Aufgabe 3: Eisbudendilemma

Teilnahme-ID: 57048

Bearbeiter/-in dieser Aufgabe:
Frederico Aberle

19. April 2021

Inhaltsverzeichnis

1 Lösungsidee

1.1 Brute-Force mit Optimierung

Die Idee ist es alle Möglichkeiten auszuprobieren. Es wird also nach und nach der Standort der ersten, zweiten und dritten Eisbude verändert und anschließend überprüft, ob der Standort der drei Eisbuden valide ist.

Um herauszufinden, ob ein Standort valide ist, berechnen wir für jeden Punkt auf dem Umfang des Sees, wie die Dorfbewohner abstimmen würden. Man weiß, dass jedes Haus bis zur nächstgelegenen Eisbude für 'Ja' stimmen würde (also für eine neue Standortverlegung), ansonsten für 'Nein'. Das gleiche würde das Haus in die andere Richtung machen mit der gleichen Distanz wie bis zur nächstgelegenen Eisbude, da sich auch hier eine Wegverkürzerung zu dem Haus stattfinden würde.

Danach soll aus allen Abstimmungen für jeden Punkt des Umfangs des Sees die größtmögliche Ausbeute bestimmt werden. Es werden also drei neue Standorte an Eisbuden gewählt, damit die Abstimmung an diesen drei Punkten zusammen die größtmöglichste Anzahl an Stimmen erfährt.

Falls nun von allen Möglichkeiten es eine gibt, bei der die größtmöglichste Anzahl an Stimmen eines Gegenvorschlags kleiner oder gleich der Hälfte der Anzahl der Häuser ist, dann soll diese Möglichkeit mit den Standorten von den Eisbuden ausgegeben werden.

2 Umsetzung

Die ersten beiden Eisbudenstandorte sollen Adressen von Häusern sein. Der dritte Standort soll eine beliebige Adresse vom See sein. Wenn man nämlich alle drei Standorte beliebig wählt, also alle Möglichkeiten durchprobiert, hat sich bei der Ausgabe herausgestellt, dass es fast immer nur Lösungen gab bei denen mindestens zwei Eisbuden Adressen von Häusern waren. Dies ist auch logisch, da bei einem Gegenvorschlag von drei neuen Eisbuden die Häuser auf denen die ursprünglichen Häuser liegen immer gegen den neuen Vorschlag stimmen würden. Am Schluss soll dies erklärt werden.

Wir haben drei ineinander geschachtelte for-Schleifen für die jeweiligen Eisbuden. Nun soll für jede Iteration die Abstimmung der Einwohner ermittelt werden und in der Liste 'result' abgespeichert werden. Um die Liste 'v' (die Abstimmung) zu bestimmen, wird für jede Adresse der Abstand d zur nächstgelegenen Eisbude berrechnet. Wenn x für die Position des Hauses steht, dann soll die Stimme für den Standort x bei v[x-d:x+d+1] hinzugefügt werden. Falls der Abstand links von der Position 0 überlappen sollte, wird getrennt hinzugefügt. Einmal bei v[-(d-x):] und bei v[:x+d+1]. Analog wird bei rechter Überlappung die Stimme bei v[x-d:] und v[:d-(c-1-x)] hinzugefügt, wobei c der Umfang des Sees sei. Am Ende soll noch für die Position der Eisbuden eine leere Menge zugeordnet werden, um sicher zu gehen, dass für diese Standortwahl niemand abgestimmen kann. Außerdem soll noch am Ende von der Liste v die Postion der drei Eisbudenstandorte gespeichert werden, damit wir bei der Ausgabe wissen für welche Standorte nun diese Abstimmung war.

