

Aufgabe 2: ChoreoGraph DAG Lösung

Team-ID: 00001

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

1. Dezember 2025

Inhaltsverzeichnis

1	Lösungsidee	1
1.1	Vereinfachungen	1
1.2	Darstellung als Graph	1
1.2.1	Beispiel	2
1.3	Erstellen des Graphs	2
1.4	DAG	2
1.5	Optimierungsziele	2
2	Umsetzung	2
3	Komplexitätsanalyse	3
4	Beispiele	3
4.1	Standardbeispiele	4
4.2	Zusatzbeispiele	10
5	Quellcode	16

1 Lösungsidee

Alle möglichen Lösungen werden durch einen DAG dargestellt, aus dem die Lösungen der verschiedenen Optimierungsziele berechnet werden.

1.1 Vereinfachungen

Ich spreche hier nur von drei Personen anstelle von 16. Die Lösung ist jedoch auf 16 Personen skalierbar.

1.2 Darstellung als Graph

Sei F ein Tupel der Figuren, sei m die Anzahl an Takten.

Daraus kann nun ein gerichteter Graph konstruiert werden, bei dem jeder Knoten einer bestimmten Figur und dessen Endzeit entspricht. Es gibt also für jede Taktzahl von 0 bis m ganze $|F|$ verschiedene Knoten. Die Kanten ausgehend aus einer Figur $f \in F$ zeigen an, welche Figur direkt nach f getanz werden kann. Jede mögliche Choreographie entspricht dann einem Pfad von den Start- zu den Endknoten.

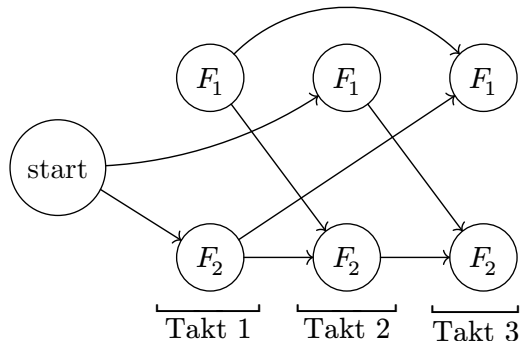
1.2.1 Beispiel

Jede Figur wird als ein tupel dargestellt mit: (dauer, Endpositionen der Personen)

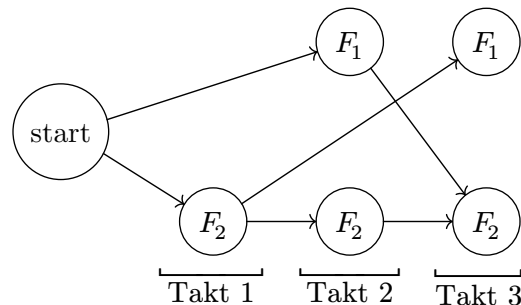
$F = ((2, ACB), (1, CBA))$

$m = 3$

Der daraus entstehende Graph:



Der Graph ohne überflüssige Kanten und Knoten:



1.3 Erstellen des Graphs

Es werden wiederholt, nach einem bestimmten Schema, Kanten aus dem Graphen entfernt, die nicht relevant sind. Am Ende entsteht ein Graph, bei dem jede Kante mindestens eine mögliche Lösung repräsentiert, die die Nebenbedingungen erfüllt.

Hierfür werden zunächst von Start bis Ende alle erreichbaren Zustände für alle Knoten ermittelt. Anschließend werden diejenigen Zustände und Kanten gelöscht, die nicht bis zum Ende führen.

Das Schema wird insgesamt dreimal mit folgenden Kriterien ausgeführt:

1. Taktlängen
2. Einzelne Positionen der Tänzer
3. Permutationen der Tänzer

Im dritten Durchlauf wird darauf geachtet, dass eine Permutation nur weiter geprüft wird, wenn auch alle Einzelpositionen dieser Permutation erreicht werden können.

1.4 DAG

Die Ergebnisse des dritten Durchlaufs werden verwendet, um einen DAG zu erstellen, bei dem jeder Knoten eine erreichbare Permutation eines Knotens des vorherigen Graphen darstellt. Dadurch entspricht jeder mögliche Pfad im DAG genau einer möglichen Lösung.

1.5 Optimierungsziele

Auf dem DAG können die Lösungen für die Optimierungsziele 2 bis 5 mittels Greedy-Verfahren im Graph ermittelt werden. Für das erste Optimierungsziel ist diese Methode jedoch nicht optimal. Deshalb werden hierfür alle möglichen Pfade berechnet und der Pfad, der das erste Optimierungsziel am besten erfüllt, als Lösung ausgewählt. Falls die Anzahl möglicher Lösungen sehr groß ist (nicht der Fall bei allen BwInf-Eingaben), wird das erste Optimierungsziel greedy ermittelt, also nur angenähert.

2 Umsetzung

Die Lösung wurde in JavaScript umgesetzt.

3 Komplexitätsanalyse

Wir setzen voraus, dass die Grundrechenarten in konstanter Zeit durchführbar sind und dass Zahlen konstant viel Speicher belegen. Wir betrachten die Laufzeit in Abhängigkeit von der Anzahl verschiedener Figuren f und der Anzahl an Takten t . Das Einlesen der Daten hat die Laufzeitkomplexität $\mathcal{O}(f)$.

Ein einzelner Durchlauf des Graphen benötigt $\mathcal{O}(tf^2)$. Für alle Optimierungsziele außer dem ersten Optimierungsziel ergibt sich damit ebenfalls $\mathcal{O}(tf^2)$. Beim ersten Optimierungsziel kommt ein zusätzlicher Term n für die Anzahl geprüfter Pfade hinzu, sodass die Komplexität $\mathcal{O}(tf^2 + n)$ ist. Im schlimmsten Fall gilt $n = f^t$, daher ist die Laufzeitkomplexität des Programms $\mathcal{O}(f^t)$.

4 Beispiele

Alle Beispiele wurden mit dem gleichen Programm berechnet. Dabei war dieses Programm in der Lage, alle Lösungen innerhalb einer Sekunde zu berechnen. Um die Leistungsfähigkeit unseres Programms zu veranschaulichen, haben wir uns dazu entschieden, weitere Beispielaufgaben zu erstellen, die Extremsituationen sowie weitere erwartbare Szenarien widerspiegeln.

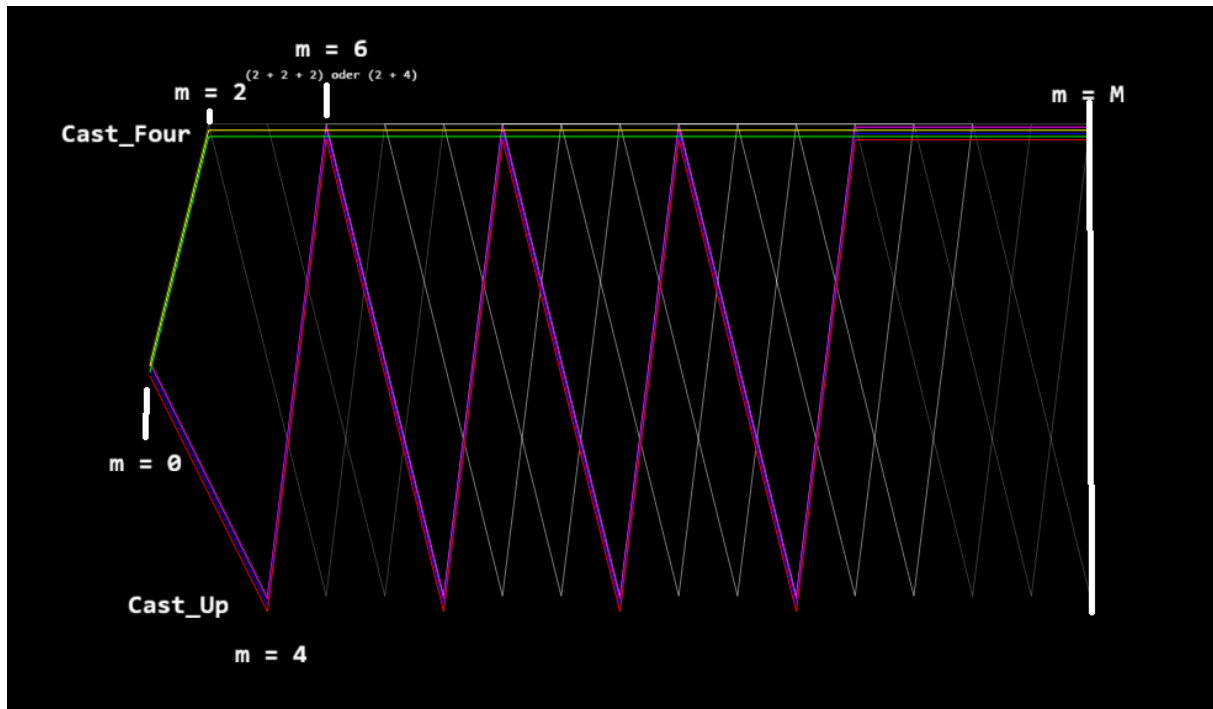
Lösungen mit große Ausgaben sind im Ordner zu finden. Eigene Aufgaben sind ebenfalls im Ordner zu finden.

