Aufgabe 1: Müllabfuhr

Teilnahme-ID: 01048

Bearbeiter/-in dieser Aufgabe:
Lorian Linnertz Arend

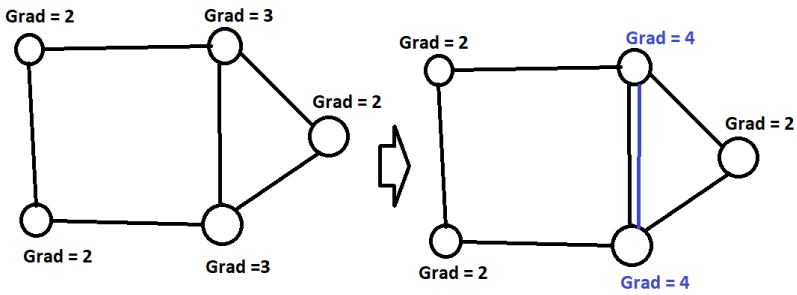
19. April 2022

Inhaltsverzeichnis

Lösungsidee

Einleitung:

Meine Lösungsidee basiert im wesentlichen auf einem “Euler-Circuit”. Ein Euler-Circuit ist ein Weg, welcher alle Straßen durchfährt aber nur einmal. Somit ist dieser Pfad der effektivst Mögliche Weg. Somit ist meine Idee erst einen großen alles um fassenden Weg zu generieren, welcher später dann, in 5 ungefähr gleich große aufgeteilt wird. Das Problem ist jedoch, dass damit der Euler-Circuit möglich muss jede Kreuzung einen Grad(1) haben, welcher gerade ist.



Anpassung der Grade

Dies trifft jedoch nicht immer zu, zum Beispiel haben die Endknoten von Sackgassen einen Grad von 1 was ungerade wäre. Um dieses Problem zu lösen, habe ich mir gedacht, dass man einfach eine zusätzliche Straße zum Knoten hinzufügt, so dass der Knoten ausgeglichen ist. Dies macht man bei allen Knoten mit einem ungeradenem Grad und am Ende hat man einen Graph mit nur graden Grade.

Erstellung des Euler-Circuit

Für die Erstellung des Euler-Circuit, habe ich mir überlegt, den Fleury-Algorithmus zu verwenden. Dieser hat ebenfalls keine Probleme mit den Zusätzlichen Straßen, was von Vorteil ist. Wenn man mit dem Fleury-Algorithmus bei deim Ursprung(TrashCity) anfängt, so wird er am Ende auch wieder automatisch dorthin zurück kommen.

Aufspaltung des Euler-Circuit

Für die Aufspaltung in die einzelnen Tages Routen, folgt man dem Euler-Circuit so lange, bis man die Entfernung erreicht hat und berechnet den Rückweg. Am nächsten Tag kommt der LKW wieder an den gleichen Punkt und beginnt seine Tages-Route. Ebenfalls muss man den Rückweg und den Euler-Circuit auf überlappungen überprüfen

(1) ein Grad ist die Summe aller Strecken, welche mit dem Knoten(Kreuzung) verbunden sind
(2) eine Brücken-Verbindungen hat die Eigenschaft, dass wenn man sie durch trennt,zwei separate Graphen entstehen

Umsetzung

Als erstes habe ich den Graph in Form einer Adjenz-Matrix und einer wie ich sie nenne weight-matrix(eine Adjenzmatrix, nur dass die 1sen durch die Distanz erstetz wurde.

Anpassung der Grade

Für die Anpassung der Grade, fängt man an mit den Knoten mit einem Grad von 1. Diese nimmt man und sucht mit hilfe des Dijkstra Algorithmuses, die nächst gelegenen Knoten mit einem Ungeraden Knoten. Nun nimmt man den Weg und fügt einen zweiten hinzu. Dadurch sind die Grade aller beteiligten Knoten gerade. (Ps: Falls dazwischen noch ein Knoten ist, welcher ein graden Grad hat, so bleibt dieser auch gerade, da ein Weg zu diesem Knoten hinführt und einer Weg, dadurch hat man zwei zusätzliche verbindungen an diesem Knoten, was bedeutet dass er gerade bleibt.) Dies tut man so lange, bis keine Knoten mehr übrig sind.
(Diese geht immer auf , der Beweis wäre: wenn man alle Gerade Verdoppt, so sind sie automatisch auch alle gerade,denn: 2*ungerade = gerade; 2*gerade = gerade)

Erstellung des Euler-Circuit

Für den Euler-Circuit zu erstellen, fängt man beim Ursprung an und folgt den Anweisungen:
Falls es eine Normale-Verbindung gibt
    -nimm ein beliebigen Knoten, welcher mit dem aktuellen Verbunden ist und keine BrückenVerbindung(Cut-Edge)(2) ist

-Falls keine Normale Verbindung mehr übrig ist:
    gehe zu dem Knoten, welcher mit der BrückenVerbindung verbunden ist

-Lösche die Stecke welche du gerade gelaufen bis, falls 2 Stecken da sind lösche eine
-Beginne so lange von vorne, bis keine Knoten mehr übrig sind

Das schwierige ist, die Brücken-Verbindungen aus zu machen, jedoch wenn man sich den Ablauf etwas genauer ansieht, liegen die Brücken-Verbindungen auf dem Rückweg, deswegen habe ich immer die Verbindung welche auf dem Rückwerts-Weg liegt, als Brücken-Verbindung behandelt, die beeinfluss den Algorithmus kaum und ist sicher

Aufspaltung des Euler-Circuit

Um den Euler-Circuit Aufzuspalten, muss man erstmal die Gesamtstrecke wissen, welche man dann durch 5 teilt. Nun folgt man dem Euler-Circuit so lange, bis man einen Weg zurückgelegt hat, welcher ungefähr dem Durchschnittsweg(Gesamtweg/5) entspricht. Anschließend berechnet man mithilfe des Dijkstra Algorithmus, den schnellst Weg zurück zum Ursprung. Dies ist der Weg für den ersten Tag. Anschließend berechnet fährt man am nächsten Tag zu dem Punkt zurück wo man aufgehört hat und wieder holt das ganze.
Ein wichtiges Element jedoch ist, dass man überprüft, dass sich die Wege nicht überlappen, denn dies würde bedeuten, dass man unnötig oft über die gleiche Straße fährt. Überlappen, bedeutet, dass wenn man 1 Schritt weiter geht, auf dem Euler-Circuit, die Gesamt-Länge(Hinfahrt + Weg + Rückfahrt) des Weges gleichbleit
Um dies zu verhindern, überprüft man einfach, ob die Gesamtstrecke gleichbleibt. Falls dies zutrifft, weiß man dass sich die Wege auf dieser Straße überlappen, denn man fährt zwar 1 weiter, der Weg bleibt aber der Gleiche.
Um dies besser zu verdeutlichen, kann man eine Angepasste Version, des Gesamtweges benutzen. Sie lautet (Hinfahrt + Weg + 2* Rückfahrt). Der unterschied ist, dass wenn man nun 1 Schritt weiter geht auf dem Euler-Circuit und dieser Schritt sich ebenfalls auf dem Rückweg befindet. Nun folgt man dem Euler-Path so lange bis der vorherige Angepasste Version des Gesamtweges größer oder gleich dem aktuellen ist oder eine Begrenzung ereicht ist.
Zum Schluss fügt man die drei Komponenten des Weges zusammen und man hat den Weg für die jeweiligen Tag, welche alle zusammen mindesten jede Staße einmal besucht haben.

Beispiele

---MUELLABFUHR0.TXT---

Tag 1: 0 -> 2 -> 1 -> 8 -> 0, Gesamtlaenge: 4
Tag 2: 0 -> 2 -> 3 -> 4 -> 0, Gesamtlaenge: 4
Tag 3: 0 -> 4 -> 5 -> 6 -> 0, Gesamtlaenge: 4
Tag 4: 0 -> 6 -> 7 -> 8 -> 0, Gesamtlaenge: 4
Tag 5: 0 -> 8 -> 9 -> 8 -> 0, Gesamtlaenge: 4
Maximale Lange einer Tagestour: 4.0

---MUELLABFUHR1.TXT---

Tag 1: 0 -> 4 -> 3 -> 6 -> 0, Gesamtlaenge: 15
Tag 2: 0 -> 6 -> 3 -> 1 -> 6 -> 0, Gesamtlaenge: 13
Tag 3: 0 -> 5 -> 4 -> 7 -> 6 -> 0, Gesamtlaenge: 21
Tag 4: 0 -> 6 -> 7 -> 5 -> 3 -> 2 -> 3 -> 6 -> 0, Gesamtlaenge: 23
Tag 5: 0 -> 6 -> 7 -> 6 -> 0, Gesamtlaenge: 4
Maximale Lange einer Tagestour: 23.0

---MUELLABFUHR2.TXT---

Tag 1: 0 -> 5 -> 11 -> 2 -> 8 -> 11 -> 3 -> 11 -> 5 -> 0, Gesamtlaenge: 9
Tag 2: 0 -> 5 -> 11 -> 7 -> 1 -> 12 -> 8 -> 12 -> 9 -> 5 -> 0, Gesamtlaenge: 10
Tag 3: 0 -> 5 -> 14 -> 2 -> 10 -> 4 -> 3 -> 13 -> 1 -> 6 -> 0, Gesamtlaenge: 10
Tag 4: 0 -> 6 -> 0 -> 6 -> 4 -> 13 -> 14 -> 7 -> 8 -> 14 -> 5 -> 0, Gesamtlaenge: 11
Tag 5: 0 -> 5 -> 14 -> 10 -> 9 -> 6 -> 14 -> 13 -> 9 -> 7 -> 9 -> 0, Gesamtlaenge: 11
Maximale Lange einer Tagestour: 11.0

---MUELLABFUHR3.TXT---

Tag 1: 0 -> 1 -> 3 -> 2 -> 1 -> 5 -> 2 -> 4 -> 1 -> 7 -> 2 -> 6 -> 1 -> 9 -> 2 -> 8 -> 1 -> 11 -> 2 -> 10 -> 1 -> 13 -> 0, Gesamtlaenge: 22
Tag 2: 0 -> 13 -> 2 -> 12 -> 1 -> 14 -> 2 -> 0 -> 3 -> 5 -> 4 -> 3 -> 7 -> 4 -> 6 -> 3 -> 9 -> 4 -> 8 -> 3 -> 11 -> 4 -> 10 -> 0, Gesamtlaenge: 23
Tag 3: 0 -> 10 -> 3 -> 13 -> 4 -> 12 -> 3 -> 14 -> 4 -> 0 -> 5 -> 7 -> 6 -> 5 -> 9 -> 6 -> 8 -> 5 -> 11 -> 6 -> 10 -> 5 -> 13 -> 0, Gesamtlaenge: 23
Tag 4: 0 -> 13 -> 6 -> 12 -> 5 -> 14 -> 6 -> 0 -> 7 -> 9 -> 8 -> 7 -> 11 -> 8 -> 10 -> 7 -> 13 -> 8 -> 12 -> 7 -> 14 -> 8 -> 0, Gesamtlaenge: 22
Tag 5: 0 -> 9 -> 11 -> 10 -> 9 -> 13 -> 10 -> 12 -> 9 -> 14 -> 10 -> 0 -> 11 -> 13 -> 12 -> 11 -> 14 -> 12 -> 0 -> 13 -> 14 -> 0, Gesamtlaenge: 21
Maximale Lange einer Tagestour: 23.0

--MUELLABFUHR4.TXT---

Tag 1: 0 -> 1 -> 2 -> 1 -> 0, Gesamtlaenge: 4
Tag 2: 0 -> 1 -> 2 -> 3 -> 4 -> 3 -> 2 -> 1 -> 0, Gesamtlaenge: 8
Tag 3: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 0, Gesamtlaenge: 10
Tag 4: 0 -> 9 -> 8 -> 7 -> 6 -> 7 -> 8 -> 9 -> 0, Gesamtlaenge: 8
Tag 5: 0 -> 9 -> 8 -> 9 -> 0, Gesamtlaenge: 4
Maximale Lange einer Tagestour: 10.0