Nachdem nun alle Abstimmungen für jede Iteration in result gespeichert sind, soll nun die Abstimmung herausgefiltert werden, bei denen ein Gegenvorschlag mit drei neuen Eisbuden maximale Stimmen erfahren würde. Sind die maximalen Stimmen vom Gegenvorschlag kleiner oder gleich die Hälfte der Anzahl der Häuser, so ist ein valider Standort der ursprüglichen Eisbuden gefunden. Bei der Bestimmung der maximalen Stimmen von den drei Eisbuden aus einem Gegenvorschlag muss man jedoch bedenken, dass es nicht immer Positionen sind, die die größte Anzahl an Stimmen aufweisen. Wenn z.B. Position 1, 2 und 3 die größte Anzahl an Stimmen aufweisen (z.B. 8, 8 und 7) sind die Stimmen von Position 1 auch Teil von den Stimmen aus Position 2 und sind somit nicht geeignet als Eisbudenstandorte für die maximale Ausbeute. Die Funktion def maxV(v) soll die maximale Ausbeute bestimmen. Zunächst iteriert man durch v und streicht alle mehrfach vorkommenden Abstimmungen von einzelen Positionen in v. Die Position mit den größten Stimmen in v wird auf 'largest' gesetzt. Danach betritt man eine while-Schleife solange noch nicht drei Eisbuden mit der maximalen Ausbeute gefunden wurden. Dann betritt man noch eine for-Schleife, die bis zur nächsten Position in v mit largest Anzahl an Stimmen geht. Falls die Position kleiner als largest-Stimmen hat soll einfach weitergegangen werden. Wenn drei Eisbuden gefunden wurden, soll die Schleife verlassen werden. An einer Position mit largest-Stimmen angekommen, soll nun überprüft werden, ob die Position mit largest-Stimmen sich auch wirklich um largest-Stimmen von den anderen schon gefundenen Eisbuden unterscheidet und nicht das oben erwähnte Problem auftritt. Falls es für largest kein Eisbudenstandort gefunden wurde, wird largest um 1 verringert, um nach den Standort mit den nächstgrößten Stimmen zu suchen. Eine Abstimmung muss ja nicht unbedingt drei oder mehr Standorte mit largest-Stimmen haben.

Dabei muss die maximale Ausbeute nicht immer aus Eisbudenstandorten bestehen von denen sich alle Stimmen unterscheiden. Oftmals ist ein Standort mit mehr Stimmen, die sich in nur wenigen Stimmen von den anderen Eisbudenstandorten der Liste 'e' unterscheiden, besser als ein Standort mit weniger Stimmen, die sich in keinen Stimmen von den anderen Eisbudenstandorten unterscheiden. Oder vielleicht sogar gleich gut. Bsp.: e = [[1, 2, 3, 4], [20, 21, 22, 23]]. Dann ist z.B. [4, 5, 6, 7] besser als [5, 6], da nach Abzug von der 4 in [4, 5, 6, 7] immer noch die drei Stimmen [5, 6, 7] sich von e[0] unterscheiden und mehr sind. Die Funktion def isDifferent(v[i], x, largest) soll das überprüfen. Dazu überprüft man, ob die von v[i] in x (Eisbudenstandort aus e) vorkommenden Stimmen sich maximal um len(v[i])-largest unterscheiden. Bei wahrer Aussage ist ein Eisbudenstandort gefunden, der der Liste e hinzugefügt wird. Der Zähler 'total' wird um largest erhöht und zählt von den drei Eisbuden, die sich unterscheidenden Stimmen. Nach Vollendung der for-Schleife wird, wie schon ob beschrieben, largest um 1 verringert.

Nach der Auswertung der Abstimmung nach der maximalen Stimmenausbeute, soll nun entschieden werden, ob die Eisbuden der Abstimmung valide Orte sind, also kein Gegenvorschlag durchkommen würde. Wenn der ausgegebene Wert result von der Funktion maxV(v) kleiner oder gleich die Anzahl der Häuser ist, sollen die letzten drei Elemente von v, also die gespeicherten Eisbudenstandorte, ausgegeben werden.

