Aufgabe 3: Wandertag

Team-ID: 00433

Team-Name: Was?

Bearbeiter dieser Aufgabe:
Mathieu de Borman

15. Oktober 2024

Inhaltsverzeichnis

1 Interpretation der Aufgabenstellung **1**

2 Lösungsidee **1**
2.1 Laufzeitanalyse . . . 2

3 Umsetzung **2**
3.1 Einlesen . . . 2
3.2 Strukturierung . . . 2
3.2.1 Speicherung als Gruppen . . . 2
3.2.2 Entfernung unnötiger Gruppen . . . 2
3.2.3 Kategoriesierung der Personen . . . 2
3.2.4 Abkürzungen . . . 3
3.3 Brute Force . . . 3
3.4 Ausgeben . . . 3

4 Beispiele **3**
4.1 Lösungen . . . 3
4.1.1 wandern1.txt . . . 3
4.1.2 wandern2.txt . . . 4
4.1.3 wandern3.txt . . . 5
4.1.4 wandern4.txt . . . 6
4.1.5 wandern5.txt . . . 6
4.1.6 wandern6.txt . . . 6
4.1.7 wandern7.txt . . . 7

5 Quellcode **7**

1 Interpretation der Aufgabenstellung

Aufgabe ist es, "drei Streckenlängen so [zu] berechne[n], dass möglichst viele Mitglieder teilnehmen."Deswegen ist es keine Option, ein Programm mit einer approximierenden Lösung zu wählen. Selbst wenn diese eine Laufzeitkomplexität von $O(n)$ hat, wobei n die Anzahl der potentiellen Teilnehmer ist.

2 Lösungsidee

Zuerst werden die Daten so strukturiert, dass die Berechnungen schneller ablaufen können. Danach wird jede mögliche Kombination an Streckenlängen ausprobiert und die Beste ausgegeben.

2.1 Laufzeitanalyse

Die Laufzeit hängt von einem Parameter ab, nämlich der Anzahl der potentiellen Teilnehmer n. Da das Programm Brute Force verwendet, hat es eine Laufzeitkomplexität von $O(n^3)$ bei einer Speicherkomplexität von $O(n^2)$.

3 Umsetzung

Die Lösungsidee ist in JavaScript implementiert.

3.1 Einlesen

Die Eingabe wird über ein Eingabefenster (prompt) ausgelesen. Anschließend werden alle ungültigen Zeichen rausgefiltert (für den Fall, dass keine saubere Eingabe gemacht wurde) und die Gültigen in Zahlen umgewandelt. Danach wird überprüft, ob die Eingabe das Format eines Wandertag-Datensatzes hat. Daraufhin werden die Daten als Personen Objekte gespeichert.

3.2 Strukturierung

Dieser Teil strukturiert die Daten so, dass das Programm eine möglichst kurze Zeit zum Ausführen benötigt.

3.2.1 Speicherung als Gruppen

Zuerst wird eine Liste erstellt in der alle Mindest-und Maximallängen der Personen enthalten sind. Diese werden dann nach Länge sortiert. Anschließend werden gleiche Längen in einer Gruppe zusammengefasst. Dies hat zur Folge, dass bei einer Veränderung der Strecke nur die Längen angesteuert werden, die von der Änderung betroffen sind. Dabei wird immer gespeichert welche Person bei welcher Strecke mitwandert und wie viele Personen mitwandern. Dadurch muss man bei einer Verlängerung einer Strecke von einer Gruppe zur nächsten, nur alle Personen der beiden Gruppen aktualisieren. Das sorgt dafür, dass man für die Berechnung der Teilnehmer bei einer Länge nicht mehr zwingend durch alle Personen durchiterieren muss.

