

Aufgabe 1: L^AT_EX-Dokument

Teilnahme-ID: ?????

Bearbeiter/-in dieser Aufgabe:
Vor- und Nachname

24. April 2025

Inhaltsverzeichnis

Anleitung: Trage oben in den Zeilen 8 bis 10 die Aufgabennummer, die Teilnahme-ID und die/den Bearbeiterin/Bearbeiter dieser Aufgabe mit Vor- und Nachnamen ein. Vergiss nicht, auch den Aufgaben-namen anzupassen (statt „L^AT_EX-Dokument“)!

Dann kannst du dieses Dokument mit deiner L^AT_EX-Umgebung übersetzen. Die Texte, die hier bereits stehen, geben ein paar Hinweise zur Einsendung. Du solltest sie aber in deiner Einsendung wieder entfernen!

1 Lösungsidee

TODO vlt kurzfassung/zusammenfassung

1.1 Huffman Codierung

TODO Quelle vlt: <https://people.eng.unimelb.edu.au/ammoffat/abstracts/compsurv19moffat.pdf>

David Huffman hat 1952 eine Methode veröffentlicht, die heute als *Huffman Codierung* bekannt ist. Er stellte eine Methode vor, um eine optimale Präfixfreie Codierung für einen bestimmten Text zu finden. Dabei geht man von einem Alphabet A mit n verschiedenen Zeichen $a_1, \dots, a_n \in A$ aus, gemeinsam mit der Häufigkeitsverteilung der Zeichen $p_i = \frac{\text{Häufigkeit von } a_i \text{ im zu codierenden Text}}{\text{Länge des Textes}}$. Man nennt diese Antreff-wahrscheinlichkeit p_i eines Zeichens auch *Frequenz* oder *Gewichtung*. Wir nehmen an, dass das Alphabet nach absteigender Häufigkeit sortiert ist, also $p_i \geq p_{i+1}$ für alle $0 \leq i < n - 1$. Der Text soll mit einem Ausgabealphabet O codiert werden, das aus r verschiedenen Zeichen $o_1, \dots, o_r \in O$ besteht. In der Bwinf-Aufgabenstellung besteht dieses Ausgabealphabet O aus den r verschiedenen Perlen (die aber alle den gleichen Durchmesser haben). Um den ursprünglichen Text zu codieren, müssen wir nun jedem Buchstaben a_i ein *Codewort* w_i zuordnen, dass aus einer Kette an Buchstaben aus dem Ausgabealphabet besteht $w_i = o_j o_k o_l \dots$.

Diese Codierung soll *Präfixfrei* sein, also kein Codewort soll Teil eines anderen Codeworts sein. Dadurch kann der Text als Aneinanderreihung von Codewörtern (ohne „Komma“ dazwischen) übertragen und ein-deutig decodiert werden.

Aufgrund von dieser Eigenschaft können wir den Code als einen Baum darstellen, der n Blätter hat und bei dem jeder Knoten höchstens r Kinder hat.

Definition 1. Wir nennen einen solchen Baum mit n blättern und höchstens r Kindern *valide*, da er eine mögliche Codetabelle darstellt.

Jedes Blatt eines solchen Baums repräsentiert ein Codewort w , dass eindeutig durch den Pfad von der Wurzel des Baums zu diesem Blatt definiert ist. Der Pfad wird durch die Kanten des Baums definiert, die mit den Perlen beschriftet sind. Da alle Perlen gleich groß sind, also die Länge aller Buchstaben des Ausgabealphabets gleich ist, ist die Beschriftung der Kanten beliebig. Es ist lediglich wichtig, dass alle Kanten eines Knotens mit unterschiedlichen Buchstaben beschriftet sind. Die Länge des Codeworts $|w|$

entspricht der Anzahl der Kanten auf dem Pfad von der Wurzel zu diesem Blatt.

Da wir eine optimale Codetabelle (also eine möglichst kurze Perlenkette) erstellen wollen, müssen wir die „Kosten“ eines Baums definieren. Diese Kosten eines Baums hängen natürlich auch davon ab, welches Codewort welchem Buchstaben zugeordnet wird. Für diese Definition gehen wir davon aus, dass wir eine solche Zuordnung $w_i \rightarrow a_i$ haben.