--MUELLABFUHR5.TXT---

Tag 1: 0 -> 2 -> 1 -> 4 -> 2 -> 5 -> 1 -> 7 -> 2 -> 6 -> 4 -> 5 -> 3 -> 1 -> 9 -> 2 -> 8 -> 1 -> 11 -> 4 -> 6 -> 5 -> 7 -> 3 -> 2 -> 12 -> 1 -> 13 -> 2 -> 16 -> 1 -> 14 -> 4 -> 8 -> 3 -> 4 -> 10 -> 1 -> 19 -> 2 -> 17 -> 1 -> 21 -> 2 -> 18 -> 3 -> 9 -> 4 -> 12 -> 5 -> 8 -> 6 -> 7 -> 8 -> 9 -> 6 -> 10 -> 2 -> 22 -> 4 -> 16 -> 6 -> 12 -> 7 -> 9 -> 10 -> 3 -> 12 -> 9 -> 11 -> 7 -> 10 -> 5 -> 13 -> 3 -> 14 -> 6 -> 13 -> 7 -> 15 -> 3 -> 17 -> 5 -> 14 -> 9 -> 13 -> 10 -> 8 -> 12 -> 13 -> 11 -> 10 -> 12 -> 14 -> 10 -> 16 -> 7 -> 17 -> 6 -> 15 -> 4 -> 20 -> 1 -> 24 -> 2 -> 23 -> 1 -> 25 -> 1 -> 28 -> 2 -> 25 -> 3 -> 19 -> 4 -> 24 -> 3 -> 20 -> 2 -> 27 -> 1 -> 30 -> 4 -> 25 -> 5 -> 18 -> 7 -> 19 -> 5 -> 20 -> 7 -> 21 -> 8 -> 14 -> 13 -> 15 -> 9 -> 15 -> 11 -> 14 -> 16 -> 8 -> 18 -> 10 -> 17 -> 9 -> 16 -> 13 -> 17 -> 12 -> 18 -> 13 -> 19 -> 6 -> 23 -> 3 -> 21 -> 10 -> 19 -> 9 -> 20 -> 8 -> 23 -> 4 -> 27 -> 3 -> 22 -> 7 -> 23 -> 5 -> 22 -> 10 -> 20 -> 11 -> 22 -> 12 -> 19 -> 14 -> 17 -> 16 -> 15 -> 12 -> 21 -> 10 -> 23 -> 9 -> 21 -> 14 -> 18 -> 15 -> 14 -> 22 -> 13 -> 20 -> 14 -> 24 -> 5 -> 26 -> 2 -> 31 -> 1 -> 32 -> 1 -> 33 -> 2 -> 32 -> 4 -> 28 -> 3 -> 5 -> 43 -> 0, Gesamtlaenge: 1465
Tag 2: 0 -> 43 -> 5 -> 3 -> 26 -> 4 -> 34 -> 1 -> 35 -> 3 -> 31 -> 6 -> 24 -> 8 -> 25 -> 6 -> 26 -> 7 -> 25 -> 9 -> 22 -> 16 -> 18 -> 17 -> 19 -> 15 -> 20 -> 16 -> 19 -> 18 -> 20 -> 17 -> 21 -> 15 -> 22 -> 18 -> 21 -> 16 -> 23 -> 11 -> 24 -> 9 -> 26 -> 8 -> 28 -> 5 -> 27 -> 6 -> 28 -> 7 -> 27 -> 9 -> 30 -> 6 -> 29 -> 1 -> 38 -> 2 -> 36 -> 1 -> 41 -> 2 -> 37 -> 2 -> 39 -> 1 -> 42 -> 2 -> 43 -> 1 -> 45 -> 2 -> 44 -> 1 -> 48 -> 3 -> 33 -> 4 -> 36 -> 3 -> 34 -> 6 -> 32 -> 5 -> 29 -> 2 -> 49 -> 1 -> 46 -> 4 -> 37 -> 3 -> 38 -> 4 -> 39 -> 3 -> 40 -> 4 -> 42 -> 5 -> 31 -> 8 -> 29 -> 7 -> 30 -> 8 -> 32 -> 7 -> 31 -> 10 -> 24 -> 12 -> 26 -> 11 -> 25 -> 10 -> 27 -> 11 -> 29 -> 10 -> 28 -> 12 -> 27 -> 13 -> 21 -> 22 -> 19 -> 20 -> 22 -> 23 -> 12 -> 31 -> 11 -> 30 -> 10 -> 32 -> 9 -> 31 -> 14 -> 24 -> 13 -> 23 -> 14 -> 29 -> 15 -> 23 -> 17 -> 22 -> 26 -> 14 -> 32 -> 11 -> 33 -> 5 -> 35 -> 4 -> 44 -> 3 -> 42 -> 8 -> 33 -> 6 -> 35 -> 7 -> 34 -> 8 -> 35 -> 10 -> 34 -> 9 -> 33 -> 12 -> 32 -> 13 -> 25 -> 16 -> 24 -> 15 -> 27 -> 16 -> 26 -> 17 -> 24 -> 18 -> 25 -> 17 -> 27 -> 18 -> 26 -> 19 -> 21 -> 26 -> 20 -> 23 -> 19 -> 25 -> 20 -> 24 -> 23 -> 25 -> 21 -> 28 -> 15 -> 30 -> 12 -> 36 -> 27 -> 0, Gesamtlaenge: 1467
Tag 3: 0 -> 27 -> 36 -> 5 -> 37 -> 6 -> 36 -> 7 -> 37 -> 9 -> 36 -> 8 -> 38 -> 5 -> 39 -> 9 -> 38 -> 6 -> 41 -> 5 -> 40 -> 7 -> 38 -> 10 -> 36 -> 11 -> 34 -> 13 -> 30 -> 14 -> 36 -> 13 -> 31 -> 16 -> 29 -> 17 -> 28 -> 18 -> 29 -> 19 -> 27 -> 21 -> 33 -> 13 -> 35 -> 12 -> 38 -> 11 -> 37 -> 10 -> 39 -> 11 -> 41 -> 8 -> 40 -> 12 -> 39 -> 13 -> 37 -> 12 -> 42 -> 9 -> 41 -> 8 -> 44 -> 6 -> 43 -> 3 -> 47 -> 4 -> 48 -> 5 -> 43 -> 4 -> 49 -> 3 -> 0 -> 5 -> 47 -> 8 -> 45 -> 5 -> 49 -> 6 -> 45 -> 7 -> 42 -> 13 -> 41 -> 12 -> 45 -> 10 -> 43 -> 7 -> 44 -> 9 -> 43 -> 12 -> 46 -> 6 -> 47 -> 10 -> 44 -> 11 -> 42 -> 14 -> 37 -> 15 -> 31 -> 19 -> 28 -> 23 -> 26 -> 24 -> 25 -> 22 -> 29 -> 20 -> 30 -> 17 -> 32 -> 15 -> 34 -> 16 -> 36 -> 15 -> 35 -> 14 -> 40 -> 15 -> 39 -> 16 -> 37 -> 17 -> 33 -> 18 -> 30 -> 19 -> 32 -> 18 -> 31 -> 22 -> 30 -> 21 -> 36 -> 18 -> 35 -> 17 -> 34 -> 20 -> 31 -> 24 -> 27 -> 22 -> 33 -> 19 -> 35 -> 20 -> 32 -> 22 -> 36 -> 19 -> 37 -> 20 -> 33 -> 24 -> 28 -> 25 -> 26 -> 27 -> 23 -> 29 -> 24 -> 30 -> 23 -> 31 -> 27 -> 25 -> 30 -> 26 -> 28 -> 27 -> 29 -> 28 -> 30 -> 27 -> 32 -> 23 -> 35 -> 21 -> 39 -> 17 -> 38 -> 14 -> 43 -> 13 -> 46 -> 7 -> 48 -> 6 -> 0 -> 7 -> 49 -> 8 -> 46 -> 9 -> 47 -> 12 -> 48 -> 43 -> 0, Gesamtlaenge: 1467
Tag 4: 0 -> 43 -> 48 -> 8 -> 0 -> 9 -> 48 -> 10 -> 46 -> 11 -> 45 -> 12 -> 49 -> 10 -> 0 -> 11 -> 47 -> 14 -> 45 -> 17 -> 40 -> 16 -> 38 -> 18 -> 39 -> 19 -> 38 -> 20 -> 39 -> 22 -> 37 -> 21 -> 41 -> 14 -> 48 -> 11 -> 49 -> 13 -> 47 -> 16 -> 41 -> 15 -> 42 -> 19 -> 40 -> 18 -> 41 -> 17 -> 44 -> 18 -> 42 -> 21 -> 43 -> 15 -> 46 -> 14 -> 49 -> 15 -> 47 -> 18 -> 43 -> 16 -> 46 -> 17 -> 47 -> 20 -> 40 -> 21 -> 45 -> 19 -> 41 -> 20 -> 43 -> 19 -> 46 -> 18 -> 48 -> 13 -> 0 -> 15 -> 48 -> 17 -> 0 -> 16 -> 49 -> 18 -> 0 -> 19 -> 47 -> 21 -> 48 -> 19 -> 49 -> 20 -> 46 -> 22 -> 38 -> 23 -> 37 -> 24 -> 32 -> 25 -> 33 -> 29 -> 30 -> 31 -> 28 -> 32 -> 26 -> 31 -> 32 -> 29 -> 31 -> 34 -> 24 -> 35 -> 22 -> 41 -> 23 -> 39 -> 24 -> 41 -> 25 -> 34 -> 26 -> 35 -> 25 -> 36 -> 26 -> 37 -> 25 -> 38 -> 26 -> 39 -> 27 -> 34 -> 28 -> 36 -> 27 -> 35 -> 29 -> 34 -> 30 -> 32 -> 33 -> 30 -> 36 -> 29 -> 37 -> 27 -> 38 -> 29 -> 40 -> 22 -> 42 -> 23 -> 40 -> 25 -> 42 -> 26 -> 40 -> 27 -> 41 -> 26 -> 43 -> 22 -> 45 -> 23 -> 42 -> 29 -> 41 -> 30 -> 37 -> 31 -> 35 -> 32 -> 34 -> 33 -> 31 -> 38 -> 30 -> 40 -> 28 -> 39 -> 31 -> 38 -> 33 -> 35 -> 34 -> 36 -> 31 -> 41 -> 32 -> 37 -> 33 -> 39 -> 34 -> 37 -> 43 -> 0, Gesamtlaenge: 1467
Tag 5: 0 -> 43 -> 37 -> 35 -> 36 -> 42 -> 30 -> 43 -> 23 -> 44 -> 21 -> 49 -> 22 -> 44 -> 24 -> 43 -> 28 -> 45 -> 25 -> 44 -> 26 -> 45 -> 27 -> 42 -> 33 -> 40 -> 31 -> 45 -> 29 -> 43 -> 31 -> 47 -> 24 -> 48 -> 20 -> 0 -> 23 -> 47 -> 29 -> 44 -> 27 -> 46 -> 23 -> 48 -> 26 -> 46 -> 25 -> 47 -> 32 -> 40 -> 34 -> 38 -> 35 -> 40 -> 38 -> 37 -> 41 -> 33 -> 43 -> 32 -> 42 -> 37 -> 43 -> 35 -> 41 -> 34 -> 42 -> 39 -> 38 -> 42 -> 41 -> 39 -> 40 -> 41 -> 43 -> 38 -> 44 -> 30 -> 45 -> 32 -> 44 -> 33 -> 45 -> 34 -> 44 -> 35 -> 45 -> 36 -> 44 -> 37 -> 45 -> 38 -> 48 -> 27 -> 49 -> 24 -> 0 -> 25 -> 49 -> 26 -> 0 -> 27 -> 49 -> 28 -> 46 -> 29 -> 48 -> 28 -> 0 -> 29 -> 49 -> 30 -> 46 -> 31 -> 49 -> 32 -> 46 -> 34 -> 47 -> 33 -> 48 -> 32 -> 0 -> 30 -> 47 -> 37 -> 46 -> 35 -> 48 -> 36 -> 47 -> 40 -> 42 -> 44 -> 39 -> 43 -> 42 -> 46 -> 39 -> 47 -> 42 -> 48 -> 40 -> 45 -> 41 -> 44 -> 43 -> 45 -> 42 -> 49 -> 33 -> 0 -> 34 -> 49 -> 35 -> 0 -> 36 -> 49 -> 37 -> 0 -> 38 -> 49 -> 39 -> 0 -> 40 -> 49 -> 41 -> 46 -> 43 -> 47 -> 45 -> 44 -> 46 -> 45 -> 48 -> 43 -> 48 -> 44 -> 0 -> 41 -> 47 -> 48 -> 46 -> 47 -> 49 -> 43 -> 0 -> 45 -> 49 -> 46 -> 0 -> 48 -> 49 -> 0, Gesamtlaenge: 1473
Maximale Lange einer Tagestour: 1473.0

--MUELLABFUHR6.TXT---

Tag 1: 0 -> 4 -> 10 -> 48 -> 25 -> 30 -> 28 -> 70 -> 28 -> 82 -> 31 -> 28 -> 83 -> 25 -> 39 -> 19 -> 48 -> 25 -> 39 -> 30 -> 33 -> 30 -> 83 -> 33 -> 60 -> 33 -> 83 -> 39 -> 60 -> 44 -> 0 -> 4 -> 93 -> 35 -> 32 -> 81 -> 51 -> 64 -> 63 -> 76 -> 63 -> 80 -> 16 -> 74 -> 92 -> 16 -> 89 -> 99 -> 27 -> 87 -> 8 -> 20 -> 73 -> 88 -> 35 -> 98 -> 58 -> 0, Gesamtlaenge: 594798
Tag 2: 0 -> 58 -> 98 -> 35 -> 88 -> 73 -> 20 -> 8 -> 87 -> 27 -> 99 -> 89 -> 40 -> 66 -> 43 -> 27 -> 43 -> 99 -> 40 -> 99 -> 89 -> 96 -> 99 -> 27 -> 87 -> 27 -> 94 -> 15 -> 7 -> 12 -> 7 -> 50 -> 12 -> 15 -> 50 -> 15 -> 85 -> 12 -> 21 -> 49 -> 1 -> 3 -> 14 -> 68 -> 3 -> 49 -> 3 -> 11 -> 1 -> 49 -> 21 -> 50 -> 91 -> 49 -> 68 -> 37 -> 11 -> 36 -> 14 -> 68 -> 91 -> 50 -> 85 -> 87 -> 8 -> 34 -> 8 -> 20 -> 34 -> 36 -> 55 -> 52 -> 20 -> 73 -> 88 -> 35 -> 98 -> 58 -> 0, Gesamtlaenge: 635804
Tag 3: 0 -> 58 -> 98 -> 35 -> 88 -> 73 -> 52 -> 55 -> 78 -> 37 -> 78 -> 69 -> 38 -> 75 -> 55 -> 97 -> 38 -> 23 -> 69 -> 75 -> 69 -> 97 -> 23 -> 75 -> 78 -> 23 -> 18 -> 77 -> 13 -> 2 -> 5 -> 2 -> 19 -> 84 -> 2 -> 72 -> 13 -> 72 -> 86 -> 5 -> 13 -> 84 -> 77 -> 67 -> 42 -> 45 -> 93 -> 0, Gesamtlaenge: 579988

Tag 4: 0 -> 93 -> 45 -> 42 -> 67 -> 5 -> 42 -> 41 -> 5 -> 77 -> 86 -> 41 -> 42 -> 67 -> 41 -> 86 -> 42 -> 45 -> 10 -> 98 -> 4 -> 44 -> 59 -> 24 -> 22 -> 9 -> 17 -> 22 -> 47 -> 26 -> 9 -> 47 -> 56 -> 17 -> 54 -> 9 -> 56 -> 26 -> 29 -> 53 -> 29 -> 57 -> 53 -> 80 -> 53 -> 95 -> 53 -> 57 -> 79 -> 92 -> 57 -> 26 -> 6 -> 62 -> 54 -> 22 -> 54 -> 62 -> 59 -> 60 -> 44 -> 0, Gesamtlaenge: 573091
Tag 5: 0 -> 44 -> 60 -> 61 -> 24 -> 31 -> 70 -> 82 -> 24 -> 61 -> 60 -> 44 -> 0 -> 58 -> 45 -> 41 -> 90 -> 41 -> 90 -> 88 -> 35 -> 46 -> 32 -> 88 -> 73 -> 88 -> 81 -> 65 -> 51 -> 64 -> 80 -> 65 -> 76 -> 80 -> 65 -> 81 -> 76 -> 81 -> 88 -> 90 -> 32 -> 98 -> 35 -> 46 -> 71 -> 0 -> 58 -> 98 -> 45 -> 93 -> 58 -> 98 -> 93 -> 0, Gesamtlaenge: 487355
Maximale Lange einer Tagestour: 635804.0