Zu den Lösungen haben wir außerdem noch eine Grafik eingefügt, die die Entscheidung des Programms darstellt. Dabei steht die x-Achse für die Anzahl der Takte (wobei der letzte Wert M sowie der Anfangswert 0 entspricht) und die y-Achse für die einzelnen Figuren (von oben nach unten in der Reihenfolge der Aufgabendatei). Alle in der Grafik aufgezeigten Wege stellen ein mögliches Ergebnis dar.

Bestimmte Wegentscheidungen, die zur Endergebnisentscheidung geführt haben, werden mit Farben markiert. Dabei steht Pink für die am meisten unterschiedlichen Figuren, Gelb für die größte absolute Anzahl an Figuren, Blau für die geringste absolute Anzahl an Figuren, Grün für den Maximalbetrag der erforderlichen Bewegungen sowie Rot für den Minimalbetrag der erforderlichen Bewegungen.

Hinweis: Die Grafik stellt in keiner Weise die Endposition der Figuren dar, sondern nur die Entscheidungen der Figuren. Gibt es für eine Aufgabe keine Lösung, so gibt es auch keine Grafik, da keine Grafik dargestellt werden kann, die zur Lösung führt.

Unten ist ein Beispiel für choreo01.txt zu finden.



4.1 Standardbeispiele

choreo01.out

Das erste Beispiel lief ohne Probleme. Durch den Verlauf der Positionen erkennt man, dass alle Positionen korrekt und problemlos ausgerechnet wurden.

Runtime: 0.015s

Distinct solutions: 13

most distinct Figures (pink):

2 distinct Figures

Cast_Up→Cast_Four→Cast_Up→Cast_Four→Cast_Up→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four

ABCDEFGHIJKLMNO

PBCDEFGHIJKLMNO

MBCDFGHEJLINO

LABCPEFGDIJKMNO

IBCDMPFGHAJLENO

HABCLEFGPIJKMNO

EBCDFGHIJKLMNO

DABCHEFGLIJKPMNO

ABCD EFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCDEFGHIJKLMNO

most Figures (yellow):

16 Figures

Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four

ABCDEFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCD EFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCD EFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCD EFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCDEFGHIJKLMNO

least Figures (blue):

12 Figures

Cast_Up→Cast_Four→Cast_Up→Cast_Four→Cast_Up→Cast_Four→Cast_Up→Cast_Four→Cast_Four→Cast_Four→Cast_Four→Cast_Four

ABCDEFGHIJKLMNO

PBCDEFGHIJKLMNO

MBCDFGHEJLINO

LABCPEFGDIJKMNO

IBCDMPFGHAJLENO

HABCLEFGPIJKMNO

EBCDFGHIJKLMNO

DABCHEFGLIJKPMNO

ABCD EFGHIJKLMNO

BCDAFGHEJLINO

CDABGHEFKLIJOPMN

DABCHEFGLIJKPMNO

ABCDEFGHIJKLMNO

longest Distance (green):

384m

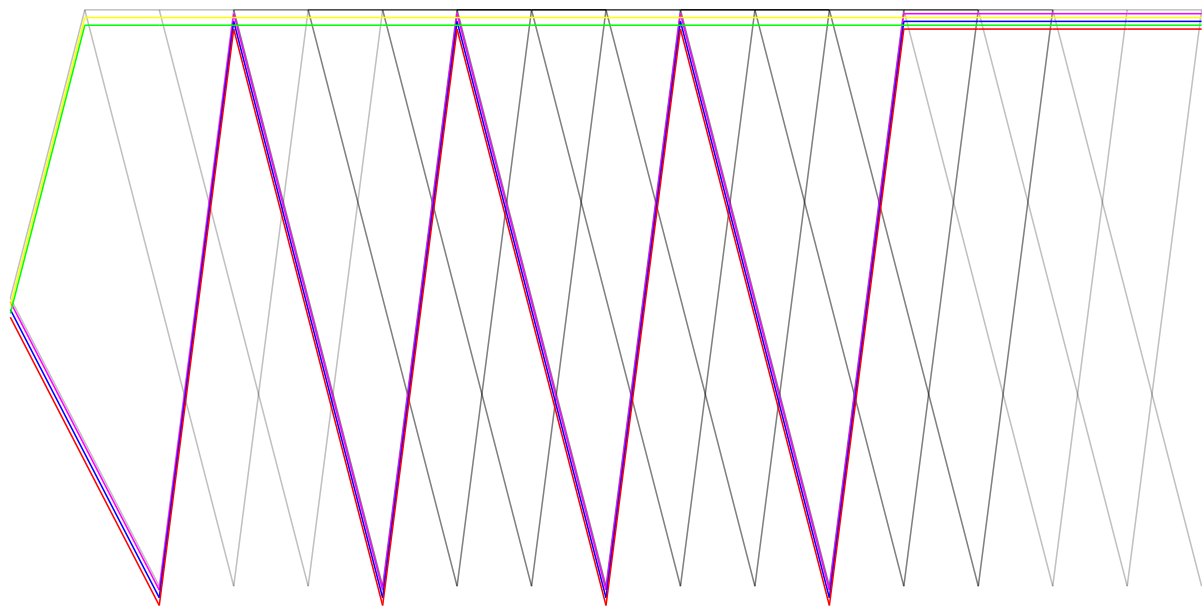
```

Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four
ABCDEF GHIJ KLMNOP
BCD AFGHEI JKLMNOP
C D AFGHEI JKLMNOP
D A B C F G H I J K L M N O
A B C D E F G H I J K L M N O
B C D A F G H I J K L M N O
C D A B G H E F K L I J O P M N
D A B C F G H I J K L M N O
A B C D E F G H I J K L M N O
B C D A F G H I J K L M N O
C D A B G H E F K L I J O P M N
D A B C F G H I J K L M N O
A B C D E F G H I J K L M N O
B C D A F G H I J K L M N O
C D A B G H E F K L I J O P M N
D A B C F G H I J K L M N O
ABCDEF GHIJ KLMNOP

shortest Distance (red):
312m
Cast_Up+Cast_Four+Cast_Up+Cast_Four+Cast_Up+Cast_Four+Cast_Up+Cast_Four+Cast_Four+Cast_Four+Cast_Four+Cast_Four
ABCDEF GHIJ KLMNOP
P A B C D E F G H I J K L M N O
M B C D A F G H E I J K L I N O P
L A B C P E F G D I J K H M N O
I B C D M F G H A J K L E N O P
H A B C L E F G P I J K D M N O
E B C D I F G H M J K L A N O P
D A B C F G H I J K P M N O
A B C D E F G H I J K L M N O P
B C D A F G H E I J K L I N O P M
C D A B G H E F K L I J O P M N
D A B C F G H I J K P M N O
A B C D E F G H I J K L M N O P

```

,



choreo02.out

Durch die verschiedenen Veränderungen der Positionen können keine Lösungen ausgerechnet werden, die genau auf 24 Takte enden und gleichzeitig zur Urposition zurückführen.

Runtime: 0.012s

Keine Lösung!

choreo03.out

Die Taktdauer (48) lässt sich nicht ohne Rest durch 5 teilen. Da nur zwei Figuren gegeben wurden und die erste Figur 5 Takte benötigt, während die zweite Figur das doppelte von der ersten Taktdauer benötigt, lassen sich keine Ergebnisse ohne Rest bilden, weshalb keine Lösungen berechnet werden können.