Hinweis: Es hat sich herausgestellt, dass maxV(v) nicht immer die größtmöglichste Anzahl an Stimmen für drei Eisbuden eines Gegenvorschlags berechnet. Es gibt einen Sonderfall. Es können aber die Lösungen drastisch minimiert werden. Unter den wenig verbliebenen Lösungen können aber diese mit Brute-Force

auf ihre Richtigkeit überprüft werden. Die Funktion def isValid(v) iteriert in drei Schleifen durch alle Möglichkeiten für die Eisbuden. Dann werden für jede Möglichkeit die Stimmen der drei Eisbuden verglichen und untersucht wie viele auch nur einzeln vorkommen. Wenn es von all diesen Möglichkeiten die einzeln vorkommenden Stimmen größer als die Hälfte der Anzahl der Häuser ist, dann ist diese Lösung ungültig, andernfalls war die Berechnung von maxV(v) richtig.

2.1 Optimierung

Im Anhang sind die Position des Sees aus Beispiel 3 zu sehen. Beide Bilder sind Lösungen dieses Beispiels mit den Eisbuden 0, 17, 42 und den Eisbuden 1, 17, 42. Beispiel 3 zeigt den Zusammenhang der Lösung besser als Beispiel 1, da es in Beispiel 1 zu wenig Häuser und Positionen und damit Stimmen gibt.

Der Grund für diese Lösung ist die perfekte Verteilung der Stimmen. Es gibt nicht an einer Postition 9 Stimmen und an allen anderen Postition 0, sondern genau umgekehrt. Von der zweiten bis zur dritten Eisbude sind es immer maximal 2 Stimmen und danach geht es auch mit 2 Stimmen für eine Zeit lang weiter bis dann 3 und 4 Stimmen kommen. Wenn man also einen Gegenvorschlag machen müsste, würde man die vier Stimmen 9, 12, 13 und 15 nehmen. Danach stehen nur noch zwei unbenutzte Stimmen zur Auswahl, sodass sich die maximale Ausbeute von 4 + 2 + 2 = 8 ergibt (gleich die Hälfte der Anzahl der Häuser).

Das lässt sich so erklären, dass zwei Eisbuden Häuser sein müssen. Diese sind fix und haben zwischen ihnen lauter Positionen mit gleich vielen Stimmen (hier von 17 bis 42). Die andere Eisbude soll sich folglich auf der anderen Seite der beiden Eisbuden (hier von 42 bis 17) hin- und herbewegen. Dabei soll sie die Stimmen möglichst minimieren. Das heißt, links bzw. rechts von der Eisbude bis zur nächsten Eisbude müssen die maximalen Stimmen genauso groß wie zwischen den beiden Eisbudenhäusern sein (also hier maximal 2 Stimmen). Schließlich muss noch überprüft werden, ob 2 mal die maximalen Stimmen zwischen den Eisbudenhäusern plus die maximalen Stimmen von dem dritten Bereich mit den restlichen Stimmen (hier von 0 bzw. 1 bis 17) kleiner oder gleich die Hälfte der Anzahl der Häuser ist. In diesem Beispiel, wie schon erwähnt, 2 * 2 + 4 <= 8. Aus dieser Erkenntnis lässt sich nun eine Bedingung herleiten. Es gilt 2*x + y = N//2, wobei x die maximale Stimmenanzahl zwischen den beiden Eisbudenhäusern ist und y die maximale Stimmenanzahl der restlichen Stimmen ist. Diese sind zusammen maximal N//2 groß. Nach y umgestellt ergibt sich: y = N//2 - 2*x. Außerdem ist bekannt, dass y > x sein muss, also folgt: y = N//2 - 2*x > x. Nach N//2 umgestellt ergibt sich N//2 > 3*x und schließlich nach x umgestellt x < (N//2) / 3. Wir können also aus der bekannten Anzahl der Häuser x bestimmen, die maximale Stimmenanzahl zwischen den beiden Eisbudenhäusern.