3.2.2 Entfernung unnötiger Gruppen

Die Anzahl der Gruppen wird wie folgt minimiert: Immer dann, wenn alle Personen einer benachbarten Gruppe auch bei der Länge der eigenen Gruppe wandern, heißt das, dass die benachbarte Länge in jedem fall uninterressant ist, da sie nicht mehr Teilnehmer haben kann als die eigene. Daher werden dann alle Personen der benachbarten Gruppe der eigenen hinzugefügt und die benachbarte Gruppe gelöscht. Alle Personenduplikate in einer Gruppe werden gelöscht.

3.2.3 Kategoriesierung der Personen

Innerhalb jeder Gruppe werden die Personen jeweils in eine der drei Kategorien eingeteilt: $Mehr$, $Weniger$ oder $Keines$. Wenn eine Person auch bei der Länge der nächstlängeren Gruppe wandern würde, dann fällt diese in die Kategorie $Mehr$. Wenn eine Person auch bei der Länge der nächstkleineren Gruppe wandern würde, dann fällt diese in die Kategorie $Weniger$. Ist eine Person weder in der $Mehr$ noch in der $Weniger$ Gruppe, so fällt sie in die Kategorie $Keines$. Diese Einteilung ermöglicht es, einfacher zu ermitteln, ob eine Person bei einer bestimmten Strecke mitwandert. Jedes Mal wenn eine Strecke die Länge einer Gruppe hat, muss man nur für alle Personen von bestimmten Kategorien die Information, ob sie bei der Strecke mitlaufen, toggeln. Aus der folgenden Tabelle ist entnehmbar, welche Kategorien bei welcher Bewegung relativ zur eigenen Gruppe überprüft werden müssen:

Kürzer ↔ Eigenlänge		Eigenlänge ↔ Länger		Überspringen der Eigenlänge	
$Keines$	$Mehr$	$Weniger$	$Keines$	$Weniger$	$Mehr$

3.2.4 Abkürzungen

Häufig wird eine Strecke von einer bestimmten Länge wieder auf die kürzeste Länge gesetzt. Also wird gespeichert, welche Personen bei welcher Länge durch einen solchen Sprung beeinflusst werden. Beim Ausführen eines Sprungs wird dann auf die Liste für diese Länge zurückgegriffen.

3.3 Brute Force

Für jede Länge der übriggebliebenen Gruppen werden alle Kombinationen durchprobiert. Währenddessen gilt immer: $\text{Strecke}_3 \leq \text{Strecke}_2 \leq \text{Strecke}_1$. Dadurch wird die Anzahl der getesteten Längen erheblich reduziert.

3.4 Ausgeben

Die maximale Anzahl an Teilnehmern und die optimalen Streckenlängen werden formatiert und in die Konsole ausgegeben. Außerdem wird ein Säulendiagramm angezeigt, um darzustellen wie viele Menschen bei welcher Streckenlänge mitwandern (Grau oder Schwarz) bzw. welcher Teilnehmer bei welcher Strecke mitwandert (Rot, Grün, Blau). Die X-Achse stellt die Distanz in m dar. Die Y-Achse enthält die Anzahl der teilnehmenden Personen.

4 Beispiele

Wir rufen nun das JavaScript-Programm mit den verschiedenen BWINF-Eingabedateien auf. Diese Dateien sind in demselben Ordner wie die Programmdatei. Das Programm wird mit Hilfe des Browsers ausgeführt. In dem Dialogfeld können die Daten der Teilnehmer eingegeben werden. Die Ausgabe kann in der Konsole ausgelesen werden. Die Teilnehmer werden als Säulendiagramm im Canvas angezeigt. Das Programm terminiert für alle getesteten Eingaben in weniger als 3s auf einem gewöhnlichen PC.