Runtime: 0.002s

Keine Lösung!

choreo04.out

Trotz ungewöhnlicher Taktdauer sowie durch die Figuren verursachter Positionen konnten Lösungen gefunden werden.

Runtime: 0.489s

Distinct solutions: 12

most distinct Figures (pink):

6 distinct Figures

Fan→Fan→Hover→Fan→Hockey_Stick→Fan→Fan→Fan→Fan→Appel→Bounce→Aida

ABCDEFGHIJKLMNPO

MKHLEIOJGDCPNABF

NCJPEGBDOLHFAMKI

OEKMPAJNHILCBFDG

BECNFMDAJGPHKILO

AFIHPEDGNJLBOMKC

MIGJFELoadPKBNCH

NGODIEPBMLFCKAHJ

AOBLGEFKNPIHCMJD

MBKPOEICAfGJHNDL

NIDGELHOJKBMCAFP

NFMHGPKBOJLIACD

ABCDEFGHIJKLMNPO

most Figures (yellow):

12 Figures

Fan→Fan→Hover→Fan→Hockey_Stick→Fan→Fan→Fan→Fan→Appel→Bounce→Aida

ABCDEFGHIJKLMNPO

MKHLEIOJGDCPNABF

NCJPEGBDOLHFAMKI

OEKMPAJNHILCBFDG

BECNFMDAJGPHKILO

AFIHPEDGNJLBOMKC

MIGJFELoadPKBNCH

NGODIEPBMLFCKAHJ

AOBLGEFKNPIHCMJD

MBKPOEICAfGJHNDL

NIDGELHOJKBMCAFP

NFMHGPKBOJLIACD

ABCDEFGHIJKLMNPO

least Figures (blue):

12 Figures

Fan→Fan→Hover→Fan→Hockey_Stick→Fan→Fan→Fan→Fan→Appel→Bounce→Aida

ABCDEFGHIJKLMNPO

MKHLEIOJGDCPNABF

NCJPEGBDOLHFAMKI

OEKMPAJNHILCBFDG

BECNFMDAJGPHKILO

AFIHPEDGNJLBOMKC

MIGJFELoadPKBNCH

NGODIEPBMLFCKAHJ

AOBLGEFKNPIHCMJD

MBKPOEICAfGJHNDL

NIDGELHOJKBMCAFP

NFMHGPKBOJLIACD

ABCDEFGHIJKLMNPO

longest Distance (green):

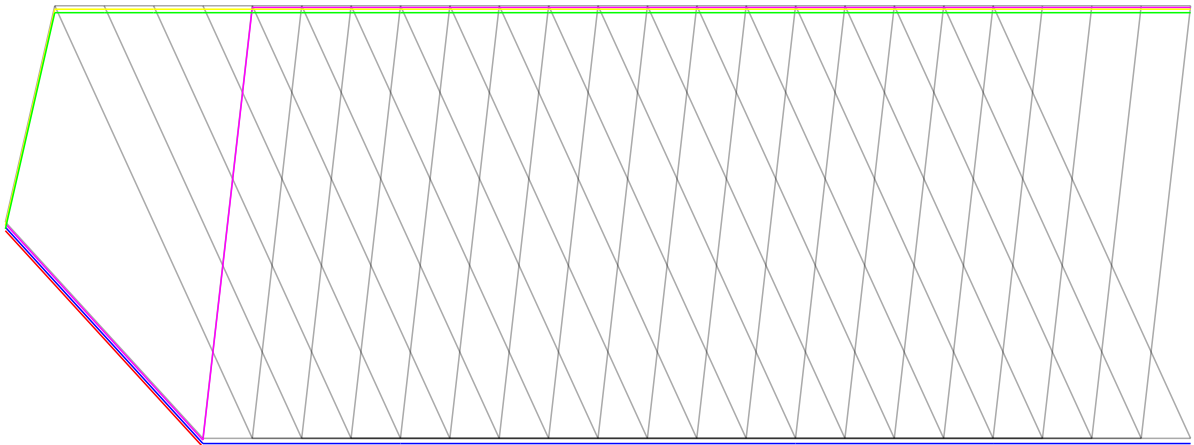
1154m

Fan→Fan→Hover→Fan→Hockey_Stick→Fan→Fan→Fan→Fan→Appel→Bounce→Aida

ABCDEFGHIJKLMNPO

MKHLEIOJGDCPNABF

,



choreo06.out

Runtime: 0.12s

Distinct solutions: 47014

most distinct Figures (pink):

4 distinct Figures

Cycle_Back→Cycle_Back→Cycle_Back→Cycle_Back→Sides→Pairs→Pairs→Cycle_4→Cycle_4→Cycle_4

ABCDEFGH IJKLMN

PABCDEFGH IJKLMN

OPABCDEFGH IJKLMN

NOPABCDEFGH IJKLM

MNOPABCDEFGH IJKL

EFGH IJKLMN

FEHG IJKLMN

EFGH IJKLMN

IJKLMN

MNOP

ABCDEFGH IJKLMN

most Figures (yellow):

16 Figures

Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGH IJKLMN

least Figures (blue):

4 Figures

Sides→Sides→Sides→Sides

ABCDEFGH IJKLMN

IJKLMN

ABCDEFGH IJKLMN

IJKLMN

ABCDEFGH IJKLMN

longest Distance (green):

768m

Cycle_4→Cycle_4→Cycle_4→Cycle_4→Cycle_4→Cycle_4→Cycle_4→Cycle_4

ABCDEFGH IJKLMN

EFGH IJKLMN

IJKLMN

MNOP

ABCDEFGH IJKLMN

EFGH IJKLMN

IJKLMN

MNOP

ABCDEFGH IJKLMN

shortest Distance (red):

256m

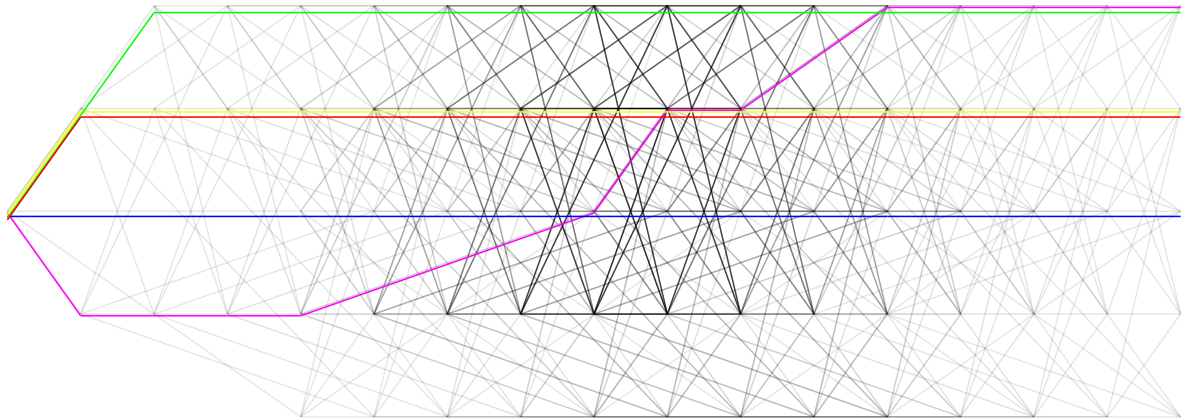
Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs→Pairs

ABCDEFGH IJKLMN

BADCFEHG IJKNMPO

ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP
 BADCFEHGJILKNMPO
 ABCDEFGHIJKLMNOP

,



choreoA.out

Bei der choreoA handelt es sich um eine erweiterte Version von choreo03.txt. Alle 23 Taktdauern können nicht ohne Rest durch 48 geteilt werden. Hinzu können keine Summen gebildet werden, die genau den Wert 48 entsprechen (beweisbar per DP). Aus diesem Grund, gibt es hier keine Lösung.

