

Aufgabe: Hex-Max

3.1 Lösungsidee

Wir möchten die größte Hex-Zahl finden, die mit maximal m Umlegungen aus einer gegebenen Hex-Zahl erzeugt werden kann.

Im Folgenden bezeichnen wir die ursprüngliche, gegebene Hex-Zahl mit A und mit A_i ($1 \leq i \leq n$) die i -te der n Ziffern. Wir beschreiben A_i als Menge, die die Nummern aller mit Stäbchen belegten Positionen (auch Segmente genannt; siehe Abbildung 3.1 für die von uns gewählte Nummerierung der Positionen) enthält: $A_i = \{1, 2, 5, 6\}$ entspräche z. B. der Ziffer 4, $A_i = \{2, 5, 6\}$ wäre keine Ziffer des Hexadezimalsystems. Aus A erzeugte Hex-Zahlen bezeichnen wir mit anderen Großbuchstaben wie B und einzelne Ziffern manchmal mit Kleinbuchstaben wie b ; die Notation gilt entsprechend.

Bei $A_i \setminus B_i$ handelt es sich daher um die Menge der Positionen, die in Ziffer A_i mit einem Stäbchen belegt sind, aber in B_i nicht. $|A_i \setminus B_i|$ und $|B_i \setminus A_i|$ sind darum die Anzahl der Stäbchen, die entfernt bzw. abgelegt werden müssen, um B_i aus A_i zu erhalten.

Wir verwenden die folgenden beiden Lemmata (mathematische Aussagen kleinen Umfangs – Hilfssätze – mit Beweis), um das Problem etwas umzuformulieren.

Lemma 1. *Wenn eine Hex-Zahl B aus der ursprünglichen Hex-Zahl A durch das Anheben von l Stäbchen und das Ablegen von l Stäbchen an bisher freien Positionen erzeugt werden kann, so kann sie auch durch das Umlegen von höchstens l Stäbchen im Sinne der Aufgabenstellung erzeugt werden, d. h. durch das einzelne Bewegen von höchstens l Stäbchen, ohne dass die Darstellung einer Ziffer dabei komplett geleert würde.*

Beweis. Wir betrachten den folgenden Algorithmus:

Bei der i -ten Ziffer müssen $|A_i \setminus B_i|$ Stäbchen angehoben und $|B_i \setminus A_i|$ Stäbchen abgelegt werden. Zunächst bewegen wir daher innerhalb jeder i -ten Ziffer $\gamma_i := \min(|A_i \setminus B_i|, |B_i \setminus A_i|)$ Stäbchen von ihrer bisherigen Position an eine zu belegende. Da wir nur innerhalb der Ziffer Stäbchen umlegen, kann die Ziffer dabei nicht geleert werden. Nun liegen noch $\alpha_i := |A_i \setminus B_i| - \gamma_i$ Stäbchen bei Ziffer i , die von dort entfernt werden müssen, und zu $\beta_i := |B_i \setminus A_i| - \gamma_i$ Positionen bei Ziffer i müssen noch Stäbchen bewegt werden. Man beachte, dass $\alpha_i = 0$ oder $\beta_i = 0$ gilt; an jeder Ziffer müssen also nur noch Stäbchen entfernt oder nur noch Stäbchen abgelegt werden. Damit kann auch die Bedingung, dass keine Darstellung einer Ziffer komplett geleert werden darf, nicht verletzt werden: Durch das Ablegen eines Stäbchens ohnehin nicht, und wenn wir von einer Ziffer lediglich Stäbchen entfernen und am Ende wieder eine (nicht leere) Ziffer erhalten, kann die Darstellung nicht dazwischen geleert gewesen sein.

Wir nehmen nun von einer beliebigen Ziffer, von der noch Stäbchen entfernt werden müssen ($\alpha_i > 0$), ein solches Stäbchen und legen es bei einer Ziffer, bei der noch Positionen belegt werden müssen ($\beta_i > 0$), an einer solchen ab. Das wiederholen wir, bis keine Stäbchen mehr angehoben bzw. abgelegt werden müssen.