4.1 Lösungen

4.1.1 wandern1.txt

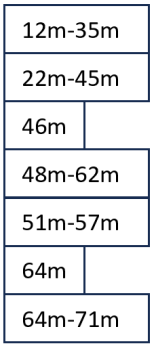


Abbildung 1: Daten nach Einlesung

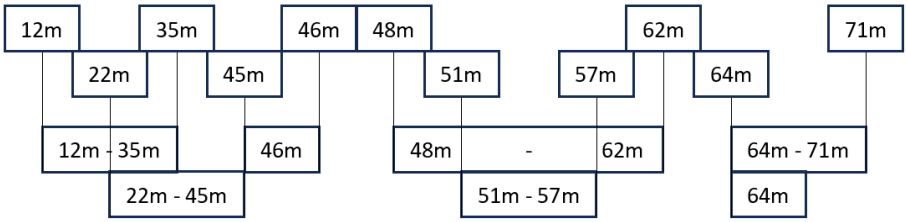


Abbildung 2: Daten nach Gruppierung

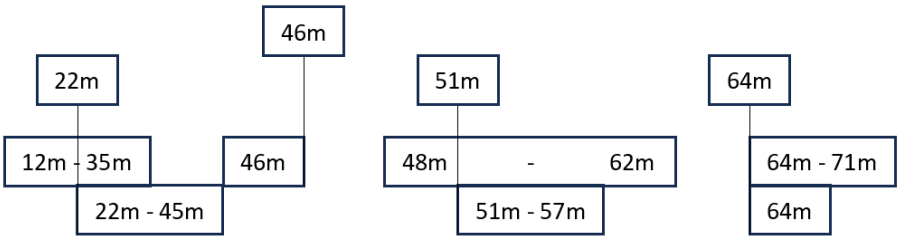
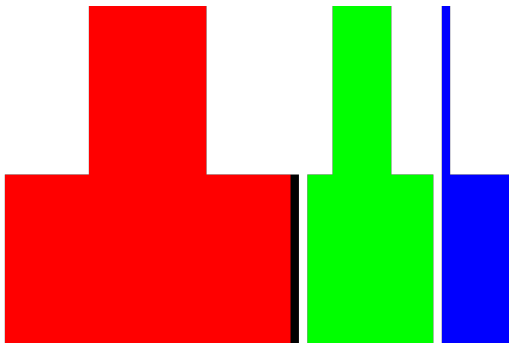


Abbildung 3: Daten nach Minimierung der Gruppen



```
6 Teilnehmer
64m
51m
22m
```