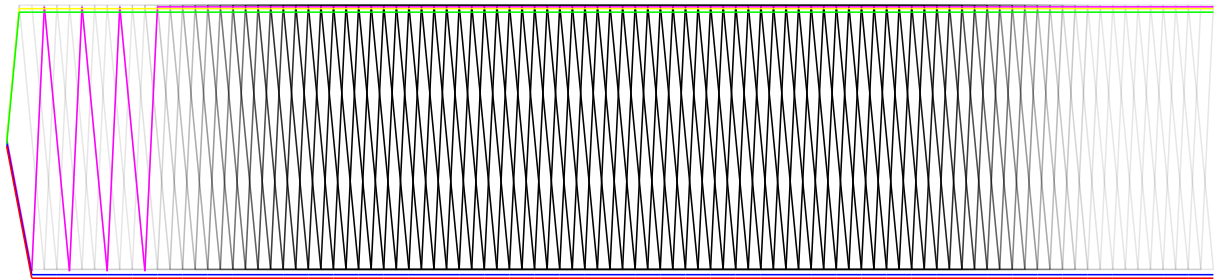
Runtime: 0.034s
 Keine Lösung!

4.2 Zusatzbeispiele

Custom1.out

Beim Custom1.txt handelt es sich um choreo01.txt, jedoch für $M = 192$. Die Ausgabe ist im Beispielsordner zu finden.

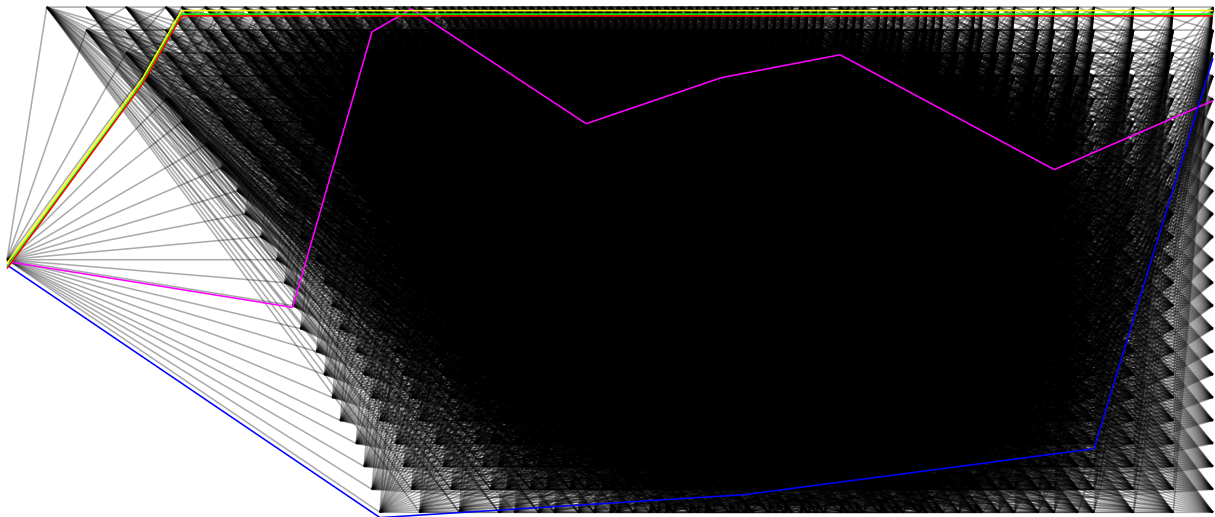
,



Custom3.out

Beim Custom3.txt handelt es sich um choreoA.txt, jedoch für $M = 152$. Die Ausgabe ist im Beispielsordner zu finden.

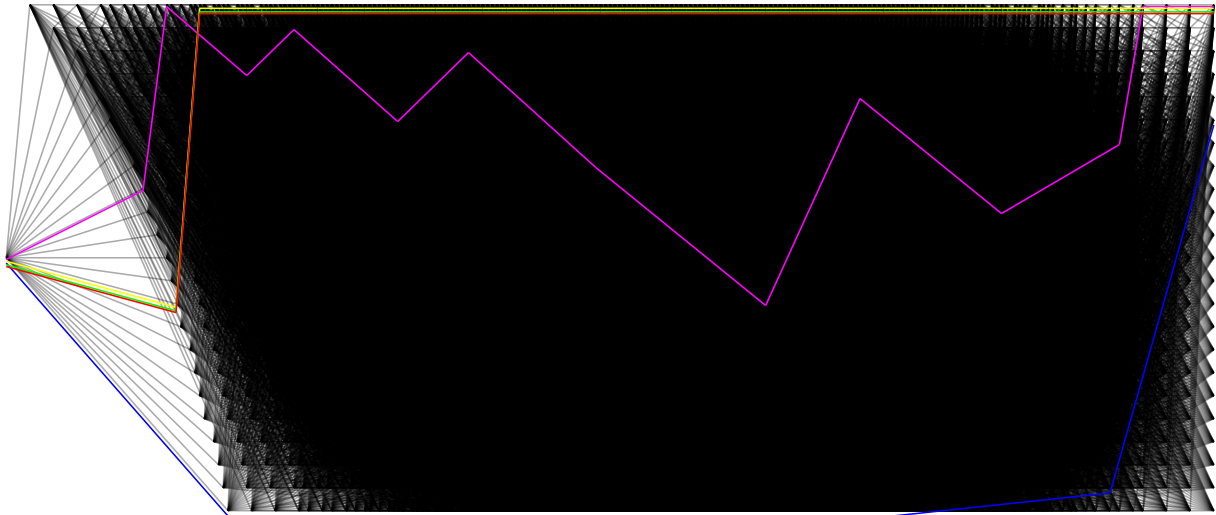
,



Custom4.out

Beim Custom4.txt handelt es sich um choreoA.txt, jedoch für $M = 256$. Die Ausgabe ist im Beispielsordner zu finden.

,



PyCustom1.out

Elf zufällig generierte Figuren mit zufällige Endpositionen sowie Taktdauer zwischen 1 und 2 (inklusive) mit $M = 8$. Durch die absolut zufällige Positionen können keine Lösungen generiert werden, die zum Startposition führen.

Runtime: 0.104s

Keine Lösung!

PyCustom2.out

Beim PyCustom2 wurde mehrere zufällige Figuren mit unterschiedlichen Taktdauern generiert. Dies wurde ebenfalls problemlos gelöst.

Runtime: 0.177s

Distinct solutions: 70410

most distinct Figures (pink):

4 distinct Figures

custom14→custom3→custom1→custom0

ABCDEFGHIJKLMNPO

CABDGEFHKIJLOMNP

BDCAFHGEJLKINPOM

CDABGHEFKLIJOPMN

ABCDEFGHIJKLMNPO

most Figures (yellow):

4 Figures

custom0→custom0→custom0→custom0

ABCDEFGHIJKLMNPO

CDABGHEFKLIJOPMN

ABCDEFGHIJKLMNPO

CDABGHEFKLIJOPMN

ABCDEFGHIJKLMNPO

least Figures (blue):

4 Figures

custom0→custom0→custom0→custom0

ABCDEFGHIJKLMNPO

CDABGHEFKLIJOPMN

ABCDEFGHIJKLMNPO

CDABGHEFKLIJOPMN

ABCDEFGHIJKLMNPO

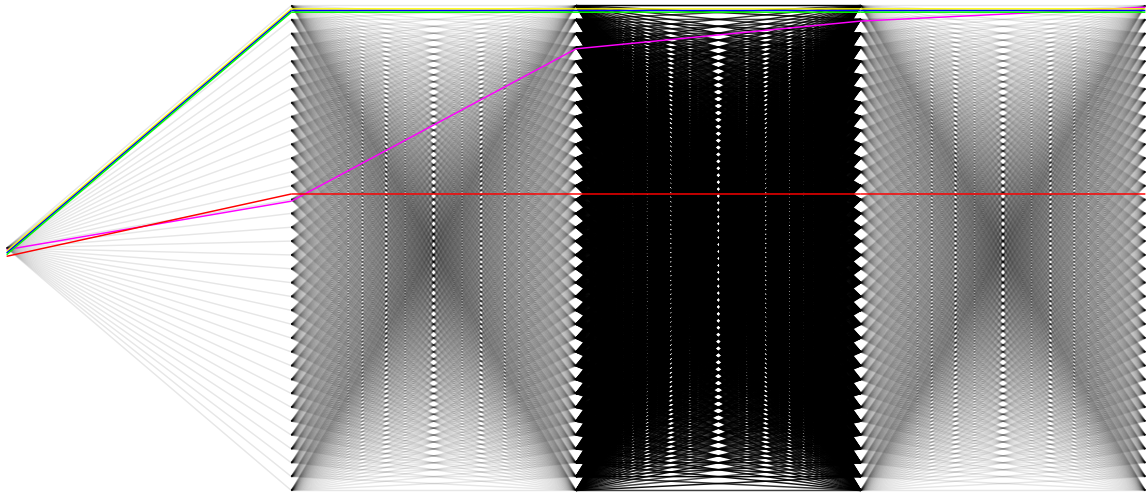
longest Distance (green):

```

128m
custom0→custom0→custom0→custom0
ABCDEFGHIJKLMNPO
CDABGHEFKLIJOPMN
ABCDEFGHIJKLMNPO
CDABGHEFKLIJOPMN
ABCDEFGHIJKLMNPO

shortest Distance (red):
32m
custom13→custom13→custom13→custom13
ABCDEFGHIJKLMNPO
BACDFEGHJIKLNMOP
ABCDEFGHIJKLMNPO
BACDFEGHJIKLNMOP
ABCDEFGHIJKLMNPO

