

Informatik 1

Assignment Project Exam Help

<https://tutorcs.com>

12 – Use Case: Huffman-Trees

WeChat: cstutorcs

Winter 2020/21

Torsten Grust

Universität Tübingen, Germany

1 | Zeichencodierungen

Zeichencodierungen bilden Zeichen auf Sequenzen von Bits ab. Die meisten dieser Codes nutzen eine *fixe Anzahl* von Bits:

Code	Details
ASCII	Codes 0-127, 7 Bit, American Standard Code for Information Interchange
ISO-8859-1	Codes 0-255, 8 Bit, 191 lateinische + Steuerzeichen
Unicode	20 Bit, Unicode 13.0 codiert 143859 Zeichen aktueller/historischer Scripts oder "Alphabete" Beispiel: Zeichen € = 0000 0010 0000 1010 1100 0 2 0 a c

```
; Zeichen "€" in Unicode-Codierung
(: euro-symbol string)
(define euro-symbol "\U020ac")
```

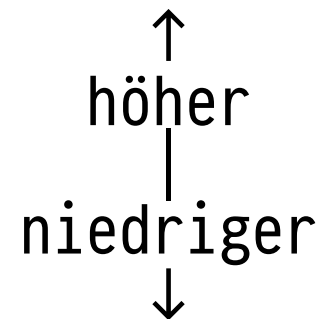
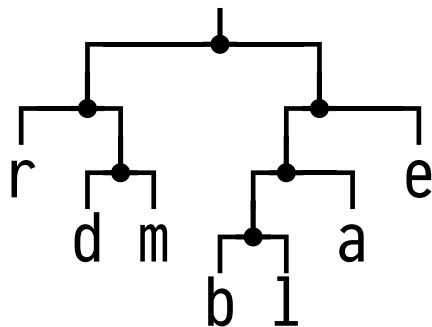
Huffman-Codes

Huffman-Codes nutzen Bitsequenzen *variabler Länge*.

- 💡 **Idee:** Zeichen mit hoher Frequenz werden mit weniger Bits codiert, als seltene Zeichen \Rightarrow **Datenkompression**. Einsatz in JPEG, MP3 und ZIP.

Assignment und Project Exam Help

- Huffman-Codes sind Binärbäume, deren Blätter Labels tragen. Beispiel: Huffman-Code für "erdbeermarmelade":¹

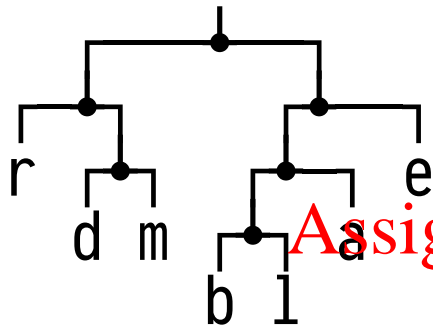


Zeichenfrequenz

¹ Dieser Huffman-Code kann Texte codieren, die (nur) die Zeichen a,b,d,e,l,m und r enthalten.

Huffman-Codes \equiv Pfade im Huffman-tree

- **Pfade** im Huffman-Tree codieren die vorhandenen Zeichen:



Code für Zeichen c:

Pfad von Wurzel bis Blatt mit Label c:

Abstieg in linken Teilbaum \equiv Bit 0

Abstieg in rechten Teilbaum \equiv Bit 1

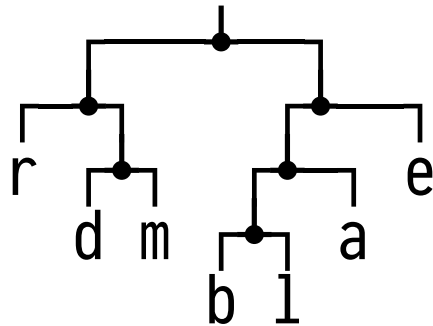
<https://tutorcs.com>

- Zeichencodes im Huffman-Tree für "erdbeermarmelade":

WeChat: cstutorcs

Zeichen	Frequenz	Code (Bits)
e	5	11
r	3	00
m	2	011
a	2	101
d	2	010
b	1	1000
l	1	1001

