Aufgabe 3: Hex-Max

Teilnahme-ID: 61045

Bearbeiter/-in dieser Aufgabe:
Linus Schumann

22. April 2022

Inhaltsverzeichnis

1 Lösungsidee

Zuerst wird die Idee zur Lösung des Problems genauer beschrieben. Dazu gehört die erste Lösungsidee, die allerdings nicht umgesetzt wurde, sondern es wurde die zweite Lösungsidee umgesetzt, die einige Verbesserungen bringt.

1.1 Erste Lösungsidee

Wie oben beschrieben ist dies die erste Lösungsidee, die nicht umgesetzt wurde. Dabei war der Ansatz im Grunde genommen nach und nach in verschiedenen Schritten immer ein Stäbchen zu verschieben und die Zwischenstände als Schritte auszugeben. Allerdings gab es dabei verschiedene Probleme, wie z.B. das bei der Umlegung eines Stäbchens, der Algorithmus berechnet müsste, wo das Stäbchen hingelegt bzw. weggenommen werden würde. Da dies eine große Schwierigkeit darstellt oder eventuell gar nicht möglich ist, wurde diese Lösungsidee zur zweiten Lösungsidee abgewandelt, die nun beschrieben wird.

1.2 Zweite umgesetzte Lösungsidee

Bei dieser Lösungsidee wird nicht, wie bei der ersten Lösungsidee, jedes Stäbchen einzeln verschoben, sondern es wird eine Ziffer immer direkt gegen eine andere Ziffer getauscht. Um dabei zu verhindern, dass am Ende zu viele Stäbchen entfernt wurden oder Stäbchen nicht verwendet wurden, wird eine Art Zwischenspeicher für übrige und benötigte Stäbchen verwendet.

Zum Beispiel würde bei der Umlegung von einem d zu einem F, dieser Zwischenspeicher auf 1 gesetzt werden, da ein d aus 5 Stäbchen und ein F aus 4 Stäbchen besteht. Dies bedeutet, dass im weiteren Verlauf falls ein Stäbchen benötigt wird, keine weitere Umlegung erfolgen muss, sondern zuerst Stäbchen aus dem Zwischenspeicher genutzt werden. Natürlich funktioniert dies auch andersrum, also wenn irgendwo ein Stäbchen übrig ist und der Zwischenspeicher auf -1 steht, würde der Zwischenspeicher auf 0 gesetzt und es müsste wieder keine Unmlegung genutzt werden.

Beachten muss man natürlich bei diesem Ansatz, dass der Line-Cache am Ende auf jeden Fall 0 sein muss, da sonst die Lösung nicht möglich ist und eine neue Lösung gesucht werden muss.

In Abb. 1 wird der Algorithmus am Beispiel der Beispieldaten hexmax0 nach dieser Lösungsidee noch einmal dargestellt und veranschaulicht.

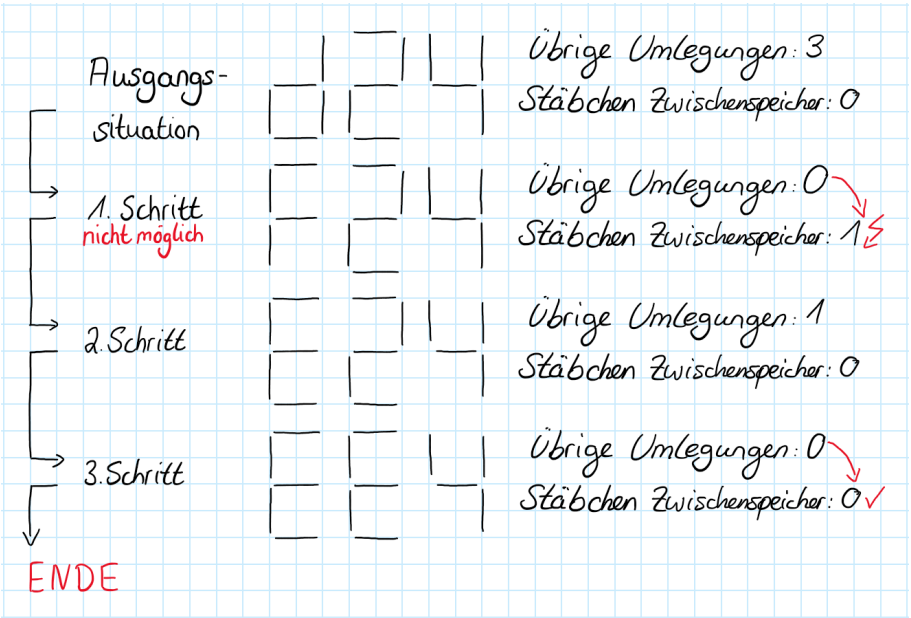


Abbildung 1: Veranschaulichter Algorithmus

1.3 Rekursion

Um die bestmögliche Lösung zu finden, wird eine rekursive Struktur genutzt. Diese nähert sich von der besten Lösung immer mehr der schlechtesten Lösung an. Dabei wird die erste gefundene Lösung, die dann automatisch die beste Lösung ist, gespeichert und der Algorithmus ist beendet.

