

1. Zug WiSe2022



# Huffman-Kodierung

# Chronologie

- Idee
- Konstruktion
- Kodierung und Dekodierung

# Idee

# Idee

- Buchstabe „e“ kommt häufiger vor als „q“
- Häufig vorkommende Zeichen kurz codieren
- Ähnlich beim Morsecode
  - e → „.“ , q → „---.-“

# Idee

- Problematik:
  - Unterschiedliche Codewortlängen
    - keine eindeutige Dekodierung
  - Bsp.: „ ..-....- “ → usa , idea
- Gewisse Codewörter Anfangswörter von anderen Codewörtern
- o.a. e (.), i (..), u (..-), oder f (..-.)

# Idee

- Grundsatz: Fano-Bedingung
- Ziel:
  - systematische Konstruktion eines Codes
    - erfüllen der Fano-Bedingung
    - Kodierung mit möglichst wenigen Bits
- Anwendung:
  - Kompression von Texten
    - Fax Übertragung
    - Bilddaten-Kompressionsverfahren  
JPEG

# Idee

- Kurze mittlere Länge eines Codeworts

$$\bar{m} = \sum_{i=1}^n p(x_i) \cdot m(x_i)$$

Formel mittlere Codewortlänge

$p(x_i)$  → Häufigkeit des entsprechenden Symbols

$m(x_i)$  → Länge des Codewortes  
des entsprechenden Symbols

# Konstruktion

Des Huffman-Codes

# Konstruktion

- Eingabe: Text in Klarschrift
- Ausgabe: Binärer Baum
  - mit einer Knotenmarkierung
  - und Kantenmarkierung

# Konstruktion

- Beispielzeichenkette: dieser satz ist falsch
- Binär:

01100100 01101001 01100101 01110011

01100101 01110010 00100000 01110011

01100001 01110100 01111010 00100000

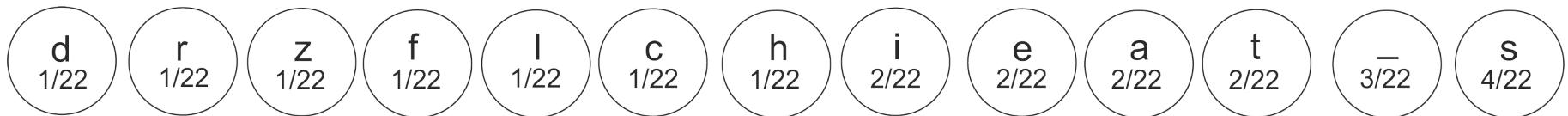
01101001 01110011 01110100 00100000

01100110 01100001 01101100 01110011

01100011 01101000 = 176 Bit

# Konstruktion

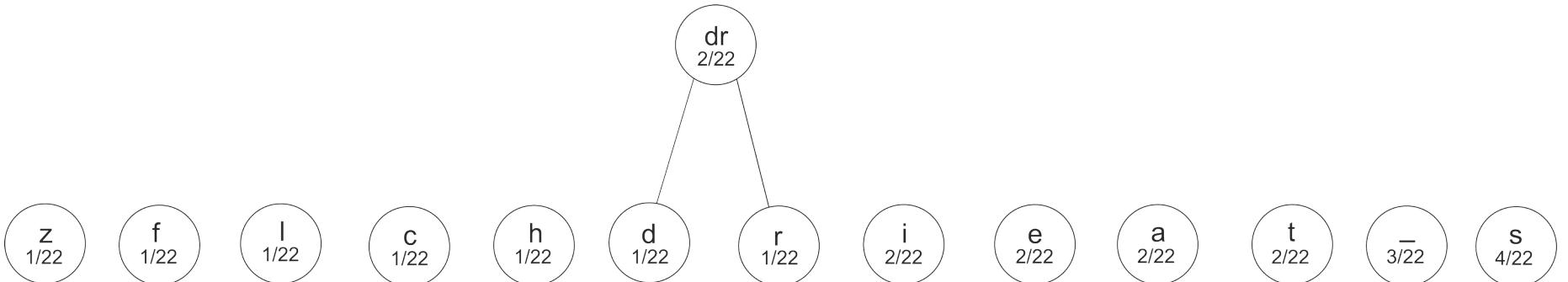
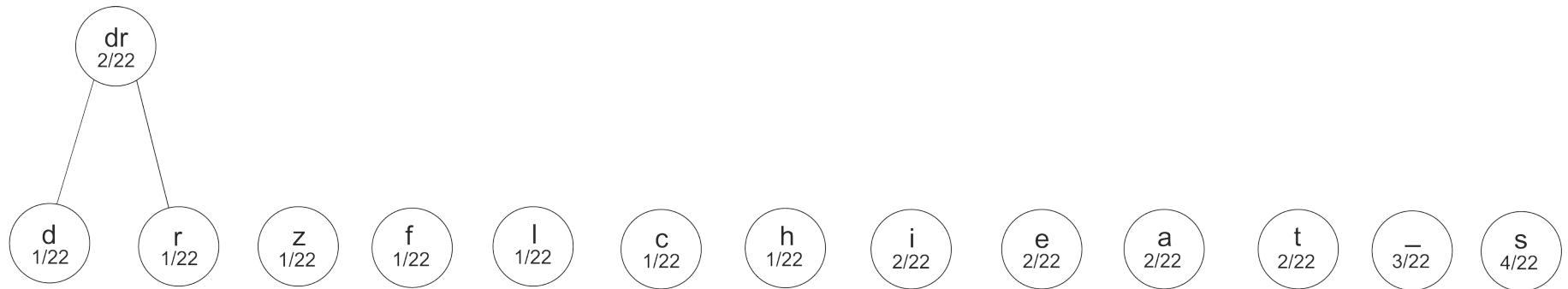
- Methode:
  1. Für jedes unterschiedliche Zeichen neuen Knoten erzeugen und mit Häufigkeit markieren
- d (1/22), i (2/22), e (2/22), s (4/22), r (1/22), a (2/22), t (2/22), z (1/22), f (1/22), l (1/22), c (1/22), h(1/22), \_ (3/22)



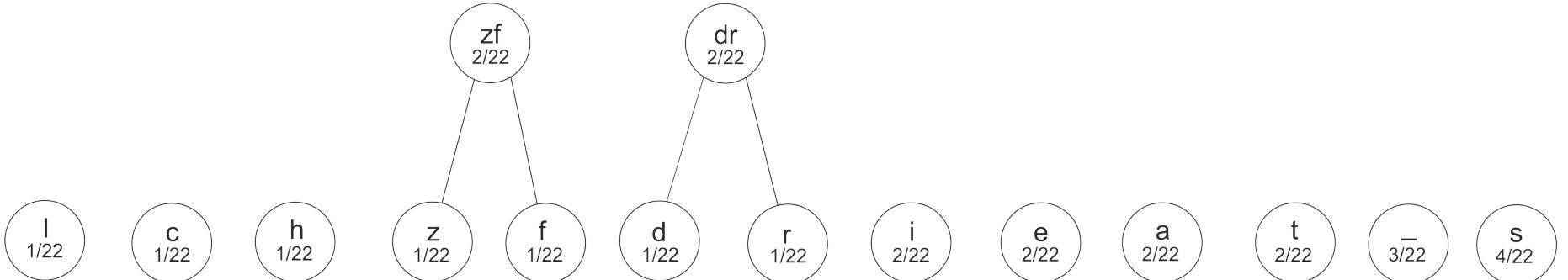
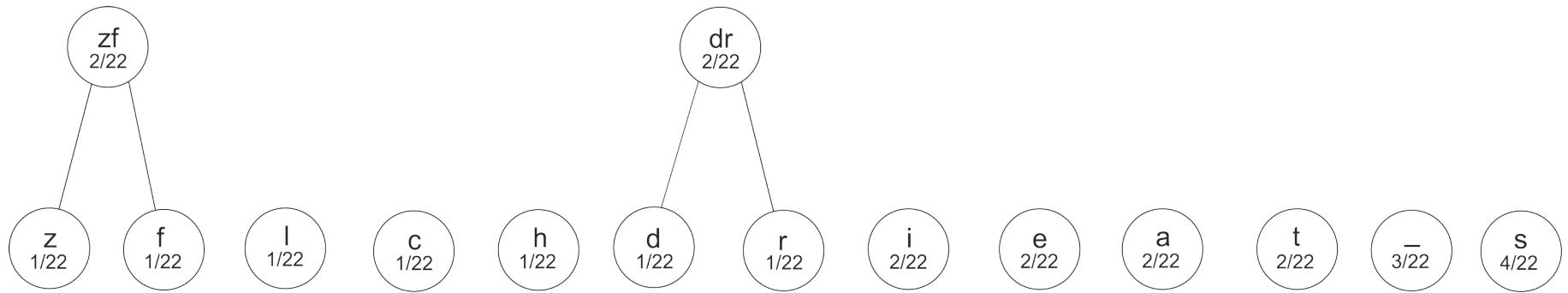
# Konstruktion