--MUELLABFUHR7.TXT---

Tag 1: 0 -> 1 -> 2 -> 19 -> 27 -> 90 -> 79 -> 118 -> 43 -> 113 -> 118 -> 184 -> 65 -> 104 -> 149 -> 65 -> 318 -> 104 -> 438 -> 65 -> 87 -> 165 -> 198 -> 87 -> 438 -> 318 -> 149 -> 104 -> 445 -> 149 -> 198 -> 184 -> 149 -> 460 -> 318 -> 184 -> 318 -> 198 -> 207 -> 43 -> 204 -> 79 -> 207 -> 79 -> 237 -> 207 -> 113 -> 240 -> 79 -> 279 -> 27 -> 90 -> 237 -> 359 -> 90 -> 335 -> 165 -> 346 -> 165 -> 433 -> 87 -> 445 -> 184 -> 207 -> 118 -> 198 -> 433 -> 90 -> 433 -> 346 -> 359 -> 433 -> 465 -> 87 -> 460 -> 438 -> 346 -> 460 -> 465 -> 335 -> 346 -> 465 -> 335 -> 279 -> 90 -> 497 -> 237 -> 279 -> 359 -> 497 -> 279 -> 27 -> 204 -> 118 -> 240 -> 204 -> 207 -> 240 -> 27 -> 19 -> 124 -> 105 -> 413 -> 383 -> 122 -> 124 -> 252 -> 105 -> 491 -> 379 -> 124 -> 272 -> 220 -> 124 -> 485 -> 122 -> 252 -> 272 -> 383 -> 413 -> 272 -> 481 -> 220 -> 252 -> 379 -> 485 -> 252 -> 481 -> 220 -> 19 -> 272 -> 485 -> 379 -> 481 -> 383 -> 491 -> 481 -> 19 -> 383 -> 491 -> 485 -> 19 -> 2 -> 126 -> 68 -> 217 -> 126 -> 174 -> 243 -> 68 -> 218 -> 68 -> 256 -> 14 -> 86 -> 64 -> 261 -> 86 -> 96 -> 261 -> 324 -> 14 -> 256 -> 217 -> 211 -> 2 -> 211 -> 68 -> 421 -> 126 -> 211 -> 243 -> 256 -> 211 -> 421 -> 217 -> 2 -> 218 -> 256 -> 386 -> 64 -> 297 -> 64 -> 330 -> 86 -> 386 -> 86 -> 297 -> 96 -> 330 -> 261 -> 334 -> 64 -> 324 -> 96 -> 429 -> 96 -> 386 -> 126 -> 372 -> 2 -> 372 -> 174 -> 297 -> 174 -> 324 -> 330 -> 334 -> 324 -> 372 -> 243 -> 218 -> 386 -> 243 -> 421 -> 372 -> 297 -> 261 -> 386 -> 297 -> 330 -> 334 -> 386 -> 324 -> 372 -> 386 -> 421 -> 2 -> 1 -> 3 -> 6 -> 3 -> 1 -> 0, Gesamtlaenge: 577428
Tag 2: 0 -> 1 -> 3 -> 6 -> 11 -> 13 -> 45 -> 77 -> 88 -> 77 -> 241 -> 88 -> 479 -> 88 -> 300 -> 155 -> 157 -> 248 -> 155 -> 248 -> 362 -> 248 -> 428 -> 157 -> 241 -> 248 -> 478 -> 185 -> 229 -> 98 -> 185 -> 428 -> 241 -> 351 -> 98 -> 236 -> 229 -> 238 -> 45 -> 179 -> 98 -> 306 -> 179 -> 229 -> 351 -> 236 -> 179 -> 367 -> 98 -> 238 -> 179 -> 367 -> 351 -> 300 -> 236 -> 306 -> 185 -> 478 -> 351 -> 306 -> 238 -> 236 -> 428 -> 362 -> 351 -> 428 -> 478 -> 362 -> 241 -> 478 -> 367 -> 300 -> 241 -> 479 -> 155 -> 355 -> 300 -> 45 -> 238 -> 351 -> 479 -> 355 -> 45 -> 13 -> 99 -> 102 -> 123 -> 15 -> 85 -> 125 -> 129 -> 15 -> 148 -> 85 -> 364 -> 129 -> 99 -> 123 -> 85 -> 472 -> 99 -> 123 -> 129 -> 226 -> 13 -> 99 -> 462 -> 15 -> 365 -> 15 -> 499 -> 85 -> 417 -> 148 -> 201 -> 152 -> 178 -> 199 -> 259 -> 461 -> 145 -> 152 -> 320 -> 145 -> 178 -> 320 -> 199 -> 286 -> 146 -> 190 -> 226 -> 99 -> 472 -> 152 -> 339 -> 201 -> 364 -> 339 -> 472 -> 364 -> 339 -> 492 -> 152 -> 417 -> 148 -> 365 -> 123 -> 148 -> 499 -> 125 -> 365 -> 462 -> 102 -> 190 -> 281 -> 146 -> 264 -> 13 -> 102 -> 365 -> 499 -> 102 -> 226 -> 146 -> 453 -> 190 -> 286 -> 264 -> 13 -> 123 -> 364 -> 417 -> 472 -> 123 -> 462 -> 125 -> 13 -> 129 -> 417 -> 492 -> 178 -> 461 -> 146 -> 281 -> 264 -> 145 -> 320 -> 461 -> 199 -> 264 -> 226 -> 275 -> 190 -> 496 -> 275 -> 453 -> 281 -> 275 -> 461 -> 286 -> 264 -> 259 -> 492 -> 320 -> 286 -> 281 -> 496 -> 281 -> 226 -> 286 -> 496 -> 226 -> 453 -> 461 -> 264 -> 13 -> 226 -> 462 -> 129 -> 499 -> 462 -> 13 -> 11 -> 60 -> 92 -> 110 -> 51 -> 216 -> 23 -> 61 -> 32 -> 69 -> 32 -> 216 -> 69 -> 267 -> 51 -> 328 -> 83 -> 156 -> 61 -> 267 -> 425 -> 32 -> 288 -> 23 -> 267 -> 425 -> 51 -> 288 -> 69 -> 476 -> 69 -> 425 -> 216 -> 328 -> 278 -> 59 -> 31 -> 91 -> 59 -> 278 -> 83 -> 288 -> 156 -> 83 -> 400 -> 59 -> 208 -> 31 -> 76 -> 11 -> 76 -> 60 -> 156 -> 262 -> 92 -> 76 -> 110 -> 288 -> 216 -> 476 -> 425 -> 156 -> 475 -> 60 -> 373 -> 76 -> 208 -> 76 -> 262 -> 475 -> 60 -> 457 -> 76 -> 457 -> 92 -> 475 -> 76 -> 455 -> 101 -> 60 -> 488 -> 373 -> 101 -> 373 -> 208 -> 328 -> 400 -> 208 -> 455 -> 101 -> 11 -> 91 -> 208 -> 457 -> 475 -> 488 -> 455 -> 11 -> 457 -> 488 -> 11 -> 6 -> 3 -> 1 -> 0, Gesamtlaenge: 823767
Tag 3: 0 -> 1 -> 3 -> 6 -> 18 -> 73 -> 180 -> 107 -> 74 -> 392 -> 202 -> 273 -> 332 -> 202 -> 302 -> 332 -> 302 -> 392 -> 273 -> 454 -> 107 -> 192 -> 180 -> 206 -> 337 -> 73 -> 391 -> 109 -> 454 -> 202 -> 107 -> 273 -> 480 -> 180 -> 337 -> 180 -> 440 -> 392 -> 332 -> 454 -> 302 -> 480 -> 192 -> 452 -> 107 -> 332 -> 480 -> 392 -> 454 -> 469 -> 107 -> 440 -> 469 -> 180 -> 442 -> 73 -> 439 -> 206 -> 442 -> 107 -> 480 -> 454 -> 442 -> 192 -> 469 -> 442 -> 439 -> 337 -> 276 -> 180 -> 452 -> 206 -> 276 -> 387 -> 73 -> 458 -> 109 -> 441 -> 387 -> 439 -> 387 -> 18 -> 276 -> 391 -> 337 -> 18 -> 391 -> 441 -> 439 -> 391 -> 458 -> 441 -> 442 -> 452 -> 469 -> 442 -> 18 -> 439 -> 458 -> 441 -> 18 -> 6 -> 81 -> 37 -> 75 -> 8 -> 48 -> 388 -> 84 -> 111 -> 171 -> 84 -> 115 -> 8 -> 53 -> 56 -> 161 -> 296 -> 70 -> 117 -> 106 -> 70 -> 150 -> 117 -> 212 -> 70 -> 296 -> 117 -> 314 -> 70 -> 494 -> 106 -> 117 -> 494 -> 166 -> 168 -> 106 -> 212 -> 296 -> 266 -> 150 -> 314 -> 150 -> 414 -> 150 -> 393 -> 266 -> 314 -> 296 -> 106 -> 486 -> 106 -> 381 -> 168 -> 233 -> 153 -> 53 -> 153 -> 56 -> 432 -> 56 -> 494 -> 212 -> 314 -> 494 -> 381 -> 266 -> 393 -> 381 -> 414 -> 393 -> 166 -> 153 -> 168 -> 265 -> 153 -> 265 -> 415 -> 153 -> 443 -> 166 -> 233 -> 265 -> 443 -> 168 -> 414 -> 486 -> 153 -> 486 -> 153 -> 263 -> 161 -> 432 -> 486 -> 168 -> 494 -> 432 -> 263 -> 53 -> 233 -> 415 -> 443 -> 381 -> 486 -> 212 -> 486 -> 443 -> 53 -> 8 -> 274 -> 341 -> 8 -> 333 -> 8 -> 418 -> 48 -> 341 -> 115 -> 341 -> 176 -> 75 -> 176 -> 37 -> 285 -> 75 -> 333 -> 274 -> 388 -> 111 -> 257 -> 162 -> 84 -> 388 -> 418 -> 341 -> 388 -> 418 -> 338 -> 8 -> 463 -> 8 -> 463 -> 84 -> 487 -> 84 -> 463 -> 115 -> 338 -> 37 -> 406 -> 81 -> 131 -> 81 -> 412 -> 37 -> 159 -> 285 -> 406 -> 131 -> 285 -> 470 -> 131 -> 6 -> 159 -> 333 -> 341 -> 463 -> 338 -> 159 -> 406 -> 187 -> 6 -> 187 -> 203 -> 171 -> 257 -> 171 -> 271 -> 162 -> 111 -> 487 -> 171 -> 162 -> 239 -> 271 -> 187 -> 257 -> 187 -> 412 -> 251 -> 6 -> 251 -> 187 -> 448 -> 162 -> 487 -> 239 -> 257 -> 203 -> 271 -> 251 -> 448 -> 203 -> 271 -> 487 -> 257 -> 448 -> 6 -> 406 -> 412 -> 6 -> 3 -> 1 -> 0, Gesamtlaenge: 725107
Tag 4: 0 -> 1 -> 3 -> 47 -> 336 -> 258 -> 16 -> 305 -> 46 -> 20 -> 46 -> 312 -> 16 -> 459 -> 46 -> 312 -> 197 -> 258 -> 20 -> 336 -> 368 -> 450 -> 47 -> 368 -> 450 -> 116 -> 214 -> 173 -> 255 -> 20 -> 459 -> 214 -> 356 -> 116 -> 255 -> 47 -> 466 -> 197 -> 305 -> 459 -> 258 -> 255 -> 183 -> 3 -> 128 -> 116 -> 371 -> 214 -> 183 -> 128 -> 450 -> 173 -> 128 -> 466 -> 255 -> 197 -> 459 -> 312 -> 258 -> 183 -> 371 -> 356 -> 173 -> 450 -> 255 -> 312 -> 466 -> 255 -> 356 -> 371 -> 255 -> 3 -> 183 -> 450 -> 356 -> 3 -> 1 -> 4 -> 7 -> 9 -> 21 -> 40 -> 130 -> 228 -> 40 -> 344 -> 232 -> 130 -> 228 -> 175 -> 191 -> 95 -> 94 -> 144 -> 9 -> 22 -> 21 -> 287 -> 22 -> 205 -> 21 -> 319 -> 22 -> 329 -> 21 -> 456 -> 21 -> 382 -> 130 -> 411 -> 40 -> 416 -> 130 -> 310 -> 175 -> 228 -> 232 -> 175 -> 246 -> 95 -> 144 -> 22 -> 456 -> 287 -> 319 -> 205 -> 94 -> 175 -> 246 -> 120 -> 191 -> 246 -> 310 -> 191 -> 94 -> 246 -> 482 -> 94 -> 310 -> 232 -> 120 -> 482 -> 95 -> 329 -> 94 -> 9 -> 95 -> 482 -> 135 -> 9 -> 135 -> 329 -> 144 -> 319 -> 329 -> 205 -> 9 -> 144 -> 456 -> 329 -> 9 -> 287 -> 411 -> 228 -> 382 -> 287 -> 416 -> 382 -> 344 -> 411 -> 416 -> 344 -> 310 -> 482 -> 9 -> 7 -> 108 -> 42 -> 210 -> 234 -> 97 -> 309 -> 147 -> 97 -> 363 -> 147 -> 353 -> 284 -> 213 -> 284 -> 289 -> 108 -> 242 -> 7 -> 188 -> 42 -> 235 -> 42 -> 294 -> 108 -> 471 -> 242 -> 42 -> 474 -> 7 -> 235 -> 108 -> 474 -> 188 -> 213 -> 301 -> 289 -> 147 -> 378 -> 234 -> 323 -> 97 -> 498 -> 147 -> 363 -> 309 -> 323 -> 363 -> 289 -> 309 -> 498 -> 301 -> 315 -> 284 -> 188 -> 294 -> 210 -> 242 -> 235 -> 471 -> 294 -> 234 -> 378 -> 309 -> 498 -> 315 -> 289 -> 353 -> 289 -> 474 -> 213 -> 315 -> 353 -> 301 -> 284 -> 474 -> 213 -> 353 -> 498 -> 289 -> 474 -> 235 -> 294 -> 242 -> 7 -> 294 -> 474 -> 7 -> 4 -> 17 -> 33 -> 34 -> 39 -> 93 -> 114 -> 221 -> 114 -> 230 -> 222 -> 230 -> 291 -> 114 -> 473 -> 39 -> 186 -> 253 -> 93 -> 186 -> 253 -> 422 -> 186 -> 490 -> 39 -> 253 -> 490 -> 93 -> 221 -> 222 -> 349 -> 182 -> 133 -> 181 -> 182 -> 292 -> 181 -> 291 -> 222 -> 483 -> 114 -> 473 -> 44 -> 34 -> 44 -> 39 -> 345 -> 44 -> 93 -> 291 -> 424 -> 221 -> 473 -> 93 -> 404 -> 39 -> 404 -> 182 -> 44 -> 473 -> 230 -> 349 -> 292 -> 426 -> 181 -> 349 -> 424 -> 444 -> 181 -> 380 -> 34 -> 270 -> 39 -> 430 -> 270 -> 44 -> 490 -> 270 -> 93 -> 422 -> 221 -> 483 -> 291 -> 426 -> 380 -> 133 -> 345 -> 182 -> 380 -> 345 -> 270 -> 404 -> 473 -> 422 -> 473 -> 490 -> 430 -> 34 -> 345 -> 292 -> 444 -> 349 -> 483 -> 444 -> 380 -> 426 -> 444 -> 380 -> 34 -> 33 -> 49 -> 322 -> 139 -> 231 -> 322 -> 268 -> 139 -> 325 -> 231 -> 384 -> 139 -> 357 -> 268 -> 322 -> 325 -> 326 -> 322 -> 384 -> 268 -> 376 -> 71 -> 303 -> 49 -> 399 -> 82 -> 231 -> 399 -> 223 -> 446 -> 71 -> 303 -> 121 -> 33 -> 71 -> 451 -> 82 -> 268 -> 399 -> 322 -> 82 -> 325 -> 357 -> 370 -> 121 -> 376 -> 82 -> 326 -> 384 -> 357 -> 376 -> 139 -> 389 -> 357 -> 82 -> 384 -> 399 -> 82 -> 33 -> 121 -> 451 -> 303 -> 223 -> 446 -> 303 -> 361 -> 451 -> 361 -> 370 -> 325 -> 376 -> 303 -> 33 -> 370 -> 389 -> 370 -> 376 -> 446 -> 33 -> 376 -> 451 -> 33 -> 17 -> 4 -> 1 -> 0, Gesamtlaenge: 624906
Tag 5: 0 -> 1 -> 4 -> 17 -> 38 -> 24 -> 26 -> 24 -> 369 -> 38 -> 26 -> 78 -> 160 -> 307 -> 137 -> 158 -> 55 -> 196 -> 119 -> 55 -> 298 -> 55 -> 385 -> 24 -> 434 -> 24 -> 299 -> 26 -> 408 -> 158 -> 160 -> 396 -> 398 -> 158 -> 307 -> 385 -> 26 -> 437 -> 78 -> 385 -> 196 -> 143 -> 119 -> 369 -> 38 -> 434 -> 119 -> 369 -> 143 -> 298 -> 408 -> 160 -> 437 -> 137 -> 396 -> 437 -> 158 -> 295 -> 137 -> 398 -> 408 -> 396 -> 437 -> 398 -> 307 -> 295 -> 196 -> 434 -> 143 -> 434 -> 295 -> 385 -> 369 -> 299 -> 78 -> 437 -> 408 -> 385 -> 299 -> 17 -> 369 -> 434 -> 17 -> 4 -> 36 -> 170 -> 151 -> 63 -> 189 -> 50 -> 35 -> 134 -> 189 -> 50 -> 177 -> 189 -> 134 -> 193 -> 63 -> 348 -> 134 -> 374 -> 63 -> 375 -> 63 -> 477 -> 134 -> 375 -> 151 -> 170 -> 304 -> 36 -> 260 -> 4 -> 215 -> 36 -> 260 -> 151 -> 193 -> 374 -> 280 -> 189 -> 407 -> 280 -> 348 -> 151 -> 477 -> 189 -> 35 -> 280 -> 477 -> 193 -> 348 -> 374 -> 35 -> 316 -> 177 -> 50 -> 225 -> 4 -> 225 -> 50 -> 250 -> 35 -> 407 -> 316 -> 250 -> 304 -> 215 -> 260 -> 170 -> 375 -> 348 -> 477 -> 375 -> 260 -> 225 -> 304 -> 225 -> 316 -> 250 -> 1 -> 4 -> 427 -> 215 -> 1 -> 10 -> 62 -> 227 -> 62 -> 290 -> 100 -> 377 -> 62 -> 423 -> 254 -> 282 -> 138 -> 290 -> 254 -> 449 -> 58 -> 269 -> 112 -> 227 -> 269 -> 112 -> 331 -> 58 -> 343 -> 112 -> 447 -> 58 -> 195 -> 269 -> 331 -> 227 -> 377 -> 360 -> 62 -> 449 -> 100 -> 360 -> 112 -> 449 -> 195 -> 331 -> 343 -> 195 -> 343 -> 269 -> 447 -> 195 -> 10 -> 138 -> 423 -> 282 -> 277 -> 254 -> 468 -> 138 -> 468 -> 290 -> 360 -> 227 -> 447 -> 331 -> 449 -> 269 -> 360 -> 10 -> 277 -> 290 -> 449 -> 277 -> 423 -> 468 -> 390 -> 343 -> 447 -> 10 -> 282 -> 449 -> 377 -> 10 -> 449 -> 468 -> 10 -> 1 -> 25 -> 89 -> 28 -> 283 -> 141 -> 142 -> 57 -> 154 -> 141 -> 154 -> 142 -> 164 -> 154 -> 340 -> 57 -> 164 -> 350 -> 57 -> 397 -> 57 -> 132 -> 142 -> 350 -> 132 -> 311 -> 132 -> 313 -> 25 -> 200 -> 28 -> 342 -> 67 -> 313 -> 29 -> 405 -> 132 -> 350 -> 141 -> 435 -> 154 -> 350 -> 340 -> 142 -> 397 -> 164 -> 340 -> 311 -> 397 -> 350 -> 405 -> 311 -> 313 -> 132 -> 397 -> 405 -> 313 -> 200 -> 89 -> 327 -> 28 -> 419 -> 200 -> 283 -> 327 -> 67 -> 219 -> 25 -> 219 -> 293 -> 219 -> 342 -> 89 -> 245 -> 200 -> 366 -> 283 -> 366 -> 435 -> 283 -> 245 -> 327 -> 200 -> 419 -> 219 -> 327 -> 245 -> 419 -> 327 -> 342 -> 313 -> 25 -> 245 -> 435 -> 327 -> 25 -> 342 -> 419 -> 25 -> 1 -> 225 -> 427 -> 260 -> 1 -> 304 -> 427 -> 1 -> 0 -> 12 -> 52 -> 5 -> 66 -> 167 -> 52 -> 163 -> 5 -> 167 -> 354 -> 12 -> 194 -> 72 -> 169 -> 72 -> 249 -> 72 -> 358 -> 169 -> 317 -> 5 -> 127 -> 12 -> 317 -> 52 -> 127 -> 163 -> 354 -> 66 -> 352 -> 167 -> 464 -> 66 -> 467 -> 72 -> 409 -> 66 -> 484 -> 167 -> 464 -> 163 -> 354 -> 464 -> 467 -> 484 -> 352 -> 409 -> 169 -> 401 -> 72 -> 495 -> 0 -> 30 -> 209 -> 80 -> 136 -> 172 -> 80 -> 402 -> 209 -> 136 -> 247 -> 136 -> 394 -> 431 -> 80 -> 410 -> 347 -> 410 -> 420 -> 140 -> 103 -> 54 -> 308 -> 41 -> 54 -> 321 -> 103 -> 172 -> 321 -> 103 -> 224 -> 395 -> 41 -> 103 -> 308 -> 41 -> 244 -> 30 -> 209 -> 172 -> 347 -> 420 -> 224 -> 140 -> 308 -> 395 -> 54 -> 493 -> 321 -> 395 -> 140 -> 244 -> 80 -> 436 -> 172 -> 395 -> 244 -> 308 -> 420 -> 395 -> 244 -> 410 -> 436 -> 172 -> 431 -> 209 -> 394 -> 489 -> 136 -> 403 -> 247 -> 209 -> 493 -> 489 -> 403 -> 402 -> 431 -> 403 -> 30 -> 244 -> 420 -> 436 -> 402 -> 30 -> 247 -> 431 -> 436 -> 30 -> 0 -> 127 -> 354 -> 484 -> 464 -> 409 -> 194 -> 169 -> 495 -> 249 -> 358 -> 317 -> 358 -> 401 -> 194 -> 249 -> 409 -> 317 -> 0 -> 194 -> 358 -> 495 -> 401 -> 0 -> 409 -> 467 -> 484 -> 409 -> 495 -> 0, Gesamtlaenge: 482824
Maximale Lange einer Tagestour: 823767.0
---MUELLABFUHR8.TXT---