Damit nicht jede Kombination ausprobiert werden muss, wird ein Abbruchkriterium definiert. Dieses überprüft, ob überhaupt noch genügend frei Plätze für die Stäbchen aus dem Zwischenspeicher zur Verfügung stehen bzw. ob noch genügend Stäbchen verfügbar sind, um den Zwischenspeicher wieder auszugleichen.

2 Umsetzung

2.1 Allgemeines

Im folgenden wird die Umsetzung, der in Abschnitt 1 beschriebene Lösungsidee, näher erläutert. Grundsätzlich wurde diese Idee dabei in Python, genauer gesagt in der Datei "Aufgabe-3-Hex-Max.py" implementiert. Diese Datei befindet sich unter dem Haupt-Verzeichnis der Aufgabe im Verzeichnis "./source/".

Um das implementierte Programm zu starten, kann das Batch-Script (Windows) oder das Shell-Script (Mac, Linux) genutzt werden. Beide befinden sich im Haupt-Verzeichnis der Aufgabe. Eine direkte Ausführung der Python-Datei ist nicht möglich, da die Verzeichnis-Pfade auf das Verzeichnis des Batch- bzw. Shell-Script angepasst wurden.

Unter dem Haupt-Verzeichnis der Aufgabe im Verzeichnis "./beispieldaten/" befinden sich alle in dieser Dokumentation aufgeführten Beispiele und unter dem Verzeichnis "./beispielausgaben/" befinden sich dementsprechend die gesicherten Ausgaben, die auch bei Ausführung des Scripts ausgegeben werden. Damit letztere besser von den beispieldaten unterschieden werden können, werden diese mit der Dateiendung ".out" gespeichert, sind aber im Klartext und können wie ganz normale ".txt" Dateien geöffnet werden.

2.2 Verwendete Module

Um das Python-Script zu starten, werden verschiede Module benötigt, deren Aufgabe und Vorteile im folgenden beschrieben wird.

copy
: Da es in der implementierten Lösung erforderlich ist eine "Deepcopy" von einer Liste zu erstellen und Python nur eine eingebaute Funktion zur Erstellung einer "Shallowcopy" beinhaltet, wird vom Module "copy" die Funktion "deepcopy()" eingebunden. Auf den Unterschied zwischen einer Deep- und einer Shallowcopy wird in Abschnitt 2.3 genauer eingegangen.

tkinter
: Das Module "tkinter" wird in der Implementierung dazu genutzt, um die Auswahl der Datei mit Hilfe eines Datei-Auswahl-Fensters zu ermöglichen.

sys
: Zuletzt wird noch aus dem Module "sys", die Funktion "setrecursionlimit" genutzt, um das Rekursionslimit für die Implementierung zu erhöhen.

2.3 Unterschied zwischen Deep- und Shallowcopy

In der Implementierung gibt es wie in Abschnitt 2.2 beschrieben Stellen an denen eine Shallowcopy, wie sie in Python eingebaut ist, nicht ausreicht und stattdessen eine Deepcopy erstellt werden muss. Der Unterschied zwischen diesen beiden Arten der Kopie einer Liste, wird dabei nur dann deutlich, wenn man Listen mit mindestens einer Unterliste verwendet. Bei einer Shallowcopy wird nämlich nur die erste Schicht der Liste kopiert und auf Elemente in Unterlisten nur referenziert. Dies kann zu Problemen führen, wenn ein Programm Elemente in Unterlisten ändert, da sich dann nicht nur die kopierte, sondern auch die Originale Liste mit ändern würde.

Damit das eben beschriebene Phänomen nicht zu Problemen führt, wurde in der Implementierung mit einer Deepcopy gearbeitet, bei der auch die Unterlisten vollständig kopiert werden und nicht nur eine Referenz erzeugt wird.

2.4 7-Segment-Anzeige

Um die aktuelle Belegung und die vorausgesetzten Belegungen der einzelnen Hexdezimalzahlen passend zu speichern, wird in der Implementierung ein Dictonary benutzt, in dem die einzelnen standard Belegungen für jede Zahl gespeichert sind. Diese Belegungen werden als Tupel bzw. an manchen Stellen als Liste von genau 7 Boolean-Werten gespeichert. Dabei repräsentiert jeder Wert eine Linie der 7-Segment-Anzeige und es wird die Reihenfolge, die in Abb. 2 durch die aufsteigenden Buchstaben deutlich wird, genutzt.

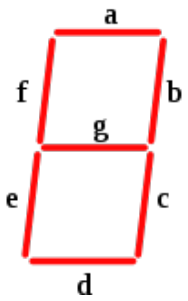


Abbildung 2: 7-Segment-Anzeige

2.5 Funktionen des Algorithmus

Im Nachfolgenden werden die einzelnen großen Funktionen, auf die die verschiedenen Aufgaben des Algorithmus verteilt wurden, näher erläutert und die jeweilige Aufgabe im Bezug auf die in Abschnitt 1 erläuterte Lösungsidee beschrieben.