4.1.2 wandern2.txt

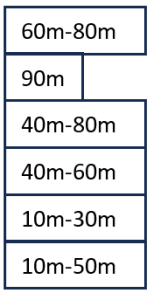


Abbildung 4: Daten nach Einlesung

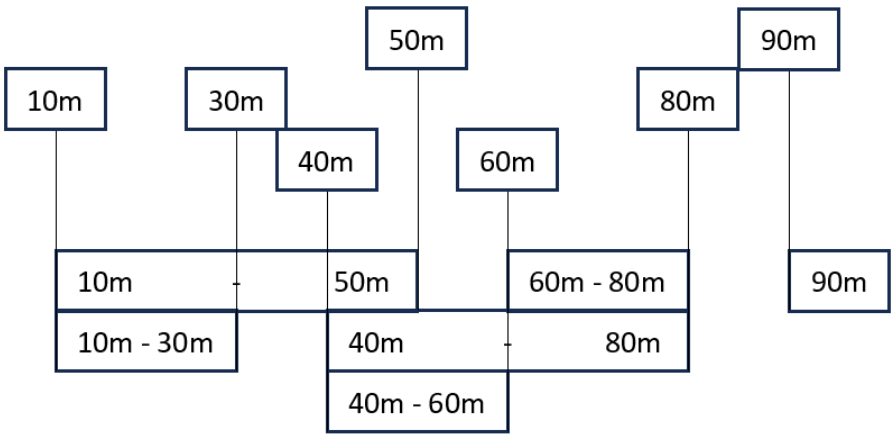


Abbildung 5: Daten nach Gruppierung

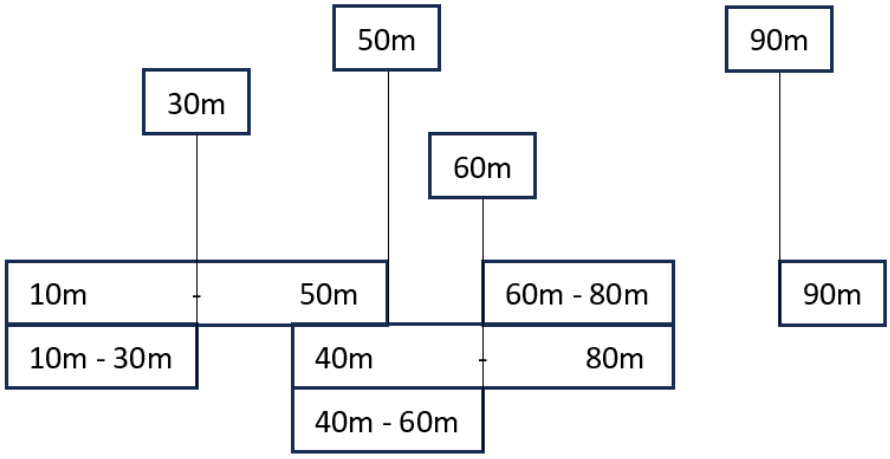
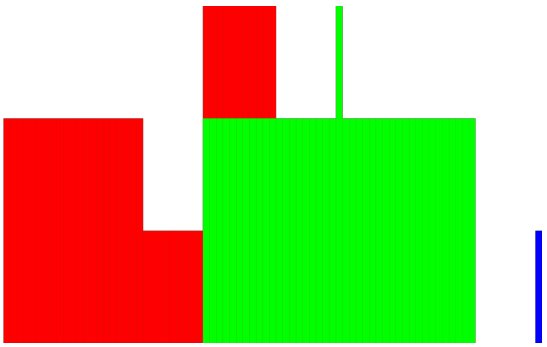
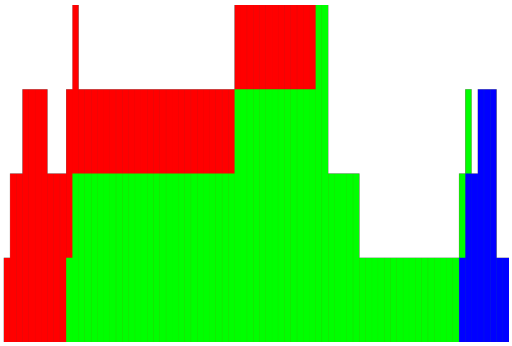