Tag 1: 0 -> 205 -> 323 -> 554 -> 21 -> 129 -> 574 -> 345 -> 767 -> 518 -> 250 -> 845 -> 188 -> 163 -> 83 -> 2 -> 173 -> 383 -> 151 -> 245 -> 383 -> 542 -> 286 -> 195 -> 831 -> 648 -> 506 -> 252 -> 468 -> 126 -> 291 -> 506 -> 612 -> 105 -> 506 -> 695 -> 105 -> 693 -> 506 -> 969 -> 252 -> 822 -> 13 -> 330 -> 113 -> 133 -> 289 -> 285 -> 114 -> 171 -> 139 -> 5 -> 504 -> 139 -> 576 -> 5 -> 659 -> 116 -> 155 -> 358 -> 10 -> 28 -> 215 -> 10 -> 226 -> 28 -> 358 -> 215 -> 91 -> 689 -> 29 -> 10 -> 254 -> 358 -> 778 -> 5 -> 140 -> 155 -> 536 -> 10 -> 504 -> 254 -> 536 -> 576 -> 254 -> 668 -> 28 -> 29 -> 226 -> 254 -> 792 -> 10 -> 854 -> 116 -> 581 -> 38 -> 531 -> 215 -> 689 -> 38 -> 813 -> 590 -> 334 -> 78 -> 64 -> 265 -> 687 -> 78 -> 199 -> 334 -> 657 -> 38 -> 998 -> 64 -> 22 -> 29 -> 668 -> 226 -> 504 -> 576 -> 668 -> 504 -> 778 -> 139 -> 659 -> 140 -> 551 -> 93 -> 234 -> 114 -> 283 -> 93 -> 386 -> 114 -> 641 -> 93 -> 508 -> 283 -> 386 -> 171 -> 194 -> 299 -> 246 -> 194 -> 311 -> 171 -> 641 -> 283 -> 234 -> 175 -> 65 -> 93 -> 650 -> 65 -> 331 -> 89 -> 659 -> 155 -> 677 -> 139 -> 551 -> 659 -> 677 -> 140 -> 721 -> 89 -> 413 -> 15 -> 553 -> 413 -> 96 -> 65 -> 418 -> 331 -> 175 -> 96 -> 234 -> 386 -> 679 -> 114 -> 866 -> 171 -> 679 -> 311 -> 672 -> 194 -> 353 -> 299 -> 264 -> 246 -> 308 -> 11 -> 246 -> 311 -> 809 -> 679 -> 866 -> 289 -> 667 -> 754 -> 774 -> 133 -> 667 -> 866 -> 641 -> 234 -> 508 -> 331 -> 96 -> 285 -> 234 -> 650 -> 418 -> 413 -> 175 -> 285 -> 508 -> 650 -> 175 -> 418 -> 721 -> 551 -> 677 -> 721 -> 175 -> 674 -> 96 -> 774 -> 667 -> 917 -> 289 -> 580 -> 113 -> 356 -> 13 -> 366 -> 172 -> 208 -> 435 -> 35 -> 102 -> 274 -> 35 -> 242 -> 236 -> 366 -> 242 -> 274 -> 275 -> 264 -> 267 -> 11 -> 264 -> 608 -> 274 -> 735 -> 35 -> 488 -> 102 -> 435 -> 236 -> 488 -> 208 -> 627 -> 35 -> 790 -> 274 -> 567 -> 13 -> 938 -> 113 -> 416 -> 580 -> 133 -> 754 -> 809 -> 866 -> 133 -> 917 -> 754 -> 866 -> 917 -> 774 -> 285 -> 413 -> 500 -> 553 -> 674 -> 413 -> 996 -> 15 -> 709 -> 590 -> 657 -> 813 -> 696 -> 199 -> 513 -> 260 -> 441 -> 63 -> 456 -> 48 -> 480 -> 17 -> 325 -> 480 -> 769 -> 17 -> 661 -> 48 -> 769 -> 232 -> 17 -> 963 -> 325 -> 661 -> 769 -> 325 -> 232 -> 456 -> 287 -> 232 -> 480 -> 963 -> 661 -> 456 -> 441 -> 344 -> 260 -> 739 -> 199 -> 687 -> 260 -> 789 -> 287 -> 867 -> 232 -> 871 -> 48 -> 963 -> 789 -> 344 -> 734 -> 199 -> 797 -> 78 -> 265 -> 797 -> 687 -> 441 -> 789 -> 687 -> 513 -> 739 -> 687 -> 295 -> 78 -> 696 -> 734 -> 334 -> 696 -> 998 -> 657 -> 734 -> 590 -> 714 -> 500 -> 851 -> 709 -> 714 -> 813 -> 734 -> 812 -> 199 -> 295 -> 334 -> 813 -> 998 -> 334 -> 812 -> 295 -> 513 -> 812 -> 233 -> 590 -> 800 -> 709 -> 233 -> 800 -> 864 -> 15 -> 714 -> 851 -> 864 -> 714 -> 996 -> 500 -> 636 -> 15 -> 851 -> 996 -> 553 -> 636 -> 416 -> 674 -> 996 -> 864 -> 233 -> 579 -> 128 -> 468 -> 291 -> 933 -> 128 -> 819 -> 295 -> 739 -> 797 -> 295 -> 734 -> 998 -> 795 -> 581 -> 215 -> 792 -> 226 -> 536 -> 668 -> 792 -> 358 -> 689 -> 531 -> 313 -> 29 -> 792 -> 536 -> 854 -> 689 -> 22 -> 313 -> 116 -> 778 -> 854 -> 313 -> 581 -> 22 -> 49 -> 458 -> 337 -> 282 -> 143 -> 84 -> 204 -> 143 -> 422 -> 204 -> 223 -> 193 -> 237 -> 223 -> 319 -> 170 -> 193 -> 473 -> 223 -> 665 -> 193 -> 455 -> 97 -> 94 -> 169 -> 444 -> 94 -> 486 -> 82 -> 23 -> 433 -> 327 -> 207 -> 486 -> 364 -> 494 -> 82 -> 97 -> 169 -> 748 -> 97 -> 640 -> 94 -> 494 -> 486 -> 947 -> 23 -> 486 -> 968 -> 94 -> 947 -> 82 -> 327 -> 544 -> 429 -> 328 -> 487 -> 309 -> 42 -> 80 -> 183 -> 382 -> 429 -> 436 -