Wir haben offensichtlich nicht mehr Stäbchen bewegt als mindestens angehoben bzw. abgelegt werden müssen, um B aus A zu erhalten, und damit insbesondere höchstens l Stäbchen. \square

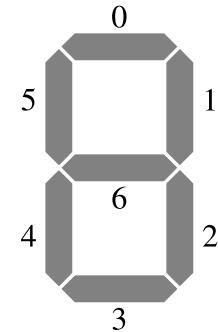


Abbildung 3.1: Nummerierung der Positionen

Lemma 2. Lässt sich eine Hex-Zahl B aus der ursprünglichen Hex-Zahl A mit höchstens m Umlegungen erzeugen, so kann sie auch durch das Anheben von l Stäbchen und das Ablegen von l Stäbchen erzeugt werden, mit $l \leq m$.

Beweis. Wenn B aus A mit $o \leq m$ Umlegungen erzeugt wird, sind danach höchstens o Positionen mit einem Stäbchen belegt, die davor nicht belegt waren ($\sum_i |B_i \setminus A_i| \leq o$), und höchstens o Positionen sind danach nicht mehr belegt, waren es aber davor ($\sum_i |A_i \setminus B_i| \leq o$). Da die Gesamtanzahl der Stäbchen gleich bleibt, sind beide Anzahlen gleich: $l := \sum_i |B_i \setminus A_i| = \sum_i |A_i \setminus B_i| \leq o \leq m$. Wenn wir also die l Stäbchen, die in A an Positionen liegen, die in B nicht mehr belegt sind, anheben, und dann an den Positionen ablegen, die in B belegt sind, aber in A noch frei, erhalten wir B auf die gewünschte Weise. \square

Wir wollen daher nun folgendes Problem lösen: Was ist die größte Hex-Zahl, die durch das Anheben von l Stäbchen und das Ablegen von l Stäbchen erzeugt werden kann, wobei $l \leq m$ sein muss? Wenn wir diese Hex-Zahl bestimmen können, können wir mit dem im Beweis von Lemma 1 skizzierten Algorithmus auch eine gültige Folge von Umlegungen finden, die sie erzeugt; außerdem bezieht dies nach Lemma 2 jede Hex-Zahl mit ein, die auch durch gültige Umlegungen zu erhalten ist, wir bestimmen so also tatsächlich die gesuchte maximale Hex-Zahl.

Eine wichtige Beobachtung benötigen wir für alle unsere Algorithmen: Eine größere Ziffer an einer Stelle der Hex-Zahl macht die Zahl größer als jegliche Änderungen darauf folgender („less significant“) Stellen. Formal: Es gilt stets $B > B'$, wenn $B_1 = B'_1, \dots, B_{i-1} = B'_{i-1}$ und $B_i > B'_i$, unabhängig von den Werten von B_{i+1}, \dots, B_n und B'_{i+1}, \dots, B'_n . Daher liegt ein Greedy-Ansatz nahe, bei dem man die Ziffern von links nach rechts jeweils so groß wie möglich zu machen versucht. Dabei muss jedoch darauf geachtet werden, dass mit den dafür nötigen Veränderungen auch noch rechts von $i + 1$ eine gültige Hex-Zahl erzeugt werden kann.

Dynamische Programmierung: Erster Ansatz

Wir definieren $r(a, b)$ als die Anzahl der Stäbchen, die übrig bleiben, wenn man die Ziffer a zur Ziffer b ändert; $r(a, b)$ kann negativ sein, dann benötigt man zusätzliche Stäbchen, um a zu b zu ändern. Außerdem sei $s(a, b)$ die Anzahl der Positionen von Stäbchen, die sich ändern, wenn man a zu b ändert. Es gilt also

$$\begin{aligned} r(a, b) &:= |a \setminus b| - |b \setminus a| \text{ und} \\ s(a, b) &:= |a \setminus b| + |b \setminus a|. \end{aligned}$$

Im Folgenden bezeichnen wir mit $[A_1 \cdots A_i]$ eine Hex-Zahl mit $1 \leq i \leq n$ Ziffern A_1, \dots, A_i .

