 -> 100 -> 43 -> 546 -> 240 -> 178 -> 675 -> 43 -> 403 -> 256 -> 36 -> 923 -> 256 -> 834 -> 643 -> 284 -> 942 -> 539 -> 301 -> 983 -> 732 -> 198 -> 62 -> 589 -> 351 -> 777 -> 785 -> 198 -> 983 -> 351 -> 904 -> 942 -> 301 -> 939 -> 589 -> 136 -> 983 -> 777 -> 198 -> 134 -> 62 -> 77 -> 198 -> 301 -> 904 -> 939 -> 62 -> 777 -> 939 -> 732 -> 942 -> 823 -> 57 -> 472 -> 231 -> 452 -> 352 -> 132 -> 225 -> 403 -> 923 -> 761 -> 857 -> 603 -> 891 -> 953 -> 635 -> 603 -> 240 -> 635 -> 178 -> 453 -> 546 -> 178 -> 891 -> 675 -> 546 -> 953 -> 453 -> 225 -> 675 -> 603 -> 178 -> 43 -> 635 -> 891 -> 240 -> 675 -> 403 -> 178 -> 225 -> 626 -> 365 -> 53 -> 132 -> 523 -> 53 -> 352 -> 605 -> 214 -> 953 -> 626 -> 453 -> 365 -> 235 -> 290 -> 496 -> 355 -> 465 -> 415 -> 30 -> 253 -> 892 -> 156 -> 770 -> 954 -> 892 -> 898 -> 253 -> 158 -> 496 -> 30 -> 911 -> 355 -> 69 -> 527 -> 596 -> 156 -> 954 -> 263 -> 596 -> 770 -> 839 -> 115 -> 117 -> 322 -> 736 -> 966 -> 540 -> 164 -> 873 -> 527 -> 760 -> 535 -> 69 -> 156 -> 898 -> 168 -> 72 -> 228 -> 69 -> 168 -> 527 -> 72 -> 535 -> 158 -> 465 -> 69 -> 898 -> 596 -> 954 -> 228 -> 263 -> 903 -> 369 -> 870 -> 210 -> 103 -> 560 -> 973 -> 326 -> 973 -> 393 -> 150 -> 191 -> 342 -> 474 -> 100 -> 304 -> 908 -> 482 -> 516 -> 540 -> 298 -> 966 -> 560 -> 973 -> 475 -> 775 -> 966 -> 560 -> 103 -> 298 -> 560 -> 775 -> 298 -> 736 -> 103 -> 210 -> 117 -> 369 -> 115 -> 623 -> 839 -> 903 -> 870 -> 333 -> 115 -> 903 -> 623 -> 770 -> 928 -> 645 -> 104 -> 928 -> 307 -> 484 -> 104 -> 277 -> 766 -> 841 -> 41 -> 841 -> 766 -> 141 -> 385 -> 71 -> 307 -> 645 -> 595 -> 385 -> 307 -> 104 -> 484 -> 277 -> 357 -> 104 -> 595 -> 928 -> 357 -> 645 -> 104 -> 385 -> 71 -> 277 -> 307 -> 595 -> 623 -> 369 -> 333 -> 117 -> 623 -> 333 -> 903 -> 117 -> 870 -> 115 -> 770 -> 263 -> 156 -> 228 -> 892 -> 596 -> 228 -> 253 -> 415 -> 290 -> 30 -> 466 -> 290 -> 355 -> 30 -> 235 -> 466 -> 523 -> 626 -> 214 -> 365 -> 446 -> 626 -> 132 -> 214 -> 466 -> 911 -> 496 -> 415 -> 355 -> 535 -> 898 -> 158 -> 290 -> 591 -> 197 -> 304 -> 622 -> 591 -> 153 -> 908 -> 829 -> 471 -> 995 -> 53 -> 605 -> 365 -> 953 -> 240 -> 43 -> 603 -> 761 -> 834 -> 923 -> 666 -> 36 -> 823 -> 904 -> 284 -> 666 -> 643 -> 231 -> 255 -> 995 -> 472 -> 407 -> 533 -> 452 -> 471 -> 523 -> 352 -> 407 -> 231 -> 533 -> 850 -> 231 -> 995 -> 452 -> 761 -> 256 -> 666 -> 857 -> 834 -> 666 -> 255 -> 407 -> 850 -> 995 -> 533 -> 431 -> 717 -> 541 -> 918 -> 75 -> 752 -> 474 -> 642 -> 124 -> 537 -> 918 -> 899 -> 663 -> 937 -> 335 -> 469 -> 921 -> 33 -> 880 -> 33 -> 921 -> 469 -> 335 -> 937 -> 663 -> 877 -> 75 -> 935 -> 918 -> 124 -> 57 -> 539 -> 284 -> 36 -> 834 -> 403 -> 857 -> 256 -> 643 -> 255 -> 36 -> 539 -> 643 -> 823 -> 301 -> 589 -> 198 -> 942 -> 939 -> 983 -> 62 -> 785 -> 399 -> 127 -> 945 -> 39 -> 218 -> 399 -> 732 -> 777 -> 399 -> 77 -> 134 -> 399 -> 945 -> 777 -> 136 -> 899 -> 270 -> 935 -> 618 -> 16 -> 270 -> 663 -> 136 -> 935 -> 541 -> 57 -> 537 -> 642 -> 717 -> 537 -> 850 -> 717 -> 533 -> 472 -> 850 -> 720 -> 918 -> 642 -> 75 -> 270 -> 877 -> 752 -> 537 -> 720 -> 57 -> 231 -> 431 -> 471 -> 605 -> 523 -> 214 -> 446 -> 466 -> 496 -> 535 -> 168 -> 873 -> 197 -> 482 -> 164 -> 613 ->

805 -> 482 -> 100 -> 752 -> 342 -> 877 -> 899 -> 75 -> 565 -> 191 -> 805 -> 150 -> 613 -> 196 -> 482 -> 622 -> 873 -> 304 -> 829 -> 591 -> 908 -> 622 -> 153 -> 197 -> 100 -> 805 -> 164 -> 150 -> 196 -> 516 -> 393 -> 326 -> 350 -> 266 -> 350 -> 326 -> 540 -> 393 -> 775 -> 973 -> 475 -> 297 -> 257 -> 369 -> 322 -> 333 -> 4 -> 137 -> 379 -> 671 -> 339 -> 338 -> 59 -> 261 -> 189 -> 700 -> 41 -> 700 -> 189 -> 261 -> 59 -> 338 -> 982 -> 459 -> 2 -> 961 -> 306 -> 398 -> 601 -> 368 -> 467 -> 943 -> 135 -> 943 -> 467 -> 556 -> 426 -> 934 -> 205 -> 0 , Gesamtlänge: 2805648

Tag 3: 0 -> 420 -> 200 -> 974 -> 554 -> 684 -> 820 -> 684 -> 554 -> 974 -> 179 -> 181 -> 190 -> 292 -> 845 -> 547 -> 549 -> 685 -> 993 -> 569 -> 384 -> 524 -> 384 -> 341 -> 50 -> 375 -> 81 -> 984 -> 941 -> 763 -> 512 -> 524 -> 86 -> 936 -> 128 -> 579 -> 819 -> 295 -> 797 -> 199 -> 334 -> 812 -> 199 -> 734 -> 590 -> 714 -> 864 -> 15 -> 996 -> 851 -> 709 -> 590 -> 800 -> 233 -> 864 -> 996 -> 553 -> 15 -> 413 -> 89 -> 331 -> 508 -> 650 -> 418 -> 65 -> 96 -> 331 -> 175 -> 285 -> 413 -> 175 -> 721 -> 677 -> 551 -> 140 -> 155 -> 659 -> 116 -> 313 -> 22 -> 64 -> 998 -> 813 -> 38 -> 657 -> 734 -> 813 -> 696 -> 334 -> 734 -> 812 -> 295 -> 739 -> 199 -> 78 -> 295 -> 687 -> 797 -> 265 -> 78 -> 696 -> 199 -> 513 -> 739 -> 687 -> 513 -> 812 -> 233 -> 579 -> 126 -> 291 -> 933 -> 468 -> 252 -> 956 -> 822 -> 356 -> 113 -> 330 -> 938 -> 113 -> 416 -> 636 -> 500 -> 996 -> 413 -> 553 -> 500 -> 851 -> 864 -> 800 -> 709 -> 15 -> 636 -> 553 -> 674 -> 96 -> 175 -> 65 -> 331 -> 418 -> 175 -> 674 -> 416 -> 468 -> 126 -> 933 -> 938 -> 356 -> 938 -> 291 -> 506 -> 648 -> 773 -> 997 -> 799 -> 997 -> 773 -> 956 -> 971 -> 13 -> 938 -> 416 -> 580 -> 113 -> 133 -> 866 -> 754 -> 667 -> 917 -> 289 -> 285 -> 114 -> 386 -> 93 -> 234 -> 285 -> 96 -> 774 -> 917 -> 133 -> 667 -> 866 -> 641 -> 283 -> 386 -> 679 -> 809 -> 754 -> 917 -> 866 -> 171 -> 311 -> 672 -> 264 -> 919 -> 615 -> 608 -> 102 -> 627 -> 208 -> 367 -> 88 -> 339 -> 671 -> 379 -> 137 -> 726 -> 617 -> 201 -> 821 -> 277 -> 484 -> 71 -> 277 -> 201 -> 484 -> 71 -> 588 -> 308 -> 246 -> 353 -> 672 -> 919 -> 275 -> 308 -> 615 -> 672 -> 275 -> 615 -> 264 -> 608 -> 274 -> 735 -> 274 -> 275 -> 264 -> 11 -> 246 -> 919 -> 267 -> 11 -> 299 -> 267 -> 915 -> 357 -> 484 -> 915 -> 308 -> 672 -> 267 -> 264 -> 299 -> 353 -> 194 -> 246 -> 311 -> 194 -> 672 -> 11 -> 919 -> 608 -> 735 -> 242 -> 366 -> 955 -> 646 -> 955 -> 195 -> 955 -> 646 -> 330 -> 646 -> 330 -> 938 -> 126 -> 933 -> 579 -> 468 -> 128 -> 819 -> 723 -> 796 -> 344 -> 789 -> 867 -> 456 -> 63 -> 532 -> 272 -> 375 -> 81 -> 984 -> 532 -> 287 -> 867 -> 232 -> 325 -> 661 -> 17 -> 232 -> 456 -> 871 -> 532 -> 456 -> 769 -> 48 -> 480 -> 17 -> 963 -> 48 -> 661 -> 769 -> 17 -> 325 -> 769 -> 480 -> 232 -> 287 -> 456 -> 48 -> 871 -> 63 -> 441 -> 456 -> 661 -> 963 -> 480 -> 325 -> 963 -> 789 -> 63 -> 920 -> 723 -> 984 -> 287 -> 789 -> 260 -> 441 -> 532 -> 81 -> 920 -> 984 -> 796 -> 819 -> 344 -> 723 -> 941 -> 796 -> 920 -> 287 -> 871 -> 232 -> 769 -> 867 -> 63 -> 984 -> 344 -> 441 -> 687 -> 78 -> 797 -> 739 -> 260 -> 687 -> 265 -> 64 -> 78 -> 334 -> 590 -> 813 -> 334 -> 295 -> 513 -> 260 -> 344 -> 734 -> 998 -> 795 -> 581 -> 38 -> 689 -> 358 -> 10 -> 28 -> 226 -> 29 -> 668 -> 576 -> 5 -> 504 -> 10 -> 854 -> 778 -> 116 -> 155 -> 358 -> 28 -> 215 -> 689 -> 531 -> 215 -> 358 -> 792 -> 10 -> 254 -> 358 -> 778 -> 5 -> 140 -> 677 -> 139 -> 5 -> 659 -> 89 -> 721 -> 551 -> 659 -> 140 -> 721 -> 418 -> 413 -> 500 -> 714 -> 15 -> 851 -> 714 -> 996 -> 674 -> 413 -> 96 -> 234 -> 641 -> 93 -> 551 -> 139 -> 576 -> 504 -> 226 -> 792 -> 29 -> 10 -> 226 -> 668 -> 254 -> 792 -> 536 -> 254 -> 226 -> 536 -> 668 -> 792 -> 215 -> 91 -> 689 -> 22 -> 581 -> 215 -> 10 -> 536 -> 854 -> 689 -> 29 -> 313 -> 854 -> 116 -> 581 -> 313 -> 531 -> 38 -> 998 -> 334 -> 657 -> 590 ->

