Aufgabe 3: Voll Daneben

3.1 Problemstellung

Al Capone Junior bekommt von den Teilnehmern eine Liste mit n Zahlen. Wir ordnen diese Liste der Größe nach und nennen ihre Einträge $t_1, \ldots, t_n$. Das Ziel von Al ist es, m Zahlen $d_1, \ldots, d_m$ so zu wählen, dass die Summe der Abstände von t_i zu den jeweils nächsten d_j möglichst gering wird. In der Aufgabe war zwar nur $m = 10$ verlangt, aber wir lösen dies allgemein.

Eine Lösung von Al heißt *optimal*, wenn er seine Auszahlung nicht mehr weiter verringern kann, egal wie er seine Zahlen wählt. Der Algorithmus in dieser Musterlösung berechnet immer eine optimale Lösung für Al.

Beachte, dass es mehrere optimale Lösungen geben kann. Angenommen Al darf nur eine Zahl bestimmen. Haben wir zwei Teilnehmer mit den Zahlen 10 und 20, so kann Al jede beliebige Zahl zwischen 10 und 20 wählen und die Ausschüttung beträgt in diesen Fällen genau 10 Taler, was auch optimal ist. Wählt Al dagegen eine Zahl unter 10 oder über 20, so liegt seine Auszahlung über 10 Taler.

3.2 Erste Ideen

Ein Problem ist, dass Al sehr viele Wahlmöglichkeiten für jede einzelne seiner Zahlen hat. Um diese Möglichkeiten etwas einzuschränken, benutzen wir folgende Information:

Für Al gibt es immer eine optimale Lösung, die ausschließlich Zahlen der Teilnehmer verwendet.

Wir wollen uns überlegen, wieso dies der Fall ist. Nehmen wir an, Als Lösung enthält eine Zahl d, die von keinem Teilnehmer gewählt wurde. Ist diese Zahl d kleiner als alle Teilnehmerwerte, so kann keine optimale Lösung vorliegen. Wir könnten dann einfach d in Richtung der Teilnehmerwerte verschieben und erhalten eventuell eine bessere Lösung (falls d die Auszahlung tatsächlich beeinflusst), jedenfalls keine schlechtere. Das Gleiche passiert natürlich auch, wenn d größer als alle Teilnehmerwerte ist. Der interessante Fall ist somit, wenn Als Zahl d zwischen zwei Teilnehmerwerten t_i und t_{i+1} liegt.

Liegt eine Teilnehmerzahl t_j näher an d als alle anderen Zahlen von Al, so sagen wir, dass t_j von d *getroffen* wird. Die Liste der von d getroffenen Teilnehmerwerte können wir in zwei Gruppen aufteilen, nämlich diejenigen Teilnehmerwerte, die kleiner als d sind und diejenigen, die größer als d sind. Zur Abkürzung schreiben wir:

$L(d) =$ Liste der von d getroffenen Teilnehmerwerte, die kleiner als d sind.
$R(d) =$ Liste der von d getroffenen Teilnehmerwerte, die größer als d sind.

Je nachdem, ob die Anzahl $|L(d)|$ oder $|R(d)|$ größer ist, verschieben wir Als Zahl d entweder auf den Wert t_i oder t_{i+1}. Alle anderen seiner Zahlen lassen wir dabei gleich. Durch die Verschiebung von d verändern sich auch die Listen $L(d)$ und $R(d)$.

Im Fall $|L(d)| \geq |R(d)|$ verschieben wir d auf t_i. Diese Verschiebung hat folgende Auswirkungen:

a) Für alle Teilnehmer aus $L(d)$ verringert sich die Auszahlung um $d - t_i$.

b) Für alle Teilnehmer aus $R(t_i)$ erhöht sich die Auszahlung um $d - t_i$.

c) Beim Wechsel von $L(d)$ zu $L(t_i)$ kommen eventuell neue Teilnehmer dazu. Für diese verringert sich die Auszahlung um einen Betrag zwischen und 0 und $d - t_i$.

d) Beim Wechsel von $R(d)$ zu $R(t_i)$ fallen eventuell alte Teilnehmer weg. Für diese erhöht sich die Auszahlung um einen Betrag zwischen und 0 und $d - t_i$.

e) Für alle restlichen Teilnehmer bleibt die Auszahlung konstant.

Wegen der Bedingung $|L(d)| \geq |R(d)|$ sind die zusätzlichen Auszahlungen bei b) und d) auf keinen Fall größer als die Einsparungen in a) und c). Durch die Verschiebung verschlechtert sich also Als Lösung nicht. Es kann zwar passieren, dass Al lediglich eine andere gleich gute Anordnung findet, aber in der Regel wird die neue Anordnung sogar günstiger sein.

Im Fall $|L(d)| < |R(d)|$ verschieben wir d auf t_{i+1}. Hier treten die gleichen Auswirkungen wie vorhin auf, außer dass jetzt alles spiegelverkehrt ist.