```



PyCustom3.out

Beim PyCustom3 wurde weitere zufällige Figuren mit unterschiedlichen Taktdauern generiert.

Runtime: 0.234s

Distinct solutions: 225631

most distinct Figures (pink, maybe not Optimal):

```

4 distinct Figures
custom37→custom2→custom1→custom0
ABCDEFGHIJKLMNPO
ADCBEGHFIKJMPON
DBCAHFGELJKIPNOM
ABDCEFHGIJLKMNP0
ABCDEFGHIJKLMNPO

```

most Figures (yellow):

```

4 Figures
custom0→custom0→custom0→custom0
ABCDEFGHIJKLMNPO
ABDCEFHGIJLKMNP0
ABCDEFGHIJKLMNPO
ABDCEFHGIJLKMNP0
ABCDEFGHIJKLMNPO

```

least Figures (blue):

```

4 Figures
custom0→custom0→custom0→custom0
ABCDEFGHIJKLMNPO
ABDCEFHGIJLKMNP0
ABCDEFGHIJKLMNPO

```

ABDCEFHGIJLKMNO
 ABCDEFGHIJLKMNO

longest Distance (green):

128m

custom3→custom3→custom3→custom3

ABDCEFHGIJLKMNO

DCBAHGFELKJIPONM

ABDCEFHGIJLKMNO

DCBAHGFELKJIPONM

ABDCEFHGIJLKMNO

shortest Distance (red):

0m

custom10→custom10→custom10→custom10

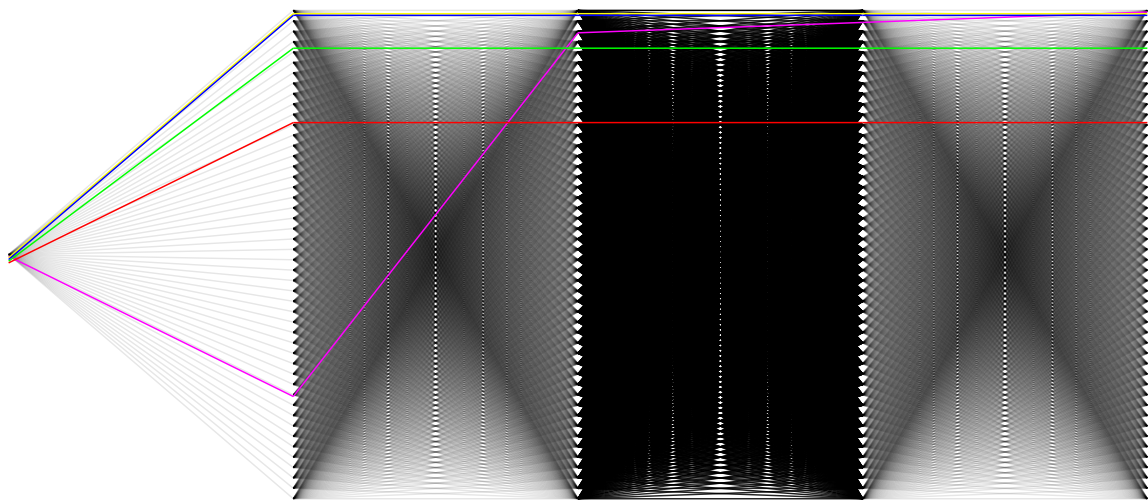
ABDCEFHGIJLKMNO

ABDCEFHGIJLKMNO

ABDCEFHGIJLKMNO

ABDCEFHGIJLKMNO

ABDCEFHGIJLKMNO



PyCustom4.out

Beim PyCustom4 wurde sehr viele zufällige Figuren mit unterschiedlichen Taktdauern für $M = 2$ generiert.

Runtime: 0.202s

Distinct solutions: 3042

most distinct Figures (pink):

2 distinct Figures

custom44→custom1

ABDCEFHGIJLKMNO

CABDGEFHKIJLOMNP

ABDCEFHGIJLKMNO

most Figures (yellow):

2 Figures

custom44→custom1

ABDCEFHGIJLKMNO

CABDGEFHKIJLOMNP

ABDCEFHGIJLKMNO

least Figures (blue):

1 Figures

custom90

ABDCEFHGIJLKMNO

ABDCEFHGIJLKMNO

longest Distance (green):

64m

custom14→custom3

ABCDEFGHIJKLMN

CDBAGHFEKLJIOPNM

ABCDEFGHIJKLMN

shortest Distance (red):