233 -> 709 -> 714 -> 813 -> 657 -> 998 -> 696 -> 734 -> 295 -> 199 -> 687 -> 789 -> 441 ->
 867 -> 871 -> 965 -> 652 -> 458 -> 801 -> 798 -> 49 -> 948 -> 458 -> 337 -> 237 -> 844 ->
 682 -> 319 -> 74 -> 143 -> 422 -> 727 -> 46 -> 965 -> 948 -> 798 -> 965 -> 49 -> 458 -> 337
 -> 237 -> 844 -> 682 -> 387 -> 192 -> 699 -> 515 -> 432 -> 833 -> 746 -> 445 -> 572 -> 767
 -> 820 -> 767 -> 572 -> 445 -> 746 -> 833 -> 432 -> 515 -> 699 -> 192 -> 387 -> 682 -> 844
 -> 237 -> 337 -> 458 -> 652 -> 948 -> 49 -> 22 -> 29 -> 28 -> 668 -> 504 -> 254 -> 576 ->
 536 -> 155 -> 677 -> 659 -> 139 -> 504 -> 778 -> 139 -> 171 -> 679 -> 114 -> 283 -> 508 ->
 234 -> 650 -> 65 -> 93 -> 508 -> 285 -> 774 -> 133 -> 754 -> 774 -> 667 -> 289 -> 580 ->
 133 -> 289 -> 866 -> 679 -> 311 -> 809 -> 866 -> 114 -> 171 -> 641 -> 114 -> 234 -> 283 ->
 93 -> 650 -> 175 -> 234 -> 386 -> 171 -> 194 -> 299 -> 246 -> 264 -> 308 -> 11 -> 915 -> 71
 -> 201 -> 588 -> 277 -> 766 -> 277 -> 357 -> 484 -> 645 -> 307 -> 385 -> 141 -> 766 -> 841
 -> 41 -> 700 -> 189 -> 261 -> 59 -> 338 -> 339 -> 338 -> 982 -> 459 -> 2 -> 961 -> 306 ->
 165 -> 619 -> 782 -> 799 -> 542 -> 887 -> 195 -> 887 -> 542 -> 799 -> 782 -> 619 -> 165 ->
 306 -> 398 -> 601 -> 368 -> 467 -> 556 -> 426 -> 934 -> 205 -> 0 , Gesamtlänge: 2601671

Tag 4: 0 -> 824 -> 788 -> 934 -> 703 -> 348 -> 960 -> 528 -> 205 -> 853 -> 974 -> 694 ->
 976 -> 79 -> 442 -> 889 -> 122 -> 1 -> 460 -> 238 -> 889 -> 1 -> 200 -> 310 -> 294 -> 420 ->
 952 -> 205 -> 934 -> 602 -> 120 -> 222 -> 826 -> 380 -> 120 -> 490 -> 419 -> 349 -> 362 ->
 370 -> 558 -> 182 -> 914 -> 980 -> 525 -> 251 -> 914 -> 111 -> 610 -> 662 -> 251 -> 894 ->
 404 -> 44 -> 251 -> 343 -> 611 -> 718 -> 827 -> 994 -> 7 -> 243 -> 692 -> 273 -> 305 ->
 827 -> 7 -> 611 -> 994 -> 305 -> 185 -> 718 -> 269 -> 7 -> 718 -> 305 -> 269 -> 552 -> 280 ->
 503 -> 552 -> 185 -> 273 -> 555 -> 209 -> 273 -> 493 -> 268 -> 269 -> 185 -> 884 -> 894 ->
 44 -> 7 -> 185 -> 493 -> 552 -> 584 -> 280 -> 268 -> 994 -> 718 -> 243 -> 884 -> 273 ->
 503 -> 584 -> 593 -> 281 -> 952 -> 824 -> 205 -> 788 -> 922 -> 664 -> 856 -> 991 -> 370 ->
 492 -> 784 -> 598 -> 768 -> 958 -> 602 -> 490 -> 598 -> 492 -> 362 -> 419 -> 859 -> 423 ->
 530 -> 644 -> 686 -> 162 -> 846 -> 967 -> 174 -> 241 -> 558 -> 314 -> 174 -> 362 -> 991 ->
 174 -> 370 -> 314 -> 991 -> 160 -> 649 -> 380 -> 85 -> 120 -> 826 -> 556 -> 426 -> 647 ->
 359 -> 628 -> 348 -> 31 -> 628 -> 703 -> 359 -> 31 -> 348 -> 528 -> 703 -> 960 -> 426 ->
 628 -> 556 -> 467 -> 40 -> 826 -> 556 -> 467 -> 943 -> 135 -> 118 -> 566 -> 730 -> 405 ->
 34 -> 258 -> 125 -> 811 -> 12 -> 751 -> 18 -> 186 -> 217 -> 986 -> 371 -> 862 -> 729 -> 371
 -> 781 -> 258 -> 34 -> 566 -> 670 -> 811 -> 781 -> 125 -> 154 -> 868 -> 18 -> 986 -> 862 ->
 868 -> 12 -> 862 -> 730 -> 909 -> 32 -> 34 -> 670 -> 520 -> 258 -> 811 -> 729 -> 751 -> 868
 -> 125 -> 371 -> 909 -> 34 -> 520 -> 566 -> 424 -> 786 -> 98 -> 258 -> 98 -> 392 -> 380 ->
 783 -> 602 -> 598 -> 856 -> 958 -> 649 -> 490 -> 160 -> 362 -> 166 -> 419 -> 154 -> 349 ->
 492 -> 768 -> 992 -> 784 -> 664 -> 768 -> 856 -> 784 -> 922 -> 952 -> 200 -> 853 -> 694 ->
 554 -> 853 -> 694 -> 179 -> 974 -> 824 -> 310 -> 420 -> 0 -> 952 -> 788 -> 602 -> 426 ->
 934 -> 323 -> 205 -> 0 -> 303 -> 294 -> 1 -> 708 -> 37 -> 238 -> 708 -> 420 -> 788 -> 281 ->
 479 -> 593 -> 503 -> 718 -> 268 -> 243 -> 209 -> 614 -> 664 -> 428 -> 479 -> 280 -> 555 ->
 692 -> 614 -> 503 -> 493 -> 280 -> 593 -> 303 -> 584 -> 555 -> 243 -> 894 -> 343 -> 525 ->
 914 -> 526 -> 722 -> 229 -> 241 -> 182 -> 111 -> 722 -> 610 -> 914 -> 722 -> 662 -> 111 ->
 526 -> 525 -> 241 -> 662 -> 182 -> 526 -> 980 -> 251 -> 229 -> 980 -> 111 -> 251 -> 182 ->
 229 -> 525 -> 722 -> 980 -> 610 -> 526 -> 251 -> 404 -> 827 -> 611 -> 404 -> 343 -> 827 ->
 44 -> 692 -> 209 -> 428 -> 273 -> 614 -> 593 -> 555 -> 493 -> 584 -> 479 -> 220 -> 303 ->
 310 -> 0 -> 593 -> 428 -> 768 -> 784 -> 958 -> 490 -> 40 -> 85 -> 125 -> 6 -> 419 -> 160 ->