Wir iterieren in einer Schleife durch die Häuser als ersten Eisbudenstandort und in einer zweiten Schleife wieder durch die Häuser als zweiten Eisbudenstandort. Dabei wird überprüft, ob die Bedingung x < (N//2) / 3 erfüllt ist, ansonsten können wir mit dem nächsten Haus fortfahren. Die Größe x können wir mit der Funktion maxV(countV(v)) aus unserer ersten Lösungsidee bestimmen. Der dritte Eisbudenstandort iteriert auf der anderen Seite zwischen den Eisbudenhäusern. Auch hier kann mit maxV(countV(v)) x und y bestimmt werden und bei falscher Bedingung x < (N//2) / 3 oder 2*x + y < N//2 zur nächsten Position gesprungen werden.

Die Laufzeit lässt sich schwer bestimmen, aber sie ist um einiges besser, da diese Lösung nicht mehr auf reines Brute-Force basiert. Vielleicht sogar gut genug für die Beispiele 6 und 7.

Aus Zeitgründen konnte diese Optimierung nicht mehr implementiert werden.

3 Laufzeit

3.1 ohne Optimierung

Wir haben drei for-Schleifen. Zwei für die Häuser und eine für jede beliebige Adresse. Dann berechnen wir noch für jedes Haus die Abstimmung. Es ergibt sich

$$\mathcal{O}(\frac{N \cdot (N-1)}{2} \cdot c \cdot N) \tag{1}$$

wobei N die Anzahl der Häuser ist und c der Umfang des Sees.

4 Beispiele

4.1 eisbuden1.txt

[2, 8, 14]
[2, 12, 14]
[2, 12, 15]
[2, 5, 14]
[2, 6, 14]
[2, 7, 14]
[2, 8, 14]
[2, 9, 14]
[2, 10, 14]
[2, 11, 14]
[2, 12, 14]
[2, 10, 15]
[2, 11, 15]
[2, 12, 15]
[2, 8, 14]
[2, 12, 14]
[2, 12, 15]
— 0.023669004440307617 seconds —
Hinweis: Lösungen werden mehrach abgedruckt, da die Eisbuden in vertauschter Reihenfolge nochmals vorkommen.

4.2 eisbuden2.txt

Keine Lösung
— 0.7086191177368164 seconds —

4.3 eisbuden3.txt

[0, 17, 42]
[1, 17, 42]
— 0.8902971744537354 seconds —
Hinweis: Hier wäre [1, 18, 42] auch eine Lösung gewesen. Jedoch ist in dieser Lösung nur eine Eisbude gleichzeitig die Adresse eines Hauses, sodass diese nicht abgedruckt wird.