> 373 -> 328 -> 544 -> 521 -> 183 -> 433 -> 639 -> 183 -> 548 -> 80 -> 309 -> 183 -> 804 -> 309 -> 427 -> 42 -> 625 -> 244 -> 324 -> 42 -> 817 -> 80 -> 461 -> 45 -> 110 -> 427 -> 45 -> 548 -> 309 -> 324 -> 45 -> 817 -> 461 -> 110 -> 443 -> 324 -> 80 -> 521 -> 382 -> 505 -> 23 -> 494 -> 929 -> 23 -> 771 -> 364 -> 837 -> 212 -> 170 -> 682 -> 212 -> 319 -> 473 -> 455 -> 381 -> 337 -> 665 -> 282 -> 381 -> 681 -> 337 -> 237 -> 665 -> 422 -> 701 -> 84 -> 223 -> 844 -> 84 -> 282 -> 681 -> 458 -> 381 -> 801 -> 458 -> 652 -> 801 -> 681 -> 665 -> 844 -> 204 -> 701 -> 143 -> 74 -> 84 -> 422 -> 727 -> 46 -> 49 -> 948 -> 46 -> 458 -> 948 -> 652 -> 965 -> 49 -> 798 -> 46 -> 801 -> 798 -> 948 -> 801 -> 727 -> 282 -> 701 -> 727 -> 616 -> 121 -> 138 -> 849 -> 108 -> 138 -> 950 -> 46 -> 965 -> 871 -> 63 -> 789 -> 867 -> 441 -> 532 -> 63 -> 867 -> 769 -> 456 -> 867 -> 871 -> 287 -> 920 -> 63 -> 984 -> 272 -> 81 -> 316 -> 272 -> 532 -> 287 -> 984 -> 344 -> 796 -> 723 -> 344 -> 819 -> 723 -> 920 -> 796 -> 819 -> 579 -> 468 -> 416 -> 938 -> 291 -> 847 -> 356 -> 822 -> 55 -> 366 -> 646 -> 13 -> 971 -> 55 -> 646 -> 955 -> 195 -> 887 -> 542 -> 799 -> 782 -> 619 -> 165 -> 306 -> 398 -> 601 -> 368 -> 467 -> 556 -> 426 -> 934 -> 205 -> 0, Gesamtlaenge: 2736375

Tag 2: 0 -> 205 -> 934 -> 426 -> 556 -> 467 -> 368 -> 601 -> 398 -> 306 -> 165 -> 619 -> 782 -> 799 -> 542 -> 887 -> 195 -> 955 -> 646 -> 330 -> 735 -> 102 -> 608 -> 615 -> 264 -> 672 -> 267 -> 299 -> 11 -> 672 -> 275 -> 615 -> 672 -> 353 -> 246 -> 919 -> 264 -> 308 -> 275 -> 919 -> 267 -> 915 -> 11 -> 919 -> 615 -> 308 -> 672 -> 919 -> 608 -> 735 -> 242 -> 567 -> 356 -> 938 -> 330 -> 822 -> 955 -> 55 -> 956 -> 252 -> 847 -> 971 -> 646 -> 567 -> 366 -> 955 -> 172 -> 488 -> 627 -> 102 -> 410 -> 435 -> 488 -> 790 -> 435 -> 627 -> 172 -> 367 -> 41 -> 410 -> 627 -> 617 -> 141 -> 71 -> 201 -> 141 -> 394 -> 617 -> 167 -> 201 -> 277 -> 71 -> 385 -> 104 -> 277 -> 357 -> 484 -> 71 -> 588 -> 277 -> 385 -> 141 -> 766 -> 167 -> 385 -> 201 -> 588 -> 308 -> 915 -> 357 -> 645 -> 104 -> 307 -> 71 -> 915 -> 484 -> 201 -> 766 -> 277 -> 484 -> 307 -> 277 -> 821 -> 201 -> 617 -> 766 -> 841 -> 41 -> 412 -> 4 -> 137 -> 41 -> 726 -> 367 -> 189 -> 261 -> 88 -> 137 -> 189 -> 400 -> 261 -> 339 -> 59 -> 261 -> 671 -> 379 -> 88 -> 367 -> 208 -> 759 -> 41 -> 700 -> 137 -> 379 -> 189 -> 567 -> 735 -> 435 -> 759 -> 189 -> 700 -> 208 -> 790 -> 627 -> 759 -> 367 -> 379 -> 261 -> 802 -> 400 -> 339 -> 88 -> 700 -> 410 -> 759 -> 700 -> 726 -> 410 -> 841 -> 394 -> 412 -> 137 -> 726 -> 617 -> 841 -> 412 -> 726 -> 759 -> 137 -> 926 -> 103 -> 4 -> 210 -> 117 -> 115 -> 333 -> 4 -> 322 -> 117 -> 333 -> 322 -> 210 -> 736 -> 4 -> 321 -> 103 -> 210 -> 870 -> 115 -> 623 -> 117 -> 870 -> 322 -> 412 -> 870 -> 333 -> 623 -> 595 -> 104 -> 357 -> 928 -> 104 -> 484 -> 645 -> 307 -> 385 -> 595 -> 307 -> 928 -> 595 -> 645 -> 928 -> 770 -> 156 -> 228 -> 72 -> 527 -> 69 -> 156 -> 263 -> 228 -> 253 -> 30 -> 235 -> 365 -> 53 -> 132 -> 214 -> 365 -> 446 -> 466 -> 30 -> 355 -> 290 -> 30 -> 496 -> 158 -> 253 -> 415 -> 30 -> 911 -> 466 -> 214 -> 446 -> 626 -> 132 -> 352 -> 53 -> 523 -> 132 -> 225 -> 403 -> 178 -> 225 -> 453 -> 365 -> 605 -> 53 -> 995 -> 255 -> 231 -> 407 -> 352 -> 523 -> 214 -> 605 -> 352 -> 452 -> 407 -> 472 -> 231 -> 431 -> 452 -> 471 -> 523 -> 605 -> 471 -> 829 -> 153 -> 304 -> 197 -> 153 -> 591 -> 290 -> 235 -> 466 -> 290 -> 415 -> 465 -> 158 -> 290 -> 496 -> 355 -> 415 -> 496 -> 466 -> 523 -> 626 -> 214 -> 953 -> 365 -> 626 -> 453 -> 546 -> 178 -> 240 -> 546 -> 675 -> 178 -> 453 -> 953 -> 546 -> 43 -> 178 -> 635 -> 240 -> 603 -> 178 -> 891 -> 603 -> 635 -> 891 -> 675 -> 225 -> 626 -> 953 -> 635 -> 43 -> 240 -> 891 -> 953 -> 240 -> 675 -> 403 -> 256 -> 643 -> 231 -> 452 -> 761 -> 603 -> 675 -> 43 -> 403 -> 857 -> 256 -> 666 -> 36 -> 255 -> 407 -> 850 -> 231 -> 533 -> 407 -> 995 -> 431 -> 471 -> 995 -> 452 -> 533 -> 431 -> 828 -> 717 -> 431 -> 850 -> 472 -> 533 -> 850 -> 717 -> 533 -> 995 -> 472 -> 57 -> 124 -> 537 -> 57 -> 231 -> 995 -> 850 -> 720 -> 537 -> 717 -> 541 -> 57 -> 720 -> 717 -> 642 -> 75 -> 124 -> 541 -> 720 -> 918 -> 124 -> 642 -> 537 -> 850 -> 828 -> 829 -> 591 -> 622 -> 153 -> 908 -> 304 -> 622 -> 873 -> 168 -> 69 -> 355 -> 465 -> 69 -> 535 -> 72 -> 168 -> 535 -> 158 -> 898 -> 69 -> 228 -> 892 -> 156 -> 898 -> 253 -> 892 -> 898 -> 535 -> 355 -> 911 -> 496 -> 535 -> 760 -> 527 -> 168 -> 898 -> 596 -> 156 -> 954 -> 228 -> 596 -> 263 -> 770 -> 596 -> 892 -> 954 -> 263 -> 903 -> 115 -> 770 -> 839 -> 623 -> 770 -> 954 -> 596 -> 527 -> 873 -> 197 -> 482 -> 100 -> 43 -> 603 -> 857 -> 761 -> 834 -> 256 -> 761 -> 923 -> 256 -> 36 -> 539 -> 57 -> 823 -> 36 -> 834 -> 403 -> 923 -> 666 -> 255 -> 643 -> 539 -> 284 -> 36 -> 923 -> 834 -> 666 -> 643 -> 823 -> 904 -> 284 -> 643 -> 834 -> 857 -> 666 -> 284 -> 942 -> 301 -> 539 -> 942 -> 732 -> 399 -> 127 -> 47 -> 39 -> 127 -> 218 -> 39 -> 945 -> 127 -> 835 -> 981 -> 39 -> 987 -> 127 -> 202 -> 218 -> 47 -> 176 -> 130 -> 248 -> 462 -> 130 -> 575 -> 219 -> 130 -> 848 -> 76 -> 176 -> 218 -> 557 -> 47 -> 964 -> 202 -> 557 -> 495 -> 346 -> 176 -> 219 -> 346 -> 575 -> 462 -> 815 -> 248 -> 848 -> 462 -> 76 -> 346 -> 815 -> 495 -> 575 -> 740 -> 202 -> 597 -> 300 -> 47 -> 981 -> 202 -> 803 -> 557 -> 597 -> 470 -> 507 -> 149 -> 109 -> 61 -> 517 -> 262 -> 510 -> 644 -> 14 -> 162 -> 430 -> 213 -> 14 -> 430 -> 749 -> 14 -> 530 -> 213 -> 644 -> 162 -> 686 -> 25 -> 262 -> 764 -> 470 -> 517 -> 630 -> 470 -> 704 -> 61 -> 630 -> 742 -> 495 -> 740 -> 557 -> 838 -> 202 -> 300 -> 557 -> 964 -> 300 -> 740 -> 742 -> 575 -> 76 -> 230 -> 219 -> 838 -> 300 -> 835 -> 987 -> 216 -> 203 -> 411 -> 469 -> 335 -> 406 -> 450 -> 335 -> 514 -> 561 -> 335 -> 885 -> 411 -> 651 -> 335 -> 937 -> 450 -> 631 -> 288 -> 297 -> 266 -> 257 -> 297 -> 475 -> 266 -> 347 -> 350 -> 266 -> 560 -> 103 -> 733 -> 321 -> 736 -> 103 -> 298 -> 560 -> 733 -> 736 -> 322 -> 369 -> 115 -> 839 -> 903 -> 117 -> 369 -> 333 -> 903 -> 623 -> 369 -> 870 -> 903 -> 369 -> 257 -> 347 -> 475 -> 326 -> 350 -> 475 -> 775 -> 298 -> 736 -> 966 -> 298 -> 540 -> 326 -> 594 -> 266 -> 863 -> 297 -> 390 -> 206 -> 16 -> 270 -> 206 -> 618 -> 16 -> 663 -> 206 -> 977 -> 288 -> 925 -> 478 -> 173 -> 619 -> 165 -> 221 -> 619 -> 637 -> 60 -> 221 -> 875 -> 619 -> 782 -> 245 -> 542 -> 887 -> 542 -> 799 -> 782 -> 619 -> 165 -> 306 -> 398 -> 601 -> 368 -> 467 -> 556 -> 426 -> 934 -> 205 -> 0, Gesamtlaenge: 2803005