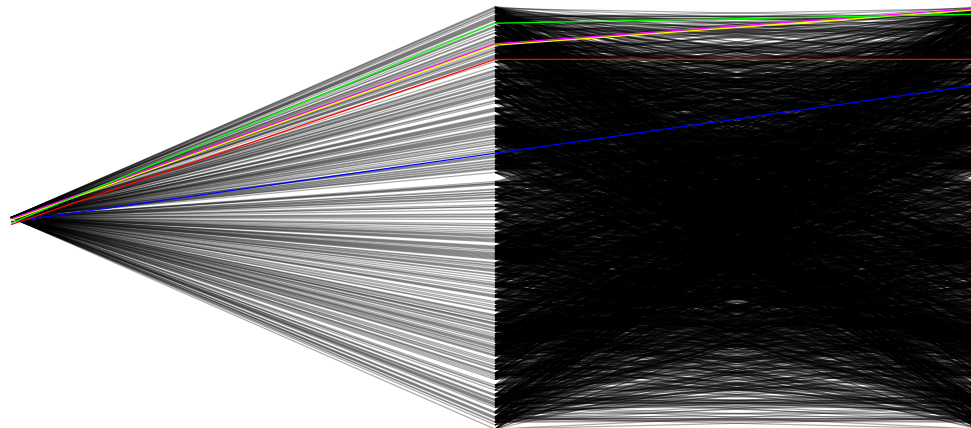
0m

custom56→custom56

ABCDEFGHIJKLMN

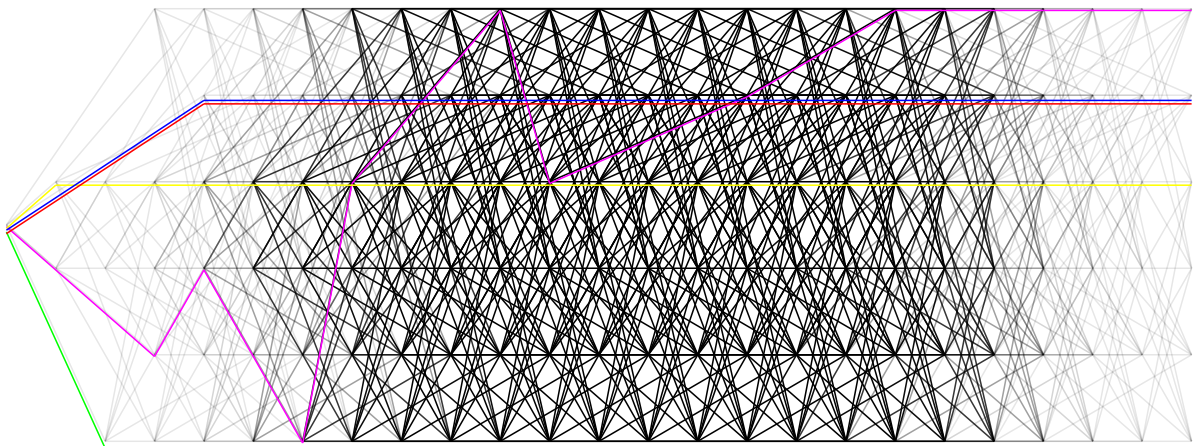
ABCDEFGHIJKLMN

ABCDEFGHIJKLMN



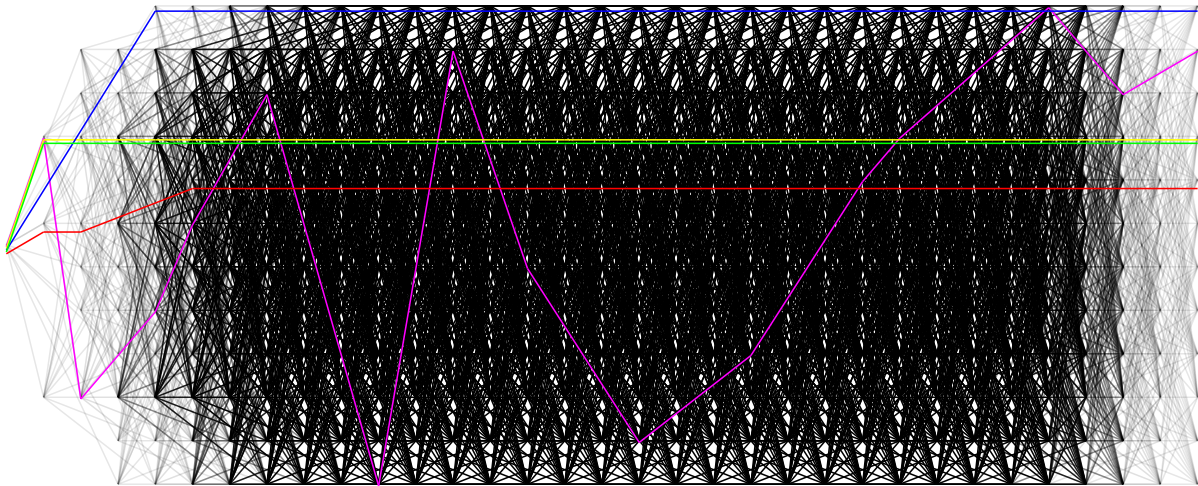
PyCustom5.out

Beim PyCustom5 wurde eine erwartbare Aufgabenstellung initiiert. Die Ausgabe ist im Beispielsordner zu finden.



PyCustom6.out

Beim PyCustom6 wurde eine erwartbare schwierige Aufgabenstellung initiiert. Die Ausgabe ist im Beispielsordner zu finden.