Huffman-Codes sind präfixfrei



e	≡	11		d	≡	010
r	≡	00		b	≡	1000
m	≡	011		l	≡	1001
a	≡	101				

- Huffman-Codes sind **präfixfrei**: die Bits eines Zeichens (= Blatt) sind *niemals* ein Präfix eines anderen Zeichens.
 - Ermöglicht die **eindeutige Decodierung** von Bitsequenzen (⤴ ≡ navigiere von der Wurzel des Huffman-Trees aus):

⤴1001⤴101⤴011⤴1000⤴010⤴101
 l a m b d a

- Länge: 20 Bit (Unicode hätte 120 Bit benötigt).

2 | Racket: Huffman-Trees (1)


```

; Ein Blatt eines Huffman-Tree (huff-leaf)
; - trägt ein Label (Zeichen c):
(: make-huff-leaf (%a -> (huff-leaf-of %a)))
(: huff-leaf-label ((huff-leaf-of %a) -> %a))
(define-record-procedures-parametric huff-leaf huff-leaf-of
  make-huff-leaf
  huff-leaf?
  (huff-leaf-label))

; Ein innerer Knoten eines Huffman-Tree (huff-node) besitzt
; - einen linken Teilbaum (left) und
; - einen rechten Teilbaum (right):
(: make-huff-node (%a %b -> (huff-node-of %a %b)))
(: huff-node-left ((huff-node-of %a %b) -> %a))
(: huff-node-right ((huff-node-of %a %b) -> %b))
(define-record-procedures-parametric huff-node huff-node-of
  make-huff-node
  huff-node?
  (huff-node-left huff-node-right))

```

Assignment Project Exam Help
<https://tutorcs.com>
 WeChat: cstutorcs



Racket: Huffman-Trees (2)

- **Huffman-Trees** sind eine Variante von Binärbäumen:

```

; Signatur (huff-tree-of t): Huffman-Tree mit Blättern
; mit Labeln der Signatur t
(define huff-tree-of
  (lambda (t)
    (signature
      (mixed (huff-leaf-of t)
              (huff-node-of (huff-tree-of t) (huff-tree-of t))
            )
    )
  )
)

```

Assignment Project Exam Help
<https://tutorcs.com>
 WeChat: cstutorcs

- Huffman-codierte Zeichen \equiv **Bit**-Sequenzen (*list-of bit*):

```

; Ein Bit eines Zeichencodes
(define bit
  (signature (one-of 0 1))) ; oder ("L", "R"), ("↙", "↘"), ...

```

3 | Ein einfaches API für Huffman-Codierung

1. Decodieren einer Bit-Sequenz:

```
(: huff-decode
  ((huff-tree-of string) (list-of bit) -> string))
```

2. Codieren eines Strings:

Assignment Project Exam Help

<https://tutorcs.com>

```
(: huff-encode
  ((huff-tree-of string) string -> (list-of bit)))
```

$\forall \text{ string } s: (\text{huff-decode } ht \ (\text{huff-encode } ht \ s)) = s$

3. Huffman-Tree für gegebenen (Referenz-)Text konstruieren:

```
(: huffman-code (string -> (huff-tree-of string)))
```


4 | API: Decodieren einer Bit-Sequenz (Erste Planskizze)

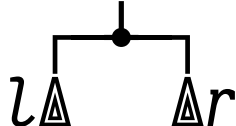
- Ein erster Plan für einen Worker `dec` für `huff-decode`:

```
(: dec (huff-tree-of %a) (list-of bit) -> (list-of %a))
```

1 (dec $\overset{|}{c}$ *bits*) = (make-pair *c* ??)

2 (dec Δ empty) = empty

3₀ (dec  (make-pair 0 *bits*)) = (dec *l* Δ *bits*)

3₁ (dec  (make-pair 1 *bits*)) = (dec Δ *r* *bits*)

- Fall 1/?: Im Huffman-Tree bis zum Blatt mit Label *c* navigiert. Wiedereinsteig an der Wurzel (s. ↑, Folie 5)?