Tag 3: 0 -> 205 -> 934 -> 426 -> 556 -> 467 -> 368 -> 601 -> 398 -> 306 -> 165 -> 619 -> 782 -> 799 -> 542 -> 887 -> 195 -> 932 -> 286 -> 887 -> 802 -> 671 -> 400 -> 379 -> 926 -> 321 -> 966 -> 560 -> 926 -> 966 -> 733 -> 926 -> 990 -> 59 -> 400 -> 932 -> 339 -> 671 -> 932 -> 802 -> 339 -> 338 -> 59 -> 932 -> 887 -> 773 -> 195 -> 955 -> 646 -> 956 -> 773 -> 648 -> 612 -> 252 -> 971 -> 822 -> 956 -> 955 -> 971 -> 956 -> 648 -> 695 -> 612 -> 693 -> 893 -> 108 -> 316 -> 391 -> 108 -> 501 -> 81 -> 532 -> 456 -> 871 -> 532 -> 984 -> 723 -> 941 -> 128 -> 936 -> 512 -> 50 -> 86 -> 184 -> 105 -> 969 -> 648 -> 972 -> 95 -> 271 -> 881 -> 957 -> 95 -> 695 -> 972 -> 105 -> 970 -> 126 -> 579 -> 933 -> 126 -> 938 -> 933 -> 468 -> 970 -> 128 -> 763 -> 512 -> 86 -> 564 -> 50 -> 375 -> 81 -> 920 -> 984 -> 81 -> 738 -> 272 -> 375 -> 108 -> 738 -> 375 -> 391 -> 849 -> 121 -> 989 -> 138 -> 927 -> 616 -> 849 -> 585 -> 108 -> 950 -> 798 -> 965 -> 948 -> 950 -> 391 -> 585 -> 316 -> 738 -> 501 -> 375 -> 585 -> 753 -> 50 -> 585 -> 946 -> 50 -> 341 -> 524 -> 86 -> 936 -> 763 -> 941 -> 796 -> 984 -> 941 -> 970 -> 763 -> 184 -> 564 -> 512 -> 753 -> 564 -> 519 -> 869 -> 384 -> 259 -> 249 -> 67 -> 92 -> 152 -> 67 -> 340 -> 92 -> 249 -> 340 -> 152 -> 74 -> 204 -> 874 -> 74 -> 237 -> 844 -> 319 -> 74 -> 701 -> 874 -> 741 -> 616 -> 989 -> 849 -> 858 -> 927 -> 849 -> 946 -> 669 -> 340 -> 807 -> 92 -> 840 -> 67 -> 509 -> 152 -> 874 -> 989 -> 858 -> 741 -> 927 -> 989 -> 741 -> 669 -> 341 -> 564 -> 893 -> 184 -> 693 -> 969 -> 612 -> 972 -> 184 -> 524 -> 512 -> 940 -> 86 -> 970 -> 936 -> 753 -> 940 -> 519 -> 509 -> 249 -> 840 -> 836 -> 336 -> 656 -> 192 -> 387 -> 212 -> 455 -> 837 -> 319 -> 682 -> 837 -> 387 -> 656 -> 464 -> 192 -> 699 -> 332 -> 583 -> 20 -> 905 -> 119 -> 545 -> 515 -> 332 -> 688 -> 387 -> 682 -> 844 -> 473 -> 212 -> 879 -> 82 -> 444 -> 97 -> 968 -> 169 -> 640 -> 82 -> 748 -> 207 -> 444 -> 748 -> 640 -> 444 -> 968 -> 640 -> 947 -> 494 -> 771 -> 929 -> 82 -> 968 -> 748 -> 947 -> 771 -> 879 -> 929 -> 947 -> 879 -> 906 -> 19 -> 951 -> 8 -> 629 -> 373 -> 429 -> 624 -> 397 -> 629 -> 624 -> 683 -> 8 -> 697 -> 397 -> 907 -> 429 -> 487 -> 373 -> 750 -> 187 -> 372 -> 8 -> 750 -> 328 -> 988 -> 373 -> 147 -> 187 -> 852 -> 8 -> 978 -> 629 -> 683 -> 550 -> 397 -> 988 -> 436 -> 487 -> 382 -> 639 -> 521 -> 436 -> 244 -> 779 -> 427 -> 324 -> 110 -> 548 -> 804 -> 110 -> 817 -> 804 -> 461 -> 324 -> 487 -> 639 -> 544 -> 505 -> 907 -> 629 -> 852 -> 147 -> 372 -> 905 -> 320 -> 131 -> 66 -> 320 -> 447 -> 147 -> 599 -> 320 -> 425 -> 147 -> 780 -> 244 -> 511 -> 324 -> 625 -> 443 -> 427 -> 511 -> 443 -> 779 -> 625 -> 511 -> 779 -> 780 -> 279 -> 239 -> 360 -> 279 -> 511 -> 780 -> 360 -> 363 -> 161 -> 52 -> 106 -> 161 -> 499 -> 52 -> 329 -> 239 -> 620 -> 279 -> 592 -> 239 -> 780 -> 620 -> 360 -> 592 -> 762 -> 329 -> 360 -> 698 -> 239 -> 882 -> 131 -> 425 -> 485 -> 20 -> 951 -> 119 -> 886 -> 19 -> 975 -> 464 -> 702 -> 387 -> 719 -> 332 -> 924 -> 336 -> 688 -> 656 -> 719 -> 336 -> 699 -> 583 -> 192 -> 825 -> 336 -> 900 -> 227 -> 559 -> 73 -> 68 -> 378 -> 142 -> 569 -> 157 -> 409 -> 73 -> 378 -> 534 -> 73 -> 724 -> 68 -> 569 -> 378 -> 559 -> 142 -> 654 -> 159 -> 559 -> 587 -> 142 -> 993 -> 68 -> 51 -> 157 -> 498 -> 259 -> 509 -> 807 -> 259 -> 869 -> 893 -> 940 -> 524 -> 564 -> 940 -> 341 -> 753 -> 946 -> 341 -> 384 -> 524 -> 893 -> 972 -> 271 -> 897 -> 245 -> 799 -> 383 -> 890 -> 151 -> 706 -> 95 -> 831 -> 972 -> 881 -> 95 -> 773 -> 997 -> 245 -> 890 -> 434 -> 481 -> 90 -> 388 -> 144 -> 408 -> 90 -> 417 -> 144 -> 782 -> 799 -> 542 -> 979 -> 286 -> 338 -> 529 -> 177 -> 402 -> 529 -> 293 -> 979 -> 383 -> 944 -> 577 -> 144 -> 890 -> 782 -> 931 -> 173 -> 459 -> 99 -> 338 -> 578 -> 99 -> 529 -> 865 -> 99 -> 962 -> 338 -> 982 -> 99 -> 901 -> 177 -> 478 -> 459 -> 177 -> 931 -> 901 -> 402 -> 578 -> 865 -> 863 -> 347 -> 594 -> 350 -> 863 -> 594 -> 257 -> 390 -> 376 -> 791 -> 631 -> 376 -> 977 -> 390 -> 477 -> 257 -> 475 -> 973 -> 266 -> 477 -> 350 -> 973 -> 326 -> 393 -> 516 -> 196 -> 164 -> 482 -> 196 -> 613 -> 164 -> 540 -> 516 -> 482 -> 613 -> 516 -> 805 -> 100 -> 197 -> 591 -> 908 -> 482 -> 622 -> 908 -> 828 -> 304 -> 829 -> 908 -> 100 -> 304 -> 873 -> 164 -> 805 -> 150 -> 164 -> 842 -> 482 -> 805 -> 196 -> 150 -> 393 -> 775 -> 560 -> 973 -> 347 -> 990 -> 733 -> 775 -> 966 -> 540 -> 393 -> 973 -> 775 -> 540 -> 613 -> 805 -> 474 -> 100 -> 752 -> 75 -> 270 -> 618 -> 877 -> 75 -> 618 -> 935 -> 75 -> 899 -> 270 -> 877 -> 565 -> 75 -> 918 -> 537 -> 752 -> 474 -> 342 -> 477 -> 977 -> 406 -> 651 -> 450 -> 737 -> 406 -> 663 -> 270 -> 935 -> 541 -> 918 -> 899 -> 663 -> 877 -> 752 -> 342 -> 565 -> 191 -> 150 -> 613 -> 842 -> 805 -> 191 -> 342 -> 877 -> 899 -> 136 -> 589 -> 198 -> 62 -> 134 -> 198 -> 301 -> 983 -> 777 -> 399 -> 218 -> 202 -> 300 -> 145 -> 747 -> 186 -> 986 -> 868 -> 125 -> 85 -> 120 -> 602 -> 788 -> 952 -> 0, Gesamtlaenge: 2791747

Tag 4: 0 -> 952 -> 788 -> 602 -> 120 -> 85 -> 125 -> 868 -> 986 -> 186 -> 747 -> 145 -> 300 -> 202 -> 218 -> 399 -> 777 -> 983 -> 301 -> 823 -> 942 -> 904 -> 301 -> 939 -> 589 -> 301 -> 983 -> 136 -> 663 -> 937 -> 514 -> 631 -> 737 -> 469 -> 514 -> 737 -> 561 -> 406 -> 937 -> 737 -> 651 -> 469 -> 561 -> 450 -> 885 -> 514 -> 651 -> 561 -> 791 -> 715 -> 459 -> 876 -> 478 -> 715 -> 606 -> 177 -> 982 -> 402 -> 962 -> 578 -> 982 -> 529 -> 901 -> 478 -> 2 -> 459 -> 931 -> 979 -> 799 -> 997 -> 383 -> 577 -> 151 -> 997 -> 979 -> 890 -> 944 -> 607 -> 90 -> 180 -> 388 -> 417 -> 180 -> 123 -> 51 -> 498 -> 384 -> 569 -> 498 -> 724 -> 51 -> 409 -> 378 -> 993 -> 73 -> 685 -> 534 -> 806 -> 159 -> 587 -> 654 -> 900 -> 587 -> 836 -> 719 -> 583 -> 545 -> 902 -> 107 -> 54 -> 26 -> 302 -> 432 -> 26 -> 454 -> 66 -> 438 -> 302 -> 454 -> 438 -> 448 -> 26 -> 913 -> 66 -> 485 -> 119 -> 975 -> 702 -> 886 -> 464 -> 906 -> 886 -> 697 -> 550 -> 624 -> 750 -> 988 -> 629 -> 550 -> 907 -> 988 -> 550 -> 395 -> 683 -> 697 -> 395 -> 886 -> 975 -> 951 -> 395 -> 620 -> 363 -> 279 -> 698 -> 363 -> 106 -> 329 -> 592 -> 882 -> 447 -> 599 -> 852 -> 978 -> 372 -> 485 -> 905 -> 66 -> 448 -> 224 -> 302 -> 861 -> 224 -> 438 -> 913 -> 302 -> 872 -> 454 -> 224 -> 833 -> 302 -> 902 -> 432 -> 54 -> 985 -> 26 -> 818 -> 107 -> 159 -> 653 -> 107 -> 432 -> 515 -> 719 -> 699 -> 515 -> 825 -> 656 -> 699 -> 688 -> 825 -> 719 -> 924 -> 654 -> 559 -> 900 -> 719 -> 930 -> 699 -> 924 -> 825 -> 900 -> 836 -> 825 -> 930 -> 836 -> 924 -> 900 -> 930 -> 924 -> 587 -> 685 -> 559 -> 993 -> 409 -> 878 -> 157 -> 724 -> 569 -> 993 -> 534 -> 653 -> 806 -> 685 -> 653 -> 818 -> 432 -> 913 -> 448 -> 861 -> 746 -> 680 -> 765 -> 224 -> 913 -> 985 -> 432 -> 833 -> 746 -> 765 -> 448 -> 872 -> 985 -> 818 -> 833 -> 861 -> 765 -> 3 -> 361 -> 374 -> 573 -> 571 -> 374 -> 830 -> 131 -> 599 -> 882 -> 698 -> 329 -> 620 -> 698 -> 762 -> 830 -> 329 -> 855 -> 106 -> 638 -> 24 -> 52 -> 762 -> 882 -> 830 -> 573 -> 728 -> 374 -> 855 -> 52 -> 638 -> 161 -> 24 -> 106 -> 755 -> 571 -> 743 -> 437 -> 24 -> 499 -> 638 -> 437 -> 489 -> 604 -> 24 -> 56 -> 437 -> 604 -> 638 -> 56 -> 604 -> 787 -> 489 -> 676 -> 56 -> 772 -> 414 -> 658 -> 457 -> 440 -> 414 -> 676 -> 658 -> 489 -> 713 -> 414 -> 691 -> 440 -> 522 -> 463 -> 396 -> 491 -> 632 -> 743 -> 604 -> 860 -> 389 -> 3 -> 445 -> 572 -> 3 -> 483 -> 361 -> 830 -> 728 -> 755 -> 573 -> 483 -> 389 -> 112 -> 3 -> 814 -> 112 -> 9 -> 483 -> 743 -> 755 -> 483 -> 571 -> 860 -> 743 -> 787 -> 676 -> 691 -> 658 -> 440 -> 563 -> 522 -> 710 -> 491 -> 744 -> 9 -> 860 -> 787 -> 632 -> 772 -> 440 -> 713 -> 563 -> 710 -> 744 -> 112 -> 572 -> 389 -> 744 -> 794 -> 112 -> 609 -> 621 -> 250 -> 883 -> 101 -> 123 -> 388 -> 434 -> 577 -> 706 -> 271 -> 957 -> 706 -> 972 -> 957 -> 897 -> 944 -> 434 -> 607 -> 388 -> 481 -> 123 -> 417 -> 408 -> 607 -> 417 -> 999 -> 123 -> 878 -> 724 -> 993 -> 685 -> 549 -> 101 -> 547 -> 247 -> 678 -> 101 -> 758 -> 247 -> 843 -> 758 -> 547 -> 276 -> 101 -> 999 -> 247 -> 549 -> 547 -> 678 -> 843 -> 999 -> 678 -> 276 -> 549 -> 758 -> 276 -> 883 -> 292 -> 87 -> 211 -> 292 -> 188 -> 354 -> 87 -> 318 -> 188 -> 888 -> 83 -> 439 -> 398 -> 27 -> 60 -> 538 -> 27 -> 421 -> 60 -> 875 -> 637 -> 221 -> 961 -> 2 -> 606 -> 901 -> 459 -> 982 -> 865 -> 962 -> 982 -> 606 -> 876 -> 631 -> 885 -> 561 -> 921 -> 216 -> 411 -> 737 -> 791 -> 876 -> 715 -> 705 -> 631 -> 925 -> 791 -> 705 -> 876 -> 925 -> 715 -> 910 -> 70 -> 315 -> 306 -> 165 -> 961 -> 306 -> 476 -> 27 -> 712 -> 306 -> 398 -> 476 -> 439 -> 497 -> 398 -> 601 -> 83 -> 476 -> 712 -> 421 -> 538 -> 637 -> 421 -> 757 -> 27 -> 912 -> 163 -> 538 -> 912 -> 476 -> 601 -> 439 -> 757 -> 83 -> 912 -> 601 -> 497 -> 543 -> 315 -> 317 -> 70 -> 600 -> 401 -> 33 -> 148 -> 673 -> 33 -> 216 -> 651 -> 885 -> 216 -> 673 -> 203 -> 987 -> 411 -> 921 -> 469 -> 987 -> 981 -> 218 -> 399 -> 134 -> 77 -> 62 -> 589 -> 351 -> 904 -> 939 -> 732 -> 777 -> 62 -> 939 -> 942 -> 198 -> 732 -> 983 -> 198 -> 777 -> 136 -> 935 -> 918 -> 642 -> 474 -> 191 -> 785 -> 62 -> 983 -> 351 -> 777 -> 399 -> 785 -> 77 -> 198 -> 785 -> 230 -> 716 -> 77 -> 399 -> 945 -> 777 -> 939 -> 983 -> 777 -> 785 -> 716 -> 34 -> 32 -> 118 -> 135 -> 312 -> 315 -> 910 -> 317 -> 690 -> 2 -> 619 -> 961 -> 315 -> 690 -> 70 -> 793 -> 317 -> 58 -> 70 -> 961 -> 690 -> 793 -> 401 -> 58 -> 600 -> 910 -> 690 -> 58 -> 793 -> 600 -> 921 -> 673 -> 880 -> 33 -> 921 -> 705 -> 793 -> 910 -> 58 -> 562 -> 118 -> 278 -> 401 -> 562 -> 278 -> 405 -> 34 -> 424 -> 118 -> 566 -> 32 -> 278 -> 880 -> 148 -> 747 -> 109 -> 217 -> 149 -> 711 -> 109 -> 296 -> 61 -> 816 -> 495 -> 838 -> 597 -> 740 -> 838 -> 816 -> 597 -> 803 -> 896 -> 964 -> 597 -> 145 -> 300 -> 981 -> 896 -> 747 -> 145 -> 803 -> 964 -> 145 -> 838 -> 964 -> 740 -> 816 -> 742 -> 815 -> 848 -> 510 -> 686 -> 213 -> 749 -> 510 -> 25 -> 731 -> 149 -> 186 -> 109 -> 470 -> 808 -> 262 -> 704 -> 517 -> 711 -> 704 -> 630 -> 808 -> 764 -> 630 -> 964 -> 816 -> 517 -> 507 -> 296 -> 186 -> 217 -> 711 -> 764 -> 704 -> 507 -> 186 -> 747 -> 217 -> 751 -> 12 -> 371 -> 18 -> 186 -> 986 -> 18 -> 868 -> 125 -> 85 -> 120 -> 602 -> 788 -> 952 -> 0, Gesamtlaenge: 2765816