492 -> 991 -> 768 -> 209 -> 884 -> 555 -> 428 -> 281 -> 922 -> 0 -> 294 -> 708 -> 122 ->
 460 -> 451 -> 976 -> 974 -> 554 -> 323 -> 528 -> 934 -> 628 -> 647 -> 31 -> 426 -> 783 ->
 222 -> 380 -> 40 -> 783 -> 120 -> 392 -> 85 -> 649 -> 40 -> 392 -> 222 -> 85 -> 6 -> 154 ->
 12 -> 371 -> 18 -> 729 -> 868 -> 371 -> 811 -> 566 -> 32 -> 258 -> 566 -> 405 -> 909 -> 34 ->
 716 -> 230 -> 219 -> 838 -> 145 -> 803 -> 896 -> 747 -> 217 -> 751 -> 862 -> 18 -> 217 ->
 149 -> 109 -> 470 -> 507 -> 704 -> 630 -> 964 -> 202 -> 300 -> 838 -> 557 -> 740 -> 816 ->
 597 -> 300 -> 202 -> 803 -> 557 -> 964 -> 740 -> 742 -> 630 -> 808 -> 764 -> 262 -> 510 ->
 25 -> 686 -> 213 -> 14 -> 423 -> 660 -> 530 -> 213 -> 644 -> 846 -> 213 -> 430 -> 749 -> 14
 -> 644 -> 423 -> 213 -> 749 -> 423 -> 686 -> 530 -> 14 -> 162 -> 430 -> 14 -> 660 -> 644 ->
 510 -> 848 -> 248 -> 462 -> 575 -> 346 -> 76 -> 575 -> 495 -> 816 -> 742 -> 815 -> 495 ->
 742 -> 575 -> 219 -> 838 -> 597 -> 740 -> 838 -> 816 -> 517 -> 711 -> 109 -> 61 -> 517 ->
 704 -> 470 -> 764 -> 711 -> 217 -> 109 -> 747 -> 145 -> 964 -> 896 -> 803 -> 597 -> 838 ->
 202 -> 557 -> 495 -> 838 -> 964 -> 300 -> 740 -> 575 -> 462 -> 848 -> 130 -> 575 -> 130 ->
 248 -> 815 -> 848 -> 76 -> 462 -> 815 -> 346 -> 495 -> 740 -> 202 -> 597 -> 964 -> 803 ->
 597 -> 470 -> 808 -> 262 -> 25 -> 731 -> 859 -> 530 -> 846 -> 749 -> 510 -> 686 -> 749 ->
 644 -> 162 -> 423 -> 731 -> 686 -> 660 -> 967 -> 558 -> 662 -> 525 -> 182 -> 314 -> 992 ->
 856 -> 160 -> 166 -> 6 -> 868 -> 986 -> 186 -> 109 -> 296 -> 186 -> 507 -> 296 -> 61 -> 704
 -> 262 -> 517 -> 630 -> 764 -> 704 -> 711 -> 149 -> 186 -> 747 -> 145 -> 300 -> 557 -> 597
 -> 145 -> 964 -> 816 -> 61 -> 630 -> 470 -> 517 -> 507 -> 149 -> 731 -> 749 -> 660 -> 859 ->
 349 -> 166 -> 154 -> 751 -> 371 -> 730 -> 811 -> 520 -> 405 -> 32 -> 520 -> 730 -> 12 ->
 781 -> 6 -> 1 -> 420 -> 200 -> 974 -> 634 -> 976 -> 570 -> 79 -> 449 -> 451 -> 959 -> 442 ->
 451 -> 442 -> 238 -> 776 -> 122 -> 37 -> 889 -> 959 -> 449 -> 634 -> 460 -> 442 -> 122 ->
 220 -> 776 -> 889 -> 460 -> 37 -> 776 -> 708 -> 303 -> 281 -> 0 , Gesamtlänge: 2586487

Tag 5: 0 -> 294 -> 1 -> 460 -> 442 -> 745 -> 442 -> 460 -> 451 -> 582 -> 570 -> 832 -> 396 ->
 502 -> 491 -> 710 -> 522 -> 440 -> 582 -> 563 -> 522 -> 396 -> 570 -> 976 -> 974 -> 200 ->
 974 -> 694 -> 725 -> 554 -> 21 -> 181 -> 916 -> 628 -> 31 -> 146 -> 810 -> 359 -> 703 ->
 628 -> 810 -> 556 -> 467 -> 826 -> 377 -> 98 -> 135 -> 786 -> 543 -> 398 -> 27 -> 712 ->
 497 -> 601 -> 439 -> 476 -> 712 -> 27 -> 60 -> 637 -> 538 -> 912 -> 163 -> 538 -> 757 -> 83
 -> 888 -> 188 -> 292 -> 318 -> 888 -> 757 -> 27 -> 538 -> 949 -> 83 -> 368 -> 377 -> 568 ->
 368 -> 439 -> 398 -> 306 -> 165 -> 961 -> 2 -> 459 -> 901 -> 606 -> 982 -> 865 -> 578 ->
 402 -> 529 -> 338 -> 286 -> 542 -> 245 -> 897 -> 957 -> 271 -> 897 -> 944 -> 577 -> 434 ->
 890 -> 383 -> 799 -> 542 -> 979 -> 890 -> 944 -> 607 -> 417 -> 408 -> 144 -> 890 -> 245 ->
 151 -> 706 -> 95 -> 881 -> 957 -> 95 -> 972 -> 648 -> 695 -> 105 -> 612 -> 969 -> 506 ->
 693 -> 893 -> 940 -> 753 -> 50 -> 341 -> 946 -> 849 -> 391 -> 108 -> 893 -> 869 -> 519 ->
 509 -> 807 -> 259 -> 249 -> 92 -> 340 -> 807 -> 92 -> 152 -> 874 -> 701 -> 84 -> 204 -> 143
 -> 74 -> 874 -> 741 -> 927 -> 849 -> 138 -> 950 -> 798 -> 46 -> 801 -> 727 -> 422 -> 143 ->
 84 -> 422 -> 701 -> 74 -> 204 -> 701 -> 143 -> 422 -> 204 -> 874 -> 989 -> 121 -> 849 ->
 585 -> 753 -> 936 -> 763 -> 184 -> 86 -> 512 -> 564 -> 341 -> 384 -> 869 -> 259 -> 509 ->
 249 -> 67 -> 840 -> 92 -> 67 -> 340 -> 249 -> 840 -> 836 -> 825 -> 336 -> 688 -> 656 -> 825
 -> 900 -> 654 -> 159 -> 653 -> 806 -> 534 -> 653 -> 107 -> 432 -> 302 -> 902 -> 985 -> 26 ->
 54 -> 985 -> 913 -> 224 -> 833 -> 895 -> 746 -> 680 -> 861 -> 746 -> 818 -> 653 -> 685 ->
 549 -> 101 -> 678 -> 999 -> 123 -> 388 -> 417 -> 90 -> 481 -> 123 -> 878 -> 157 -> 569 ->
 993 -> 378 -> 569 -> 498 -> 384 -> 524 -> 512 -> 763 -> 941 -> 984 -> 272 -> 738 -> 501 ->