Definition 2. Die Kosten eines Baums T ist definiert als das Produkt der Länge jedes Codeworts w_i mit der Frequenz p_i des codierten Buchstabens a_i :

$$\text{cost}(T) = \sum_{i=0}^n |w_i| \cdot p_i$$

Da die Kosten eines Baumes proportional zu der Länge der resultierenden Perlenkette sind, kann die Aufgabenstellung darauf reduziert werden, den validen Baum T mit den minimalen Kosten zu finden.

Dafür ist aber auch die Zuordnung, welcher Buchstabe von welchem Codewort codiert werden soll, entscheidend. Für diese Aufgabenstellung ist die Zuordnung aber für jeden Baum eindeutig, da wir die Kosten des Baums minimieren wollen. Um das zu erreichen, müssen wir das längste Codewort zu dem Buchstaben zugeordnet, der am seltensten vorkommt und das kürzeste Codewort zu dem Buchstaben zuordnen, der am häufigsten vorkommt. Dafür ordnen wir im Folgenden die Blätter Baumes in aufsteigender Reihenfolge, also $|w_0| \leq \dots \leq |w_n|$. Da das Alphabet nach absteigender Reihenfolge sortiert ist, also $p_0 \geq \dots \geq p_n$, liefert die Zuordnung von w_i zu a_i für jeden Baum die geringsten Kosten.

Definition 3. Wir nennen einen solchen Baum, der die minimalen Kosten (kürzeste Perlenkette) für eine bestimmte Häufigkeitsverteilung hat **optimal**

1.1.1 Konstruktion des optimalen Baums

Definition 4. Wir nennen einen r -när Baum, bei dem jeder Knoten maximal r Kinder hat **vollständig**, wenn jeder Knoten genau r Kinder hat.

Lemma 1. Ein optimaler binärer Huffman Baum ist vollständig. TODO braucht man das??

Beweis. Ein optimaler binärer Huffman Baum ist im allgemeinen vollständig, da in einem Optimalen Baum kein Knoten mit nur einem Kind existieren kann. Würde man diesen Knoten zu einem Blatt machen hätte man einen Baum mit der gleichen Anzahl an Blättern aber geringeren Kosten. \square

Das gilt im allgemeinen Fall bei Bäumen mit $r > 2$ Kindern aber nicht. Für die allgemein bekannte Konstruktion (TODO Quelle) eines binären Huffman Baums werden wiederholt zwei Teilbäume zu einem zusammengefasst, bis schließlich nur noch einer übrig ist. Im allgemeinen Fall müssen immer r Teilbäume zu einem zusammengefasst werden, was aber nicht für jede Anzahl an Buchstaben n im Alphabet immer funktioniert.

Beispiel 1. Für $n = 4$ werden im ersten Schritt drei Knoten zu einem zusammengefasst. Im nächsten sind nur noch zwei Knoten übrig. Würde man diese Knoten nun zu einem fertigen Baum kombinieren wäre dieser für bestimmte Häufigkeitsverteilungen (z.B. $p_0 = p_1 = p_2 = p_3$) nicht optimal.

Daher muss das Alphabet zuerst mit so vielen Platzhalterbuchstaben, deren Auftretenswahrscheinlichkeit 0 ist, aufgefüllt werden, dass gilt: $n \bmod (r - 1) = 1$.

Jetzt können wir mit der Konstruktion beginnen:

1. Erstelle für jeden Buchstaben a_i im Alphabet (inklusive Platzhalter) einen Teilbaum (der aus einem Wurzelknoten besteht) und beschrifte diesen mit der Frequenz p_i des Buchstabens a_i .
2. Wähle die r Teilbäume mit den kleinsten Auftretenswahrscheinlichkeiten. (Diese Wahl ist nicht eindeutig)
3. Beschrifte einen neuen Knoten mit der Summe der Auftretenswahrscheinlichkeiten der r gewählten Teilbäume und erstelle einen neuen Teilbaum mit dem neuen Knoten als Wurzel und den gewählten Teilbäumen als Kinder.
4. Wieder hole Schritt 2 und 3, bis nur noch ein Baum übrig ist.