2.5.1 read_file und convert_digits

In der Funktion read_file wird zuerst ein Fenster erzeugt in dem eine bestimmte Beispieldatei ausgewählt werden kann. Dannach wird aus dieser Beispieldatei die Hexadezimalzahl und die Maximale Zahl n an Umlegungen eingelesen. Nachdem Einlesen wird in der Funktion convert_digits die Hexadezimalzahl in das in Abschnitt 2.4 beschriebene Speicherkonstrukt umgewandelt.

2.5.2 calculate_final_digits

Diese Funktion stellt den Start des eigentlichen Algorithmus da. Dabei wird zuerst die Rekursive Funktion calculate_next_digit, die innerhalb dieser Funktion definiert wird, zum ersten Mal aufgerufen. Dabei werden die Start-Parameter der rekuriven Funktion angegeben. Genauer gesagt startet die Funktion bei der ersten Zahl, mit einem leeren Linienzwischenspeicher und mit den maximalen Schritten als Startwert für die übrigen Schritte.

2.5.3 calculate_next_digit

In dieser rekurisven Funktion wird zuerst überprüft, ob die aktuelle Lösung überhaupt noch möglich ist. Danach wird absteigend beginnend mit der höchsten Zahl (F) für alle erlaubten Möglichkeiten, an dieser Stelle, die Funktion erneut aufgerufen. Dabei wird die alte Zahl gegen die neue getauscht. Um zu überprüfen ob eine Zahl an dieser Stelle erlaubt ist, werden dabei die Unterschiede der Linien im 7-Segment-Format und die Anzahl an Umlegungen, die mindestens gebraucht werden um die Zahl zu erreichen, berechnet. Dabei wird auch auf den in Abschnitt 1 erklärten Linienzwischenspeicher zurückgegriffen und dieser, wenn möglich, vor den richtigen Umlegungen genutzt.

Abbruch der Rekursion Bevor die eben beschriebene rekursive Funktion ein weiteres Mal aufgerufen wird, wird nach dem Tauschen der Zahlen immer überprüft, ob das Ende der gesamten Zahl erreicht wurde. Wenn dies der Fall ist, wird überprüft, ob die Lösung erlaubt ist. Genauer gesagt wird geprüft, ob weitere Linien im Linienzwischenspeicher vorhaden sind oder weitere Linien benötigt werden.

Wenn die Lösung dann nach diesen Kriterien erlaubt ist, wird die Rekursion gestoppt und die Endwerte werden gespeichert. Damit die gesamte Rekurion stoppt, wird bei jedem Aufruf dieser Funktion immer geprüft, ob es schon eine erlaubte Lösung gibt. Wenn ja endet dieser Aufruf der Funktion sofort.

2.5.4 calculate_steps

Um alle nötigen Schritte, die durch die Aufgabenstellung gefordert bei Beispiel 0-2 ausgegeben werden müssen, zu berechnen wird diese Funktion aufgerufen. Dabei wird zuerst die Startzahl mit der Lösung verglichen und alle Unterschiedlichen Linien berechnet. Danach wird die Startzahl immer weiter an die Endzahl angepasst und Linien miteinander getauscht. Nach jedem Tausch wird dementsprechend, die aktuelle Belegung in einer weiteren Liste gespeichert, sodass hinter alle Schritte zurückgeben werden können.

2.5.5 Ausgabe der Ergebnisse

Am Ende werden nur noch die Ergebnisse wieder in eine Hexadezimalzahl zurückkonvertiert und ausgeben. Falls zusätzlich noch die im vorherigen Abschnitt beschriebenen Schritte berechnet werden, werden diese natürlich auch noch ausgegeben.

3 Beispiele

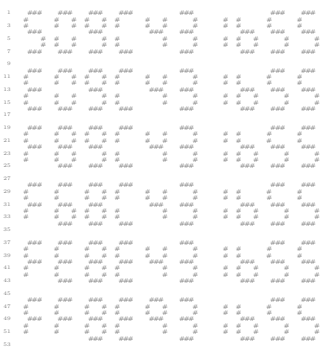
Hier werden alle Ausgaben der 6 Beispieldaten und auch eines eigenen Beispiels angegeben. Am eigenem Beispiel kann man nochmal sehen was genau passiert, wenn trotz übrigen Umlegungen kein besseres Ergebnis erzielt werden kann.