Wir möchten die folgende Funktion berechnen:

$$\begin{aligned} dp : \{0, \dots, n\} \times \{0, \dots, m\} \times \{-m, \dots, m\} &\rightarrow \{\text{Hex-Zahlen}\} \cup \{-\infty\}, \\ (i, x, y) &\mapsto \begin{cases} \text{Größte Hex-Zahl } [B_1 \cdots B_i] \text{ mit } i \text{ Stellen,} \\ \text{die an genau } x \text{ Positionen anders als } [A_1 \cdots A_i] \text{ ist und} \\ y \text{ mehr Stäbchen als } [A_1 \cdots A_i] \text{ enthält.} \end{cases} \end{aligned}$$

Dabei darf y auch negativ sein; in dem Fall handelt es sich um eine Hex-Zahl mit weniger Stäbchen als $[A_1 \cdots A_i]$. Für jede Kombination von i, x, y , für die es keine solche Hex-Zahl gibt, setzen wir $dp(i, x, y) := -\infty$.

Wir stellen fest, dass $\max_{0 \leq x \leq 2m} dp(n, x, 0)$, also die größte Hex-Zahl mit höchstens $2m$ geänderten Positionen und unveränderter Anzahl Stäbchen, gerade die gesuchte größte Hex-Zahl nach Anheben von $l \leq m$ Stäbchen und Ablegen von l Stäbchen ist. Man beachte, dass zwangsläufig $dp(n, x, 0) = -\infty$ für ungerade x gilt: Wenn die Anzahl der Stäbchen gleich geblieben ist, muss von ebenso vielen Positionen ein Stäbchen entfernt worden sein, wie Positionen mit einem belegt wurden, d. h. die Summe dieser beiden gleichen Anzahlen ist gerade.

Außerdem ist $dp(0, 0, 0) = []$ die aus 0 Ziffern bestehende „leere“ Hex-Zahl.

Wie berechnen wir also diese Funktion? Es gilt die Rekursionsgleichung

$$dp(i, x, y) = \max_{b \in \mathcal{H}} \left[\underbrace{dp(i-1, x - s(A_i, b), y - r(A_i, b))}_{\text{Bisherige Hex-Zahl mit } i-1 \text{ Ziffern}} \underbrace{b}_{\text{Neue Ziffer}} \right]$$

für alle i, x, y , wobei $\mathcal{H} := \{0, 1, \dots, e, f\}$ die Menge der 16 Hex-Ziffern ist.

Wir können also für $i = 1, 2, \dots, n$ jeweils die Werte von $dp(i, x, y)$ für alle x, y berechnen und müssen dabei jeweils nur auf (schon berechnete) Werte von $dp(i-1, \cdot, \cdot)$ zurückgreifen.

Beim Berechnen eines Wertes der Funktion müssen 16 Möglichkeiten für b probiert werden; der Vergleich der Hex-Zahlen („Ist die aktuelle Hex-Zahl größer oder kleiner als die bisher größte gefundene?“) benötigt jeweils $i \leq n$ Operationen. Es gibt $(n+1)(m+1)(2m+1) \in \mathcal{O}(nm^2)$ Kombinationen von i, x, y , also ergibt sich eine Laufzeit von $\mathcal{O}(nm^2 \cdot 16 \cdot n) = \mathcal{O}(n^2m^2)$.⁹

Dynamische Programmierung: Zweiter Ansatz

Wir stellen einen zweiten Algorithmus, mit deutlich geringerer Laufzeit, vor.

Wir möchten dazu zunächst die folgende Funktion vorberechnen:

$$\begin{aligned} e : \{0, \dots, n\} \times \{-m, \dots, m\} &\rightarrow \{0, \dots, 7n, \infty\}, \\ (i, y) &\mapsto \min \left\{ \sum_{i < j \leq n} s(A_j, b_j) \mid b_{i+1}, \dots, b_n \in \mathcal{H}, y = \sum_{i < j \leq n} r(A_j, b_j) \right\}, \end{aligned}$$

die minimal nötige Anzahl an veränderten Stäbchen-Positionen in den Ziffern A_{i+1}, \dots, A_n , um von dort genau y Stäbchen übrig zu haben, wobei diese Ziffern wieder eine gültige Hex-Zahl ergeben sollen. Ist dies nicht möglich, definieren wir $e(i, y) := \infty$.