Beispiel:

Die Teilnehmer wählen die Zahlen 7, 10, 14, 16, 18, 19, 20, 24, 26, 29, 30 und 32. Al soll drei Zahlen wählen. Diese wählt er als 10, 23 und 30.

Die blauen Punkte sind die Zahlen der Teilnehmer, die roten Striche sind Als Zahlen. Der eingefärbte Bereich wird von der Zahl 23 getroffen. Wir sehen, dass auf der linken Seite mehr Zahlen von 23 getroffen werden als auf der rechten Seite. Folglich verschieben wir die Zahl 23 auf den linken Nachbar 20.

Durch die Verschiebung $23 \to 20$ wird jetzt 16 getroffen und 26 nicht mehr. Die Auszahlungen ändern sich wie folgt:

Teilnehmer	7	10	14	16	18	19	20	24	26	29	30	32
Ausz. vorher	3	0	4	6	5	4	3	1	3	1	0	2
Ausz. nachher	3	0	4	4	2	1	0	4	4	1	0	2
Änderung	0	0	0	-2	-3	-3	-3	+3	+1	0	0	0

Insgesamt zahlt Al so 7 Taler weniger.

3.3 Zerlegung in Teilprobleme

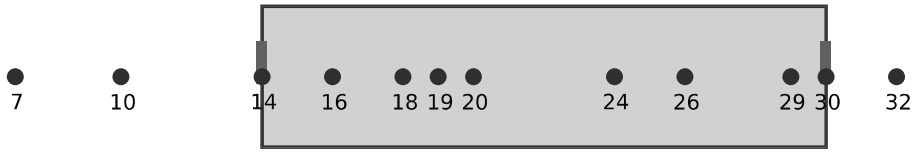
Dank der vorherigen Überlegung können wir davon ausgehen, dass Al alle seine Zahlen aus der Liste der Teilnehmer auswählt. Dies reduziert schon mal die Anzahl der Möglichkeiten, aber es sind immer noch zu viele. Deswegen brauchen wir weitere Ideen.

Oft lassen sich Probleme in der Informatik dadurch lösen, indem man sie in kleinere *Teilprobleme* zerlegt, diese jeweils für sich löst und anschließend die Teilergebnisse wieder zusammensetzt. Dieser Ansatz funktioniert auch hier, wenn wir uns klarmachen, was unsere Teilprobleme sind.

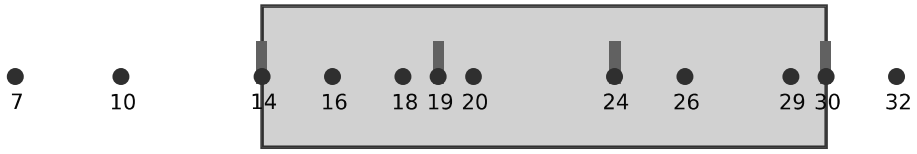
Nehmen wir an, Al hat sich schon zwei Teilnehmerwerte t_i und t_j auf irgendeine Art und Weise ausgesucht. Zwischen t_i und t_j sollen sich bislang noch keine weiteren Zahlen von Al befinden. Nun möchte sich Al l weitere Zahlen in diesem Intervall $[t_i, t_j]$ aussuchen. Unser Teilproblem besteht nun darin, diese l Zahlen so zu berechnen, dass die Summe der Auszahlungen für alle Teilnehmer innerhalb dieses Intervalls $[t_i, t_j]$ minimal wird (die anderen Teilnehmer interessieren uns erstmal nicht). Haben wir l solcher passenden Zahlen gefunden, so sprechen wir von einer *optimalen Teillösung* für $[t_i, t_j]$ mit l Zahlen.

Beispiel:

Zwischen 14 und 30 sollen zwei weitere Zahlen bestimmt werden:



Eine optimale Teillösung schaut so aus:



Anfangsbelegung:

Bei den Teillösungen gehen wir davon aus, dass Al bereits zwei Zahlen gewählt hat. Da wir am Anfang noch keine zwei Zahlen zur Verfügung haben, behelfen wir uns eines kleinen Tricks: Wir fügen zu der Zahlenliste der Teilnehmer noch zwei Dummyzahlen $-\infty$ und ∞ hinzu (in der Praxis gingen auch große Zahlen wie -1000 und 2000). Al bekommt ebenfalls zwei weitere Zahlen, die er allerdings auf diese Dummyzahlen festlegen muss. Diese Operation hat keinen Einfluss auf die Auszahlung, allerdings stehen uns jetzt zwei Ausgangszahlen zur Verfügung.