2. Solange mehr als einen Knoten ohne Kante:
  - a. Zwei Knoten mit minimaler Häufigkeit ohne Kanten suchen
  - b. Neuen Knoten erzeugen und mit den beiden o.g. Knoten verbinden
  - c. neue Häufigkeit markieren
  - d. neu sortieren

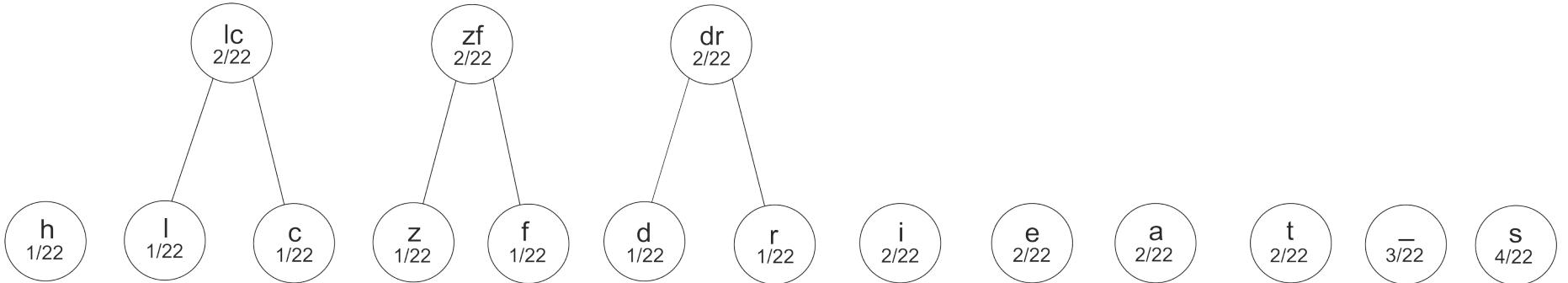
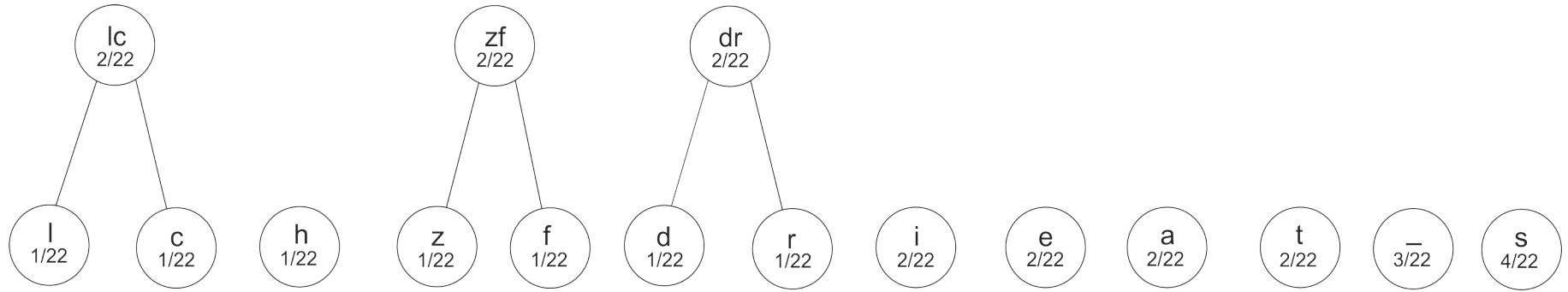
# Konstruktion