Dazu verwenden wir

$$\begin{aligned} e(n, 0) &= 0, \\ e(n, y) &= \infty \text{ für } y > 0 \text{ und die Rekursionsgleichung} \\ e(i, y) &= \min_{b \in \mathcal{H}} e(i+1, y - r(A_{i+1}, b)) + s(A_{i+1}, b) \text{ für } i < n, y \end{aligned}$$

(mit $e(\cdot, y) := \infty$ für alle $y \notin \{-m, \dots, m\}$).

Wir können die größte erreichbare Hex-Zahl bestimmen, indem wir von links nach rechts jeweils die größte Ziffer finden und wählen, mit der sich auch rechts der aktuellen Position noch eine gültige Hex-Zahl bilden lässt.

⁹Falls $m > \frac{7}{2}n$, können wir m durch $\frac{7}{2}n$ ersetzen (denn wir werden ein Stäbchen nie mehrfach umlegen), womit die Laufzeit sogar in $\mathcal{O}(n^2 \min(n, m)^2)$ liegt.

Wenn vor Stelle i bisher x Positionen von Stäbchen (bei Stelle $j < i$) geändert wurden und y Stäbchen „fehlten“, also von Stelle $j \geq i$ geholt werden müssen, ist

$$\max \{b \in \mathcal{H} \mid x + s(A_i, b) + e(i, y - r(A_i, b)) \leq 2m\}$$

die größte Ziffer, die an Stelle i gelegt werden kann. Dies lässt sich in folgenden *Greedy-Algorithmus* umsetzen:

Algorithmus 1 Greedy-Algorithmus zum Berechnen der größten erreichbaren Hex-Zahl

Berechne e vor

```

 $x \leftarrow 0$                                 ▷ Anzahl geänderter Positionen
 $y \leftarrow 0$                                 ▷ Anzahl fehlender Stäbchen
for  $i = 1, 2, \dots, n$  do
    for  $b = f, e, \dots, 1, 0$  do                ▷ Wähle greedy die größtmögliche Ziffer
         $x' \leftarrow x + s(A_i, b)$                   ▷ Neue Anzahl geänderter Positionen
         $y' \leftarrow y - r(A_i, b)$                   ▷ Neue Anzahl fehlender Stäbchen
        if  $x' + e(i, y') \leq 2m$  then            ▷ Anzahl mind. nötiger Änderungen  $\leq 2m$ ?
             $B_i \leftarrow b$ 
             $x \leftarrow x', y \leftarrow y'$ 
            break
        end if
    end for
end for
return  $B$ 
```

Algorithmus 1 hat abseits der Vorberechnung eine Laufzeit von nur $\mathcal{O}(n \cdot 16) = \mathcal{O}(n)$. In der Vorberechnung wird jedoch für alle $(n+1)(2m+1) \in \mathcal{O}(nm)$ Kombinationen von i, y jeweils in Laufzeit $\mathcal{O}(16) = \mathcal{O}(1)$ der Wert von $e(i, y)$ berechnet. Die Gesamlaufzeit ist also in $\mathcal{O}(nm)$.¹⁰

Berechnen der einzelnen Umlegungen

Sobald wir die größte Hex-Zahl berechnet haben, die mit maximal m Umlegungen erzeugt werden kann, können wir den in Lemma 1 skizzierten Algorithmus anwenden, um eine gültige Folge von Umlegungen zu finden, die sie erzeugt. Wir zeigen im Folgenden eine effiziente Umsetzung dieses Algorithmus mit Laufzeit $\mathcal{O}(n)$. Die Queues q_1, q_2 dienen dazu, nacheinander Paare von belegten, zu leerenden und leeren, zu belegenden Positionen abzuarbeiten.

¹⁰Analog zu Fußnote 9 liegt die Laufzeit sogar in $\mathcal{O}(n \min(n, m))$.