Tag 5: 0 -> 952 -> 788 -> 602 -> 120 -> 85 -> 125 -> 868 -> 18 -> 217 -> 986 -> 371 -> 729 -> 18 -> 751 -> 371 -> 730 -> 405 -> 424 -> 566 -> 34 -> 520 -> 32 -> 405 -> 566 -> 520 -> 405 -> 909 -> 32 -> 424 -> 786 -> 98 -> 135 -> 543 -> 398 -> 712 -> 497 -> 312 -> 786 -> 135 -> 943 -> 439 -> 368 -> 83 -> 949 -> 163 -> 888 -> 354 -> 292 -> 318 -> 354 -> 586 -> 188 -> 949 -> 538 -> 757 -> 601 -> 543 -> 786 -> 34 -> 670 -> 520 -> 730 -> 566 -> 670 -> 811 -> 12 -> 730 -> 862 -> 12 -> 781 -> 125 -> 85 -> 40 -> 380 -> 120 -> 85 -> 222 -> 380 -> 392 -> 40 -> 467 -> 368 -> 568 -> 98 -> 377 -> 368 -> 601 -> 943 -> 467 -> 377 -> 568 -> 467 -> 647 -> 31 -> 146 -> 633 -> 756 -> 87 -> 586 -> 292 -> 621 -> 586 -> 318 -> 845 -> 292 -> 190 -> 87 -> 655 -> 129 -> 707 -> 574 -> 684 -> 181 -> 21 -> 570 -> 79 -> 442 -> 238 -> 37 -> 122 -> 220 -> 479 -> 280 -> 268 -> 269 -> 7 -> 44 -> 251 -> 111 -> 526 -> 251 -> 182 -> 111 -> 610 -> 662 -> 111 -> 722 -> 525 -> 229 -> 241 -> 182 -> 229 -> 722 -> 610 -> 914 -> 111 -> 980 -> 229 -> 251 -> 343 -> 404 -> 251 -> 525 -> 241 -> 558 -> 182 -> 525 -> 526 -> 610 -> 980 -> 525 -> 662 -> 241 -> 174 -> 362 -> 160 -> 166 -> 154 -> 12 -> 868 -> 18 -> 862 -> 371 -> 781 -> 258 -> 32 -> 562 -> 880 -> 909 -> 34 -> 258 -> 98 -> 392 -> 120 -> 222 -> 392 -> 85 -> 380 -> 783 -> 40 -> 649 -> 85 -> 6 -> 125 -> 258 -> 520 -> 811 -> 125 -> 371 -> 811 -> 258 -> 566 -> 811 -> 729 -> 751 -> 862 -> 729 -> 868 -> 125 -> 154 -> 349 -> 166 -> 362 -> 349 -> 419 -> 154 -> 751 -> 868 -> 154 -> 6 -> 166 -> 419 -> 160 -> 490 -> 40 -> 826 -> 222 -> 783 -> 120 -> 490 -> 419 -> 362 -> 370 -> 174 -> 967 -> 660 -> 14 -> 423 -> 162 -> 846 -> 213 -> 423 -> 530 -> 644 -> 660 -> 423 -> 644 -> 749 -> 660 -> 530 -> 686 -> 644 -> 846 -> 749 -> 686 -> 660 -> 859 -> 419 -> 6 -> 781 -> 811 -> 730 -> 909 -> 371 -> 868 -> 862 -> 986 -> 868 -> 6 -> 1 -> 122 -> 442 -> 451 -> 449 -> 79 -> 976 -> 451 -> 582 -> 440 -> 745 -> 457 -> 713 -> 772

-> 563 -> 745 -> 522 -> 396 -> 502 -> 491 -> 794 -> 345 -> 820 -> 129 -> 190 -> 318 -> 888 -> 757 -> 949 -> 888 -> 756 -> 354 -> 190 -> 655 -> 211 -> 586 -> 883 -> 211 -> 845 -> 547 -> 883 -> 621 -> 518 -> 445 -> 680 -> 833 -> 895 -> 680 -> 861 -> 895 -> 746 -> 818 -> 902 -> 985 -> 746 -> 445 -> 814 -> 518 -> 572 -> 767 -> 707 -> 655 -> 181 -> 190 -> 756 -> 810 -> 31 -> 359 -> 647 -> 146 -> 810 -> 359 -> 703 -> 348 -> 31 -> 628 -> 348 -> 633 -> 810 -> 556 -> 467 -> 826 -> 377 -> 943 -> 568 -> 826 -> 380 -> 649 -> 160 -> 856 -> 598 -> 490 -> 649 -> 958 -> 490 -> 602 -> 120 -> 826 -> 556 -> 628 -> 359 -> 916 -> 31 -> 426 -> 556 -> 647 -> 628 -> 426 -> 647 -> 810 -> 628 -> 703 -> 934 -> 323 -> 528 -> 348 -> 810 -> 960 -> 146 -> 916 -> 348 -> 960 -> 426 -> 783 -> 602 -> 426 -> 934 -> 528 -> 703 -> 960 -> 633 -> 916 -> 628 -> 934 -> 602 -> 598 -> 492 -> 160 -> 991 -> 174 -> 314 -> 182 -> 526 -> 722 -> 914 -> 182 -> 662 -> 558 -> 370 -> 314 -> 558 -> 967 -> 846 -> 530 -> 859 -> 423 -> 686 -> 731 -> 423 -> 749 -> 731 -> 859 -> 349 -> 492 -> 362 -> 991 -> 314 -> 992 -> 768 -> 209 -> 243 -> 7 -> 185 -> 269 -> 305 -> 185 -> 493 -> 268 -> 718 -> 7 -> 827 -> 44 -> 404 -> 827 -> 305 -> 718 -> 185 -> 552 -> 269 -> 718 -> 243 -> 268 -> 994 -> 7 -> 611 -> 343 -> 525 -> 914 -> 526 -> 980 -> 722 -> 662 -> 251 -> 914 -> 980 -> 251 -> 894 -> 44 -> 692 -> 209 -> 273 -> 185 -> 884 -> 209 -> 555 -> 243 -> 884 -> 273 -> 305 -> 994 -> 611 -> 404 -> 894 -> 243 -> 692 -> 273 -> 493 -> 503 -> 273 -> 555 -> 280 -> 493 -> 552 -> 503 -> 280 -> 552 -> 584 -> 280 -> 593 -> 281 -> 303 -> 220 -> 776 -> 37 -> 460 -> 122 -> 776 -> 238 -> 460 -> 442 -> 745 -> 691 -> 772 -> 745 -> 832 -> 396 -> 744 -> 814 -> 609 -> 518 -> 794 -> 463 -> 582 -> 502 -> 570 -> 396 -> 794 -> 710 -> 463 -> 570 -> 582 -> 563 -> 832 -> 570 -> 976 -> 634 -> 449 -> 959 -> 442 -> 889 -> 37 -> 708 -> 1 -> 200 -> 420 -> 1 -> 460 -> 451 -> 959 -> 889 -> 122 -> 708 -> 238 -> 889 -> 460 -> 634 -> 974 -> 179 -> 21 -> 574 -> 767 -> 820 -> 21 -> 725 -> 179 -> 181 -> 820 -> 574 -> 725 -> 181 -> 916 -> 960 -> 528 -> 205 -> 788 -> 420 -> 294 -> 1 -> 889 -> 776 -> 708 -> 303 -> 294 -> 310 -> 200 -> 853 -> 554 -> 179 -> 694 -> 554 -> 684 -> 707 -> 820 -> 684 -> 725 -> 694 -> 853 -> 725 -> 554 -> 974 -> 200 -> 952 -> 205 -> 853 -> 974 -> 694 -> 976 -> 974 -> 824 -> 205 -> 934 -> 788 -> 602 -> 958 -> 768 -> 492 -> 370 -> 991 -> 768 -> 598 -> 784 -> 492 -> 991 -> 856 -> 768 -> 664 -> 428 -> 209 -> 614 -> 273 -> 428 -> 479 -> 584 -> 493 -> 555 -> 428 -> 593 -> 303 -> 310 -> 420 -> 708 -> 294 -> 0 -> 281 -> 428 -> 768 -> 784 -> 664 -> 614 -> 503 -> 584 -> 555 -> 692 -> 614 -> 593 -> 479 -> 281 -> 788 -> 922 -> 281 -> 952 -> 420 -> 0 -> 303 -> 584 -> 593 -> 503 -> 718 -> 611 -> 827 -> 718 -> 994 -> 827 -> 343 -> 894 -> 884 -> 555 -> 593 -> 0 -> 310 -> 824 -> 788 -> 952 -> 824 -> 0 -> 922 -> 664 -> 856 -> 958 -> 784 -> 856 -> 992 -> 784 -> 922 -> 952 -> 0, Gesamtlaenge: 2711172
Maximale Lange einer Tagestour: 2803005.0