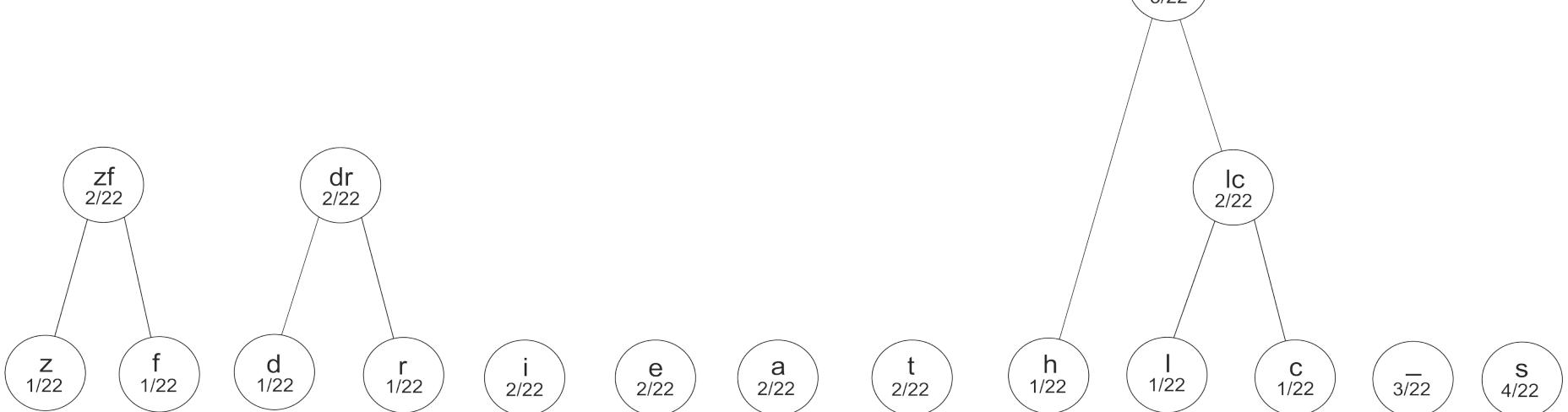
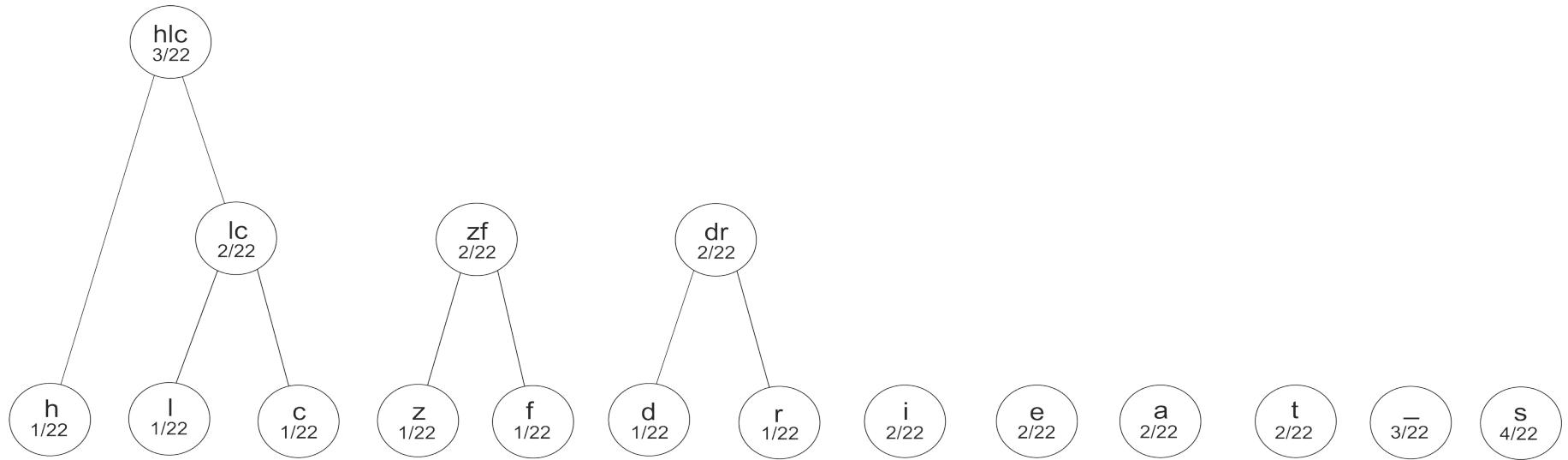
# Konstruktion