Algorithmus 2 Berechnen einzelner Umlegungen, um B aus A zu erhalten

```

for all  $1 \leq i \leq n$  do
    Lege  $\gamma_i$  Stäbchen von  $A_i \setminus B_i$  nach  $B_i \setminus A_i$  um
end for
 $q_1 \leftarrow$  Queue,  $q_2 \leftarrow$  Queue
for all  $1 \leq i \leq n$  do
    for all  $p \in A_i \setminus B_i$  do
         $q_1.\text{PUSH}((i, p))$ 
    end for
    for all  $p \in B_i \setminus A_i$  do
         $q_2.\text{PUSH}((i, p))$ 
    end for
end for
while  $|q_1| (= |q_2|) > 0$  do
     $(i_1, p_1) \leftarrow q_1.\text{POP}()$ 
     $(i_2, p_2) \leftarrow q_2.\text{POP}()$ 
    Lege Stäbchen von Position  $p_1$  bei Ziffer  $i_1$  zu Position  $p_2$  bei Ziffer  $i_2$  um
end while

```

3.2 Beispiele

Es folgen die Lösungen zu den gegebenen BWINF-Beispielen. Wie schon in der Beschreibung der Lösungsidee behandeln wir auch hier die Umlegungen gesondert und zeigen sie in einem eigenen, unteren Abschnitt.

Größte Hex-Zahl

Zunächst geben wir für jedes Beispiel die größte erreichbare Hex-Zahl an. Ziffern, die sich an mindestens einer Position geändert haben, sind rot markiert. Die Hex-Zahlen beginnen meist mit vielen fs (der größten Ziffer) und enden manchmal mit einigen 8en (der Ziffer mit den meisten Stäbchen – da das f unterdurchschnittlich viele Stäbchen hat, liegt die Erwartung nahe, dass am Ende ein Überschuss an Stäbchen besteht); der Teil dazwischen ist unterstrichen.

hexmax0.txt

EE4

hexmax1.txt

FFFEA97B55

hexmax2.txt

FFFFFFFFFF FFFFFFD9A9 BEAEE8EDA8 BDA989D9F8

hexmax3.txt

FFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF
 FFFFFFFFFFF FFFFAA98BB 8B9DFAEAE 888DD888AD 8BA8EA8888

hexmax4.txt

FFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFEB 8DE8BAA8A
DD88898E9 BA88AD9898 8F898AB7AF 7BDA8A61BA 7D4AD8F888

hexmax5.txt

FFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF
 FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF FFFFFFFFFFF
F88EFA9EBE 89EFA99FBD AA8E8EAD88
AB899F8E8F 9AA9E9AD88 988EDA9A99 888EDAD989 A8BAFD8A88
 8888888888 8888888888 8888888888 8888888888 8888888888
 8888888888 8888888888 8888888888 8888888888 8888888888
 8888888888 8888888888 8888888888 8888888888 8888888888
 8888888888 8888888888 8888888888 8888888888 8888888888
 8888888888 8888888888 8888888888 8888888888 8888888888

Einzelne Umlegungen

Wie vorgeschlagen geben wir nur für die Beispiele 0 bis 2 die einzelnen Umlegungen an; für die Beispiele 3 bis 5 belassen wir es bei der Ausgabe der jeweils größten erreichbaren Hex-Zahl. Im Folgenden zeigen wir jeweils den Zwischenstand nach jeder Umlegung.

hexmax0.txt

d24
 624
 E24
 EE4

hexmax1.txt

509C43 1655
609C43 1655
6A9C43 1655
6AAE43 1655
FFAE93 1655
FFAE93 1655
FFAE99 1655
FFFFEA97 1655

hexmax2.txt

FFFFFFFFFFFFFFFAd9A9bEAE8EdA8bdA989 19F8
FFFFFFFFFFFFFFFAd9A9bEAE8EdA8bdA989+9F8
FFFFFFFFFFFFFFFd9A9bEAE8EdA8bdA989-9F8
FFFFFFFFFFFFFFFd9A9bEAE8EdA8bdA989d9F8