TODO Bilder

TODO Quelle für allgemeine Konstruktion

1.1.2 Beweis der optimalität

Der Standardbeweis für die Optimalität von Binären Huffman Bäumen kann leicht verallgemeinert werden, daher Dazu müssen wir zuerst das „Sibling-Lemma“ auf r -när Bäume verallgemeinern:

Lemma 2. *Seien l_0, l_1, \dots, l_{r-1} die r Buchstaben mit der geringsten Antreffwahrscheinlichkeit. Sie sind in einem Huffman Baum „Geschwister“, also Kinder des gleichen Knotens Y und liegen auf der untersten Ebene des Baums.*

Beweis. Die Buchstaben l_0, l_1, \dots, l_{r-1} sind sicher Geschwister, da sie als erstes ausgewählt werden. Wenn l_0, l_1, \dots, l_{r-1} nicht auf der untersten Ebene des Baums liegen würden, dann müsste es einen internen Knoten X geben, dessen Antreffwahrscheinlichkeit kleiner ist als die des Knotens Y , dessen Kinder l_0, l_1, \dots, l_{r-1} sind. Dafür müsste aber mindestens einer der Kinder von X eine geringere Antreffwahrscheinlichkeit haben als einer der Kinder l_0, l_1, \dots, l_{r-1} von Y . Das widerspricht aber unserer anfänglichen Annahme, dass l_0, l_1, \dots, l_{r-1} die r Buchstaben mit der geringsten Antreffwahrscheinlichkeit sind. Daher können wir sagen, dass l_0, l_1, \dots, l_{r-1} immer auf der untersten Ebene des Baums liegen. \square

Damit können wir nun mithilfe von Induktion über die Größe n des Alphabets den Beweis durchführen:

Satz 1. *Die oben beschriebene Konstruktion für r -näre Huffman Bäume liefert einen optimalen Baum und damit eine bestmögliche Codetabelle.*

Beweis. Induktionsanfang: Für $n \leq r$ ist ein Huffman Baum offensichtlich optimal.

Induktionshypothese: Alle Huffman Bäume mit $< n$ Blättern sind optimal.

Induktionsschritt: Nun müssen wir zeigen, ein Huffman Baum T mit n Blättern optimal ist. Seien l_0, l_1, \dots, l_{r-1} wieder die r Buchstaben mit der geringsten Antreffwahrscheinlichkeit. Sei Y der Knoten in T , dessen Kinder l_0, l_1, \dots, l_{r-1} sind. Sei T' nun ein Huffman Baum identisch zu T , bei dem Y zu einem Blattknoten konvertiert wurde (mit der gleichen Antreffwahrscheinlichkeit wie der Knoten Y in T). Aufgrund der Induktionshypothese ist T' optimal, da er $n - (r - 1) < n$ Blätter hat.

Wenn T nicht optimal wäre, gäbe es einen Baum T_1 mit geringeren Kosten als T . Aus T_1 könnte man wieder einen Baum T'_1 mit $n - (r - 1)$ Blättern erstellen, in dem man Y zu einem Knoten konvertiert. Wenn T_1 geringere Kosten als T hätte, hätte auch T'_1 geringere Kosten als T' , woraus man schließen kann, dass T optimal sein muss. \square

1.1.3 Pseudocode

1.1.4 Laufzeit

1.2 TODO Titel

TODO entweder bei 1 anfangen und bis r oder bei 0 anfangen und bis $r-1$ Im Aufgabenteil b) haben die Perlen o_1, \cdot, o_r unterschiedliche, ganzzahlige Durchmesser c_1, \cdot, c_r . Wir sortieren die Perlen ab jetzt so, dass gilt: $c_1 \leq c_2 \leq \dots \leq c_r = C$ und wir bezeichnen den Durchmesser der größten Perle c_r mit C .

Wir eine Codetabelle, die mit unterschiedlich großen Perlen erstellt wurde ähnlich wie in Aufgabenteil a) wieder als Baum darstellen.

Definition 5. *Wir bezeichnen die Tiefe eines Knotens V mit $\text{depth}(V)$. Die Tiefe des Wurzelknotens ist 0.*

Bis jetzt waren die Kinder v_0 bis v_{r-1} eines Knotens V immer genau eine Ebene tiefer als V , also $\text{depth}(v_i) = \text{depth}(V) - 1$. Wir wollen die Eigenschaft, dass die Pfadlänge von der Wurzel eines Baumes zu einem Blatt $|w|$ der Länge der entsprechenden Aneinanderreihung an Perlen entspricht aber natürlich weiterhin erhalten. Wenn wir also die Kinder v_0 bis v_{r-1} des Knotens V mit den Perlen o_1 bis o_r beschriften $v_i \rightarrow o_i$, soll gelten: $\text{depth}(V) = \text{depth}(v_i) + c_i$.