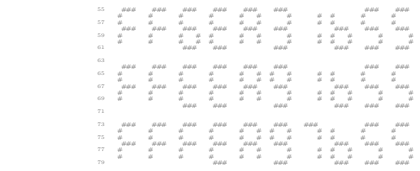
3.1 Vorgegebene Beispiele

3.1.1 Beispiel 0

```
Before  -->  D 2 4
After   -->  E E 4
Left Changes  -->  0
```

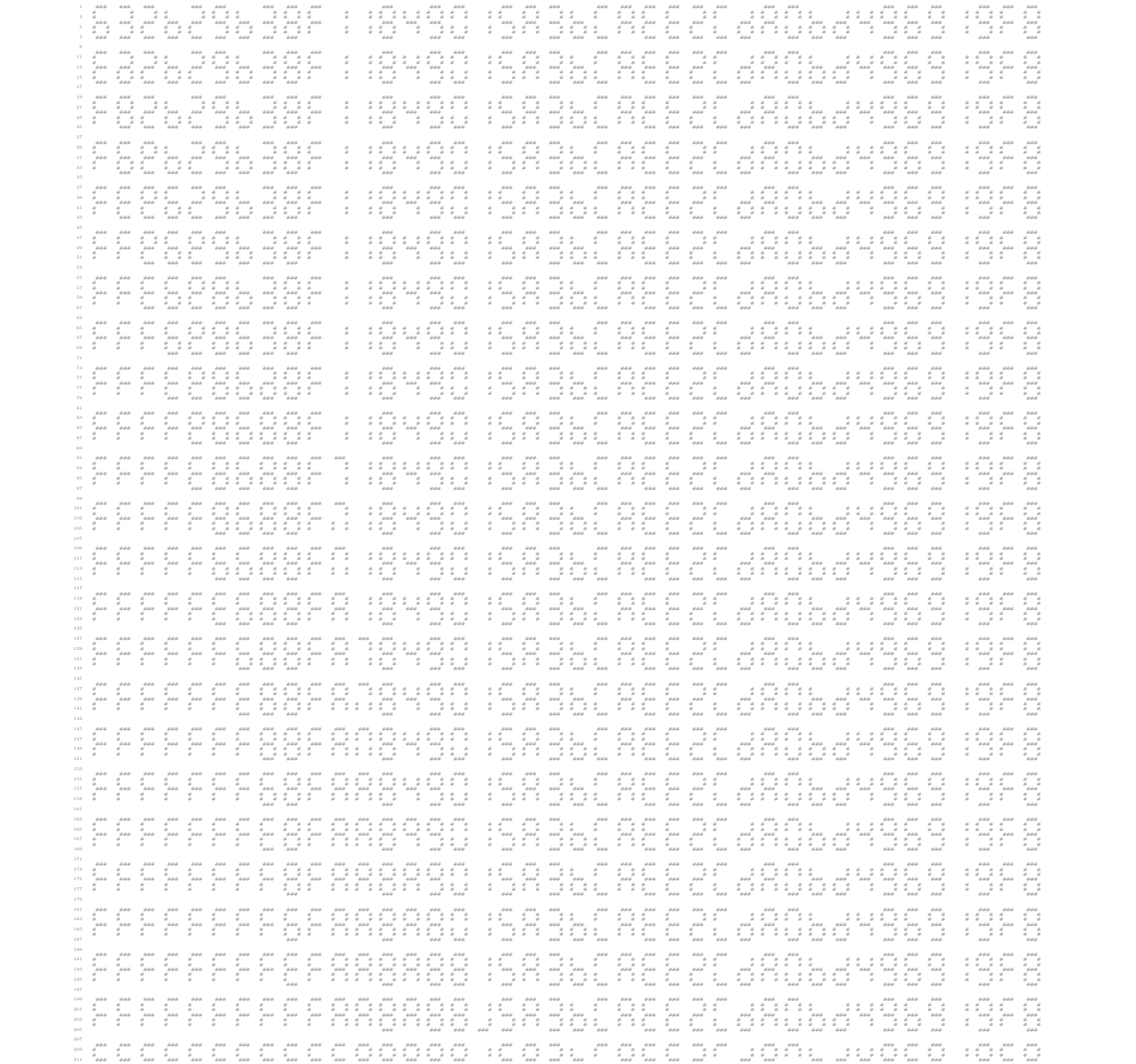
3.1.2 Beispiel 1

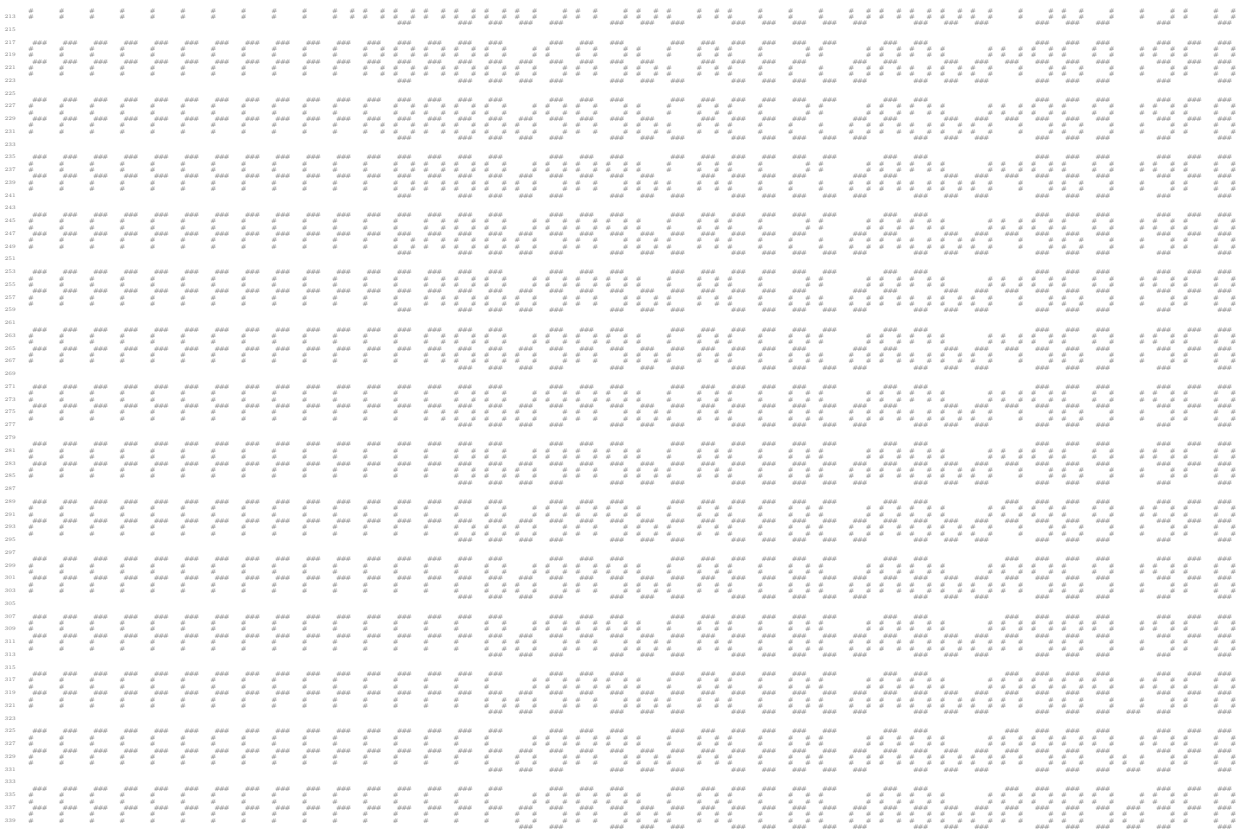




```
Before  -->  5 0 9 C 4 3 1 B 5 5
After   -->  F F F E A 9 7 B 5 5
Left Changes  -->  0
```