3.4 Optimalitätsprinzip

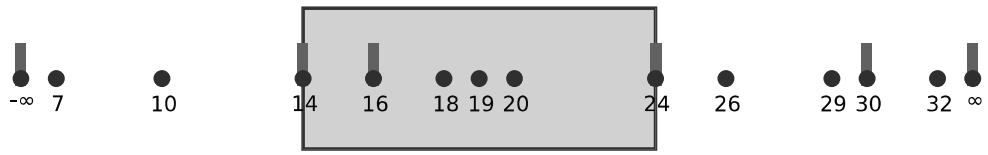
Ein *Optimalitätsprinzip* liegt vor, wenn sich eine optimale Gesamtlösung aus mehreren optimalen Teillösungen zusammensetzen lässt. In solchen Fällen findet man oft effiziente Algorithmen, wie auch hier:

Wählen wir aus einer optimalen Lösung Als $d_1, \ldots, d_m$ zwei Zahlen d_i und d_j aus, so minimiert die Verteilung seiner l Zahlen im Intervall $[d_i, d_j]$ die Summe der Auszahlungen an die Teilnehmer in diesem Intervall $[d_i, d_j]$.

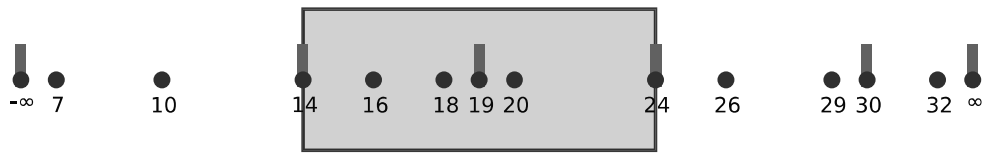
Aber wieso können wir das Optimalitätsprinzip hier überhaupt anwenden? Finden wir innerhalb des Intervalls $[d_i, d_j]$ eine bessere Verteilung mit l Zahlen (wobei d_i und d_j weiterhin gesetzt bleiben), so können wir in der Gesamtlösung ebenfalls diesen Bereich durch die bessere Teillösung austauschen. Die Teilnehmer im Intervall $[d_i, d_j]$ bekommen durch den Austausch insgesamt eine geringere Auszahlung. Für die Teilnehmer außerhalb des Intervalls $[d_i, d_j]$ bleibt die Auszahlung durch den Austausch unbeeinflusst. Für solche Teilnehmer ist Als nächste Zahl entweder höchstens d_i oder mindestens d_j, jedoch nie eine Zahl zwischen d_i und d_j.

Beispiel:

Folgende Teillösung im Bereich $[14, 24]$ ist nicht optimal, statt der Zahl 16 sollte Al lieber die Zahl 19 wählen:



Im Bereich $[14, 24]$ wurde die suboptimale Teillösung durch eine verbesserte Variante ersetzt:



Durch diese Ersetzung sind die Gesamtkosten um 5 Taler gesunken.

3.5 Berechnung einer optimalen Lösung

Wir lesen die Zahlen der Teilnehmer ein und sortieren sie der Größe nach. Dabei fügen wir auch die beiden Dummyzahlen hinzu. Wir erhalten so eine Liste $t_1, \ldots, t_n$, wobei die Einträge t_1 und t_n unsere beiden Dummyzahlen sind.

Wenn sich Al mindestens so viele Zahlen aussuchen darf, als es Werte von Teilnehmern gibt, können wir Al einfach diese Werte auswählen lassen. Alle Teilnehmer gehen dann leer aus.

Ansonsten berechnen wir die einzelnen Teillösungen. Hierbei müssen wir nicht alle Intervalle $[t_i, t_j]$ behandeln. Stattdessen genügt es, sich auf diejenigen Intervalle zu beschränken, die von einem t_i bis ganz zum Ende t_n reichen. Wir brauchen folgende Variablen:

> LÖSUNG(i,l) speichert die optimale Teillösung für die Teilnehmer t_i bis t_n mit l Zahlen.
>
> KOSTEN(i,l) gibt die Auszahlung von LÖSUNG(i,l) an die Teilnehmer t_i bis t_n an.
>
> KOSTENABSCHNITT(i,j) gibt die Summe der Auszahlungen an, die an Teilnehmer t_i bis t_j ausgezahlt werden müssen, sofern Al die Zahlen t_i und t_j festgelegt hat und es dazwischen keine weiteren Zahlen von ihm mehr gibt.

Zunächst berechnen wir die Tabelle KOSTENABSCHNITT und legen die Anfangsbedingungen

$$\text{LÖSUNG}(i,0) = \textit{leer} \text{ und } \text{KOSTEN}(i,0) = \text{KOSTENABSCHNITT}(i,n)$$

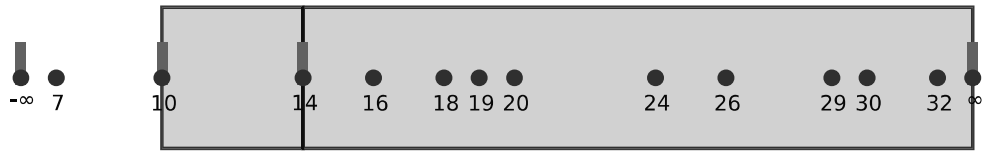
fest. Um LÖSUNG(i,l) für $l > 0$ zu berechnen, führen wir folgende Schritte aus:

1. Wir lassen j alle Zahlen von $i+1$ bis $n-1$ durchlaufen.
2. Für jedes j teilen wir das Intervall $[t_i, t_n]$ in die Teilintervalle $[t_i, t_j]$ und $[t_j, t_n]$ auf.
3. Die l Zahlen im Intervall $[t_i, t_n]$ werden wie folgt auf die Teilintervalle verteilt:
   a) Die Zahl t_j wird für die Teillösung gewählt.
   b) Im Innern von $[t_i, t_j]$ wird keine weitere Zahl gewählt. Die Kosten dieses Abschnittes sind damit KOSTENABSCHNITT(i,j).
   c) Im Innern von $[t_j, t_n]$ werden $l-1$ weitere Zahlen gewählt. Die Wahl dieser Zahlen wird rekursiv durch LÖSUNG$(j, l-1)$ bestimmt. Die Kosten dieses Abschnitts sind KOSTEN$(j, l-1)$.
4. Auf diese Weise erhalten wir einen möglichen Kandidaten für LÖSUNG(i,l) mit Kosten KOSTENABSCHNITT(i,j) + KOSTEN$(j, l-1)$.
5. Unter allen diesen Kandidaten suchen wir uns den günstigsten heraus. Diesen speichern wir in LÖSUNG(i,l) und dessen Kosten in KOSTEN(i,l).

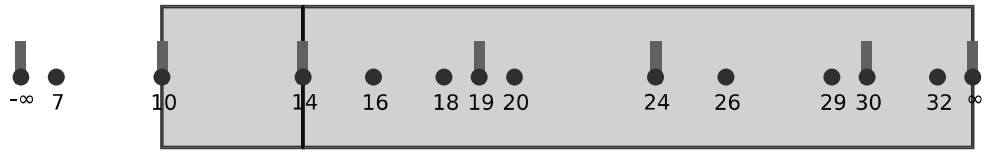
Die Gesamtlösung für $[t_1, t_n]$ mit m Zahlen erhalten wir dann durch Auslesen von LÖSUNG$(1,m)$ und deren Kosten aus KOSTEN$(1,m)$.

Beispiel:

Wir wollen LÖSUNG$(3,4)$ bestimmen, also eine Teillösung von $t_3 = 10$ bis $t_n = \infty$ mit vier weiteren Zahlen zwischen 10 und ∞. Dazu können wir das Intervall bei $j = 4$ aufteilen

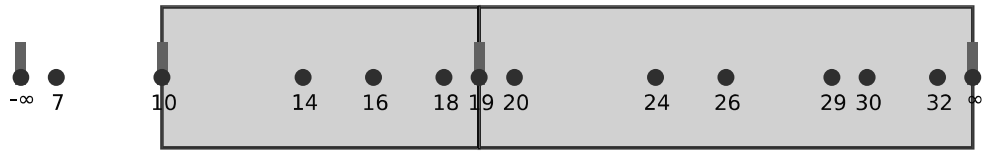


und anschließend LÖSUNG$(4,3)$ berechnen:

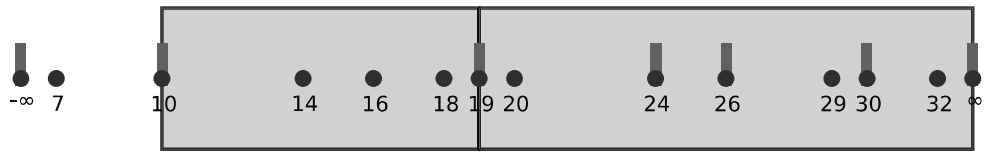


Es ist KOSTENABSCHNITT$(3,4) = 0$ und KOSTEN$(4,3) = 9$. Die Kosten im eingefärbten Bereich belaufen sich also auf 9 Taler.

Alternativ können wir das Intervall beispielsweise bei $j = 7$ aufteilen

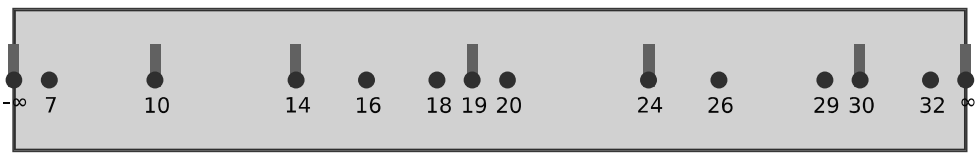


und anschließend LÖSUNG$(7,3)$ berechnen:



Es ist KOSTENABSCHNITT$(3,7) = 8$ und KOSTEN$(7,3) = 4$. Die Kosten im eingefärbten Bereich belaufen sich also auf 12 Taler.