1.2.1 Huffman

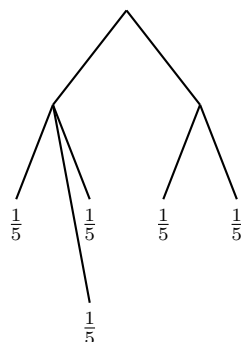
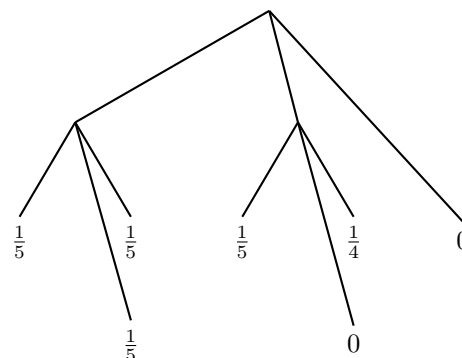
Da die oben beschriebene Huffman-Codierung den Spezialfall $c_1 = c_2 = \dots = c_r$ darstellt, ist es zunächst sinnvoll, zu betrachten, weshalb der Huffman Algorithmus hier keinen optimalen Baum liefert. TODO (Die tiefsten Knoten sind nicht zwingend Nachbarn, weil $c_1 = c_2$ etc)

1.2.2 Vollständige Bäume

TODO Quelle hier (TODO vlt kuuz auf Paper eingehen) Der und der haben 19irgendwas einen neuen Ansatz vorgestellt... sehr ausführlich hier fokussiert auf das wesentliche)

Da es in diesem Aufgabenteil also nicht möglich ist, die Bäume von unten nach oben mit einem Greedy-Algorithmus zu erstellen, müssen wir alle mögliche Lösungen betrachten. In diesem Kapitel werden wir uns zuerst auf vollständige Bäume beschränken, indem wir ähnlich wie in Teil a) fehlende Buchstaben durch Platzhalter mit Frequenz 0 ersetzen:

Definition 6. Für jeden Baum T erhält man einen vollständigen Baum $Fill(T)$, indem man an jeden Knoten so viele zusätzliche Blätter mit Frequenz 0 hinzufügt, das dieser genau r Blätter besitzt.

Abbildung 1: Ein nicht vollständiger Baum T Abbildung 2: Der vervollständigte Baum $Fill(T)$

Lemma 3. Sei T ein optimaler Baum mit n Blättern. Die Anzahl der Blätter m von $Fill(T)$ beträgt höchstens $m \leq n(r-1)$.

Beweis. TODO □

Da $cost(Fill(T)) = cost(T)$ offensichtlich gilt, können wir nun das Problem umformulieren: Finde für die Perlen Durchmesser c_1, \dots, c_r und die Frequenzen $p_1 \geq \dots \geq p_n$ (und $p_i = 0$ für alle $i > n$) den vollen Baum T_{opt} mit m Blättern ($n \leq m \leq n(r-1)$) mit minimalen Kosten:

$$cost(T_{opt}) = \min\{cost(T) : T \text{ ist vollständig und hat } m \text{ Blätter } (n \leq m \leq n(r-1))\}$$

Wenn wir T_{opt} konstruiert haben, können wir die Blätter mit Frequenz 0 entfernen und erhalten eine optimale Codierungstabelle für n Buchstaben.

Dafür müssen wir aber zuerst betrachten, wie wir vollständige Bäume allgemein darstellen und Ebene für Ebene konstruieren können.

Definition 7. TODO braucht man das und wenn ja hier? Wir nennen einen Baum Ebene- i -Baum, wenn alle internen Knoten auf einer Ebene $\leq i$ liegen.

Definition 8. Wir kürzen einen Baum T zu dem Ebene- i -Baum $Cut_i(T)$, indem wir alle Knoten entfernen, deren Eltern Tiefer als Ebene- i liegen:

$$Cut_i(T) = T - \{v \in T \mid |parent(v)| > i\}$$

TODO notation für |Tiefe| einführen TODO Beispiel

Definition 9. Die Signatur einer Ebene i des Baums T ist das $C+1$ -Tupel

$$sig_i(T) = (m; l_1, l_2, \dots, l_C)$$

wobei $m = |\{v \in T \mid v \text{ ist ein Blatt, tiefe}(v) \leq i\}|$ der Anzahl der Blätter von T mit einer Tiefe von höchstens i entspricht und

$$l_k = |\{u \in T \mid tiefe(u) = i + k\}, k \in \{1, \dots, C\}$$

die Anzahl der Knoten auf Ebene $i+k$ ist.