375 -> 272 -> 81 -> 316 -> 585 -> 50 -> 512 -> 936 -> 970 -> 941 -> 128 -> 970 -> 86 -> 524 -> 893 -> 184 -> 524 -> 341 -> 669 -> 946 -> 50 -> 86 -> 564 -> 753 -> 512 -> 940 -> 341 -> 753 -> 946 -> 585 -> 108 -> 501 -> 81 -> 375 -> 108 -> 316 -> 738 -> 81 -> 272 -> 316 -> 391 -> 950 -> 948 -> 458 -> 801 -> 798 -> 46 -> 49 -> 458 -> 652 -> 801 -> 948 -> 46 -> 727 -> 701 -> 282 -> 665 -> 844 -> 237 -> 337 -> 458 -> 46 -> 950 -> 108 -> 849 -> 616 -> 741 -> 989 -> 927 -> 616 -> 727 -> 282 -> 665 -> 844 -> 682 -> 319 -> 74 -> 84 -> 143 -> 74 -> 152 -> 67 -> 509 -> 152 -> 340 -> 669 -> 741 -> 858 -> 927 -> 138 -> 989 -> 858 -> 849 -> 989 -> 616 -> 121 -> 138 -> 108 -> 738 -> 375 -> 391 -> 585 -> 375 -> 50 -> 564 -> 940 -> 524 -> 564 -> 184 -> 972 -> 706 -> 271 -> 972 -> 695 -> 506 -> 612 -> 648 -> 831 -> 972 -> 105 -> 970 -> 468 -> 252 -> 969 -> 693 -> 105 -> 969 -> 648 -> 956 -> 955 -> 646 -> 971 -> 847 -> 291 -> 126 -> 970 -> 763 -> 128 -> 936 -> 86 -> 940 -> 519 -> 564 -> 893 -> 972 -> 612 -> 252 -> 956 -> 55 -> 822 -> 330 -> 13 -> 356 -> 847 -> 252 -> 971 -> 955 -> 366 -> 236 -> 488 -> 35 -> 274 -> 735 -> 567 -> 242 -> 735 -> 435 -> 759 -> 627 -> 410 -> 726 -> 137 -> 379 -> 671 -> 802 -> 932 -> 339 -> 802 -> 261 -> 88 -> 339 -> 261 -> 189 -> 367 -> 379 -> 400 -> 261 -> 59 -> 400 -> 802 -> 887 -> 932 -> 59 -> 339 -> 400 -> 671 -> 261 -> 379 -> 926 -> 733 -> 966 -> 321 -> 736 -> 4 -> 412 -> 322 -> 210 -> 103 -> 560 -> 266 -> 863 -> 347 -> 475 -> 326 -> 594 -> 266 -> 257 -> 390 -> 206 -> 663 -> 937 -> 737 -> 631 -> 376 -> 791 -> 925 -> 288 -> 297 -> 266 -> 350 -> 594 -> 347 -> 350 -> 475 -> 775 -> 733 -> 321 -> 4 -> 210 -> 870 -> 412 -> 41 -> 726 -> 700 -> 189 -> 137 -> 700 -> 759 -> 367 -> 208 -> 790 -> 35 -> 627 -> 617 -> 841 -> 394 -> 141 -> 617 -> 201 -> 141 -> 71 -> 277 -> 766 -> 841 -> 410 -> 700 -> 41 -> 841 -> 412 -> 137 -> 926 -> 103 -> 4 -> 210 -> 736 -> 733 -> 560 -> 103 -> 736 -> 322 -> 870 -> 322 -> 4 -> 137 -> 41 -> 410 -> 102 -> 627 -> 488 -> 435 -> 102 -> 488 -> 790 -> 274 -> 242 -> 366 -> 13 -> 646 -> 330 -> 735 -> 35 -> 435 -> 627 -> 172 -> 366 -> 646 -> 55 -> 955 -> 822 -> 252 -> 506 -> 648 -> 773 -> 95 -> 831 -> 195 -> 955 -> 172 -> 208 -> 488 -> 172 -> 367 -> 88 -> 379 -> 189 -> 759 -> 726 -> 412 -> 394 -> 617 -> 167 -> 385 -> 277 -> 385 -> 141 -> 766 -> 167 -> 201 -> 385 -> 201 -> 766 -> 617 -> 726 -> 367 -> 41 -> 759 -> 208 -> 700 -> 88 -> 137 -> 759 -> 410 -> 435 -> 790 -> 627 -> 208 -> 435 -> 236 -> 242 -> 35 -> 102 -> 735 -> 102 -> 274 -> 567 -> 189 -> 400 -> 932 -> 195 -> 286 -> 887 -> 195 -> 773 -> 997 -> 151 -> 577 -> 383 -> 979 -> 799 -> 782 -> 931 -> 459 -> 876 -> 606 -> 715 -> 705 -> 921 -> 600 -> 70 -> 793 -> 600 -> 401 -> 33 -> 880 -> 278 -> 118 -> 135 -> 312 -> 315 -> 690 -> 961 -> 70 -> 317 -> 58 -> 910 -> 315 -> 317 -> 690 -> 910 -> 715 -> 791 -> 705 -> 876 -> 478 -> 2 -> 619 -> 961 -> 315 -> 70 -> 690 -> 58 -> 562 -> 278 -> 405 -> 424 -> 32 -> 258 -> 34 -> 786 -> 312 -> 497 -> 439 -> 83 -> 2 -> 173 -> 459 -> 982 -> 962 -> 99 -> 578 -> 962 -> 865 -> 99 -> 529 -> 293 -> 979 -> 286 -> 932 -> 671 -> 339 -> 338 -> 578 -> 982 -> 177 -> 478 -> 459 -> 715 -> 478 -> 901 -> 402 -> 982 -> 338 -> 962 -> 402 -> 177 -> 459 -> 99 -> 982 -> 529 -> 177 -> 931 -> 173 -> 383 -> 997 -> 245 -> 383 -> 944 -> 434 -> 607 -> 90 -> 180 -> 417 -> 999 -> 247 -> 843 -> 678 -> 276 -> 883 -> 292 -> 621 -> 586 -> 354 -> 756 -> 87 -> 292 -> 211 -> 87 -> 586 -> 883 -> 547 -> 101 -> 123 -> 51 -> 724 -> 569 -> 142 -> 559 -> 900 -> 930 -> 699 -> 719 -> 332 -> 583 -> 719 -> 336 -> 900 -> 924 -> 654 -> 559 -> 587 -> 159 -> 559 -> 378 -> 68 -> 724 -> 993 -> 73 -> 559 -> 685 -> 534 -> 73 -> 409 -> 51 -> 498 -> 259 -> 384 -> 569 -> 68 -> 993 -> 142 -> 587 -> 685 -> 73 -> 378 -> 534 -> 993 -> 685 -> 806 -> 159 -> 107 -> 818 -> 902 -> 107 -> 54 -> 432 -> 515 ->

332 -> 515 -> 719 -> 656 -> 336 -> 699 -> 924 -> 719 -> 836 -> 900 -> 587 -> 836 -> 930 -> 825 -> 192 -> 387 -> 682 -> 387 -> 192 -> 699 -> 719 -> 825 -> 924 -> 930 -> 719 -> 900 -> 227 -> 559 -> 993 -> 409 -> 378 -> 142 -> 654 -> 587 -> 924 -> 836 -> 336 -> 924 -> 332 -> 699 -> 688 -> 656 -> 699 -> 515 -> 825 -> 688 -> 332 -> 583 -> 545 -> 902 -> 432 -> 302 -> 861 -> 224 -> 861 -> 833 -> 432 -> 985 -> 26 -> 302 -> 833 -> 746 -> 445 -> 680 -> 746 -> 985 -> 818 -> 432 -> 913 -> 985 -> 432 -> 26 -> 818 -> 833 -> 680 -> 861 -> 895 -> 680 -> 445 -> 518 -> 572 -> 389 -> 3 -> 814 -> 744 -> 112 -> 794 -> 345 -> 820 -> 181 -> 725 -> 684 -> 574 -> 767 -> 345 -> 574 -> 707 -> 684 -> 820 -> 129 -> 190 -> 756 -> 633 -> 348 -> 916 -> 633 -> 146 -> 960 -> 528 -> 205 -> 528 -> 348 -> 810 -> 633 -> 810 -> 960 -> 916 -> 146 -> 647 -> 426 -> 934 -> 205 -> 0 -> 952 -> 788 -> 602 -> 120 -> 826 -> 568 -> 943 -> 377 -> 467 -> 647 -> 556 -> 810 -> 31 -> 916 -> 359 -> 703 -> 528 -> 960 -> 633 -> 916 -> 181 -> 179 -> 21 -> 574 -> 725 -> 179 -> 181 -> 190 -> 318 -> 845 -> 547 -> 758 -> 101 -> 999 -> 843 -> 758 -> 247 -> 549 -> 276 -> 547 -> 678 -> 247 -> 547 -> 549 -> 758 -> 276 -> 101 -> 883 -> 250 -> 621 -> 586 -> 292 -> 354 -> 190 -> 655 -> 129 -> 21 -> 820 -> 707 -> 767 -> 572 -> 112 -> 9 -> 483 -> 389 -> 744 -> 396 -> 491 -> 794 -> 710 -> 563 -> 772 -> 745 -> 832 -> 563 -> 745 -> 522 -> 463 -> 570 -> 396 -> 491 -> 744 -> 710 -> 463 -> 582 -> 502 -> 570 -> 21 -> 570 -> 396 -> 744 -> 9 -> 744 -> 794 -> 463 -> 396 -> 794 -> 518 -> 250 -> 845 -> 211 -> 586 -> 188 -> 354 -> 888 -> 163 -> 912 -> 476 -> 601 -> 912 -> 27 -> 476 -> 306 -> 712 -> 421 -> 637 -> 221 -> 619 -> 165 -> 221 -> 875 -> 60 -> 421 -> 757 -> 949 -> 188 -> 163 -> 949 -> 888 -> 756 -> 810 -> 647 -> 426 -> 556 -> 467 -> 943 -> 601 -> 368 -> 467 -> 568 -> 98 -> 258 -> 34 -> 424 -> 118 -> 32 -> 278 -> 401 -> 562 -> 880 -> 148 -> 673 -> 203 -> 411 -> 469 -> 561 -> 885 -> 216 -> 203 -> 987 -> 469 -> 651 -> 514 -> 737 -> 469 -> 921 -> 33 -> 148 -> 747 -> 896 -> 981 -> 987 -> 127 -> 202 -> 300 -> 835 -> 300 -> 145 -> 747 -> 186 -> 986 -> 868 -> 125 -> 85 -> 120 -> 392 -> 98 -> 786 -> 424 -> 118 -> 562 -> 32 -> 424 -> 405 -> 730 -> 811 -> 125 -> 371 -> 909 -> 880 -> 673 -> 33 -> 216 -> 987 -> 39 -> 981 -> 835 -> 987 -> 411 -> 737 -> 791 -> 715 -> 876 -> 925 -> 715 -> 910 -> 793 -> 705 -> 631 -> 514 -> 885 -> 631 -> 288 -> 977 -> 390 -> 297 -> 475 -> 266 -> 347 -> 257 -> 475 -> 973 -> 347 -> 990 -> 733 -> 103 -> 321 -> 926 -> 560 -> 966 -> 926 -> 990 -> 59 -> 338 -> 99 -> 901 -> 529 -> 865 -> 863 -> 297 -> 257 -> 477 -> 977 -> 206 -> 663 -> 406 -> 561 -> 335 -> 651 -> 561 -> 514 -> 937 -> 335 -> 469 -> 514 -> 335 -> 450 -> 406 -> 651 -> 216 -> 411 -> 651 -> 450 -> 885 -> 335 -> 406 -> 937 -> 450 -> 631 -> 791 -> 376 -> 390 -> 477 -> 350 -> 266 -> 257 -> 594 -> 863 -> 350 -> 326 -> 350 -> 973 -> 266 -> 477 -> 390 -> 376 -> 977 -> 406 -> 737 -> 561 -> 450 -> 737 -> 651 -> 885 -> 411 -> 921 -> 216 -> 673 -> 921 -> 561 -> 791 -> 876 -> 631 -> 925 -> 478 -> 173 -> 619 -> 875 -> 637 -> 619 -> 782 -> 890 -> 151 -> 383 -> 542 -> 887 -> 773 -> 956 -> 822 -> 13 -> 971 -> 822 -> 356 -> 567 -> 646 -> 956 -> 971 -> 55 -> 366 -> 567 -> 13 -> 356 -> 847 -> 291 -> 933 -> 128 -> 970 -> 468 -> 291 -> 506 -> 105 -> 184 -> 693 -> 612 -> 695 -> 95 -> 271 -> 881 -> 972 -> 957 -> 706 -> 577 -> 144 -> 388 -> 481 -> 434 -> 388 -> 607 -> 408 -> 90 -> 388 -> 180 -> 123 -> 417 -> 144 -> 782 -> 245 -> 799 -> 997 -> 979 -> 931 -> 901 -> 177 -> 606 -> 2 -> 690 -> 793 -> 58 -> 401 -> 793 -> 317 -> 910 -> 600 -> 58 -> 70 -> 910 -> 315 -> 306 -> 315 -> 543 -> 135 -> 943 -> 439 -> 757 -> 601 -> 398 -> 497 -> 543 -> 601 -> 83 -> 912 -> 27 -> 421 -> 538 -> 60 -> 221 -> 961 -> 306 -> 712 -> 398 -> 476 -> 83 -> 163 -> 188 -> 318 -> 87 -> 190 -> 292 -> 845 -> 188 -> 586 -> 318 -> 354 -> 87 -> 655 -> 707 -> 129 -> 574 -> 820 -> 767 -> 518 ->