Wir würden also die Aufteilung bei $j = 4$ wegen der geringeren Kosten bevorzugen. Natürlich müssen wir jetzt noch alle möglichen Aufteilungen überprüfen, aber es wird sich zeigen, dass die Aufteilung bei $j = 4$ tatsächlich optimal ist. Eine optimale Gesamtlösung für fünf Zahlen schaut so aus:



3.6 Kosten der einzelnen Abschnitte

Für unseren Algorithmus benötigen wir noch eine effiziente Methode, um alle Werte in der Tabelle KOSTENABSCHNITT zu bestimmen. Ein naiver Ansatz wäre es, für jedes Paar (i, j) die Summe

$$\text{KOSTENABSCHNITT}(i,j) = \sum_{k=i}^{j} \min\{t_k - t_i, t_j - t_k\}$$

zu berechnen. Dieser Ansatz führt zu einer Laufzeit von $\Theta(n^3)$ in diesem Zwischenschritt. Eine bessere Laufzeit erhalten wir, wenn wir zuerst Hilfssummen

$$\text{SUMME}(i,j) = \sum_{k \in \{i,\ldots,j\}} |t_k - t_i|$$

bestimmen. Summe(i, j) ist der Auszahlungsbetrag für t_i bis t_j, der entsteht, wenn die jeweiligen Teilnehmerzahlen von t_i getroffen werden. Diese Hilfssummen können wir iterativ berechnen:

$$\text{SUMME}(i,i) = 0$$

$$\text{SUMME}(i,j) = \begin{cases} \text{SUMME}(i,j-1) + t_j - t_i & \text{für } j > i \\ \text{SUMME}(i,j+1) + t_i - t_j & \text{für } j < i \end{cases}$$

Die Kosten für einen Abschnitt erhalten wir dann mit Hilfe der Formel

$$\text{KOSTENABSCHNITT}(i,j) = \text{SUMME}(i,h-1) + \text{SUMME}(j,h).$$

Der Wert h ist so bestimmt, dass t_{h-1} näher an t_i liegt sowie t_h näher an t_j.

Benutzt man für jedes Paar (i, j) eine Binärsuche, um dieses h zu finden, so erhält man eine Laufzeit von $\Theta(n^2 \cdot \log(n))$. Mit einem Trick geht es aber noch schneller: Kennen wir nämlich h für KOSTENABSCHNITT(i, j), so müssen wir das h für KOSTENABSCHNITT$(i, j+1)$ nur um einen nicht allzu großen Wert erhöhen. Berechnen wir für jedes i die KOSTEN(i, j) nach aufsteigenden j, so erzielen wir eine Laufzeit von $\Theta(n^2)$.

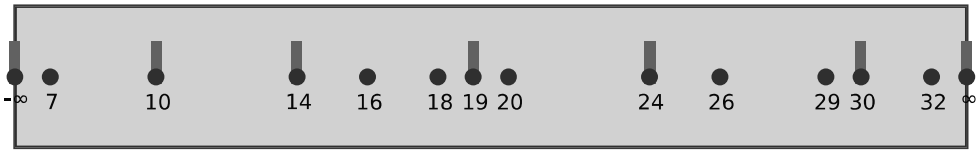
Algorithmus 1 Diese Funktion berechnet die Tabelle KOSTENABSCHNITT

```
function BERECHNEKOSTENFUERABSCHNITTE(t_1, ..., t_n)
    SUMME ← BERECHNESUMMEN(t_1, ..., t_n)
    for i von 1 bis n do
        h ← i
        for j von i + 1 bis n do
            while (t_h − t_i) < (t_j − t_h) do
                h ← h + 1
            end while
            KOSTENABSCHNITT(i, j) ← SUMME(i, h − 1) + SUMME(j, h)
        end for
    end for
    return KOSTENABSCHNITT
end function
```

3.7 Tricks zur Effizienzsteigerung

Speichern der Teillösungen: Prinzipiell ist es möglich, in LÖSUNG(i,l) eine komplette Zahlenliste der Teillösung zu speichern. Dies ist aber keine effiziente Vorgehensweise. Stattdessen speichern wir in LÖSUNG(i,l) lediglich einen Index j. Dieser Index j gibt die kleinste verwendete Zahl t_j der Teillösung in $[t_i,t_n]$ an, die nicht t_i ist. Diese Information reicht bereits aus, um daraus die komplette Gesamtlösung zu rekonstruieren. Hat Al für seine Lösung die i-te Teilnehmerzahl t_i bereits ausgewählt, so lautet seine nächste Zahl t_j mit $j =$ LÖSUNG(i,l), wobei l die Anzahl der noch unbestimmten Zahlen Als ist.

Schauen wir nochmal unsere optimale Lösung an:



Wir würden LÖSUNG$(1,5) = 3$ speichern, denn $t_3 = 10$ ist die erste Zahl. Die nächste Zahl wird durch LÖSUNG$(3,4) = 4$ mit $t_4 = 14$ angegeben, die übernächste durch LÖSUNG$(4,3) = 7$ mit $t_7 = 19$ und so weiter.

Dynamische Programmierung: In der Rekursion müssen wir LÖSUNG(i,l) mehrfach bestimmen. Es wäre nicht gerade schlau, diese Berechnung jedes Mal komplett neu auszuführen. Stattdessen berechnen wir die optimalen Teillösungen LÖSUNG(i,l) nur einmal und speichern dieses Ergebnis dann. Diese Technik ist auch unter dem Begriff der dynamischen Programmierung[5] bekannt. Dabei ist es wichtig, dass wir die Teillösungen LÖSUNG(i,l) in aufsteigenden Reihenfolge der Längen l berechnen, denn nur so können wir auf die bereits berechneten kürzeren Teillösungen LÖSUNG$(j,l-1)$ zurückgreifen.