Diese Signatur wird es uns später erlauben, alle möglichen Bäume zu betrachten und dabei aber kleine Unterschiede, die nichts an den Kosten eines Baums ändern, zu vernachlässigen.

TODO Beispiele mit signatur und expand

Über die Signatur einer Ebene- i -Baums können wir nun die Kosten dieser Ebene definieren, was uns später dabei helfen wird, den „Billigsten“ Baum zu finden.

Definition 10. Sei T ein Ebene- i Baum mit der Signatur $\text{sig}_i(T) = (m, l_1, l_2, \dots, l_C)$. Für $m \leq n$ definieren wir die Kosten der Ebene- i als:

$$\text{cost}_i(T) = \sum_{j=1}^m |w_j| \cdot p_j + i \cdot \sum_{j=m+1}^n p_j$$

wobei $|w_1| \leq \dots \leq |w_m|$ die m höchsten Blätter nach Tiefe geordnet sind.

Der erste Term der Summe beschreibt Kosten der Blätter, für die bereits sicher ist, dass es sich um Blätter handelt, während der zweite Term die Kosten für die übrigen Blätter um die Ebene i zu erreichen darstellt. Für $m \geq n$ erübrigt sich der zweite Term, da $p_i = 0$ für $i > m$. D.h. wenn T ein Ebene- i -Baum mit $\text{sig}_i(T) = (m, l_1, \dots, l_C)$ ist und $m \geq n$, dann gilt $\text{cost}_i(T) = \text{cost}(T)$.

Definition 11. Sei (m, l_1, \dots, l_C) eine valide Signatur. Wir bezeichnen die minimalen Kosten eines Baumes mit dieser Signatur mit $\text{OPT}[m, l_1, \dots, l_C]$:

$$\text{OPT}[m, l_1, \dots, l_C] = \min\{\text{cost}_i(T) \mid T \text{ ist ein Ebene-}i\text{-Baum mit } \text{sig}_i(T) = (m, l_1, \dots, l_C)\}$$

Da die maximale Anzahl an Blättern des optimalen vervollständigen Baums $\text{Fill}(T_{\text{opt}})$ $n(r-1)$ beträgt, entsprechen die Kosten dieses Baums dem Minimum aller Kosten $\text{OPT}[m, l_1, \dots, l_C]$ mit $m \geq n$ und $m + l_1 + \dots + l_C \leq n(r-1)$:

$$\text{cost}(\text{Fill}(T_{\text{opt}})) = \text{cost}(T_{\text{opt}}) = \min\{\text{OPT}[m, l_1, \dots, l_C] \mid m \geq n \wedge m + l_1 + \dots + l_C \leq n(r-1)\}.$$

Um dieses Minimum zu ermitteln müssen wir die OPT -Tabelle Schritt für Schritt auffüllen, beginnend bei einem Ebene-0-Baum, der nur aus der Wurzel und dessen Kindern besteht. Sei T also ein Ebene- i -Baum und T' ein Ebene- $i+1$ -Baum, der bis auf Ebene- i identisch zu T ist, also: $\text{Cut}_i(T') = T$. T und T' haben also die selbe Anzahl an Knoten auf den Ebenen 0 bis i . Daher ist auch die Anzahl der Knoten auf Ebene $i+1$ identisch, der einzige Unterschied besteht darin, ob diese Knoten Blätter oder interne Knoten sind. In T sind alle diese Knoten Blätter, da T ein Ebene- i -Baum ist. In T' könnten auch alle Knoten auf Ebene $i+1$ Blätter sein, es könnte aber auch eine bestimmte Anzahl q an Knoten interne Knoten sein, maximal aber natürlich so viele, wie T Blätter in Ebene $i+1$ hat. Sei $\text{sig}_i(T) = (m, l_1, \dots, l_C)$ die Ebene- i Signatur von T . Damit können wir sagen, dass $0 \leq q \leq l_1$, da l_1 die Anzahl der Blätter von T auf Ebene $i+1$ ist.