Abbildung 6: Daten nach Minimierung der Gruppen



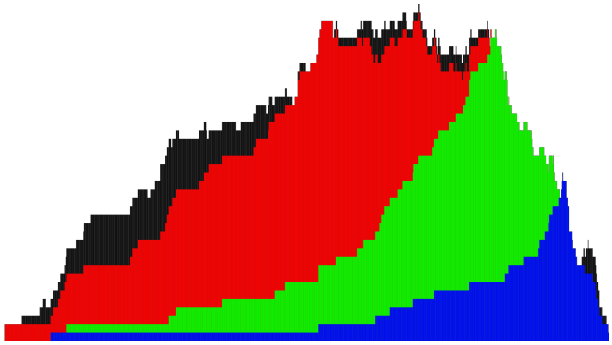
```
6 Teilnehmer
90m
60m
30m
```

4.1.3 wandern3.txt



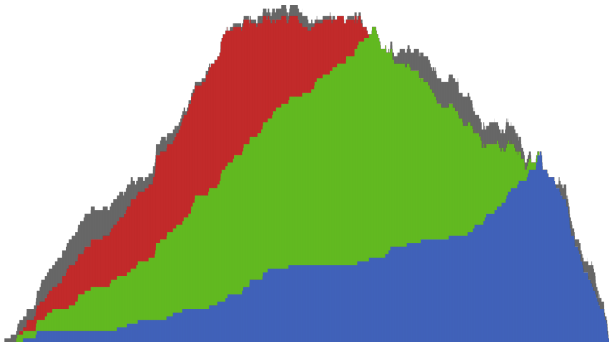
```
10 Teilnehmer
94m
67m
22m
```

4.1.4 wandern4.txt



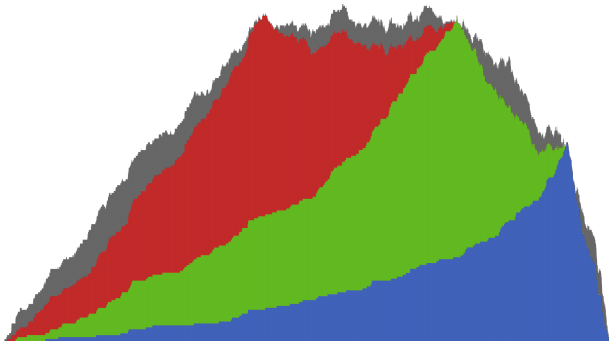
```
79 Teilnehmer
922m
812m
542m
```

4.1.5 wandern5.txt



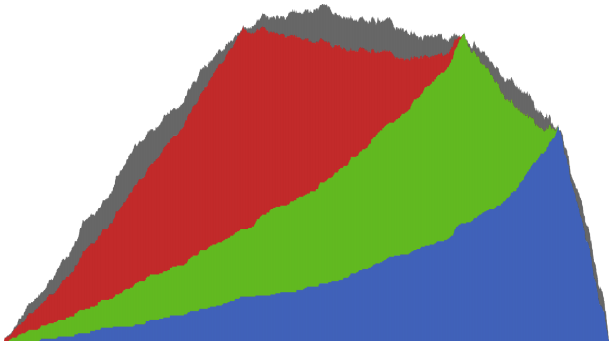
```
153 Teilnehmer
88801m
61478m
36728m
```