621 -> 609 -> 112 -> 814 -> 609 -> 518 -> 445 -> 572 -> 3 -> 112 -> 389 -> 3 -> 445 -> 814 -> 518 -> 621 -> 883 -> 211 -> 655 -> 181 -> 684 -> 554 -> 974 -> 179 -> 554 -> 21 -> 725 -> 853 -> 200 -> 420 -> 0 , Gesamtlänge: 5370110

Maximale Lange einer Tagestour: 5370110

Quellcode

```
3  public class Edge {
4      private int a;
5      private int b;
6
7      private int cost;
8
9     public Edge(int a, int b, int cost) {
10        this.a = a;
11        this.b = b;
12        this.cost = cost;
13    }
14
15    public int getOppositVertex(int vertex) {
16        if(a == vertex) {
17            return b;
18        }
19        else if(vertex == b){
20            return a;
21        }
22        else {
23            return -1;
24        }
25    }
26
27    public int getA() {
28        return a;
29    }
30
31    public int getB() {
32        return b;
33    }
34
35    public int getCost() {
36        return cost;
37    }
38
39    public void setCost(int cost) {
40        this.cost = cost;
41    }
42
43    @Override
44    public boolean equals(Object object) {
45        Edge edge = (Edge) object;
46
47        if(edge.getA() == a && edge.getB() == b && edge.getCost() == cost) {
48            return true;
49        }
50        else {
51            return false;
52        }
53    }
54
55    @Override
56    public Edge clone() {
57        return new Edge(a, b, cost);
58    }
59
60    @Override
61    public String toString() {
62        String val = "Edge: " + a + " -> " + b + " Cost: " + cost;
63        return val;
64    }
65 }
```

Codeausschnitt 1: Edge-Klasse

```

142 public HashMap<Integer, HashMap<Integer, Path>> getShortestPaths() {
143     HashMap<Integer, HashMap<Integer, Path>> shortestPaths = new HashMap<Integer, HashMap<Integer, Path>>();
144
145     for(Integer v : vertexSet) {
146         shortestPaths.put(v, getShortestPaths(v));
147     }
148
149     return shortestPaths;
150 }
151
152 public HashMap<Integer, Path> getShortestPaths(int startingVertex) {
153     HashSet<Integer> visitedVertecies = new HashSet<Integer>();
154     PriorityQueue<Path> paths = new PriorityQueue<Path>();
155     HashMap<Integer, Path> shortestPaths = new HashMap<Integer, Path>();
156
157     for(Edge edge : adjacencyList.get(startingVertex)) {
158         Path newPath = new Path(startingVertex);
159         newPath.append(edge);
160
161         paths.add(newPath);
162     }
163
164     visitedVertecies.add(startingVertex);
165
166     while(paths.peek() != null) {
167         Path bestPath = paths.poll();
168
169         if(visitedVertecies.contains(bestPath.getEndVertex())){
170             continue;
171         }
172         else {
173             shortestPaths.put(bestPath.getEndVertex(), bestPath);
174
175             visitedVertecies.add(bestPath.getEndVertex());
176
177             for(Edge edge : adjacencyList.get(bestPath.getEndVertex())) {
178                 Path newPath = bestPath.clone();
179                 newPath.append(edge);
180
181                 paths.offer(newPath);
182             }
183         }
184     }
185
186     shortestPaths.put(startingVertex, new Path(startingVertex));
187
188     return shortestPaths;
189 }
190

```

Codeausschnitt 2: getShortestPaths

```

6  public class Path implements Comparable<Path>{
7      private ArrayList<Edge> edgeList = new ArrayList<Edge>();
8      private HashSet<Edge> edgeSet = new HashSet<Edge>();
9
10     private ArrayList<Integer> vertexList = new ArrayList<Integer>();
11     private HashSet<Integer> vertexSet = new HashSet<Integer>();
12
13     private int startVertex = -1;
14     private int endVertex = -1;
15
16     public Path(int startVertex) {
17         this.startVertex = startVertex;
18         endVertex = startVertex;
19
20         vertexList.add(startVertex);
21         vertexSet.add(startVertex);
22     }
23
24     private Path(ArrayList<Edge> edgeList, HashSet<Edge> edgeSet, ArrayList<Integer> vertexList, HashSet<Integer> vertexSet, int startVertex, int endVertex) {
25         this.edgeList.addAll(edgeList);
26         this.edgeSet.addAll(edgeSet);
27
28         this.vertexList.addAll(vertexList);
29         this.vertexSet.addAll(vertexSet);
30
31         this.startVertex = startVertex;
32         this.endVertex = endVertex;
33     }
34
35     public void append(Edge edge) {
36         edgeSet.add(edge);
37         edgeList.add(edge);
38
39         endVertex = edge.getOppositeVertex(endVertex);
40
41         vertexList.add(endVertex);
42         vertexSet.add(endVertex);
43     }
44
45     public void insertCycle(Path path) {
46         vertexSet.addAll(path.getVertexSet());
47         edgeSet.addAll(path.getEdgeSet());
48
49         int insertionPoint = -1;
50
51         for(int n = 0; n < vertexList.size(); n++) {
52             if(vertexList.get(n) == path.getStartVertex()) {
53                 insertionPoint = n;
54             }
55         }
56
57         vertexList.remove(insertionPoint);
58         vertexList.addAll(insertionPoint, path.getVertexList());
59         edgeList.addAll(insertionPoint, path.getEdgeList());
60
61     }
62 }

```

Codeausschnitt 3: Path-Klasse

```

98     /*
99      * Determine edge frequency
100     */
101    HashMap<Edge, Integer> edgeFrequency = new HashMap<Edge, Integer>();
102
103    for(Edge candidate : graph.getEdgeSet()) {
104        if(!edgeFrequency.containsKey(candidate)) {
105            edgeFrequency.put(candidate, 0);
106        }
107
108        Path[] connectingPathes = new Path[2];
109        connectingPathes[0] = shortestPaths.get(0).get(candidate.getA());
110        connectingPathes[1] = shortestPaths.get(0).get(candidate.getB());
111
112        for(int n = 0; n < 2; n++) {
113            for(Edge edge : connectingPathes[n].getEdgeSet()) {
114                if(edgeFrequency.containsKey(edge)) {
115                    edgeFrequency.put(edge, edgeFrequency.get(edge) + 1);
116                }
117                else {
118                    edgeFrequency.put(edge, 1);
119                }
120            }
121        }
122    }
123

```

Codeausschnitt 4: getDayRoutes-Methode – Bestimmung der Edge Frequenz

```

124      /*
125       * Determine required edges
126       */
127     HashSet<Edge> requiredEdges = new HashSet<Edge>();
128
129     for(Edge edge : graph.getEdgeSet()) {
130         if(edgeFrequency.get(edge) <= 1) {
131             requiredEdges.add(edge);
132         }
133     }
134

```

Codeausschnitt 5: getDayRoutes-Methode – Bestimmung der kritischen Kanten

```

135     /*
136      * Determine representative edges
137      */
138     HashSet<Edge> representativeEdges = new HashSet<Edge>();
139
140     for(int n = 0; n < 5 && n < requiredEdges.size(); n++) {
141         if(representativeEdges.isEmpty()) {
142             Edge representativeEdge = null;
143             for(Edge edge : requiredEdges) {
144                 if(representativeEdge == null) {
145                     representativeEdge = edge;
146                 }
147                 else if(getDistanceToDepot(representativeEdge, shortestPaths) < getDistanceToDepot(edge, shortestPaths)) {
148                     representativeEdge = edge;
149                 }
150             }
151             representativeEdges.add(representativeEdge);
152         }
153         else {
154             Edge representativeEdge = null;
155             for(Edge edge : requiredEdges) {
156                 if(!representativeEdges.contains(edge) && representativeEdge == null) {
157                     representativeEdge = edge;
158                 }
159                 else if(!representativeEdges.contains(edge) && getDistance(representativeEdge, representativeEdges, shortestPaths)
160                         < getDistance(edge, representativeEdges, shortestPaths)) {
161                     representativeEdge = edge;
162                 }
163             }
164             representativeEdges.add(representativeEdge);
165         }
166     }
167     requiredEdges.removeAll(representativeEdges);

```

Codeausschnitt 6: getDayRoutes-Methode – Bestimmung der repräsentativen Kanten

```

168  /*
169   * Make clusters with required Edges
170   */
171  HashMap<Edge, HashSet<Edge>> clusters = new HashMap<Edge, HashSet<Edge>>();
172
173  for(Edge representativeEdge : representativeEdges) {
174      HashSet<Edge> cluster = new HashSet<Edge>();
175      cluster.add(representativeEdge);
176      clusters.put(representativeEdge, cluster);
177  }
178
179  for(Edge requiredEdge : requiredEdges) {
180      Edge chosenRepresentativ = null;
181
182      for(Edge edge : representativeEdges) {
183          if(chosenRepresentativ == null) {
184              chosenRepresentativ = edge;
185          }
186          else if(getDistance(chosenRepresentativ, requiredEdge, shortestPaths) > getDistance(edge, requiredEdge, shortestPaths)) {
187              chosenRepresentativ = edge;
188          }
189          else if (getDistance(chosenRepresentativ, requiredEdge, shortestPaths) == getDistance(edge, requiredEdge, shortestPaths)) {
190              if(clusters.get(edge).size() < clusters.get(chosenRepresentativ).size()) {
191                  chosenRepresentativ = edge;
192              }
193          }
194      }
195
196      clusters.get(chosenRepresentativ).add(requiredEdge);
197  }
198
199 /**
200  * Supplement clusters with edges on shortest paths to required edges;
201 */
203 ArrayList<HashSet<Edge>> completeClusters = new ArrayList<HashSet<Edge>>();
204
205 for(Edge representativeEdge : representativeEdges) {
206     HashSet<Edge> cluster = clusters.get(representativeEdge);
207     HashSet<Edge> clusterRequiredEdges = new HashSet<Edge>();
208     clusterRequiredEdges.addAll(cluster);
209
210     for(Edge requiredEdge : clusterRequiredEdges) {
211         cluster.addAll(shortestPaths.get(0).get(requiredEdge.getA()).getEdgeSet());
212         cluster.addAll(shortestPaths.get(0).get(requiredEdge.getB()).getEdgeSet());
213     }
214
215     completeClusters.add(cluster);
216 }
217