3.8 Laufzeitüberlegungen

Der Algorithmus zur Bestimmung aller Teillösungen LÖSUNG(i,l) mittels dynamischer Programmierung hat eine Laufzeit von $\Theta(n^2 \cdot m)$. Es gibt nämlich insgesamt $n \cdot m$ Teillösungen, und zur Berechnung einer einzelnen Teillösung müssen wir im Schnitt etwa $\frac{n}{2}$ Kandidaten berücksichtigen. Das Sortieren der Teilnehmerwerte hat eine Laufzeit von $\Theta(n \cdot \log(n))$. Wie wir gerade gesehen haben, liegt die Berechnung aller KOSTENABSCHNITT(i,j) in $\Theta(n^2)$. Die beiden letzten Operationen sind also für große Eingaben laufzeittechnisch vernachlässigbar und die Gesamtlaufzeit liegt bei $\Theta(n^2 \cdot m)$.

3.9 Beispiele

Im Folgenden sind die Ergebnisse des Programms für obiges Beispiel sowie für die drei vorgegebenen Beispiele angegeben. Außerdem ist extra ein Beispiel angefügt, wie Al Verlust machen kann.

[5] https://de.wikipedia.org/wiki/Dynamische_Programmierung

Algorithmus 2 Berechnung der optimalen Lösung mittels dynamischer Programmierung

```
function BERECHNEOPTIMALELOESUNG(t_1, ..., t_n, m)
    Eingabe: Sortierte Teilnehmerwerte t_1, ..., t_n inkl. Dummies, Anzahl m für Al
    Ausgabe: Als Zahlen d_1, ..., d_m sowie deren Kosten

    ▷ Lege die Anfangsbedingungen fest.
    KOSTENABSCHNITT ← BERECHNEKOSTENFUERABSCHNITTE(t_1, ..., t_n)
    for i von 1 bis n do
        KOSTEN(i, 0) = KOSTENABSCHNITT(i, n)
    end for

    ▷ Berechne alle optimalen Teillösungen.
    for l von 1 bis m do
        for i von 1 bis n do
            KOSTEN(i, l) ← ∞
            for j von i + 1 bis n − 1 do
                if KOSTENABSCHNITT(i, j) + KOSTEN(j, l − 1) < KOSTEN(i, l) then
                    LÖSUNG(i, l) ← j
                    KOSTEN(i, l) ← KOSTENABSCHNITT(i, j) + KOSTEN(j, l − 1)
                end if
            end for
        end for
    end for

    ▷ Baue die Zahlenfolge für Al zusammen.
    i ← 1, AUSGABE ← leer
    for l von m bis 1 abwärts do
        i ← LÖSUNG(i, l)
        Füge t_i zu AUSGABE hinzu.
    end for
    return AUSGABE, KOSTEN(1, m)
end function
```

Obiges Beispiel

```
> python volldaneben.py beispiel-skizzen.txt 5
Zahlen der Teilnehmer:
 7 10 14 16 18 19 20 24 26 29 30 32
Al wählt die Zahlen:
 7 14 19 24 30
Einnahmen durch die Einsätze: 300 Taler
Auszahlungen an die Teilnehmer: 12 Taler
Gewinn für Al: 288 Taler
```

Beispiel 1

```
> python volldaneben.py beispiel1.txt
Zahlen der Teilnehmer:
 5 10 15 20 25 30 ... 990 995 1000
Al wählt die Zahlen:
 50 145 240 335 430 525 630 735 840 945
Einnahmen durch die Einsätze: 4975 Taler
Auszahlungen an die Teilnehmer: 4950 Taler
Gewinn für Al: 25 Taler
```

Beispiel 2

```
> python volldaneben.py beispiel2.txt
Zahlen der Teilnehmer:
 11 12 22 27 42 43 58 59 60 67 69 84 87 91 94 123 124 135 167 170 172
 178 198 211 226 229 276 281 305 313 315 324 327 335 336 362 364 367 368
 368 370 373 383 386 393 399 403 413 421 421 426 429 434 456 492 505 526
 530 537 539 540 545 567 582 584 586 649 651 676 690 729 736 739 750 754
 763 777 782 784 788 793 802 808 814 846 857 862 862 873 886 895 915 919
 925 926 929 932 956 980 996
Al wählt die Zahlen:
 59 172 315 368 421 539 651 777 862 929
Einnahmen durch die Einsätze: 2500 Taler
Auszahlungen an die Teilnehmer: 1924 Taler
Gewinn für Al: 576 Taler
```