Quellcode

```
def get_short_path(startpoint,matrix):
    gone_nodes = []#Diese Variable gibt an welche knoten bereits überprüft worden sind
    waiting_nodes = [startpoint] #Diese Variable speichert alle Knoten welche noch überprüft werden müssen
    memory = [[startpoint]] #Diese Variable specihert die Wege der noch zu überprüfenden Knoten
    grads = Graph.get_grad(matrix)[0]
    while True and len(waiting_nodes) != 0:
        if grads[waiting_nodes[0]]%2 == 1 and waiting_nodes[0] != startpoint: #überprüft ob der Knoten ungerade ist
            return memory[0] #Falls die zutrifft, so wird der Weg diese Graphen zurück gegeben
        new_nodes = get_connections(waiting_nodes[0],matrix,gone_nodes) #In der Variable new_Nodes werden die nächst möglichen Knoten
gespeichert(aufsegehend von waiting_nodes[0])
        gone_nodes.append(waiting_nodes[0])
        for n in new_nodes:
            if n not in waiting_nodes:
                waiting_nodes.append(n)
                path = tuple(memory[0])
                path = list(path)
                path.append(n)
                memory.append(path)
        waiting_nodes.pop(0)
        memory.pop(0)

def get_connections(point,matrix,gone_nodes): #die Parameter sind zum einen die aktuelste Verion der angepassten Matrix und zum anderen der
Bereits zurückgelete weg
    row = matrix[:,point] #gibt die Spalte des Knoten auf dem wir uns gerade Befinden wieder
    row.tolist()
    possible_connections = []
    for i,value in enumerate(row):
        if value != 0  and i not in gone_nodes:
            possible_connections.append(i)
    return possible_connections
#-----------------------------------------------------------------------------------------------
class Graph():
    def __init__(self,data,nodes,edges):
        self.nodes = nodes
        self.edges = edges
        self.matrix = Graph.create_matrix(self,data)
        self.weight_matrix = Graph.create_weight_matrix(self,data)
        self.adjusted_matrix = Graph.create_adjusted_matrix(self,data)

    def create_matrix(self,data):
        zero_matrix = np.zeros([self.nodes,self.nodes]) #erstellt eine Null-Matrix mit der Form nodes x nodes
        for i in data: #iteriert durch alle Daten
            zero_matrix.itemset((i[0],i[1]),1) #bei einer Adjenzmatrix werden normalerweise die 1sen nach den richtungen der Edges gesetzt zum Beispiel
A -> B aber da unser Graph ungerichtet ist müssen wir sowohl bei A->B als auch bei B->A eine 1 einfügen
            zero_matrix.itemset((i[1],i[0]),1)
        return zero_matrix

    def create_weight_matrix(self,data):
        zero_matrix = np.zeros([self.nodes,self.nodes]) #erstellt eine Null-Matrix mit der Form nodes x nodes
        for i in data: #iteriert durch alle Daten
            zero_matrix.itemset((i[0],i[1]),i[2]) #Diese Matrix ist von der Form her eine Adjenz-Matrix nur, dass statt den 1sen die Gewichte da stehen
            zero_matrix.itemset((i[1],i[0]),i[2])
        return zero_matrix

    def create_adjusted_matrix(self,data):
        adjusted_matrix = Graph.create_matrix(self,data)
```

```
        odd_index,control_bool = Graph.get_grad(adjusted_matrix)[1:]
        while control_bool:
            adjusted_matrix = Graph.adjust_grad(odd_index[0],adjusted_matrix,self.weight_matrix)
            odd_index,control_bool = Graph.get_grad(adjusted_matrix)[1:]
        return adjusted_matrix

    def adjust_grad(point,matrix,weight_matrix): #Dies Methode soll die einzelnen Elemente in der Matrix anpassen
        #path = Graph.get_short_path(point,matrix)
        odd_list = Graph.get_grad(matrix)[1]
        odd_list.remove(point)
        path = Daijkstra(point,odd_list,weight_matrix).path
        for i in range(len(path)-1):
            actual_num = matrix.item((path[i],path[i+1]))
            matrix.itemset((path[i],path[i+1]),actual_num+1)#Erhöht die Anzahl der Straßen um eins sowohl bei A->B als auch bei B->A
            matrix.itemset((path[i+1],path[i]),actual_num+1)
        return matrix


    def get_grad(matrix):#soll eine Liste mit den Grade der einzelnen Kreuzungen wiedergeben
        list = matrix.sum(axis=0).tolist() #summjert alle Zahlen von jeder Spalte auf und fügt es in ein Numpy Array, dieser Array wird anschließend in
eine Liste convertiert
        odd_list = []
        control_bool = False
        for i,b in enumerate(list):
            if b%2 == 1:
                odd_list.append([i,b])
                control_bool = True
        if len(odd_list) > 0:
            odd_list = SortIndexes_MinToMax(odd_list)[0]
        return list,odd_list,control_bool #[liste mit den jeweiligen Grade, liste mit den Sortierten Indexen,Kontrolle(Falls False, so sind keine ungeraden
Grade mehr da)]
#-----------------------------------------------------------------------------------------------------
class Fleury():
    def __init__(self,nodes,adjusted_matrix):
        self.nodes = nodes
        self.maxCounter = adjusted_matrix.sum()/2
        self.euler_circuit = Fleury.create_fleury_matrix(self,adjusted_matrix)

    def create_fleury_matrix(self,matrix):
        startpoint = 0
        current_matrix = copy.deepcopy(matrix)
        euler_circuit = [0] #Dies ist die Liste in welche nach her die besuchten Knoten der Reihenfolge nach eingefügt werden. 0 ist bereits gegeben, da
es immer bei 0 startet
        counter = 0
        while True:
            if counter % 10 == 0:
                print(f"{round(counter/self.maxCounter *100,1)}%")
            counter +=1
            if euler_circuit[-1] == 0: #Falls wir uns auf dem Startpunkt befinden, so gibt es keine möglichen "Cut edges"
                delta_matrix = np.zeros([self.nodes,self.nodes]) #erstellt eine Null-Matrix mit der Form nodes x nodes
            else:
                delta_matrix = Fleury.get_delta_matrix(self,euler_circuit[-1],current_matrix) #speichert die delta_matrix
            adapted_matrix = np.subtract(current_matrix,delta_matrix) #Diese Matrix zeigt alle möglichen Wege, ohne dass man ein Teil der original
Matrix abschneidet
            next_node = Fleury.choose_nodes(self,euler_circuit[-1],adapted_matrix) #gibt den nächsten Knoten an ohne das Risiko eine "Bridge" zu
entfernen
            if next_node != None: #Überprüft ob es ein Resultat gibt
                euler_circuit.append(next_node)
            else:
                next_node = Fleury.choose_nodes(self,euler_circuit[-1],current_matrix) #Falls der einzige verbleibende Knoten über eine "Cut Edge" führt
so wird diese genommen
                euler_circuit.append(next_node)
            actual_num = current_matrix.item((euler_circuit[-1],euler_circuit[-2])) #Entnimmt die Anzahl der Straßen, zwischen den beiden letzten
gegangenen Knoten
            current_matrix.itemset((euler_circuit[-1],euler_circuit[-2]),actual_num-1)#Verringer die Anzahl der Straßen um eins sowohl bei A->B als auch
bei B->A
            current_matrix.itemset((euler_circuit[-2],euler_circuit[-1]),actual_num-1)
            remain_edges = current_matrix.sum()/2 #Diese Funktion gibt die Anzahl der Verbleibenden Straßen an. Die Funktioniert indem man die
Elemente der aktuellen Matrix aufsummiert und anschließend wird das ganz noch durch 2 geteilt, da bei ungerichteten Adjenzmatrixen, eine Ecke
immer 2 mal dagestellt wird A->B, B->A
            if next_node == 0 and remain_edges == 0: #Falls wir wieder am Start
                return euler_circuit
```

```
    def get_delta_matrix(self,prev_node,matrix): #Diese Matrix soll garantieren, dass keine Bridge abgeschnitten wird
        zero_matrix = np.zeros([self.nodes,self.nodes]) #erstellt eine Null-Matrix mit der Form nodes x nodes
        path =get_short_path(prev_node,matrix)
        for i in range(len(path)-1):
            zero_matrix.itemset((path[i],path[i+1]),1)#Gibt die Edges an, welche nicht abgeschnitten werden dürfen
            zero_matrix.itemset((path[i+1],path[i]),1)
        return zero_matrix

    def choose_nodes(self, prev_node, matrix): #Diese Methode soll den nächsten Knoten finden
        possible_nodes = matrix[:,prev_node]#gibt die Spalte des Knoten auf dem wir uns gerade Befinden wieder
        for node,b in enumerate(possible_nodes):
            if b > 0:
                return node #Gibt den ersten Knoten zurück zu dem mindestens eine Ecke führt
        return None #Falls kein Knoten in frage kommt, so gib None zurück
#-----------------------------------------------------------------------------------------
class Daijkstra():
    def __init__(self,startpoint,endpoint,matrix):
        self.startpoint = startpoint
        self.endpoint = Daijkstra.get_endpoint(endpoint)
        self.weight_matrix = matrix
        self.visited = []
        self.visited_nodes = [] #Hier werden die nur die Namen aller bereits besuchten Knoten eingefügt
        self.unvisited = Daijkstra.get_unvisited_list(self)
        self.path = []
        Daijkstra.main(self)

    def get_endpoint(endpoint):#Dies dient dazu, dass man sowohl, eine Liste von möglichen Endzielen angeben kann oder nur ein einzelner Wert oder
None
        result = []
        if isinstance(endpoint,list):
            result.extend(endpoint)
        else:
            result.append(endpoint)
        return result

    def main(self):
        for _ in range(len(self.unvisited)):
            current_set = self.unvisited[0]
            x = self.unvisited.pop(0)
            self.visited.append(x)
            self.visited_nodes.append(x[0])
            Daijkstra.update_unvisited(self,current_set)
            if self.endpoint[0] == None:
                pass
            elif len(self.endpoint) == 1 and current_set[0] == self.endpoint[0]: #Überprüft, ob man den Algorithmus bereits beenden kann
                self.path.extend(Daijkstra.get_path(self,self.endpoint[0]))
                break
            elif len(self.endpoint) > 1 and current_set[0] in self.endpoint:#Überprüft, ob man den Algorithmus bereits beenden kann
                self.path.extend(Daijkstra.get_path(self,self.visited[-1][0]))
                break
            if len(self.unvisited) == 1:#Falls nur noch 1 Knoten nicht besucht wurde,so ist der Algorithmus fertig
                if self.unvisited[0] == self.endpoint[0]:
                    Daijkstra.get_path(self)
                x = self.unvisited.pop(0)
                self.visited.append(x)
                self.visited_nodes.append(x[0])
                break

    def update_unvisited(self,current_set):
        neighbors = Daijkstra.get_neighbors(self,current_set[0])
        for i,set in enumerate(self.unvisited): #iteriert durch alle unbesuchten Knoten
            if set in neighbors: #Überprüft ob der Knoten auch ein Nachbar ist
                additional_weight = self.weight_matrix.item((current_set[0],set[0])) #sucht das Gewicht zwischen dem Knoten current_set und set
                possible_new_weight = current_set[1]+additional_weight
                if set[1] is None:
                    self.unvisited[i] = [set[0],possible_new_weight,current_set[0]]
                elif possible_new_weight < set[1]:
                    self.unvisited[i] = [set[0],possible_new_weight,current_set[0]]
            else:
                continue
        self.unvisited = SortIndexes_MinToMax(self.unvisited)[1]

    def get_neighbors(self,node): #Der aktuelle Knoten und die Matrix
```

```
        row = self.weight_matrix[:,node] #gibt die Spalte des Knoten auf dem wir uns gerade Befinden wieder
        row.tolist()
        neighbors = []
        for i,value in enumerate(row): #iteriert durch alle Knoten
            if value != 0  and i not in self.visited_nodes: #überprüft ob der Wert größer 0 ist und ob der Knoten nicht bereits besucht wurde
                for set in self.unvisited: #Falls beide bedingungen zutreffen, so wird in unvisited nach dem passenden set gesucht und zu neighbors
angefügt
                    if set[0] == i:
                        neighbors.append(set)
        return neighbors

    def get_unvisited_list(self): #Erstell eine Liste, mit der Benötigten Form, [Index,None,None] und setzt den Startpunkt auf 0. [Startpoint,0,None]
        lenght = self.weight_matrix.shape[0] #Gibt mir die Breite der Matrix zurück
        unvisited = []
        for i in range(0,lenght):
            if i == self.startpoint:
                unvisited.append([i,0,None])
            else:
                unvisited.append([i,None,None])
        unvisited = SortIndexes_MinToMax(unvisited)[1] #Sortiert die Liste, sodass das Startelement am Anfang steht
        return unvisited

    def get_path(self,endpoint):
        path = []
        current_set = Daijkstra.search_node(self,endpoint)
        while True:
            if current_set[2] == None:
                path.append(current_set[0])
                break
            path.append(current_set[0])
            current_set = Daijkstra.search_node(self,current_set[2])
        return list(reversed(path)) #Die Liste wird umgedreht, da sie aktuell den Weg vom Endpunkt zum Startpunkt zeigt, sie soll aber den Weg vom
Startpunkt zum Endpunkt zeigen

    def get_distance(self,endpoint):
        return Daijkstra.search_node(self,endpoint)[1]

    def search_node(self,node):
        for i in range(len(self.visited_nodes)):
            b = self.visited[i]
            if b[0] == node:
                return b
        print("Not found")

    def merge_adjenz_weight(adjence_matrix,weight_matrix): #
        shape = adjence_matrix.shape[0]
        new_matrix = np.zeros((shape,shape))
        for i in range(shape):
            for j in range(shape):
                actual_num1 = weight_matrix.item((i,j))#Entnimmt der Matrix die Werte an der Stelle (i,j)
                if  adjence_matrix.item((i,j)) > 0: #Falls die Adjenzmatrix größer 1 ist so wird actual_num2 automatisch auf 1 gesetzt
                    actual_num2 = 1
                else: #Falls dies nicht zutrifft,so wird actual_num2 auf 0 gesetzt
                    actual_num2 = 0
                product = actual_num1*actual_num2 #Multipliziert beide Werte
                new_matrix.itemset((i,j),product)
        return new_matrix
#-----------------------------------------------------------------------------------------------------
class Split():
    def __init__(self,nodes,adjusted_matrix,weight_matrix,euler_circuit):
        self.nodes = nodes
        self.adjusted_matrix = adjusted_matrix
        self.weight_matrix = weight_matrix
        self.euler_circuit = euler_circuit
        self.total_lenght = Split.complete_lenght(self)
        self.average_lenght = self.total_lenght/5
        self.mst = Daijkstra(0,None,weight_matrix)
        self.day_paths = []
        self.daily_lenght = []
        Split.main(self)
#Therorie: Wir teilen jede Strecke in 3 teile hinfahrt,weg,rückfahrt -> es muss immer ein weg geben, jedoch eine hin und rückfahrt sind nicht zwingen
notwendig
    def get_final_paths(self):
```

```
        days = []
        for i in self.day_paths:
            path = []
            if i[0] != []:
                i[0].pop(-1)
                for j in i[0]:
                    path.append(j)
            if i[2] != []:
                i[1].pop(-1)
                for j in i[1]:
                    path.append(j)
                for j in i[2]:
                    path.append(j)
            else:
                for j in i[1]:
                    path.append(j)
            days.append(path)
        max_lenght = self.daily_lenght[int(Find_Max_Basic(self.daily_lenght)[1])]
        return days, self.daily_lenght, max_lenght

    def main(self):
        remain_path = self.euler_circuit
        for i in range(5):
            if i == 4:
                Split.calc_best_cutting(self,remain_path,True)
            else:
                remain_path = Split.calc_best_cutting(self,remain_path,False)

    def calc_best_cutting(self,remain_path,isFriday):
        full_daily_path=[[],[],[]] #In diese Variable wird der ganz Weg engespeichert, mit den einzelnen etappen
        path_distance_list = Split.calc_distance_path(self,remain_path) #speichert in path_distance_list die distanzen aller möglichkeiten
        if isFriday == False:
            full_distance_list=[]
            for path in path_distance_list: #erstellt die Variable full_distance_list
                reference_distance = self.mst.get_distance(remain_path[0]) + path[1]+ 2*self.mst.get_distance(path[0])
                full_distance_list.append([path[0],path[1], reference_distance,self.mst.get_distance(path[0])])#Form: [Name des
Knoten,Weg_distanz,distanz zum Vergleichen,distanz zum Ursprung]
            main_path = []
            main_path_info = []
            for node in full_distance_list:
                if node[1] <= self.average_lenght//1: #das //1 soll die Zahl abrunden
                    main_path.append(node[0])
                    main_path_info.append(node)
                elif node[2] <= main_path_info[-1][2] and len(main_path)/(self.average_lenght)-1 <= 0.15: #Beschränkung, auf maximal 15%
überschreitung des durchschnittes
                    main_path.append(node[0])
                    main_path_info.append(node)
                else:
                    break
            full_daily_path[1].extend(main_path)
            if main_path[-1] != 0:#Überprüft ob eine Rückfahrt nötig ist
                return_path = self.mst.get_path(main_path[-1])#Falls sie nötig sein sollt, so wird die kürzeste gesucht
                return_path = list(reversed(return_path)) #In diesem Fall muss man die Liste noch mal um drehen, weil es in diesem Fall vom einem
Bestimmten Punkt zurück zum Ursprung geht und nicht wie normalerweise
                full_daily_path[2].extend(return_path)
            self.daily_lenght.append(self.mst.get_distance(remain_path[0])+main_path_info[-1][1]+self.mst.get_distance(main_path[-1])) #Berechnet die
gesamte stecke und fügt es in die Variable
        else:
            full_daily_path[1].extend(remain_path)
            self.daily_lenght.append(self.mst.get_distance(remain_path[0])+path_distance_list[-1][1]) #Berechnet die gesamte Strecke speziell für den
Freitag

        if remain_path[0] != 0: #Überprüft ob eine Hinfahrt nötig ist
            outward_path = self.mst.get_path(remain_path[0]) #Falls sie nötig sein sollt, so wird die kürzeste gesucht
            full_daily_path[0].extend(outward_path)

        self.day_paths.append(full_daily_path)
        if isFriday == False:
            remain_path = remain_path[len(main_path)-1:] #Dieser Code soll den schon abgefahrenen Weg entfernen, lässt aber noch den End Knoten da,
da er später als Anfangs Knoten dienen wird
        return remain_path

    def calc_distance_path(self,path):
```

```
        distance = 0
        memory = [[path[0],0]] #Hier sollen die Distanzen aller Punkte gespeichert werden Form: [Name des Knoten, Entfernung vom Startknoten]
        for node in path[1:]: #Wir iterieren durch alle Knoten im Path außer dem ersten,weil wir den bereits im memory gespeichert haben.
            distance_delta = self.weight_matrix.item((memory[-1][0],node)) #Wir speichern die distanz zwischen dem Letzten Knoten und dem aktuellen
in der Variable distance_delta
            distance = distance +distance_delta
            memory.append([node,distance])
        return memory

    def complete_lenght(self):
        shape = self.adjusted_matrix.shape[0]
        total_lenght = 0
        for i in range(shape):
            for j in range(shape):
                actual_num1 = self.weight_matrix.item((i,j))#Entnimmt der Matrix die Werte an der Stelle (i,j)
                actual_num2 = self.adjusted_matrix.item((i,j))
                product = actual_num1*actual_num2 #Multipliziert beide Werte
                total_lenght = total_lenght + product
        return total_lenght/2 #Da bei der Adjenzmatrix jeder Weg doppelt eingetragen ist muss man durch 2 teilen
```