```

Codeausschnitt 7: getDayRoutes-Methode – Erstellung der Cluster und fülle sie mit den kritischen Kanten und den Kanten auf den kürzesten Pfaden, durch dies Kanten

```

192     /*
193      * Find critical vertices
194      * (vertices with an uneven degree)
195      */
196  ArrayList<Integer> criticalVertices = new ArrayList<Integer>();
197
198  for(Integer v : vertexSet) {
199      if(adjacencyList.get(v).size() % 2 != 0) {
200          criticalVertices.add(v);
201      }
202  }

```

Codeausschnitt 8: getPostmanTour-Methode – Findung aller kritischen Knoten

```

204  /*
205   * Extend graph to make an eulerian Graph
206   */
207  if(!criticalVertices.isEmpty()) {
208      /*
209       * Calculate edges for matching graph
210       */
211      HashMap<Integer, HashMap<Integer, Path>> shortestPaths = getShortestPaths();
212
213      ArrayList<Edge> matchingEdges = new ArrayList<Edge>();
214      HashMap<Edge, Path> edgePathMap = new HashMap<Edge, Path>();
215
216      for(int i1 = 0; i1 < criticalVertices.size(); i1++) {
217          for(Integer i2 = i1 + 1; i2 < criticalVertices.size(); i2++) {
218              int v1 = criticalVertices.get(i1);
219              int v2 = criticalVertices.get(i2);
220
221              Edge matchingEdge = new Edge(v1, v2, shortestPaths.get(v1).get(v2).getCost());
222
223              matchingEdges.add(matchingEdge);
224              edgePathMap.put(matchingEdge, shortestPaths.get(v1).get(v2));
225          }
226      }
227
228      /*
229       * Calculate minimal cost perfect matching
230       */
231      Graph matchingGraph = new Graph(matchingEdges);
232      HashSet<Edge> perfectMatching = matchingGraph.getMinimalCostPerfectMatching();
233
234      /*
235       * Supplement graph with edges on paths corresponding to matching edges in the matching.
236       */
237      for(Edge matchingEdge : perfectMatching) {
238          for(Edge pathEdge : edgePathMap.get(matchingEdge).getEdgeSet()) {
239              Edge supplementEdge = new Edge(pathEdge.getA(), pathEdge.getB(), pathEdge.getCost());
240              edgeSet.add(supplementEdge);
241              adjacencyList.get(supplementEdge.getA()).add(supplementEdge);
242              adjacencyList.get(supplementEdge.getB()).add(supplementEdge);
243              edgeCount++;
244          }
245      }
246  }

```

Codeausschnitt 9: getPostmanTour-Methode – Erweiterung des Graph, sodass er eulersch wird

```

248     /*
249      * Calculate euler Tour
250     */
251     HashSet<Edge> coveredEdges = new HashSet<Edge>();
252
253     Path eulerTour = null;
254
255     while(!coveredEdges.containsAll(edgeSet)) {
256         if(eulerTour == null) {
257             int base = 0;
258             int currentVertex = base;
259
260             Path subCycle = new Path(currentVertex);
261
262             do {
263                 for(Edge edge : adjacencyList.get(currentVertex)){
264                     if(!coveredEdges.contains(edge)) {
265                         subCycle.append(edge);
266                         coveredEdges.add(edge);
267                         currentVertex = edge.getOppositVertex(currentVertex);
268                         break;
269                     }
270                 }
271             }
272             while(currentVertex != base);
273
274             eulerTour = subCycle;
275         }
276         else {
277             int base = -1;
278
279             for(Integer vertex : eulerTour.getVertexSet()) {
280                 for(Edge edge : adjacencyList.get(vertex)) {
281                     if(!coveredEdges.contains(edge)) {
282                         base = vertex;
283                     }
284                 }
285             }
286
287             int currentVertex = base;
288
289             Path subCycle = new Path(base);
290
291             do {
292                 for(Edge edge : adjacencyList.get(currentVertex)){
293                     if(!coveredEdges.contains(edge)) {
294                         subCycle.append(edge);
295                         coveredEdges.add(edge);
296                         currentVertex = edge.getOppositVertex(currentVertex);
297                         break;
298                     }
299                 }
300             }
301             while(currentVertex != base);
302
303             eulerTour.insertCycle(subCycle);
304         }
305     }
306
307     return eulerTour;
308 }

```

Codeausschnitt 10: getPostmanTour-Methode – Bestimmung des Eulerkreis

```
45④ public void insertCycle(Path path) {  
46     vertexSet.addAll(path.getVertexSet());  
47     edgeSet.addAll(path.getEdgeSet());  
48  
49     int insertionPoint = -1;  
50  
51     for(int n = 0; n < vertexList.size(); n++) {  
52         if(vertexList.get(n) == path.getStartVertex()) {  
53             insertionPoint = n;  
54         }  
55     }  
56  
57     vertexList.remove(insertionPoint);  
58     vertexList.addAll(insertionPoint, path.getVertexList());  
59     edgeList.addAll(insertionPoint, path.getEdgeList());  
60  
61 }  
62 }
```

Codeausschnitt 11: insertCycle-Methode

```

public HashSet<Edge> getMinimalCostPerfectMatching(){
/*
 * This method is based on Galil's paper, released March 1986,
 * "Efficient Algorithms for Finding Maximum Matching in Graphs"
 * its recommended to read said paper before reading this method.
 */

/*
 * Invert edge cost
 */
int highestCost = -1;

for(Edge edge : edgeSet) {
    if(highestCost == -1) {
        highestCost = edge.getCost();
    }
    else if(highestCost < edge.getCost()) {
        highestCost = edge.getCost();
    }
}
highestCost++;

for(Edge edge : edgeSet) {
    edge.setCost(highestCost - edge.getCost());
}

HashSet<Edge> matchingEdges = new HashSet<Edge>();

HashMap<Integer, Double> dualVar = new HashMap<Integer, Double>();

/*
 * Assign dual variable to v (u(v))
 */
for(Integer v : vertexSet) {
    double maxEdgeCost = 0;
    boolean maxEdgeCostFound = false;
    for(Edge edge : adjacencyList.get(v)) {
        if(!maxEdgeCostFound) {
            maxEdgeCost = edge.getCost();
            maxEdgeCostFound = true;
        }
        else if(edge.getCost() > maxEdgeCost) {
            maxEdgeCost = edge.getCost();
        }
    }
    dualVar.put(v, maxEdgeCost / 2.0);
}

```

Codeausschnitt 12: getMinimalCostPerfectMatching-Methode – Bestimmung der anfängliche Dual Variable

```

360     /*
361      * Create initial surface graph and link it to the base graph
362      * (make a blossom for each vertex containing only this vertex)
363      * (blossoms like this (containing only one vertex) are referred to as trivial)
364      */
365     HashSet<Blossom> surfaceGraph = new HashSet<Blossom>();
366     HashMap<Integer, Blossom> associationMap = new HashMap<Integer, Blossom>();
367
368     for(Integer v : vertexSet) {
369         Blossom b = new Blossom(v, adjacencyList.get(v));
370         surfaceGraph.add(b);
371         associationMap.put(v, b);
372     }
373
374     while(true) {
375         /*
376          * Reset labels
377          */
378         for(Blossom b : surfaceGraph) {
379             b.resetLabel();
380         }
381
382         /*
383          * Find all single blossoms
384          * Label them by 'S' and insert them into Q
385          */
386         Queue<Blossom> q = new LinkedList<Blossom>();
387
388         for(Blossom b : surfaceGraph) {
389             boolean isMatched = false;
390
391             for(Edge outerEdge : b.getOuterEdges()) {
392                 if(matchingEdges.contains(outerEdge)) {
393                     isMatched = true;
394                     break;
395                 }
396             }
397
398             if(!isMatched) {
399                 b.setLabel('S');
400                 q.offer(b);
401             }
402         }
403     }

```

Codeausschnitt 13: getMinimalCostPerfectMatching-Methode – Erstellung des SurfaceGraph. Start der Iteration. Lösung der Label. Labelung alle Surface Blossoms

```

404     boolean hasBeenAugmented = false;
405
406     while(!hasBeenAugmented) {
407         while(!q.isEmpty() && !hasBeenAugmented) {
408             Blossom b = q.poll();
409             if(b.onSurface()) {
410                 for(Edge edge : b.getOuterEdges()) {
411                     if(getSlack(edge, dualVar) == 0) {
412                         Blossom d = associationMap.get(b.getOppositVertex(edge));
413
414                         if(d.getLabel() == '/') {
415                             /*
416                             * Case C1
417                             * Apply R12
418                             */
419                             d.setLabel('T', b);
420
421                             Edge matchingEdge = null;
422                             for(Edge outerEdge : d.getOuterEdges()) {
423                                 if(matchingEdges.contains(outerEdge)) {
424                                     matchingEdge = outerEdge;
425                                     break;
426                                 }
427                             }
428
429                             Blossom dSpouse = associationMap.get(d.getOppositVertex(matchingEdge));
430
431                             int debug = d.getOppositVertex(matchingEdge);
432
433                             dSpouse.setLabel('S', d);
434                             q.offer(dSpouse);
435                         }
436                         ....
437                     }
438                 }
439             }
440         }
441     }