Beispiel 3

```
> python volldaneben.py beispiel3.txt
Zahlen der Teilnehmer:
 20 40 60 80 100 100 120 120 140 160 160 180 200 220 220 240 240 240 240 260 260
 260 280 280 300 300 340 340 340 340 360 360 380 380 400 400 420 420 440 440 460
 460 460 480 480 500 520 520 520 520 520 520 540 540 540 560 580 580 580 580 600
 600 620 640 640 640 660 680 680 680 680 700 700 720 720 720 720 720 740 740 760
 780 780 800 800 820 840 840 860 860 860 880 900 900 920 920 960 980 980 1000
Al wählt die Zahlen:
 100 240 340 440 520 580 660 720 860 960
```

```
Einnahmen durch die Einsätze: 2500 Taler
Auszahlungen an die Teilnehmer: 2160 Taler
Gewinn für Al: 340 Taler
```

Extra-Beispiel für Als Verlust

Wir wählen uns 21 Zahlen zwischen 1 und 1000, so dass aufeinanderfolgende Zahlen genau Abstand 49 haben.

```
> python volldaneben.py beispiel-verlust.txt
Zahlen der Teilnehmer:
 1 50 99 148 197 246 295 344 393 442 491 540 589 638 687 736 785 834 883 932 981
Al wählt die Zahlen:
 1 50 99 148 197 344 491 638 785 932
Einnahmen durch die Einsätze: 525 Taler
Auszahlungen an die Teilnehmer: 539 Taler
Gewinn für Al: -14 Taler
```

Wie wir sehen, macht Al in diesem Fall Verlust. Auch ohne das Programm auszuführen kann man leicht den Grund sehen. Von den 21 Zahlen der Teilnehmer können nämlich nur 10 Zahlen von Al gewählt werden. Die restlichen 11 Zahlen haben mindestens einen Abstand von 49 zu den nächsten Zahlen Als. Damit muss Al mindestens $11 \cdot 49 = 539$ Taler auszahlen, aber seine Einnahmen betragen nur $21 \cdot 25 = 525$.

3.10 Alternative Lösungsansätze

Einleitung

Diese Aufgabe ermöglicht eine optimale Lösung in polynomieller Zeit, wie sie oben beschrieben wird. Diese ist allerdings nicht leicht zu finden. Im Folgenden werden einige Ansätze erläutert, welche mit heuristischen Vorgehensweisen zu guten oder schlechten Ergebnissen für die vorgegebenen Beispiele führen.

Wir erhalten als Eingabe eine variable Anzahl an Zahlen und benötigen zehn Zahlen für Al (Al-Zahlen), die Als Kosten minimieren. Für jede Eingabezahl wird jeweils die Al-Zahl gesucht, die am nächsten liegt, und der Abstand beider Zahlen ergibt die jeweiligen Kosten, die zum Gesamtergebnis aufaddiert werden.

Zunächst einmal bemerken wir, dass wir eine gültige Lösung finden können, indem wir die durch die Spieler gewählten Zahlen in zehn Gruppen einteilen und dann für jede Gruppe eine Al-Zahl auswählen. Für jede Eingabezahl in einer Gruppe wird am Schluss maximal der Abstand zu der Al-Zahl in ihrer Gruppe bezahlt oder weniger, falls eine Al-Zahl einer fremden Gruppe versehentlich näher liegt. Diese Grundstrategie kann zu einer ausreichend guten Lösung führen, je nachdem, wie die Auswahl der Gruppen und die Auswahl der Al-Zahlen erfolgt.

Wir beschäftigen uns nun im Folgenden mit beiden Lösungskomponenten, also Gruppeneinteilung und Auswahl einer Al-Zahl für die Gruppe:

Auswahl einer Al-Zahl für eine Gruppe

Für eine vorgegebene Menge an Zahlen eine einzige Al-Zahl optimal auszuwählen, ist polynomiell lösbar. Hierzu berechnet man den Median der Zahlen. Der Median ist die Eingabezahl, für die genauso viele Zahlen kleiner sind wie größer. (Ist die Anzahl der Eingabezahlen gerade, muss man sich überlegen, ob es besser ist, eine kleinere Zahl mehr zu haben oder eine größere Zahl; das spielt aber in den vorgegebenen Beispielen der Aufgabe keine Rolle). Wichtig ist, dass man wirklich den Median verwendet, und nicht z. B. den Durchschnitt. Man denke dabei z.B. an eine Gruppe mit sehr vielen kleinen Zahlen und einer großen Zahl: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 und 50. Hier ist es deutlich teurer, den Durchschnitt zu verwenden (der ca. bei 4,54 liegt, was Kosten von ca. 90 erzeugt), als einfach die Al-Zahl auf die 1 zu legen und nur einmal für die 50 eine längere Strecke zu bezahlen.

Einteilung in Gruppen

Das Problem der Gruppeneinteilung ist deutlich schwieriger; man kann nicht auf einfache Weise im Vorhinein abschätzen, welche Auswahl von Gruppen optimal sein wird. Für eine optimale Gruppeneinteilung benötigt man kompliziertere Ansätze. Hierfür gibt es verschiedene Möglichkeiten wie zum Beispiel:

- Einteilen anhand fixer Gruppenbreite, z. B. alle Zahlen von 1-100, von 101-200, usw.
- Wie eben, aber es werden zumindest leere Intervalle erkannt, die keine Zahl enthalten, und dafür werden dann andere Intervalle weiter aufgeteilt.
- Einteilen nach Anzahlen, d.h. Sortieren der Zahlen und dann die ersten zehn Zahlen in dieser Sortierung, die nächsten zehn Zahlen, usw., als Gruppe wählen.