API: Decodieren einer Bit-Sequenz (Fertiger Plan)


💡 **Idee:** Schleife die Wurzel des Huffman-Tree $ht\Delta$ durch die Rekursion. Wiedereinstieg an der Wurzel (\uparrow) ist nun einfach:

- 1 $(\text{dec } ht\Delta \overset{c}{\Delta} bits) = (\text{make-pair } c \text{ } (\text{dec } ht\Delta \overset{\uparrow}{ht\Delta} bits))$
- 2 $(\text{dec } ht\Delta \Delta \text{ empty}) = \text{empty}$
- 3 $_0 (\text{dec } ht\Delta \overset{\text{tree}}{\Delta} (\text{make-pair } 0 \text{ } bits)) = (\text{dec } ht\Delta \overset{\text{left child}}{\Delta} bits)$
- 3 $_1 (\text{dec } ht\Delta \overset{\text{tree}}{\Delta} (\text{make-pair } 1 \text{ } bits)) = (\text{dec } ht\Delta \overset{\text{right child}}{\Delta} bits)$

- **Quiz:** Ein endrekursives dec ist einfach zu erhalten. Wie?

5 | API: Codieren eines Strings s (Plan ①)

1. Codiere ein einzelnes Zeichen c :

- Suche c via Tiefensuche von der Wurzel von ht aus. Protokolliere den Pfad beim Abstieg als Bit-Sequenz.
- **Q:** Wie reagieren wir, wenn uns die Tiefensuche zu einem Blatt mit Label $x \neq c$ führt?
- **💡:** Verfolge in jedem inneren Knoten  Teilbäume $l\Delta$ und Δr . Suche schlägt entweder in $l\Delta$ oder Δr fehl. Liefere leere Bit-Sequenz bei Fehlschlag \downarrow . Beachte:

$$\forall xs: (\text{append empty } xs) = xs = (\text{append } xs \text{ empty})$$

- ### 2. Codiere Zeichen des Strings s wie in Schritt 1 (map).
- Verbinde Bit-Sequenzen zur gesamten Codierung (concat).

API: Codieren eines Strings s (Plan ②)

- Plan ② für `huff-encode` geht zweiphasig vor (👉 Übung).
1. **Phase 1:** Führe *einmalig* eine Tiefensuche im Huffman-Tree durch. **Konstruiere dabei eine sortierte Code-Tabelle:**

Assignment Project Exam Help

Zeichen	Code
a	(list 0 1 1 0 ...)
b	(list 0 1 1 1 ...)
⋮	
z	(list 1 0 1 0 ...)

<https://tutorcs.com>

WeChat: cstutorcs

2. **Phase 2:** **Codiere ein Zeichen c** durch *Lookup* in dieser Tabelle. Dazu ist der Huffman-Tree unnötig.
3. **Codiere Zeichen des Strings s** wie in Schritt 2 (`map`).
Verbinde Bit-Sequenzen zur gesamten Codierung (`concat`).

6 | API: Huffman-Tree für Text *txt* konstruieren (Schritt 1.)

Plan zur Erstellung eines optimalen Huffman-Trees für einen gegebenen (Referenz-)Text *txt*:

1. Stelle **Häufigkeit des Vorkommens jedes Zeichens** in *txt* fest. Organisiere die Ergebnisse in Liste nach steigender Häufigkeit (Funktion *occurrences*).
<https://tutorcs.com>
 Assignment Project Exam Help
 WeChat: cstutores
- Definiere dazu *occur*-Records $\langle\langle i, n \rangle\rangle$:
 Ding *i* (*item*) kommt mit Häufigkeit *n* (*freq*) vor.