```

Codeausschnitt 14: getMinimalCostPerfectMatching-Methode – Iteration über den Surface Graphen.
Ausführung von R12, tritt C1 ein

```

436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779

```

Codeausschnitt 15: getMinimalCostPerfectMatching-Methode – Konstruktion des Pfades zwischen den beiden S-Blossoms. Augmentierung des Pfades zwischen den beiden S-Blossoms

```

495
496
497     else {
498         /*
499          * Uneven alternating cycle has been discovered!
500          * Make a new blossom
501         */
502
503         /*
504          * Cut off blossom stem
505         */
506         Blossom labelOrigin = null;
507         Blossom base = null;
508
509         while(pathBlossoms.get(0) == pathBlossoms.get(pathBlossoms.size() - 1)) {
510             labelOrigin = base;
511             base = pathBlossoms.get(0);
512
513             pathBlossoms.remove(pathBlossoms.size() - 1);
514             pathBlossoms.remove(0);
515         }
516
517         /*
518          * Make the new blossom and label it by 'S'
519         */
520         Blossom newBlossom = new Blossom(base, pathBlossoms);
521
522         newBlossom.setLabel('S', labelOrigin);
523         q.offer(newBlossom);
524
525         surfaceGraph.removeAll(newBlossom.getSubblossoms());
526         surfaceGraph.add(newBlossom);
527
528         for(Integer vertex : newBlossom.getVertices()) {
529             associationMap.put(vertex, newBlossom);
530         }
531     }
532 }
533 }
534 }
535 }
536 }
```

Codeausschnitt 16: getMinimalCostPerfectMatching-Methode – Konstruktion einer neuen Blossom. Labelung der Blossom als S-Blossom

```

537         if(!hasBeenAugmented) {
538             /*
539             * Calculate Delta Values
540             */
541
542             /*
543             * Delta 1 (minimum dual variable of a S-Vertex)
544             */
545             boolean delta1Found = false;
546             double delta1 = -1;
547
548             for(Integer vertex : vertexSet) {
549                 if(associationMap.get(vertex).getLabel() == 'S') {
550                     double vertexDualVar = dualVar.get(vertex);
551
552                     if(!delta1Found) {
553                         delta1 = vertexDualVar;
554                         delta1Found = true;
555                     }
556                     else if(vertexDualVar < delta1){
557                         delta1 = vertexDualVar;
558                     }
559                 }
560             }
561
562             /*
563             * Delta 2 (minimum slack on an edge between an S-Vertex and an unlabeled vertex)
564             */
565             boolean delta2Found = false;
566             double delta2 = -1;
567             HashSet<Blossom> blossomsToRevisitD2 = new HashSet<Blossom>();
568
569             for(Integer vertex : vertexSet) {
570                 if(associationMap.get(vertex).getLabel() == 'S') {
571                     for(Edge edge : adjacencyList.get(vertex)) {
572                         if(associationMap.get(edge.getOppositVertex(vertex)).getLabel() == '/') {
573                             if(!delta2Found) {
574                                 delta2 = getSlack(edge, dualVar);
575                                 blossomsToRevisitD2.add(associationMap.get(vertex));
576                                 delta2Found = true;
577                             }
578                             else if(getSlack(edge, dualVar) == delta2){
579                                 blossomsToRevisitD2.add(associationMap.get(vertex));
580                             }
581                             else if (getSlack(edge, dualVar) < delta2) {
582                                 blossomsToRevisitD2 = new HashSet<Blossom>();
583                                 blossomsToRevisitD2.add(associationMap.get(vertex));
584                                 delta2 = getSlack(edge, dualVar);
585                             }
586                         }
587                     }
588                 }
589             }
590         }

```

Codeausschnitt 17: getMinimalCostPerfectMatching-Methode – Bestimmung von Delta eins und zwei

```

591
592     /*
593      * Delta 3 (minimum slack on an edge between two surface S-Blossoms)
594      */
595     boolean delta3Found = false;
596     double delta3 = -1;
597     HashSet<Blossom> blossomsToRevisitD3 = new HashSet<Blossom>();
598
599     for(Blossom blossom : surfaceGraph) {
600         if(blossom.getLabel() == 'S') {
601             for(Edge edge : blossom.getOuterEdges()) {
602                 if(associationMap.get(blossom.getOppositVertex(edge)).getLabel() == 'S') {
603                     if(!delta3Found) {
604                         delta3 = getSlack(edge, dualVar) / 2.0;
605                         blossomsToRevisitD3.add(blossom);
606                         delta3Found = true;
607                     }
608                     else if(getSlack(edge, dualVar) / 2.0 == delta3){
609                         blossomsToRevisitD3.add(blossom);
610                     }
611                     else if (getSlack(edge, dualVar) / 2.0 < delta3) {
612                         delta3 = getSlack(edge, dualVar) / 2.0;
613                         blossomsToRevisitD3 = new HashSet<Blossom>();
614                         blossomsToRevisitD3.add(blossom);
615                     }
616                 }
617             }
618         }
619     }
620
621     /*
622      * Delta 4 (minimum dual variable of a (nontrivial) T-Blossom)
623      */
624     boolean delta4Found = false;
625     double delta4 = -1;
626     HashSet<Blossom> blossomsToRevisitD4 = new HashSet<Blossom>();
627
628     for(Blossom blossom : surfaceGraph) {
629         if(!blossom.getSubblossoms().isEmpty() && blossom.getLabel() == 'T') {
630             if(!delta4Found) {
631                 delta4 = blossom.getDualVar();
632                 blossomsToRevisitD4.add(blossom);
633                 delta4Found = true;
634             }
635             else if (blossom.getDualVar() == delta4) {
636                 blossomsToRevisitD4.add(blossom);
637             }
638             else if (blossom.getDualVar() < delta4) {
639                 delta4 = blossom.getDualVar();
640                 blossomsToRevisitD4 = new HashSet<Blossom>();
641                 blossomsToRevisitD4.add(blossom);
642             }
643         }
644     }

```

Codeausschnitt 18: getMinimalCostPerfectMatching-Methode – Bestimmung von Delta drei und vier

```

649      */
650      double delta = -1;
651
652      if((!delta2Found || (delta2Found && delta1 < delta2)) && (!delta3Found || (delta3Found && delta1 < delta3))
653          && (!delta4Found || (delta4Found && delta1 < delta4))) {
654          /*
655          * Delta 1
656          */
657          System.out.println("Delta 1 chosen");
658          HashSet<Integer> matchedVertices = new HashSet<Integer>();
659          for(Edge matchingEdge : matchingEdges) {
660              matchedVertices.add(matchingEdge.getA());
661              matchedVertices.add(matchingEdge.getB());
662          }
663
664          for(Integer v : vertexSet) {
665              if(!matchedVertices.contains(v)) {
666                  for(Integer w : vertexSet) {
667                      if(!matchedVertices.contains(w) && v != w) {
668                          matchingEdges.add(adjacencyMap.get(v).get(w));
669                          matchedVertices.add(w);
670                          matchedVertices.add(v);
671                      }
672                  }
673              }
674          }
675
676          break;
677      }
678      if(delta2Found && delta2 <= delta1 && (!delta3Found || (delta3Found && delta2 <= delta3))
679          && (!delta4Found || (delta4Found && delta2 <= delta4))) {
680          /*
681          * Delta 2
682          */
683          System.out.println("Delta 2 chosen");
684          delta = delta2;
685          for(Blossom blossom : blossomsToRevisitD2) {
686              q.offer(blossom);
687          }
688      }
689      else if(delta3Found && delta3 <= delta1 && (!delta2Found || (delta2Found && delta3 <= delta2))
690          && !delta4Found || (delta4Found && delta3 <= delta4)) {
691          /*
692          * Delta 3
693          */
694          System.out.println("Delta 3 chosen");
695          delta = delta3;
696          for(Blossom blossom : blossomsToRevisitD3) {
697              q.offer(blossom);
698          }
699      }
700      else if (delta4Found && delta4 <= delta1 && (!delta2Found || (delta2Found && delta4 <= delta2))
701          && (!delta3Found || (delta3Found && delta4 <= delta3))) {
702          /*
703          * Delta 4
704          */
705          System.out.println("Delta 4 chosen");
706          delta = delta4;
707
708          for(Blossom blossom : blossomsToRevisitD4) {
709              expandTBlossom(blossom, q, matchingEdges, surfaceGraph, associationMap);
710          }
711      }

```

Codeausschnitt 19: getMinimalCostPerfectMatching-Methode – Bestimmung des geringsten Delta Werts.

```

713     for(Integer vertex : vertexSet) {
714         double dual = dualVar.get(vertex);
715
716         if(associationMap.get(vertex).getLabel() == 'S') {
717             dual -= delta;
718         }
719         else if (associationMap.get(vertex).getLabel() == 'T') {
720             dual += delta;
721         }
722
723         dualVar.put(vertex, dual);
724     }
725
726     for(Blossom blossom : surfaceGraph) {
727         if(!blossom.getSubblossoms().isEmpty()) {
728             double dual = blossom.getDualVar();
729
730             if(blossom.getLabel() == 'S') {
731                 dual += 2.0 * delta;
732             }
733             else if(blossom.getLabel() == 'T') {
734                 dual -= 2.0 * delta;
735             }
736
737             blossom.setDualVar(dual);
738         }
739     }
740 }
741
742 if(!hasBeenAugmented) {
743     break;
744 }
745
746 HashSet<Blossom> newSurfaceGraph = new HashSet<Blossom>();
747 newSurfaceGraph.addAll(surfaceGraph);
748 for(Blossom blossom : surfaceGraph) {
749     if(!blossom.getSubblossoms().isEmpty() && blossom.getLabel() == 'S' && blossom.getDualVar() == 0) {
750         expandSBlossom(blossom, newSurfaceGraph, associationMap);
751     }
752 }
753 surfaceGraph = newSurfaceGraph;
754 }
755 return matchingEdges;
756 }