3.1.3 Beispiel 2





```
Before  -->  6 3 2 B 2 9 B 3 8 F 1 1 8 4 9 0 1 5 A 3 B C A E E 2 C D A 0 B D 4 9 6 9 1 9
    ↪ F 8
After   -->  F F F F F F F F F F F F F F F F D 9 A 9 B E A E E 8 E D A 8 B D A 9 8 9 D 9
    ↪ F 8
Left Changes  -->  0
```

3.1.4 Beispiel 3

```
Before  -->  0 E 9 F 1 D B 4 6 B 1 E 2 C 0 8 1 B 0 5 9 E A F 1 9 8 F D 4 9 1 F 4 7 7 C E
    ↪ 1 C D 3 7 E B F B 6 5 F 8 D 7 6 5 0 5 5 7 5 7 C 6 F 4 7 9 6 B B 8 B 3 D F 7 F C A
    ↪ C 6 0 6 D D 0 6 2 7 D 6 B 4 8 C 1 7 C 0 9
After   -->  F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F A A 9 8 B B 8 B 9 D F A F E A
    ↪ E 8 8 8 D D 8 8 8 A D 8 B A 8 E A 8 8 8 8
Left Changes  -->  0
```

3.1.5 Beispiel 4

```
Before  -->  1 A 0 2 B 6 B 5 0 D 7 4 8 9 D 7 7 0 8 A 6 7 8 5 9 3 0 3 6 F A 2 6 5 F 2 9 2
    ↪ 5 B 2 1 C 2 8 B 4 7 2 4 D D 8 2 2 0 3 8 E 3 B 4 8 0 4 1 9 2 3 2 2 F 2 3 0 A B 7 A
    ↪ F 7 B D A 0 A 6 1 B A 7 D 4 A D 8 F 8 8 8
After   -->  F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ E B 8 D E 8 8 B A A 8 A D D 8 8 8 8 9 8 E 9 B A 8 8 A D 9 8 9 8 8 F 8 9 8 A B 7 A
    ↪ F 7 B D A 8 A 6 1 B A 7 D 4 A D 8 F 8 8 8
Left Changes  -->  0
```