3.3 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

- (1) *Problem adäquat modelliert:* Die Ziffern der Siebensegmentanzeige müssen geeignet modelliert und im Programm dargestellt werden. Für die Lösungsbeschreibung eignet sich die Darstellung als Menge von belegten Positionen oder Bitset. Bei der Implementierung des ersten Aufgabenteils ist die Darstellung als Zahl zwischen 1 und 16 am besten, im zweiten Teil aber nicht möglich. Für Darstellungen, die die Verarbeitung ineffizient machen, können Punkte abgezogen werden.
- (2) *Laufzeit des Verfahrens in Ordnung:* Für die ersten vier Beispiele müssen in kurzer Zeit Ergebnisse berechnet werden können. Liegt die Laufzeit auf dem Niveau des zweiten DP-Ansatzes oder besser, ist auch das fünfte Beispiel in kurzer Zeit lösbar; dies wurde in vielen Einsendungen erreicht. Wurde das 5. Beispiel nicht gelöst, so gibt es 2 Punkte Abzug, in noch schlechteren Fällen bis zu 4 Punkte.
- (3) *Verfahren nicht unnötig ineffizient:* Umständlichkeiten, etwa bei der Implementierung, welche die Laufzeit nicht prinzipiell, aber praktisch beeinträchtigen, können hier zu (eher geringem) Punktabzug führen.
- (4) *Speicherbedarf in Ordnung:* Die gegebene Hex-Zahl lässt sich problemlos direkt mit einem Speicherbedarf in $\mathcal{O}(n)$ darstellen. Der Speicherbedarf der gesamten DP-Tabelle soll $\mathcal{O}(n^2m)$ nicht deutlich überschreiten. Andersartige Verfahren sollten ebenfalls nicht mehr Speicher benötigen.
- (5) *Verfahren mit korrekten Ergebnissen:* Die ausgegebene Hex-Zahl muss jeweils die größte sein, die mit maximal m Umlegungen erzeugt werden kann. Das Verfahren muss außerdem eine gültige Folge an Umlegungen berechnen können, welche die berechnete Zahl erzeugt.
- (6) *Bedingungen werden eingehalten:*
 - Es muss garantiert sein, dass keine Ziffer je geleert wird; dies kann unter anderem bei globalen Stapelstrukturen passieren.
 - Es dürfen maximal m Umlegungen erzeugt werden.
 - Die erzeugte Zahl muss eine gültige Hex-Zahl in der der Siebensegmentdarstellung sein. So ist zum Beispiel eine Ziffer \dashv nicht erlaubt.

2. Theoretische Analyse

- (1) *Verfahren / Qualität insgesamt gut begründet:* Es muss gut begründet werden, dass das Verfahren korrekt ist, also u. a. die größte Hex-Zahl berechnet und die dazu berechnete Folge an Umlegungen gültig ist (keine Leerung von Ziffern).
- (2) *Gute Überlegungen zur Laufzeit des Verfahrens:* Die Laufzeit des Verfahrens muss nicht zwingend formal, aber nachvollziehbar und korrekt charakterisiert werden.

3. Dokumentation

- (3) *Vorgegebene Beispiele dokumentiert:* Es sollten alle vorgegebenen Beispiele bearbeitet und dokumentiert worden sein. Wenn nicht alle, aber mindestens 4 der 6 Beispiele dokumentiert wurden, gibt es 2 Punkte, bei weniger bis zu 4 Punkte Abzug. Wer nicht

den effizienteren DP-Ansatz implementiert hat, kann Beispiel 5 nicht in vertretbarer Zeit lösen; das sollte dann aber dokumentiert sein. Die Ausgabe der einzelnen Umlegungen ist nur für die Beispiele 0, 1 und 2 Pflicht.

(5) *Ergebnisse nachvollziehbar dargestellt:*

- Die Hex-Zahl muss in einem geeigneten Format ausgegeben werden.
- Der Zwischenstand nach jeder Umlegung muss lesbar ausgegeben werden. In der Regel ergibt sich nicht nach jeder Umlegung eine gültige Hex-Zahl, weshalb eine graphische Ausgabe der Siebensegmentdarstellung notwendig ist.
- In der Ausgabe der Zwischenstände muss jeder Schritt einer einzelnen Umlegung, die erste (noch nicht unbedingt ausgegebene) Zahl der Hex-Zahl der Eingabe und die letzte der Lösungs-Hex-Zahl entsprechen.