Partitionierung um Medoiden (PAM)

Statt mit dynamischer Programmierung lässt sich die Aufgabe auch mit PAM (Partitioning Around Medoids)[6] lösen[7]. Hierbei sucht man sich anfangs m zufällige Teilnehmerwerte für Als Zahlen heraus. Anschließend probiert man alle Möglichkeiten aus, diese m Punkte mit einem der $n - m$ unbenutzten Teilnehmerwerte zu tauschen. Es wird immer derjenige Tausch für Al ausgewählt, der ihm am meisten Geld einspart. Dies tun wir solange, bis keine weiteren Vertauschungen mehr möglich sind. Bei PAM handelt es auch um ein Greedy-Verfahren[8], welches in jedem Rechenschritt die lokal beste Lösung sucht. Es ist aber nicht klar, ob die erhaltene Lösung zum Schluss immer die optimale Gesamtlösung für Al darstellt.

Randomisierte Algorithmen

Es lässt sich auch ein randomisiertes Vorgehen anwenden, bei dem die von Al gewählten Zahlen zufällig bestimmt werden. Ein einfaches zufälliges Bestimmen führt allerdings noch nicht zu ausreichend guten Ergebnissen und muss daher noch verbessert werden.

[6] https://de.wikipedia.org/wiki/K-Means-Algorithmus

[7] https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Clustering/Partitioning_Around_Medoids_-(PAM)

[8] https://de.wikipedia.org/wiki/Greedy-Algorithmus

Aufgrund der Erkenntnis, dass die optimale Lösung immer erreicht werden kann, wenn Zahlen der Teilnehmer gewählt werden, lassen sich allerdings bereits bessere Lösungen erzielen.

Ebenso lässt sich eine Verbesserung dadurch erzielen, indem mehrere Zufallswahlen von Als Zahlen generiert werden, von denen diejenige ausgewählt wird, die die Kosten für Al minimiert. Auch eine schrittweise Verbesserung durch das Verschieben einzelner Werte der Auswahl ist hier denkbar.

3.11 Bewertungskriterien

Die Bewertungskriterien (Fettdruck) vom Bewertungsbogen werden hier näher erläutert (Punktabzug in []).

- (1) [-1] **Dokumentation sehr unverständlich bzw. unvollständig**.
- (2) [-1] **Vorgegebenes Dateneingabeformat mangelhaft umgesetzt**:
  Die vorgegebenen Daten sollten aus den Dateien korrekt eingelesen werden und intern geeignet gespeichert werden. Insbesondere sind lineare Datenstrukturen dafür geeignet.
- (3) [-1] **Lösungsverfahren fehlerhaft**:
  Das Lösungsverfahren zur Gewinnoptimierung sollte allgemein genug sein, d.h. nicht auf spezielle Beispiele zugeschnitten. Auch Greedy-Verfahren zur Gewinnoptimierung sind in Ordnung. Der seltene Verlustfall mit einem negativen „Gewinn“ muss nicht extra berücksichtigt und erläutert werden.
- (4) [-1] **Strategie zur Gewinnoptimierung mangelhaft / fehlt**:
  Es sollte eine nachvollziehbare Strategie zur Auswahl von Als Zahlen angewandt und erläutert werden; sie soll zumindest für die vorgegebenen Beispiel zu einem Gewinn für Al führen (Auszahlung $<$ Einsätze). Allzu einfache Strategien sind nicht akzeptabel, wie etwa die Einteilung der Glückszahlen in Gruppen mit Berechnung des Durchschnitts oder gar eine von den Glückszahlen der Teilnehmer unabhängig festgelegte Auswahl der 10 Zahlen für Al.
- (5) [-1] **Verfahren bzw. Implementierung unnötig aufwendig**:
  Nur besonders umständliche Verfahren bzw. Implementierungen führen zu Punktabzug. Die Laufzeiteffizienz wird nicht bewertet.
- (6) [-1] **Programmausgabe ungeeignet**: Die Programmausgabe sollte verständlich sein. Die berechneten Zahlen von Al sowie der Gesamtgewinn und/oder die Gesamtauszahlung sollten ausgegeben werden, auch wenn die letztere Angabe nicht explizit von der Aufgabenstellung verlangt war. Wenn die Angabe des Gesamtgewinns fehlt, ist dies als ein Indiz für ein zweifelhaftes Lösungsverfahren zu werten.
- (7) [-1] **Beispiele fehlerhaft bzw. zu wenige (mind. 2/3)**:
  Es wird die Dokumentation der Ergebnisse von mind. 2 der 3 vorgegebenen Beispiele erwartet.