3.1.6 Beispiel 5

```
Before   -->  E F 5 0 A A 7 7 E C A D 2 5 F 5 E 1 1 A 3 0 7 B 7 1 3 E A A E C 5 5 2 1 5 E
    ↪ 7 E 6 4 0 F D 2 6 3 F A 5 2 9 B B B 4 8 D C 8 F A F E 1 4 D 5 B 0 2 E B F 7 9 2 B
    ↪ 5 C C B B E 9 F A 1 3 3 0 B 8 6 7 E 3 3 0 A 6 4 1 2 8 7 0 D D 2 B A 6 E D 0 D B C
    ↪ A E 5 5 3 1 1 5 C 9 A 3 1 F F 3 5 0 C 5 D F 9 9 3 8 2 4 8 8 6 D B 5 1 1 1 A 8 3 E
    ↪ 7 7 3 F 2 3 A D 7 F A 8 1 A 8 4 5 C 1 1 E 2 2 C 4 C 4 5 0 0 5 D 1 9 2 A D E 6 8 A
    ↪ A 9 A A 5 7 4 0 6 E B 0 E 7 C 9 C A 1 3 A D 0 3 8 8 8 F 6 A B E D F 1 4 7 5 F E 9
    ↪ 8 3 2 C 6 6 B F D C 2 8 9 6 4 B 7 0 2 2 B D D 9 6 9 E 5 5 3 3 E A 4 F 2 E 4 E A B
    ↪ A 7 5 B 5 D C 1 1 9 7 2 8 2 4 8 9 6 7 8 6 B D 1 E 4 A 7 A 7 7 4 8 F D F 1 4 5 2 A
    ↪ 5 0 7 9 E 0 F 9 E 6 0 0 5 F 0 4 0 5 9 4 1 8 5 E A 0 3 B 5 A 8 6 9 B 1 0 9 A 2 8 3
    ↪ 7 9 7 A B 3 1 3 9 4 9 4 1 B F E 4 D 3 8 3 9 2 A D 1 2 1 8 6 F F 6 D 2 3 3 5 8 5 D
    ↪ 8 C 8 2 0 F 1 9 7 F B A 9 F 6 F 0 6 3 A 0 8 7 7 A 9 1 2 C C B D C B 1 4 B E E C B
    ↪ A E C 0 E D 0 6 1 C F F 6 0 B D 5 1 7 B 6 8 7 9 B 7 2 B 9 E F E 9 7 7 A 9 D 3 2 5
    ↪ 9 6 3 2 C 7 1 8 F B F 4 5 1 5 6 A 1 6 5 7 6 A A 7 F 9 A 4 F A D 4 0 A D 8 B C 8 7
    ↪ E C 5 6 9 F 9 C 1 3 6 4 A 6 3 B 1 6 2 3 A 5 A D 5 5 9 A A F 6 2 5 2 0 5 2 7 8 2 B
    ↪ F 9 A 4 6 1 0 4 E 4 4 3 A 3 9 3 2 D 2 5 A A E 8 F 8 C 5 9 F 1 0 8 7 5 F A D 3 C B
    ↪ D 8 8 5 C E 6 8 6 6 5 F 2 C 8 2 6 B 1 E 1 7 3 5 E E 2 F D F 0 A 1 9 6 5 1 4 9 D F
    ↪ 3 5 3 E E 0 B E 8 1 F 3 E C 1 3 3 9 2 2 E F 4 3 E B C 0 9 E F 7 5 5 F B D 7 4 0 C
    ↪ 8 E 4 D 0 2 4 B 0 3 3 F 0 E 8 F 3 4 4 9 C 9 4 1 0 2 9 0 2 E 1 4 3 4 3 3 2 6 2 C D
    ↪ A 1 9 2 5 A 2 B 7 F D 0 1 B E F 2 6 C D 5 1 A 1 F C 2 2 E D D 4 9 6 2 3 E E 9 D E
    ↪ B 1 4 C 1 3 8 A 7 A 6 C 4 7 B 6 7 7 F 0 3 3 B D E B 8 4 9 7 3 8 C 3 A E 5 9 3 5 A
    ↪ 2 F 5 4 B 9 9 2 3 7 9 1 2 F 2 9 5 8 F D F B 8 2 2 1 7 C 1 7 5 4 4 8 A A 8 2 3 0 F
    ↪ D C B 3 B 3 8 6 9 8 2 4 A 8 2 6 6 3 5 B 5 3 8 D 4 7 D 8 4 7 D 8 4 7 9 A 8 8 F 3 5
    ↪ 0 E 2 4 B 3 1 7 8 7 D F D 6 0 D E 5 E 2 6 0 B 2 6 5 8 2 9 E 0 3 6 B E 3 4 0 F F C
    ↪ 0 D 8 C 0 5 5 5 5 E 7 5 0 9 2 2 2 6 E 7 D 5 4 D E B 4 2 E 1 B B 2 C A 9 6 6 1 A 8
    ↪ 8 2 F B 7 1 8 E 7 A A 5 3 F 1 E 6 0 6
After    -->  F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F
    ↪ F F F F F F F F F F F F F F F F F F 8 8 E F A 9 E B E 8 9 E F A 9 9 F B D A A 8 E
    ↪ 8 E A D 8 8 A B 8 9 9 F 8 E 8 F 9 A A 9 E 9 A D 8 8 9 8 8 E D A 9 A 9 9 8 8 8 E D
    ↪ A D 9 8 9 A 8 B A F D 8 A 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
    ↪ 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
Left Changes   -->  0
```

3.2 Eigene Beispiele

3.2.1 Eigenes Beispiel 1

```
        ###
    #      #  #   #
    #      #  #   #
 ###    ###    ###
#   #  #          #
#   #  #          #
 ###    ###


 ###    ###
           #  #   #
           #  #   #
 ###    ###    ###
#   #  #          #
#   #  #          #
 ###    ###


 ###    ###
#          #  #   #
#          #  #   #
 ###    ###    ###
#      #          #
#      #          #
 ###    ###


 ###    ###
#      #   #  #   #
#      #   #  #   #
 ###    ###    ###
#      #          #
#      #          #
        ###


 ###    ###    ###
#      #      #   #
#      #      #   #
 ###    ###    ###
#      #          #
```

```
#   #   ###   #
#
# ### ### ### ###
# # # # # # # #
# ### ### # # ###
# #   # # # # #
```

```
Before  -->  D 2 4
After   -->  F F A
Left Changes  -->  15
```