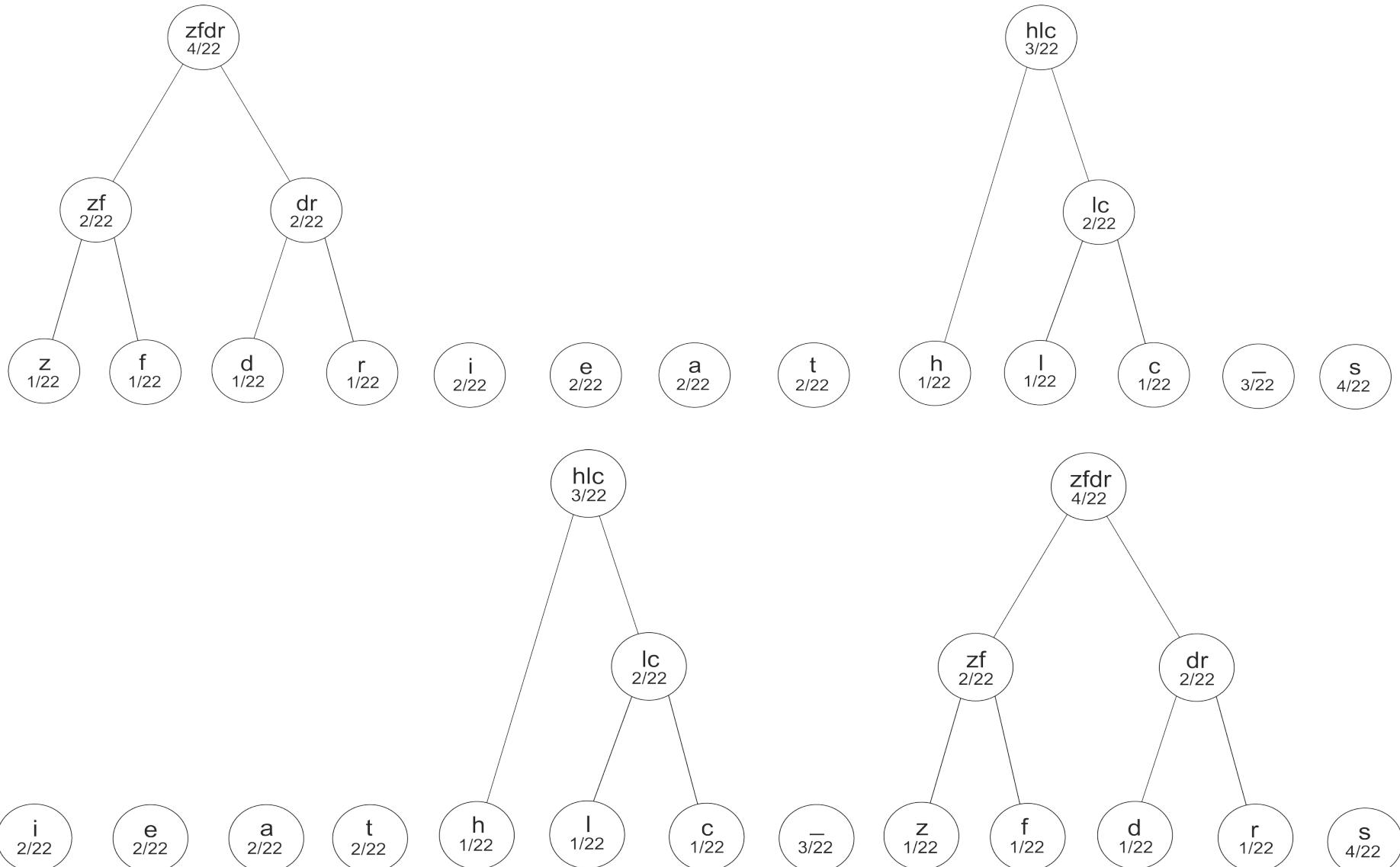
# Konstruktion



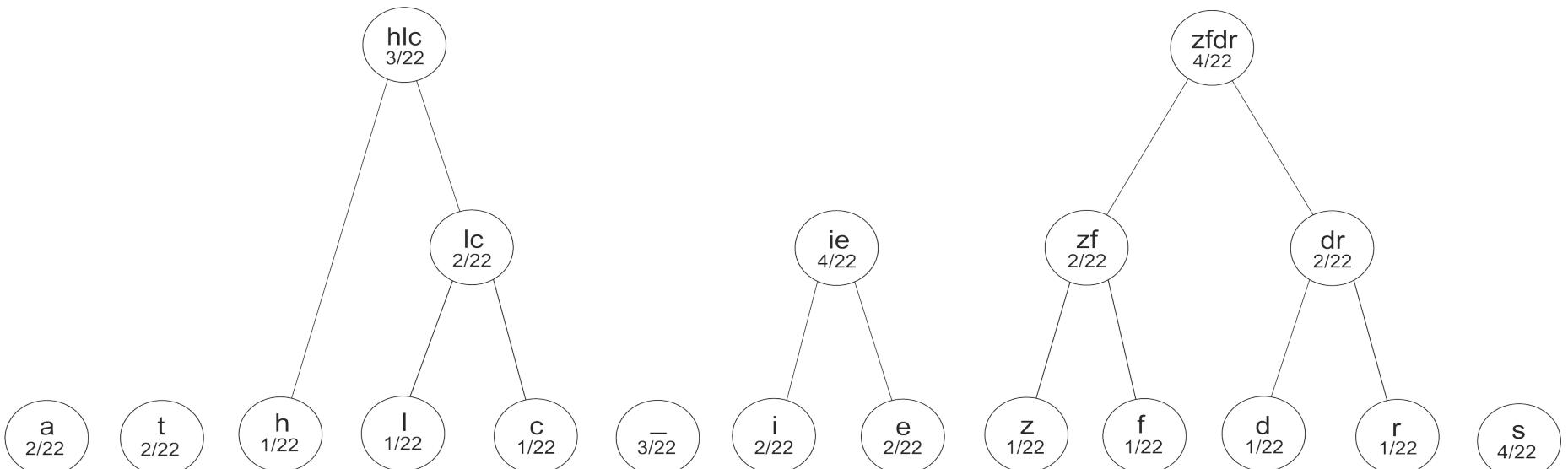