4.4 eisbuden4.txt

Keine Lösung
— 4.386654376983643 seconds —

4.5 eisbuden5.txt

Keine Lösung
— 37.710702896118164 seconds —

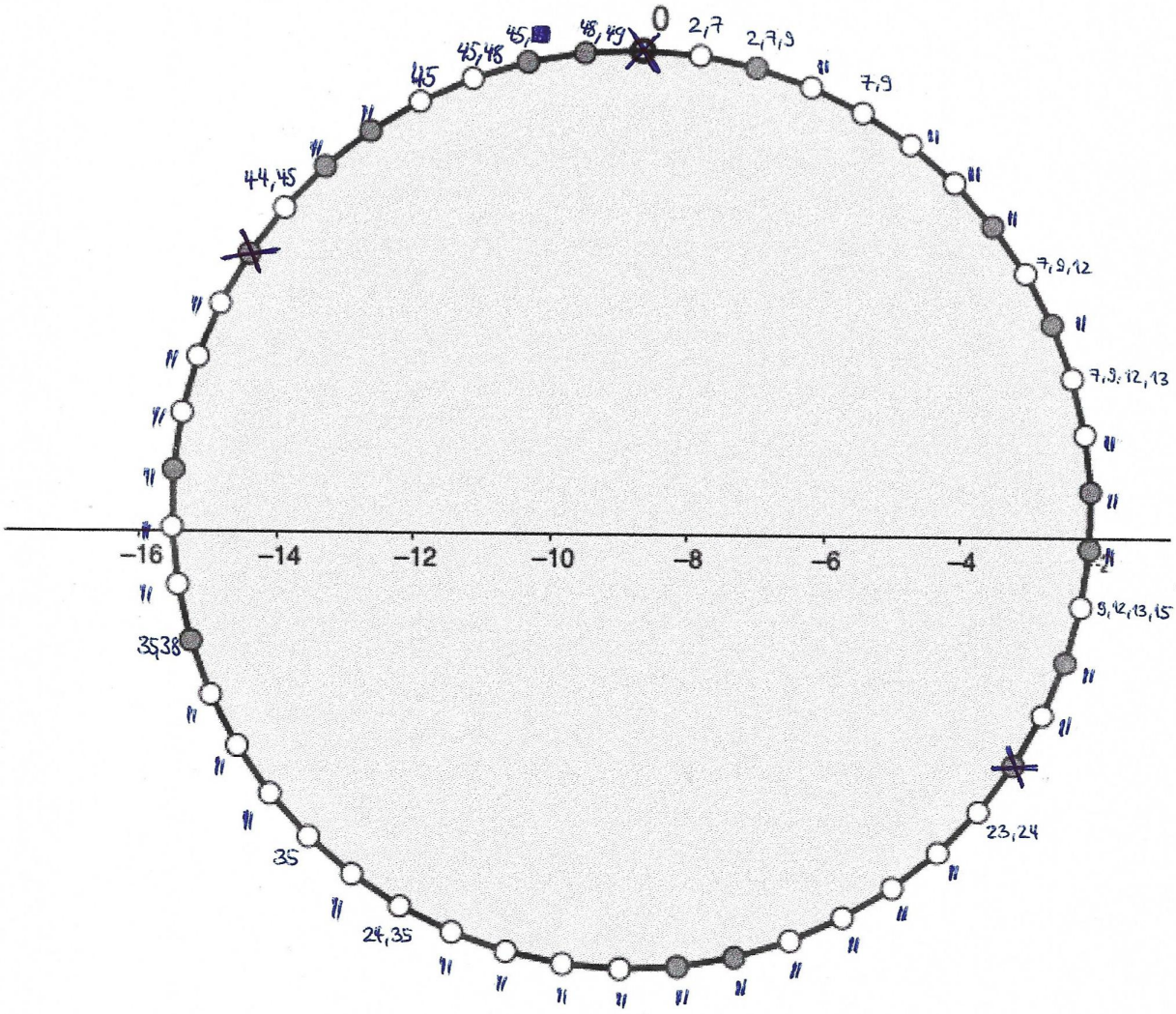
5 Quellcode

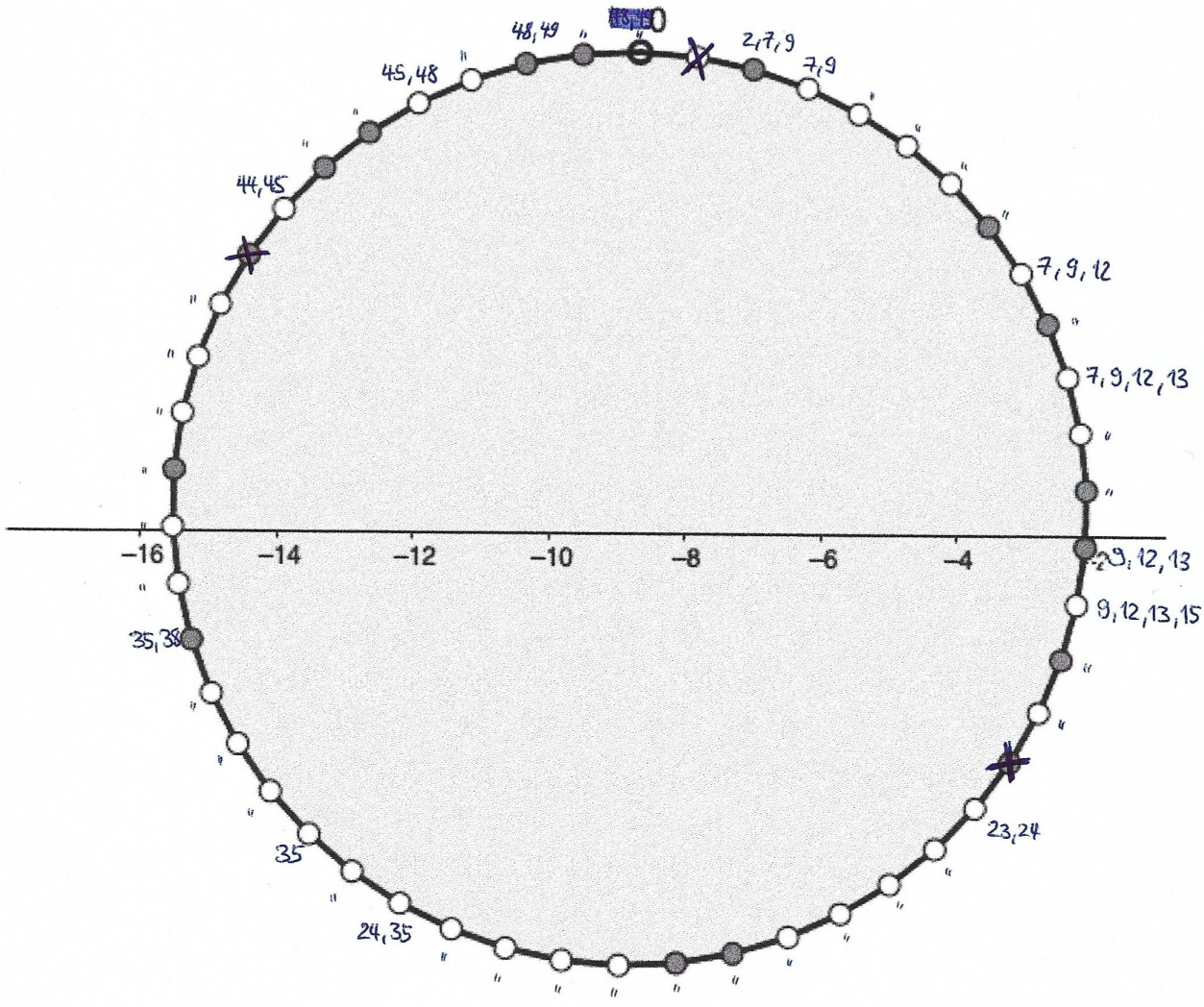
```
"""Eingabe"""
file = open('eisbuden5.txt')
firstLine = file.readline().split()
c = int(firstLine[0])  # Umfang des Sees ('circumference')
N = int(firstLine[1])  # Anzahl der Haeuser
# print(c, N)
a = [int(x) for x in file.readline().split()]  # Adressen der Haeuser ('address')
# print(a)


"""Teil 1"""
```

```
# bestimmt die Distanz von einer Adresse zur Eisbude
def dist(x, e, c):
    if x == e:
        return 0
    elif abs(x-e) > c//2:
        return abs(c-abs(x-e))-1
    else:
        return abs(x-e)-1


# fuegt in der Liste v an jeder Position des Sees die Haeuser ein,
# die fuer eine Eisbude an dieser Position stimmen wuerden
def countV(x, c, v, d):
    if x-d < 0:  # distance ueberlappt links von 0
        for i in range(x+d+1):
            # v[i] += 1
            v[i].append(x)
        for i in range(-(d-x), 0):
            # v[i] += 1
            v[i].append(x)
    elif x+d >= c:  # distance ueberlappt rechts von 0
        for i in range(x-d, len(v)):
            # v[i] += 1
            v[i].append(x)
        for i in range(d-(c-1-x)):
            # v[i] += 1
            v[i].append(x)
    else:  # distance ueberlappt gar nicht
        for i in range(x-d, x+d+1):
            # v[i] += 1
            v[i].append(x)


result = [] # speichert alle votes, also alle Listen v
for x in range(0, N-1):
    i = a[x]