5 Quellcode

script.js

JS JavaScript

```
161 //Prüfe, ob die Liste einen Eintrag hat
162 function con(connections) {
163   for (let i = 0; i < connections.length; i++) {
164     if (connections[i]) {
165       return 1
166     }
167   }
168   return 0
169 }
170 //Erstelle Anfangsgraphen mit Knoten, die zeitlich passen
171 function allOpt(m, figuren) {
172   let n = figuren.length
173   let nodes = []
174   for (let i = 0; i <= m; i++) {
175     nodes[i] = []
176     for (let j = 0; j < n; j++) {
177       nodes[i][j] = [[], []]
178     }
179   }
180   nodes[0] = [[[]], [[]]]
181   let time = 0
182   let childTime = 1 => figuren[1][0] + time
183   let getNode = 1 => nodes[childTime(1)][1]
184   let addChilds = (i, j) => {
185     for (let l = 0; l < n; l++) {
186       if (childTime(l) <= m) {
187         nodes[i][j][1][l] = getNode(l)
188         getNode(l)[0][j] = nodes[i][j]
189       }
190     }
191   }
```



```
191  }
192  addChilds(0, 0)
193  for (let i = 1; i < m; i++) {
194    time++
195    for (let j = 0; j < n; j++) {
196      if (con(nodes[i][j][0])) {
197        addChilds(i, j)
198      }
199    }
200  }
201  return nodes
202 }
203 //Lösche gerichtete Kanten zu einer Position
204 function clearDir(node, dir, pos) {
205   for (let i = 0; i < node[dir].length; i++) {
206     if (node[dir][i]) {
207       node[dir][i][0][1 - dir][pos] = 0
208     }
209   }
210   node[dir].length = 0
211 }
212 //Vereine zwei Positionslisten (parent[ind] ∪ list2)
213 function unionList(parent, list2, ind) {
214   let list1 = parent[ind]
215   if (list2 == undefined || list1 == list2) {
216     return
217   }
218   if (list1 == undefined) {
219     if (typeof list2 == "number") {
220       parent[ind] = list2
221     } else {
222       parent[ind] = [...list2]
223     }
224     return
225   }
226   if (typeof list1 == "number") {
227     parent[ind] = []
228     parent[ind][list1] = 1
229     list1 = parent[ind]
230   }
231   if (typeof list2 == "number") {
232     list1[list2] = 1
233     return
234   }
235   for (let i = 0; i < list2.length; i++) {
```

```
236   list1[i] |= list2[i]
237 }
238 }
239 //Berechne, wie sich Positionen nach dem Tanz der Figur verändern
240 function playList(parent, conv, ind) {
241   let list = parent[ind]
242   if (list == undefined) {
243     return
244   }
245   if (typeof list == "number") {
246     parent[ind] = conv[list]
247     return
248   }
249   parent[ind] = []
250   for (let i = 0; i < list.length; i++) {
251     parent[ind][conv[i]] = list[i]
252   }
253 }
254 //Schnittmenge zweier Positionslisten (Positionen, die beide erreichen)
255 function interList(list1, list2) {
256   if (list1 == undefined || list2 == undefined) {
257     return undefined
258   }
259   if (typeof list1 == "number" && typeof list2 == "number") {
260     return list1 == list2 ? list1 : undefined
261   }
262   if (typeof list1 == "number") {
263     return list2[list1] ? list1 : undefined
264   }
265   if (typeof list2 == "number") {
266     return list1[list2] ? list2 : undefined
267   }
268   let sum = 0
269   let out = []
270   for (let i = 0; i < list1.length; i++) {
271     if (list1[i] && list2[i]) {
272       sum++
273       out[i] = 1
274     }
275   }
276   if (!sum) {
277     return undefined
278   } else if (sum == 1) {
279     return out.length - 1
280   }
```

```
281  return out
282  }
283  //Bestimme für eine Person, welche Positionen erreichbar sind
284  function filter(figuren, nodes, person) {
285    let m = nodes.length - 1
286    let n = figuren.length
287    nodes[0][0][2] = person
288    for (let i = 1; i <= m; i++) {
289      for (let j = 0; j < n; j++) {
290        let node = nodes[i][j]
291        for (let l = 0; l < node[0].length; l++) {
292          if (node[0][l]) {
293            unionList(node, node[0][l][0][2], 2)
294          }
295        }
296        playList(node, figuren[j][1], 2)
297      }
298    }
299    for (let i = 0; i < n; i++) {
300      nodes[m][i][3] = interList(person, nodes[m][i][2])
301    }
302    for (let i = m; i > 0; i--) {
303      for (let j = 0; j < n; j++) {
304        let node = nodes[i][j]
305        playList(node, figuren[j][2], 3)
306        for (let l = 0; l < node[0].length; l++) {
307          if (node[0][l]) {
308            unionList(node[0][l][0], interList(node[0][l][0][2], node[3]), 3)
309          }
310        }
311      }
312    }
313    for (let i = 0; i <= m; i++) {
314      for (let j = 0; j < nodes[i].length; j++) {
315        let node = nodes[i][j]
316        if (node[3] == undefined) {
317          clearDir(node, 0, j)
318          clearDir(node, 1, j)
319        } else {
320          for (let l = 0; l < node[1].length; l++) {
321            if (node[1][l]) {
322              let inter = interList(node[2], node[1][l][0][3])
323              if (inter == undefined) {
324                node[1][l][0][0][j] = 0
325                node[1][l] = 0
```

```
326     } else {
327         node[1][1][1][person] = inter
328     }
329 }
330 }
331 }
332 node.length = 2
333 }
334 }
335 }
336 //Berechne die Umkehrabbildung einer Permutation
337 function convBack(conv) {
338     let out = []
339     for (let i = 0; i < 16; i++) {
340         out[conv[i]] = i
341     }
342     return out
343 }
344 //Erweitere Kantenstruktur um Speicher für Referenzen
345 function extendEdges(nodes) {
346     for (let i = 0; i < nodes.length; i++) {
347         for (let j = 0; j < nodes[i].length; j++) {
348             let node = nodes[i][j]
349             for (let l = 0; l < node[1].length; l++) {
350                 if (node[1][l]) {
351                     let keys = []
352                     node[1][l][0][j] = [node[1][l][0][j], keys]
353                     node[1][l] = [node[1][l], keys]
354                 }
355             }
356         }
357     }
358 }
359 //Simuliere den Tanz einer Figur auf einer Positionsliste
360 function playFigur(pos, conv) {
361     let out = []
362     for (let i = 0; i < 16; i++) {
363         out[i] = conv[pos[i]]
364     }
365     return out
366 }
367 //Bestimme Teilbaum, der in der nächsten Figur noch vorkommen könnte
368 function passesTree(tree, edge, ind = 0) {
369     if (tree[0] == -1) {
370         for (let i = 1; i < tree.length; i++) {
```

```
371     let isNum = typeof edge[16 - i] == "number"
372     if ((isNum && edge[16 - i] != tree[i]) || (!isNum && !edge[16 - i][tree[i]])) {
373         return []
374     }
375 }
376 return [...tree]
377 }
378 if (typeof edge[ind] == "number") {
379     if (tree[edge[ind] + 1]) {
380         let child = passesTree(tree[edge[ind] + 1], edge, ind + 1)
381         if (child[0] == -1) {
382             child.push(edge[ind])
383             return child
384         }
385         let out = [tree[0]]
386         out[edge[ind] + 1] = child
387         return out
388     }
389     return []
390 }
391 let treeCopy = []
392 treeCopy[0] = 0
393 for (let i = 1; i < tree.length; i++) {
394     if (tree[i] && edge[ind][i - 1]) {
395         let child = passesTree(tree[i], edge, ind + 1)
396         if (child[0]) {
397             treeCopy[i] = child
398             treeCopy[0]++
399         }
400     }
401 }
402 if (treeCopy[0] == 1 && treeCopy[treeCopy.length - 1][0] == -1) {
403     treeCopy[treeCopy.length - 1].push(treeCopy.length - 2)
404     return treeCopy[treeCopy.length - 1]
405 }
406 return treeCopy
407 }
408 //Wende die Figurrrotation auf einen Baum an
409 function playTree(tree, conv) {
410     if (tree[0] == -1) {
411         for (let i = 1; i < tree.length; i++) {
412             tree[i] = conv[tree[i]]
413         }
414         return
415     }
```

```
416 let treeCopy = [...tree]
417 tree.length = 1
418 for (let i = 1; i < treeCopy.length; i++) {
419   if (treeCopy[i]) {
420     playTree(treeCopy[i], conv)
421     tree[conv[i - 1] + 1] = treeCopy[i]
422   }
423 }
424 }
425 //Vereinigung zweier (Teil-)Bäume
426 function unionTree(tree1, tree2) {
427   if (!tree2[0]) {
428     return
429   }
430   if (!tree1[0]) {
431     for (let i = 0; i < tree2.length; i++) {
432       tree1[i] = tree2[i]
433     }
434     return
435   }
436   if (tree1[0] == -1 && tree2[0] == -1) {
437     for (let i = tree1.length - 1; i > 0; i--) {
438       if (tree1[i] != tree2[i]) {
439         let tree1c = [...tree1]
440         tree1.length = 1
441         tree1[0] = 1
442         let head = tree1
443         for (let j = tree1c.length - 1; j > i; j--) {
444           head[0] = 1
445           let newHead = []
446           head[tree1c[j] + 1] = newHead
447           head = newHead
448         }
449         head[0] = 2
450         let ind1 = tree1c[i] + 1
451         let ind2 = tree2[i] + 1
452         tree1c.length = i
453         tree2.length = i
454         head[ind1] = tree1c
455         head[ind2] = tree2
456         return
457       }
458     }
459     return
460   }
```

```
461  if (tree1[0] == -1) {
462    let tree = [...tree1]
463    tree1.length = 1
464    tree1[0] = 1
465    tree1[tree.pop() + 1] = tree
466  }
467  if (tree2[0] == -1) {
468    let tree = [...tree2]
469    tree2.length = 1
470    tree2[0] = 1
471    tree2[tree.pop() + 1] = tree
472  }
473  for (let i = 1; i < tree2.length; i++) {
474    if (tree2[i]) {
475      if (tree1[i]) {
476        unionTree(tree1[i], tree2[i])
477      } else {
478        tree1[i] = tree2[i]
479      }
480    }
481  }
482 }
483 //Schnittmenge zweier (Teil-)Bäume
484 function interTree(tree1, tree2, ind = 0) {
485   if (tree1[0] == -1 && tree2[0] == -1) {
486     for (let i = 16 - ind; i > 0; i--) {
487       if (tree1[i] != tree2[i]) {
488         return []
489       }
490     }
491     return tree1.slice(0, 17 - ind)
492   }
493   if (tree1[0] == -1) {
494     if (tree2[tree1[16 - ind] + 1]) {
495       let out = interTree(tree1, tree2[tree1[16 - ind] + 1], ind + 1)
496       if (out[0] == -1) {
497         out.push(tree1[16 - ind])
498       }
499       return out
500     }
501     return []
502   }
503   if (tree2[0] == -1) {
504     if (tree1[tree2[16 - ind] + 1]) {
505       let out = interTree(tree2, tree1[tree2[16 - ind] + 1], ind + 1)
```

```
506     if (out[0] == -1) {
507         out.push(tree2[16 - ind])
508     }
509     return out
510 }
511 return []
512 }
513 let tree = []
514 tree[0] = 0
515 for (let i = 1; i < Math.min(tree1.length, tree2.length); i++) {
516     if (tree1[i] && tree2[i]) {
517         let child = interTree(tree1[i], tree2[i], ind + 1)
518         if (child[0]) {
519             tree[i] = child
520             tree[0]++
521         }
522     }
523 }
524 if (tree[0] == 1 && tree[tree.length - 1][0] == -1) {
525     tree[tree.length - 1].push(tree.length - 2)
526     return tree[tree.length - 1]
527 }
528 return tree
529 }
530 //Berechne alle möglichen Positionskombinationen für alle Knoten
531 function brute(figuren, nodes) {
532     let m = nodes.length - 1
533     for (let i = 0; i <= m; i++) {
534         for (let j = 0; j < nodes[i].length; j++) {
535             nodes[i][j][2] = []
536             nodes[i][j][3] = []
537         }
538     }
539     nodes[0][0][2] = [-1, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
540     for (let i = 1; i <= m; i++) {
541         for (let j = 0; j < nodes[i].length; j++) {
542             let node = nodes[i][j]
543             for (let l = 0; l < node[0].length; l++) {
544                 if (node[0][l]) {
545                     unionTree(node[2], passesTree(node[0][l][0][2], node[0][l][1]))
546                 }
547             }
548             playTree(node[2], figuren[j][1])
549         }
550     }
```



```
551   for (let i = 0; i < nodes[m].length; i++) {
552     nodes[m][i][3] = [-1, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
553   }
554   for (let i = m; i >= 0; i--) {
555     for (let j = 0; j < nodes[i].length; j++) {
556       let node = nodes[i][j]
557       playTree(node[3], figuren[j][2])
558       for (let l = 0; l < node[0].length; l++) {
559         if (node[0][l]) {
560           unionTree(node[0][l][0][3], interTree(node[0][l][0][2], node[3]))
561         }
562       }
563     }
564   }
565 }
566 //Konvertiere Positions-String zu Positionsliste und wende Permutation an
567 function convertStr(pos, conv) {
568   pos = pos.split(",")
569   for (let i = 0; i < 16; i++) {
570     pos[i] = +pos[i]
571   }
572   return playFigur(pos, conv) + ""
573 }
574 //Lese alle möglichen Pfade aus einem Baum aus
575 function treeToList(tree, path = []) {
576   if (tree[0] == -1) {
577     for (let i = tree.length - 1; i > 0; i--) {
578       path.push(tree[i])
579     }
580     return [path]
581   }
582   let dump = []
583   for (let i = 1; i < tree.length; i++) {
584     if (tree[i]) {
585       if (tree[i] === true) {
586         dump.push([...path, i - 1])
587       } else {
588         dump.push(...treeToList(tree[i], [...path, i - 1]))
589       }
590     }
591   }
592   return dump
593 }
594 //Konvertiere den Graphen in einen DAG
595 function convertGraph(figuren, nodes) {
```

```
596 let m = nodes.length - 1
597 let n = figuren.length
598 for (let i = 0; i < nodes.length; i++) {
599   for (let j = 0; j < nodes[i].length; j++) {
600     let node = nodes[i][j]
601     node[2] = new Map()
602     node[3] = treeToList(node[3])
603     for (let l = 0; l < node[3].length; l++) {
604       node[3][l] += ""
605       node[2][node[3][l]] = []
606     }
607     node[3] = new Set(node[3])
608   }
609 }
610 let dag = []
611 for (let i = 0; i <= m; i++) {
612   for (let j = 0; j < nodes[i].length; j++) {
613     let node = nodes[i][j]
614     node[3] = [...node[3]]
615     for (let l = 0; l < node[3].length; l++) {
616       let id = node[3][l]
617       let newNode = node[2][id]
618       let edges = []
619       newNode.push(edges, i, j)
620       id = convertStr(id, figuren[j][1])
621       for (let y = 0; y < node[1].length; y++) {
622         if (node[1][y] && node[1][y][0][3].has(id)) {
623           edges.push(node[1][y][0][2][id])
624         }
625       }
626       if (i == m || edges.length) {
627         dag.push(newNode)
628       }
629     }
630   }
631 }
632 return dag
633 }
634 //Ergänze Figuren um Gewichte/Metriken für Optimierungsziele
635 function addWeights(figuren) {
636   for (let i = 0; i < figuren.length; i++) {
637     let dis = 0
638     for (let j = 0; j < 16; j++) {
639       dis += Math.abs(figuren[i][2][j] - j)
640     }