```

Codeausschnitt 20: getMinimalCostPerfectMatching-Methode – Veränderung der Dual Variablen

```

762 public void expandTBlossom(Blossom blossom, Queue<Blossom> q, HashSet<Edge> matchingEdges,
763                                     HashSet<Blossom> surfaceGraph, HashMap<Integer, Blossom> associationMap) {
764     /*
765      * Label the inner path
766      */
767     Edge labelOriginEdge = blossom.getEdgeTo(blossom.getLabelOrigin(), matchingEdges);
768
769     Blossom newPathStart = blossom.getConnectedSubblossom(labelOriginEdge);
770     ArrayList<Blossom> innerPath = new ArrayList<Blossom>();
771     innerPath.addAll(blossom.getInnerPath());
772
773     /*
774      * Determine which way to go around the blossom
775      */
776     int newPathStartIndex = innerPath.indexOf(newPathStart);
777
778     int direction;
779     int endpoint;
780
781     if(newPathStartIndex % 2 == 0) {
782         /*
783          * Move right
784          */
785         direction = 1;
786         innerPath.add(blossom.getBase());
787         endpoint = innerPath.size() - 1;
788     }
789     else {
790         /*
791          * Move left
792          */
793         direction = -1;
794         innerPath.add(0, blossom.getBase());
795         endpoint = 0;
796     }
797
798     /*
799      * Label new path
800      */
801     newPathStart.setLabel('T', blossom.getLabelOrigin());
802
803     for(int i = innerPath.indexOf(newPathStart); i != endpoint; i += direction) {
804         Blossom curr = innerPath.get(i);
805         Blossom next = innerPath.get(i + direction);
806
807         if(curr.getLabel() == 'T') {
808             next.setLabel('S', curr);
809             q.offer(next);
810         }
811         else if(curr.getLabel() == 'S') {
812             next.setLabel('T', curr);
813         }
814     }
815
816     Edge matchingEdge = null;
817     for(Edge outerEdge : blossom.getOuterEdges()) {
818         if(matchingEdges.contains(outerEdge)) {
819             matchingEdge = outerEdge;
820             break;
821         }
822     }

```

Codeausschnitt 21: expandTBlossom-Methode – Bestimmung der Richtung, zum Durchlaufen der Blossom. Relabelung des inneren Pfades

```

820     Blossom spouse = associationMap.get(blossom.getOppositVertex(matchingEdge));
821
822     spouse.setLabel('S', blossom.getBase());
823
824     surfaceGraph.remove(blossom);
825     for(Blossom subblossom : blossom.getSubblossoms()) {
826         for(Integer vertex : subblossom.getVertices()) {
827             associationMap.put(vertex, subblossom);
828         }
829         surfaceGraph.add(subblossom);
830         subblossom.setOnSurface(true);
831         subblossom.setParent(null);
832     }
833 }
```

Codeausschnitt 22: expandTBlossom-Methode – Auflösung der Blossom

```

835⊕ public void expandSBlossom(Blossom blossom, HashSet<Blossom> surfaceGraph, HashMap<Integer, Blossom> associationMap) {
836     surfaceGraph.remove(blossom);
837     for(Blossom subblossom : blossom.getSubblossoms()) {
838         if(!subblossom.getSubblossoms().isEmpty() && subblossom.getDualVar() == 0) {
839             expandSBlossom(subblossom, surfaceGraph, associationMap);
840         }
841     } else {
842         for(Integer vertex : subblossom.getVertices()) {
843             associationMap.put(vertex, subblossom);
844         }
845         surfaceGraph.add(subblossom);
846         subblossom.setOnSurface(true);
847         subblossom.setParent(null);
848     }
849 }
850 }
```

Codeausschnitt 23: expandSBlossom-Methode

```

852⊕ private double getSlack(Edge edge, HashMap<Integer, Double> dualVar) {
853     double slack = dualVar.get(edge.getA()) + dualVar.get(edge.getB()) - edge.getCost();
854     return slack;
855 }
```

Codeausschnitt 24: getSlack-Methode

```

9  public class Blossom {
10    private Blossom base;
11    private ArrayList<Blossom> innerPath = new ArrayList<Blossom>();
12
13    private HashSet<Blossom> subblossoms = new HashSet<Blossom>();
14    private HashSet<Integer> vertices = new HashSet<Integer>();
15    private HashSet<Edge> outerEdges = new HashSet<Edge>();
16
17    private boolean onSurface = true;
18    private Blossom parent = null;
19
20    private char label = '/';
21    private Blossom labelOrigin = null;
22
23    private double dualVar = 0;
24
25    public Blossom(int vertex, HashSet<Edge> edges) {
26      vertices.add(vertex);
27      outerEdges.addAll(edges);
28    }
29
30    public Blossom(Blossom base, ArrayList<Blossom> innerPath) {
31      this.base = base;
32      this.innerPath = innerPath;
33
34      subblossoms.add(base);
35      subblossoms.addAll(innerPath);
36
37      HashSet<Edge> oldOuterEdges = new HashSet<Edge>();
38
39      for(Blossom subblossom : subblossoms) {
40        vertices.addAll(subblossom.getVertices());
41        oldOuterEdges.addAll(subblossom.getOuterEdges());
42
43        subblossom.setOnSurface(false);
44        subblossom.setParent(this);
45      }
46
47      for(Edge oldOuterEdge : oldOuterEdges) {
48        if(vertices.contains(oldOuterEdge.getA()) && vertices.contains(oldOuterEdge.getB())) {
49          continue;
50        }
51        else {
52          outerEdges.add(oldOuterEdge);
53        }
54      }
55    }

```

Codeausschnitt 25: Blossom-Klasse

```

57④ public void augment(Edge matchingEdge, HashSet<Edge> matchingEdges) {
58     Blossom newBase = getConnectedSubblossom(matchingEdge);
59     if(!subblossoms.isEmpty() && newBase != base) {
60
61         /*
62          * Determine which way to go around the blossom
63          */
64         int newBaseIndex = innerPath.indexOf(newBase);
65
66         int startingPoint;
67         int direction;
68
69         if(newBaseIndex % 2 == 0) {
70             /*
71              * Right
72              */
73             direction = -1;
74             startingPoint = innerPath.size() - 1;
75         }
76         else {
77             /*
78              * Left
79              */
80             direction = 1;
81             startingPoint = 0;
82         }
83
84         /*
85          * Augment the blossom going from the old base, to the new base using the determined direction
86          */
87         Blossom curr = base;
88         Blossom next = innerPath.get(startingPoint);
89
90         Edge connectingEdge = curr.getEdgeTo(next, matchingEdges);
91
92         if(matchingEdges.contains(connectingEdge)) {
93             matchingEdges.remove(connectingEdge);
94         }
95         else {
96             matchingEdges.add(connectingEdge);
97             curr.augment(connectingEdge, matchingEdges);
98             next.augment(connectingEdge, matchingEdges);
99         }
100
101        for(int i = startingPoint; i != newBaseIndex; i += direction) {
102            curr = innerPath.get(i);
103            next = innerPath.get(i + direction);
104
105            connectingEdge = curr.getEdgeTo(next, matchingEdges);
106
107            if(matchingEdges.contains(connectingEdge)) {
108                matchingEdges.remove(connectingEdge);
109            }
110            else {
111                matchingEdges.add(connectingEdge);
112                curr.augment(connectingEdge, matchingEdges);
113                next.augment(connectingEdge, matchingEdges);
114            }
115        }
116    }

```

Codeausschnitt 26: augment-Methode – Augmentierung des inneren Pfades

```

117         /*
118          * Adjust inner path
119         */
120
121     do {
122         innerPath.add(base);
123         base = innerPath.get(0);
124         innerPath.remove(0);
125
126     }while (base != newBase);
127
128     base.augment(matchingEdge, matchingEdges);
129 }
130 else if(!subblossoms.isEmpty()){
131     base.augment(matchingEdge, matchingEdges);
132 }
133 }
```

Codeausschnitt 27: augment-Methode – Anpassung des inneren Pfades

```

166 public Edge getEdgeTo(Blossom b, HashSet<Edge> matchingEdges) {
167     Edge ret = null;
168
169     for(Edge e : b.getOuterEdges()) {
170         if(outerEdges.contains(e)) {
171             ret = e;
172
173             if(matchingEdges.contains(e)) {
174                 return e;
175             }
176         }
177     }
178
179     return ret;
180 }
```

Codeausschnitt 28: getEdgeTo-Methode

```

147 public Blossom getConnectedSubblossom(Edge edge) {
148     if(outerEdges.contains(edge)) {
149         if(subblossoms.isEmpty()) {
150             return this;
151         }
152         else {
153             for(Blossom subblossom : subblossoms) {
154                 if(subblossom.getOuterEdges().contains(edge)) {
155                     return subblossom;
156                 }
157             }
158             return null;
159         }
160     }
161     else {
162         return null;
163     }
164 }
```

Codeausschnitt 29: getConnectedSubblossom-Methode

Quellen

- [1] Wikipedia: Route inspection problem
(https://en.wikipedia.org/wiki/Route_inspection_problem)
- [2] G. N. Frederickson, M. S. Hecht und C. E. Kim „Approximation Algorithms for some routing problems“. *SIAM Journal on Computing*, 7(2):178-193, Mai 1978
- [3] D. Ahr und G. Reinelt „New Heuristics and Lower Bounds for the Min-Max k-Chinese Postman Problem“. *ESA 2002*, Seite 64-74, 2002
- [4] J. Edmonds und E. L. Johnson „Matching, Euler Tours and the Chinese Postman“. *Mathematical Programming*, 5:88-124, 1973
- [5] Z. Galil „Efficient Algorithms for Finding Maximum Matching in Graphs“. *Colloquium on Trees in Algebra and Programming*, Seite 90-113, 1983
- [6] Wikipedia: Algorithmus von Hierholzer
(https://de.wikipedia.org/wiki/Algorithmus_von_Hierholzer)
- [7] Wikipedia: Dijkstra-Algorithmus (<https://de.wikipedia.org/wiki/Dijkstra-Algorithmus>)