    for y in range(x+1, N):
        j = a[y]
        for k in range(0, c):
            if (k == i) or (k == j):  # es koennen keine zwei oder drei Eisbuden am gleichen Ort sein
                continue
            v = [[] for x in range(c)]  # speichert die Stimmen fuer jede Position am See
            for x in a:
                d = min(dist(x, i, c), dist(x, j, c), dist(x, k, c))
                countV(x, c, v, d)
            v[i] = []
            v[j] = []
            v[k] = []
            v.append(i)
            v.append(j)
            v.append(k)
            result.append(v)


"""Teil 2"""
# berechnet die unterschiedlichen Stimmen zu den Stimmen von den bereits bestimmten Eisbuden in e
def isDifferent(l1, l2, largest):
    counter = 0
    for x in l1:
        if x in l2:
            counter += 1
    if counter <= len(l1)-largest:
        return True
    return False


# berechnet (fast immer) die groesstmoeglichste Anzahl an Stimmen
# fuer drei Eisbuden eines Gegenvorschlags
def maxV(v):
    v = v[:-3]
    temp = []
    # streicht alle mehrfach vorkommenden Stimmen
    for i in range(len(v)):
```

```
        if v[i] != v[i-1] and v[i] != []:
            temp.append(v[i])
    v = temp
    e = [] # hier sollen die drei groessten Stimmen gespeichert werden
    total = 0 # die Anzahl der unterscheidbaren Elemente von den Stimmen (maximale Stimmen)
    largest = max([len(x) for x in v]) # groesste Laenge von den Stimmen
    while len(e) != 3:
        for i in range(len(v)):
            if len(e) == 3:
                break
            elif len(v[i]) < largest:
                continue
            else:
                bool = True
                for x in e:
                    if not isDifferent(v[i], x, largest):
                        bool = False
                if bool:
                    e.append(v[i])
                    total += largest
        largest -= 1
    return [total, e, v]


# mit Brute-Force werden die wenigen verbliebenen Loesungen auf Richtigkeit ueberprueft
def isValid(v):
    for i in range(len(v)-2):
        for j in range(i+1, len(v)-1):
            for k in range(j+1, len(v)):
                counter = 0
                for x in v[i]:
                    if x in v[j]:
                        counter += 1
                    if x in v[k]:
                        counter += 1
                for y in v[j]:
                    if y in v[k] and y not in v[i]:
                        counter += 1
                # nun werden alle unterschiedlichen Elemente ausgegeben
                if len(v[i]) + len(v[j]) + len(v[k]) - counter > N//2:
                    return False
    return True


#result = sorted(result, key=lambda x: max([len(y) for y in x[:-3]]))

"""Ausgabe"""
bool = True
for i in range(len(result)):
    v = result[i]
    # print(v)
    x = maxV(v)
    # print(x[0],x[1], x[2])
    if x[0] <= N//2:
        if isValid(x[2]):
            print(sorted([v[-3], v[-2], v[-1]]))
            bool = False
if bool:
    print('Keine Loesung')
```

6 Anhang




0
48, 49
2, 7, 9
7, 9
45, 48
44, 45
7, 9, 12
7, 9, 12, 13
9, 12, 13
9, 12, 13, 15
23, 24
35
24, 35
35, 38
−16
−14
−12
−10
−8
−6
−4