```

```
641   figuren[i].push([- 1, 1, - dis, dis])
642 }
643 }
644 //Berechne den minimalen Pfad für ein gegebenes Optimierungsziel
645 function minPath(dag, figuren, goal) {
646   for (let i = 0; i < dag.length; i++) {
647     dag[i].push(1 / 0, [])
648   }
649   dag[0][3] = 0
650   dag[0][4] = []
651   for (let i = 0; i < dag.length; i++) {
652     for (let j = 0; j < dag[i][0].length; j++) {
653       let node = dag[i][0][j]
654       let newSum = dag[i][3] + figuren[node[2]][3][goal]
655       if (newSum < node[3]) {
656         node[3] = newSum
657         node[4] = [...dag[i][4], [node[1], node[2]]]
658       }
659     }
660   }
661   let best = 1 / 0
662   let bestPath = []
663   for (let i = dag.length - figuren.length; i < dag.length; i++) {
664     if (dag[i][3] < best) {
665       best = dag[i][3]
666       bestPath = dag[i][4]
667     }
668   }
669   for (let i = 0; i < dag.length; i++) {
670     dag[i].splice(-2, 2)
671   }
672   return [bestPath, best]
673 }
674 //Heuristische Berechnung: maximieren unterschiedlicher Figuren (Annäherung)
675 function heurMaxDistinct(n, dag) {
676   for (let i = 0; i < dag.length; i++) {
677     dag[i].push(0, new Set([]), [])
678   }
679   dag[0][5] = []
680   for (let i = 0; i < dag.length; i++) {
681     for (let j = 0; j < dag[i][0].length; j++) {
682       let node = dag[i][0][j]
683       let newSum = dag[i][3] + 1 - dag[i][4].has(node[2])
684       if (newSum > node[3]) {
685         node[3] = newSum
```

```
686     node[4] = new Set(dag[i][4])
687     node[4].add(node[2])
688     node[5] = [...dag[i][5], [node[1], node[2]]]
689 }
690 }
691 }
692 let best = 0
693 let bestPath = []
694 for (let i = dag.length - n; i < dag.length; i++) {
695     if (dag[i][3] > best) {
696         best = dag[i][3]
697         bestPath = dag[i][5]
698     }
699 }
700 for (let i = 0; i < dag.length; i++) {
701     dag[i].length = 3
702 }
703 return [bestPath, best]
704 }
705 //Zähle die Anzahl verschiedener Pfade im DAG
706 function getCombsAmount(dag) {
707     for (let i = 0; i < dag.length; i++) {
708         dag[i][3] = 0
709     }
710     dag[0][3] = 1
711     let sum = 0
712     for (let i = 0; i < dag.length; i++) {
713         for (let j = 0; j < dag[i][0].length; j++) {
714             dag[i][0][j][3] += dag[i][3]
715         }
716         if (!dag[i][0].length) {
717             sum += dag[i][3]
718         }
719         dag[i].pop()
720     }
721     return sum
722 }
723 //Berechne die optimale Lösung für das erste Optimierungsziel (exakt)
724 function maxDistinct(dag, figuren) {
725     for (let i = 0; i < dag.length; i++) {
726         dag[i][3] = []
727     }
728     dag[0][3] = [[]]
729     let allCombs = []
730     for (let i = 0; i < dag.length; i++) {
```

```
731   for (let j = 0; j < dag[i][0].length; j++) {
732     for (let l = 0; l < dag[i][3].length; l++) {
733       dag[i][0][j][3].push(...dag[i][3][l], dag[i][0][j][2])
734     }
735   }
736   if (!dag[i][0].length) {
737     allCombs.push(...dag[i][3])
738   }
739   dag[i].pop()
740 }
741 let bestNum = 0
742 let bestPath = []
743 for (let i = 0; i < allCombs.length; i++) {
744   let compressed = new Set(allCombs[i])
745   if (compressed.size > bestNum) {
746     bestNum = compressed.size
747     bestPath = allCombs[i]
748   }
749 }
750 let sum = 0
751 for (let i = 0; i < bestPath.length; i++) {
752   sum += figuren[bestPath[i]][0]
753   bestPath[i] = [sum, bestPath[i]]
754 }
755 return [bestPath, bestNum]
756 }
757 //Konvertiere Positionsarray in Buchstabenfolge (A-P)
758 function posToAlph(pos) {
759   let out = ""
760   for (let i = 0; i < 16; i++) {
761     out += String.fromCharCode(pos[i] + 65)
762   }
763   return out
764 }
765 //Berechne alle Optimierungsziele und formatiere die lesbare Ausgabe
766 function main(m, figuren, names) {
767   let n = figuren.length
768   let durations = []
769   for (let i = 0; i < n; i++) {
770     durations[i] = figuren[i][0]
771   }
772   for (let i = 0; i < n; i++) {
773     figuren[i].push(convBack(figuren[i][1]))
774   }
775   let nodes = allOpt(m, figuren)
```

```
776 extendEdges(nodes)
777 for (let i = 0; i < 16; i++) {
778   filter(figuren, nodes, i)
779 }
780 brute(figuren, nodes)
781 let dag = convertGraph(figuren, nodes)
782 addWeights(figuren)
783 let allCombsNum = getCombsAmount(dag)
784 let isOptimal = allCombsNum < 10 ** 5
785 let solutions = isOptimal ? [maxDistinct(dag, figuren)] : [heurMaxDistinct(n, dag)]
786 for (let i = 0; i < 4; i++) {
787   solutions.push(minPath(dag, figuren, i))
788 }
789 let solPaths = []
790 for (let i = 0; i < 5; i++) {
791   solPaths.push(solutions[i][0])
792   if (i & 1) {
793     solutions[i][1] *= -1
794   }
795 }
796 draw(n, m, dag, solPaths, figuren)
797 return dag.length == n ? "\nKeine Lösung!" : makeReadable(solutions, figuren, names,
798   isOptimal, allCombsNum)
```