Definition 12. Sei T ein Ebene- i -Baum mit der Signatur $\text{sig}_T = (m; l_1, l_2, \dots, l_C)$. Die Expand Operation wandelt $0 \leq q \leq l_1$ Blätter auf Ebene $i+1$ in interne Knoten um, indem r Kinder an jedes dieser Blätter angehängt werden:

$$T' = \text{Expand}_i(T, q)$$

Die Expand-Operation legt nicht fest, welche der l_1 Blättern zu Knoten werden. Da wir uns auf vollständige Bäume beschränken, sind die Kosten dieser Bäume aber gleich, selbst wenn die Bäume selbst geringfügige Unterschiede haben. Um die Änderung der Signatur des Baums zu quantifizieren, müssen wir den charakteristischen Vektor einer Perlensammlung mit den Durchmessern c_1, c_2, \dots, c_r definieren:

Definition 13. Der charakteristische Vektor (d_1, d_2, \dots, d_C) einer Perlensammlung mit den Durchmessern $D = (c_1, c_2, \dots, c_r)$ entspricht den Anzahlen, wie oft ein bestimmter Durchmesser in der Perlensammlung vertreten ist:

$$d_i = \text{Anzahl von } i \text{ in } D$$

Beispiel 2. Der charakteristische Vektor der Beispieldatei 7 mit den Durchmessern $D = (1, 1, 1, 1, 1, 1, 1, 2, 3, 4)$ beträgt $(7, 1, 1, 1)$ und der charakteristische Vektor der Beispieldatei 5 mit den Durchmessern $D = (1, 1, 2, 2, 3)$ beträgt $(2, 2, 1)$.

Lemma 4. Sei T ein Ebene- i -Baum mit der Signatur $\text{sig}_T = (m; l_1, l_2, \dots, l_C)$ und $T' = \text{Expand}_i(T, q)$ die Erweiterung um q . Die Signatur von T' ist dann:

$$\text{sig}_{i+1}(T') = (m + l_1, l_2, \dots, l_C, 0) + q \cdot (-1, d_1, d_2, \dots, d_C)$$

(Multiplikation und Addition werden Elementweise ausgeführt). Die Kosten der Erweiterung betragen:

$$\text{cost}_{i+1}(\text{Expand}_i(T, q)) = \text{cost}_i(T) + \sum_{m < j \leq n} p_j$$

Beweis. TODO

□

Beispiel 3. TODO

Lemma 5. WICHTIG

$\text{cost}(\text{Fill}(T_{\text{opt}})) = \text{cost}(T_{\text{opt}})$ -> ist glaub schon fix

$\text{Fill}(T_{\text{opt}})$ wird von der naiven Implementierung gefunden ()

Für alle T mit $\text{sig}_i = (m, l_1, l_2, \text{etc})$ und $m \geq n$ gilt $\text{cost}(T) = \text{cost}(\text{sig}_i)$ -> Damit ist die naive methode optimal

Nein... Wir können nun einen Graph $G = (V, E)$ definieren, dessen Knoten die Signaturen der Bäume darstellen und dessen Kanten Erweiterungen auf die nächste Ebene repräsentieren:

$$V = \{(m, l_1, \dots, l_C) \mid m, l_1, \dots, l_2 \geq 0 \wedge m + l_1 + \dots + l_C \leq n(r-1)\}$$

$$E = \{(S = (m, l_1, \dots, l_C), S') \in V \times V \mid \exists q, 0 \leq q \leq l_1, S' = (m + l_1, l_2, \dots, l_C, 0) + q \cdot (-1, d_1, \dots, d_C)\}$$

In diesem Graph könnte man nun mithilfe von Breitensuche den Pfad finden, der zu

TODO man könnte jetzt bfs machen kurz laufzeit und ist schlecht TODO (wieso darf man bfs, eigentlich doch dijkstra?) des eine ist gar kein bfs!!!!

1.3 TODO Titel

Da dieser Algorithmus für große Eingabealphabeten ($n \gg 100$) deutlich zu langsam ist und zu viel Speicher verwendet, ...

2 Umsetzung

Hier wird kurz erläutert, wie die Lösungsidee im Programm tatsächlich umgesetzt wurde. Hier können auch Implementierungsdetails erwähnt werden. TODO erweiterung: die Kosten der Perlen sind keine ganzen Zahlen mehr, sondern reelle Zahlen.

3 Beispiele

Genügend Beispiele einbinden! Die Beispiele von der BwInf-Webseite sollten hier diskutiert werden, aber auch eigene Beispiele sind sehr gut – besonders wenn sie Spezialfälle abdecken. Aber bitte nicht 30 Seiten Programmausgabe hier einfügen!

4 Quellcode

Unwichtige Teile des Programms sollen hier nicht abgedruckt werden. Dieser Teil sollte nicht mehr als 2-3 Seiten umfassen, maximal 10.