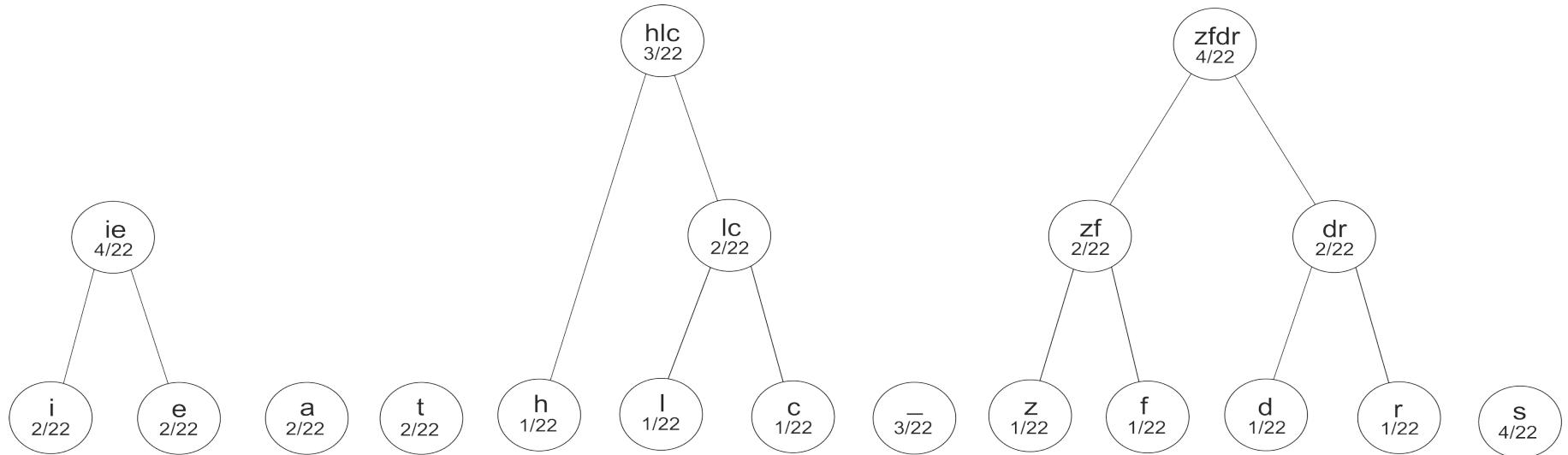
# Konstruktion