4.1.6 wandern6.txt



```
330 Teilnehmer
93177m
74810m
42898m
```

4.1.7 wandern7.txt



```
551 Teilnehmer
91601m
76088m
39543m
```

5 Quellcode

```
const c = document.getElementById("myCanvas")
const ctx = c.getContext("2d")
c.height = window.innerHeight*0.9
c.width = window.innerWidth*0.9
//Lese Eingabe aus
let input = (prompt("Wandertag")|| "0").split(/[\s\n]+/)
let vals = []
//Wandle String in Zahlen um und filtere ungültige Zahlen raus
for(let i = 0;i<input.length;i++){
 let maybe = Number(input[i])|| 0
 if(maybe.toString() == input[i]){
  vals.push(maybe)
 }
}
//Wenn die Eingabe keine Teilnehmer enthielt, stoppe das Programm
if(!(vals.length>2&&(vals.length-1)%2 == 0)){
 throw new Error("Ungültige Eingabe")
}
const size = (vals.length-1)>>2
//klasse für alle mindest oder maximallängen aller Personen
class point {
 constructor(pos,person){
  this.pos = pos
  this.person = person
 }
}
let score = 0
//klasse einer person
class person{
 constructor(line){
  this.start = Number(line[0]<line[1]?line[0]:line[1])
  this.stop = Number(line[0]<line[1]?line[1]:line[0])
  this.count = 0
  this.lines = Array(3).fill(false)
 }
 //funktion um die Teilnahme eines Teilnehmers potentiell zu ändern
 change(path){
  this.lines[path] = !this.lines[path]
```

```
  const dif = (this.lines[path]<<1)-1
  this.count += dif
  if(this.lines[path] == this.count){
   score += dif
  }
 }
}
//klasse mit allen punkten der position
class batch{
 constructor(pos){
 this.pos = pos
 this.persons = []
 this.bigger = []
 this.smaller = []
 this.neither = []
 }
 iter(list,path){
  for(let i = 0;i<list.length;i++){
   list[i].change(path)
  }
 }
 //funktion um alle zu ändern
 all(path){
  this.iter(this.persons,path)
 }
 //funktion wenn diese position nur überflogen wird
 pass(path){
  this.iter(this.bigger,path)
  this.iter(this.smaller,path)
 }
 //funktion falls diese position in richtung einer größeren verlassen wird
 next(path){
  this.iter(this.smaller,path)
  this.iter(this.neither,path)
 }
 //funktion falls diese position in richtung einer kleineren verlassen wird
 before(path){
  this.iter(this.bigger,path)
  this.iter(this.neither,path)
 }
}
let points= []
let persons = []
//Formatiere die Werte als personen
while(vals.length>1){
 let line = vals.splice(-2,2)
 let ref = new person(line)
 persons.push(ref)
 points.push(new point(Number(line[0]),ref))
 points.push(new point(Number(line[1]),ref))
}
//sortiere die punkte und gruppiere sie
points.sort((a,b)=>a.pos-b.pos)
let startPos = points[0].pos
let batches = [new batch(startPos)]
for(let i = 0;i<points.length;i++){
 if(startPos != points[i].pos){
  startPos = points[i].pos
```

```
  batches.push(new batch(startPos))
 }
 batches[batches.length-1].persons.push(points[i].person)
}
//funktion um die anzahl der gruppen zu minimieren
function minimize(a,b,cond,attr){
 for(let i = 0;i<batches.length-1;i++){
  let equal = true
  for(let j = 0;j<batches[i+b].persons.length;j++){
   if(cond(attr(batches[i+b].persons[j]),batches[i+a].pos)){
    equal = false
   }
  }
  if(equal){
   batches[i+a].persons.push(...batches[i+b].persons)
   batches.splice(i+b,1)
   i-=b
  }
 }
}
//wenn die nächstkleinere position eine Teilmenge der eigenen an Teilnehmer ist,
//werden alle personen referenzen kopiert und die kleinere position gelöscht
minimize(1,0,(a,b)=>a<b,a=>a.stop)
//wenn die nächstgrößere position eine Teilmenge der eigenen an Teilnehmer ist,
//werden alle personen referenzen kopiert und die größere position gelöscht
minimize(0,1,(a,b)=>a>b,a=>a.start)
//Merke dir welche Teilnehmer nicht mehr Teilnehmen wenn ich von x nach 0 springe
let jumps = Array(batches.length)
let notStart = batches[0].pos
for(let i = 0;i<batches.length;i++){
 jumps[i] = []
 let pos = batches[i].pos
 for(let j = 0;j<persons.length;j++){
  let start = persons[j].start
  let stop = persons[j].stop
  if((start<=pos&&pos<=stop&&start>notStart)||(start<=notStart&&stop<pos)){
   jumps[i].push(persons[j])
  }
 }
}
//Lösche alle duplikate die es an Personen referenzen in den gruppen gibt