---

4 Quellcode

```python
# import Tkinter for file selection
import tkinter as tk
from tkinter.filedialog import askopenfilename

# import deepcopy
from copy import deepcopy

# increase Recursion Limit, because of the size of hexmax5.txt
from sys import setrecursionlimit
setrecursionlimit(1010)

# Setting for step output
print_steps_var = False

# Create default Digit Dict for each possible digit
digit_dict = {
    '0' : (True, True, True, True, True, True, False),      # 0
    '1' : (False, True, True, False, False, False, False),  # 1
    '2' : (True, True, False, True, True, False, True),     # 2
    '3' : (True, True, True, True, False, False, True),     # 3
    '4' : (False, True, True, False, False, True, True),    # 4
    '5' : (True, False, True, True, False, True, True),     # 5
    '6' : (True, False, True, True, True, True, True),      # 6
    '7' : (True, True, True, False, False, False, False),   # 7
    '8' : (True, True, True, True, True, True, True),       # 8
    '9' : (True, True, True, True, False, True, True),      # 9
    'A' : (True, True, True, False, True, True, True),      # 10 = A
    'B' : (False, False, True, True, True, True, True),     # 11 = b
    'C' : (True, False, False, True, True, True, False),    # 12 = C
    'D' : (False, True, True, True, True, False, True),     # 13 = d
    'E' : (True, False, False, True, True, True, True),     # 14 = E
    'F' : (True, False, False, False, True, True, True)     # 15 = F
}
# reverse Digit Dict
reversed_digit_dict = {v: k for k, v in digit_dict.items()}

def print_and_write(message, filename):
    """ Function that prints result to console and output file
    p1 -> message -- message to be output
    p2 -> filename -- name of output file

    r  -> None
    """
    with open(filename, 'a') as f:
        print(message, file=f)
    print(message)
def calculate_steps(start_digits, final_digits):
    """ Function that calculates the necessary steps to reach the final digits
    p1 -> start_digits -- original digits before change
    p2 -> final_digits -- final digits after change

    r  -> list of steps

    """
    dif = [[],[]]
    steps = []
    step = deepcopy(start_digits)
```

```
    steps.append(deepcopy(start_digits))

    for i,(x1,x2) in enumerate(zip(start_digits, final_digits)):
        for j,(y1,y2) in enumerate(zip(x1, x2)):
            if y1 and not y2:
                dif[0].append((i,j))
            if not y1 and y2:
                dif[1].append((i,j))

    for x1,x2 in zip(dif[0],dif[1]):
        step[x1[0]][x1[1]], step[x2[0]][x2[1]] = step[x2[0]][x2[1]], step[x1[0]][x1[1]]
        steps.append(deepcopy(step))
    return steps
def print_steps(steps, filename):
    """
    p1 -> steps -- list of steps
    p2 -> filename -- name of output file

    r  -> None
    """
    for step in steps:
        # create empty lines list
        lines = ['' for _ in range(7)]
        # append each digit to lines
        for digit in step:
            for i,x in enumerate(reversed(digit)):
                # reverse i
                j = 6 - i
                match j:
                    case 6:
                        lines[3] += '␣###␣' if x else '␣␣␣␣␣'
                    case 5:
                        lines[1] += '#␣␣␣' if x else '␣␣␣␣'
                        lines[2] += '#␣␣␣' if x else '␣␣␣␣'
                    case 4:
                        lines[4] += '#␣␣␣' if x else '␣␣␣␣'
                        lines[5] += '#␣␣␣' if x else '␣␣␣␣'
                    case 3:
                        lines[6] += '␣###␣' if x else '␣␣␣␣␣'
                    case 2:
                        lines[4] += '#' if x else '␣'
                        lines[5] += '#' if x else '␣'
                    case 1:
                        lines[1] += '#' if x else '␣'
                        lines[2] += '#' if x else '␣'
                    case 0:
                        lines[0] += '␣###␣' if x else '␣␣␣␣␣'
            # create space between digits
            for i,_ in enumerate(lines):
                lines[i] += '␣␣'
        # print lines
        for x in lines:
            print_and_write(x, filename)
        print_and_write('\n', filename)
def read_file():
    """ Function that reads the input file.
    r1 -> digits -- list of input digits
    r2 -> n_of_changes -- max number of changes

    """
    def choose_file():
        """ Function that creates a window to choose a file
        r  -> filename -- filename of choosen file

        """
        root = tk.Tk()
        root.withdraw()
        root.overrideredirect(True)
        root.geometry('0x0+0+0')
        root.deiconify()
        root.lift()
        root.focus_force()
        filename = askopenfilename(parent=root, initialdir='beispieldaten/')
```

```
        root.destroy()
        return filename
    filename = choose_file()
    with open(filename) as f:
        digits = (x for x in f.readline().strip())
        n_of_changes = int(f.readline().strip())
    return digits, n_of_changes, filename
def convert_digits(intDigits):
    """ Function that converts digits to 7-Segment format
    p1 -> intDigits -- tuple of digits

    r  -> digits -- list of digits in 7-Segment format

    """
    return [list(digit_dict[x]) for x in intDigits]
def calculate_final_digits(digits, max_n_of_changes):
    """ Function that calculates the final digits after n-Changes
    p1 -> digits -- list of digits in
    p2 -> max_n_of_changes -- maximum number of line-changes

    r1 -> best_digits -- best digits after line-changes
    r2 -> best_left_n_of_changes -- left line-changes after line-changes

    """
    def calculate_dismatchings(d1,d2):