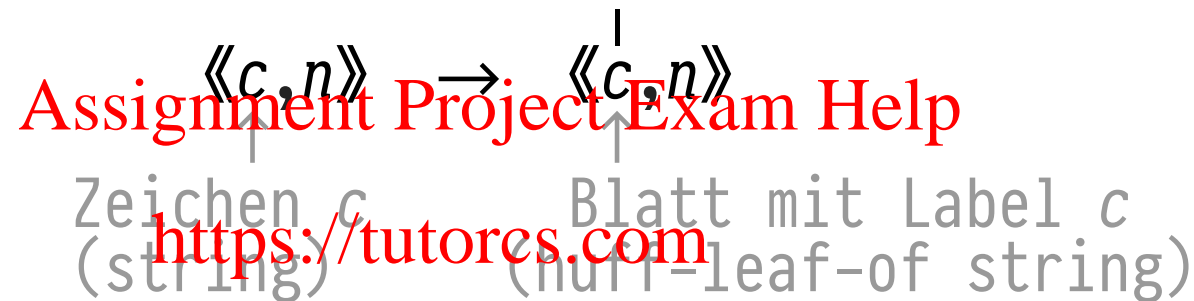
Beispiel:

```
(occurrences "erdbeermarmelade")
  ↳ (list ⟨⟨"l",1⟩⟩ ⟨⟨"b",1⟩⟩ ⟨⟨"d",2⟩⟩ ⟨⟨"a",2⟩⟩
        ⟨⟨"m",2⟩⟩ ⟨⟨"r",3⟩⟩ ⟨⟨"e",5⟩⟩)
```

API: Huffman-Tree für Text *txt* konstruieren (Schritt 2.)

2. Baue den Huffman-Tree von den Blättern her auf.

Konstruiere Liste *hts* trivialer Huffman-Trees:



WeChat: cstutorcs

- Bewahre jetzt die folgende Eigenschaft (**Invariante ①**): Die beiden Huffman-Trees, die die seltensten Zeichen in *txt* repräsentieren, stehen am Anfang der Liste *hts*:

$$(\text{list } \langle\langle c_1, n_1 \rangle\rangle \langle\langle c_2, n_2 \rangle\rangle \dots)$$

API: Huffman-Tree für Text *txt* konstruieren (Schritte 3.+4.)

3. **Wiederhole** bis Liste *hts* nur (noch) ein Element trägt:

- Fasse die zwei ersten *occur*-Records in *hts* zusammen:

$$\text{merge}(\langle\langle\Delta_1, n_1\rangle\rangle, \langle\langle\Delta_2, n_2\rangle\rangle) = \langle\langle \begin{array}{c} | \\ \Delta_1 \quad \Delta_2 \end{array}, n_1+n_2 \rangle\rangle$$

- **Bewahre Invariante** ①: Sortiere den neuen *occur*-Record bzgl. n_1+n_2 in *hts* ein.

4. *hts* = (list $\langle\langle\Delta, n\rangle\rangle$).² Δ ist der gesuchte Huffman-Tree für den gegebenen (Referenz-)Text *txt*. *Done*.

² Überlege, warum jetzt $n = (\text{string-length } \textit{txt})$ gilt.

7 | Neue Kontrollstrukturen durch H.O.F (Typ ⓘ)

Racket lässt sich mit Einsatz von H.O.F leicht um **neue Kontrollstrukturen** erweitern. Ein Beispiel ist `until`:

- Iteriere `f` auf `x`, bis Endebedingung `done?` erfüllt ist:

Assignment Project Exam Help

```
(: until ((%a -> boolean) (%a -> %a) %a -> %a)))
(define until https://tutorcs.com
  (lambda (done? f x)
    (if (done? x) WeChat: cstutorcs
        (until done? f (f x)))))
```

- Echte Iteration: `until` ist endrekursiv und der erzeugte Reduktionsprozess ist damit tatsächlich **iterativ**.

8 | API: Huffman-Tree für Text *txt* konstruieren (Racket)

```
; Generiere optimalen Huffman-Tree für String s
(: huffman-code (string -> (huff-tree-of string)))
```

```
(define huffman-code
```

```
  (lambda (s)
```

```
    (match (until singleton? merge
```

Schritt 3.

```
      (huffman-leaves
```

2.

Assignment Project Exam Help

```
      (occurrences s)))
```

1.

```
    ((list (make-occur ht _) ht))))
```

4.

<https://tutorcs.com>

- Huffman-Code (Referenz: Scroll vor *STAR WARS* Episode IV):

WeChat: cstutorcs