for(let i = 0;i<batches.length;i++){
 batches[i].persons = [...new Set(batches[i].persons)]
}
//Kategorisiere alle personen in einer gruppe
for(let i = 0;i<batches.length-1;i++){
 for(let j = 0;j<batches[i].persons.length;j++){
  if(batches[i].persons[j].stop>=batches[i+1].pos){
   //alle die auch bei größeren positionen laufen
   batches[i].bigger.push(batches[i].persons[j])
  }else{
   //alle die nur bei dieser position laufen
   batches[i].neither.push(batches[i].persons[j])
  }
 }
}
batches[batches.length-1].neither = [...batches[batches.length-1].persons]
for(let i = batches.length-1;i>0;i--){
```

```
 for(let j = 0;j<batches[i].neither.length;j++){
  if(batches[i].neither[j].start<=batches[i-1].pos){
   //alle die bei einer kleineren position laufen
   batches[i].smaller.push(batches[i].neither[j])
   batches[i].neither.splice(j,1)
   j -= 1
  }
 }
}
//klasse für jede organisierte strecke
class path{
 constructor(id){
  this.batchPos = 0
  this.id = id
  batches[0].all(this.id)
 }
 //funktion um zur position 0 zu springen
 jump(){
  for(let i = 0;i<jumps[this.batchPos].length;i++){
   jumps[this.batchPos][i].change(this.id)
  }
  this.batchPos = 0
 }
 //funktion um sich um eins vorwärts zu bewegen
 step(){
  batches[this.batchPos].next(this.id)
  this.batchPos++
  batches[this.batchPos].before(this.id)
 }
}
let paths = [new path(0),new path(1),new path(2)]
let highscore = score
let bestConfig = [0,0,0]
//Probiere alle streckenkombinationen durch
for(let i = 0;i<batches.length-1;i++){
 paths[0].step()
 paths[1].jump()
 for(let j = 0;j<=i;j++){
  paths[2].jump()
  for(let l = 0;l<=j;l++){
   //merke dir die beste kombination
   if(highscore<score){
    highscore = score
    bestConfig = [i+1,j,l]
   }
   paths[2].step()
  }
  paths[1].step()
 }
}
//Formatiere das ergtebnis
let out = highscore + " Teilnehmer"
for(let i = 0;i<3;i++){
 bestConfig[i] = batches[bestConfig[i]].pos
 out += "\n"+bestConfig[i] + "m"
}
//gebe die lösung in die console aus
console.log(out)
```

```
let max = 0
//funktion um ein Säulendiagramm anhand einer liste zu malen
function draw(list,r,g,b){
 for(let i = 0;i<list.length;i++){
  max = Math.max(max,list[i])
 }
 mulx = c.width/list.length
 muly = c.height/max
 for(let i = 0;i<list.length;i++){
  ctx.fillStyle = `rgb(${255*r} ${255*g} ${255*b})`
  ctx.fillRect(mulx*i,c.height-list[i]*muly,mulx,list[i]*muly)
 }
}
let total = Array(points[points.length-1].pos+1)
let pathSum = [...total]
let last = 0
let pathLast = [0,0,0]
let pointPos = 0
let pathPointPos = [0,0,0]
//setze die Daten der personen zurück
for(let i = 0;i<persons.length;i++){
 persons[i].count = 0
 persons[i].lines = Array(3).fill(false)
}
//Für jede mögliche position
for(let i = 0;i<total.length;i++){
 //kopiere die werte der letzten position
 total[i] = last
 pathSum[i] = [...pathLast]
 //für alle verlassenen Teilnehmer
 for(let j = pointPos-1;j>-1&&points[j].pos==i-1;j--){
  let chosen = points[j].person
  //reduziere sie vom ergebnis
  if(chosen.stop == i-1&&chosen.count == 1){
   total[i]--
   chosen.count--
   for(let l = 0;l<3;l++){
    if(chosen.lines[l]){
     chosen.lines[l] = false
     pathSum[i][l]--
    }
   }
  }
 }
 //für alle neuen Teilnehmer
 for(;pointPos<points.length&&points[pointPos].pos==i;pointPos++){
  let chosen = points[pointPos].person
  //füge sie zum ergebnis zu
  if(chosen.start == i&&chosen.count == 0){
   total[i]++
   chosen.count++
   let tick = false
   for(let j = 0;j<3;j++){
    if(((chosen.start<=bestConfig[j]&&bestConfig[j]<=chosen.stop)||tick)&&!chosen.lines[j]){
     chosen.lines[j] = true
     pathSum[i][j]++
     tick =true
    }
```

```
   }
  }
 }
 pathLast = pathSum[i]
 last = total[i]
}
//male das Diagramm für alle positionen
draw(total,0,0,0)
//male die Diagramme für die einzelnen Strecken über das gesammtdiagramm
pathTotal = [[],[],[]]
for(let i = 0;i<pathSum.length;i++){
 pathTotal[0][i] = pathSum[i][0]
 pathTotal[1][i] = pathSum[i][1]
 pathTotal[2][i] = pathSum[i][2]
}
draw(pathTotal[2],1,0,0)
draw(pathTotal[1],0,1,0)
draw(pathTotal[0],0,0,1)
```