        """ This function calculates the number of dismachings between two digits and the
    ↪   difference between the number of lines
        p1 -> d1 -- digit 1
        p2 -> d2 -- digit 2

        r1 -> number of machings (number of required changes)
        r2 -> difference between number of lines

        """
        n = [0,0]
        for x1,x2 in zip(d1,d2):
            if x1 and not x2:
                n[0]+=1
            elif not x1 and x2:
                n[1]+=1
        return max(n), n[0]-n[1]
    def check_lines(digit_index, current_digits, lines_cache):
        """ This function checks if interim result is stil possible
        p1 -> digit_index -- current index of digit
        p2 -> current_digits -- current interim digits
        p3 -> lines_cache -- interim cache of lines

        r  -> possibleness of the result

        """
        _sum = 0
        # calculate sum of lines (start at current index)
        for digit in current_digits[digit_index:]:
            _sum += sum(digit)
        if lines_cache > 0:
            # return if there is enough space for all lines that are in lines_cache
            return len(current_digits[digit_index:])*7-_sum >= lines_cache
        elif lines_cache < 0:
            # return if there are enough lines to balance lines_cache
            return _sum >= abs(lines_cache)
        return True
    def calculate_next_digit(digit_index, current_digits, lines_cache, left_n_of_changes)
    ↪ :
        """ Function that works recursive and calculates the next digit
        p1 -> digit_index -- current index of digit
        p2 -> current_digits- current interim digits
        p3 -> lines_cache -- interim cache of lines
        p4 -> left_n_of_changes -- left number of changes

        r  -> None

        """
        # reference on global variables
```

```
            global finished
            global best_digits
            global best_left_n_of_changes

            # check if interim solution is possible and the algorithm isn't finished
            if not finished and check_lines(digit_index, current_digits, lines_cache):
                cur_digit = current_digits[digit_index]
                # loop through digit dict reversed
                for new_digit in reversed(digit_dict.values()):
                    lines_cache_copy = lines_cache
                    # calculate matchings and difference between line numbers
                    n_this_change, this_lines_cache = calculate_dismatchings(cur_digit,
        ↪ new_digit)
                    # use lines cache to minimize changes
                    while lines_cache_copy > 0 and this_lines_cache < 0:
                        this_lines_cache += 1
                        n_this_change -= 1
                        lines_cache_copy -= 1
                    while lines_cache_copy < 0 and this_lines_cache > 0:
                        this_lines_cache -= 1
                        n_this_change -= 1
                        lines_cache_copy += 1
                    # check if enough changes are left
                    if n_this_change <= left_n_of_changes:
                        lines_cache_copy+=this_lines_cache
                        new_current_digits = current_digits
                        new_current_digits[digit_index] = list(new_digit)
                        # if this isn't the last digit calculate next
                        if digit_index+1 < len(current_digits):
                            # recursive call of this function
                            calculate_next_digit(digit_index+1, new_current_digits,
        ↪ lines_cache_copy, left_n_of_changes-n_this_change)
                        else:
                            # Check if solution is valid
                            if lines_cache_copy == 0:
                                # Save solution and finish the algorithm
                                finished = True
                                best_digits = new_current_digits.copy()
                                best_left_n_of_changes = left_n_of_changes-n_this_change
                                break
        global finished
        finished = False
        # start recursive function
        calculate_next_digit(0, digits.copy(), 0, max_n_of_changes)
        return best_digits, best_left_n_of_changes
    def main():
        """  Main Function that is executed at the beginning"""

        # get file
        digits, max_n_of_changes,filename = read_file()
        output_filename = 'beispielausgaben/{}.out'.format(filename.split('/')[-1].split('.')
        ↪ [0])

        # convert digits to 7-Segment format
        digits = convert_digits(digits)

        # Calculate digits
        final_digits,left_n_of_changes = calculate_final_digits(digits, max_n_of_changes)

        # print out steps
        if print_steps_var:
            steps = calculate_steps(digits, final_digits)
            step_output_filename = output_filename.split('.')[0]+'_steps.'+output_filename.
        ↪ split('.')[1]
            open(step_output_filename,'w').close()
            print_steps(steps, step_output_filename)

        # prinout results
        open(output_filename,'w').close()
        print_and_write('Before␣␣␣-->␣␣␣'+''.join(str(y)+'␣' for y in [reversed_digit_dict[
        ↪ tuple(x)] for x in digits]), output_filename)
        print_and_write('After␣␣␣␣-->␣␣␣'+''.join(str(y)+'␣' for y in [reversed_digit_dict[
        ↪ tuple(x)] for x in final_digits]), output_filename)
```

```
269     print_and_write('Left␣Changes␣␣␣-->␣␣'+str(left_n_of_changes), output_filename)

271 # only execute if script is direct executed
    if __name__ == '__main__':
273     main()
```

../source/Aufgabe-3-Hex-Max.py