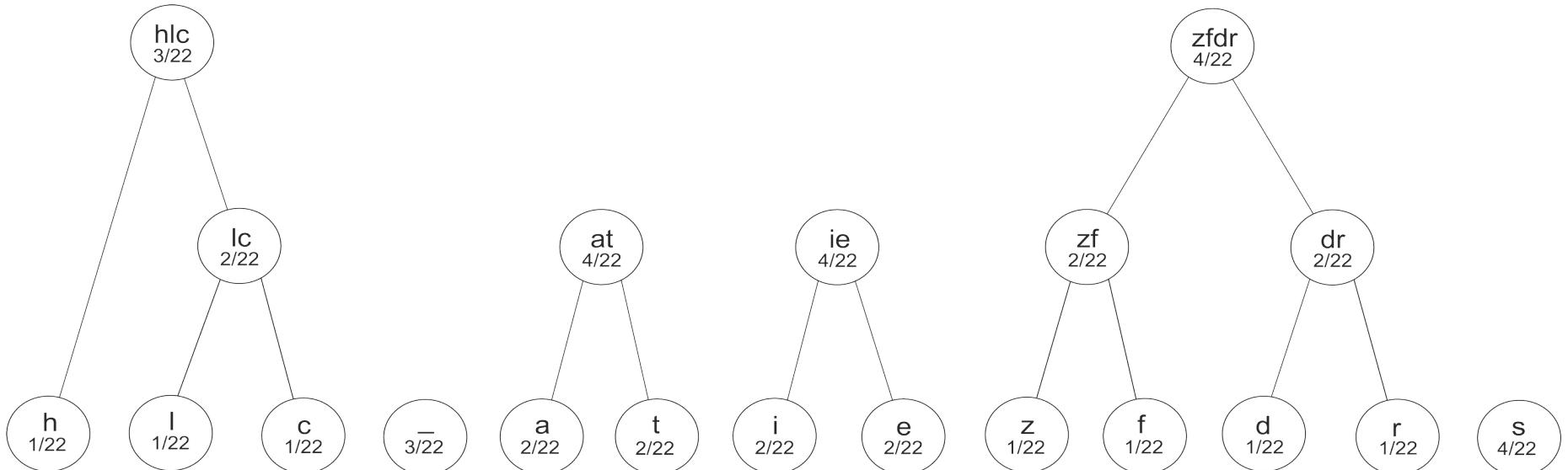
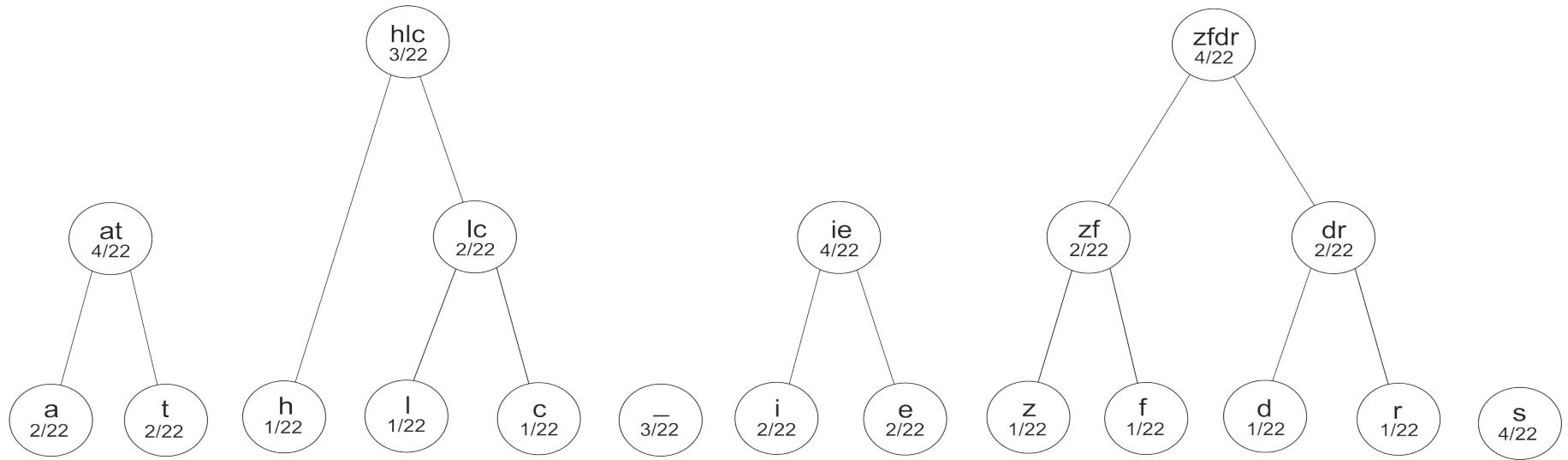
# Konstruktion



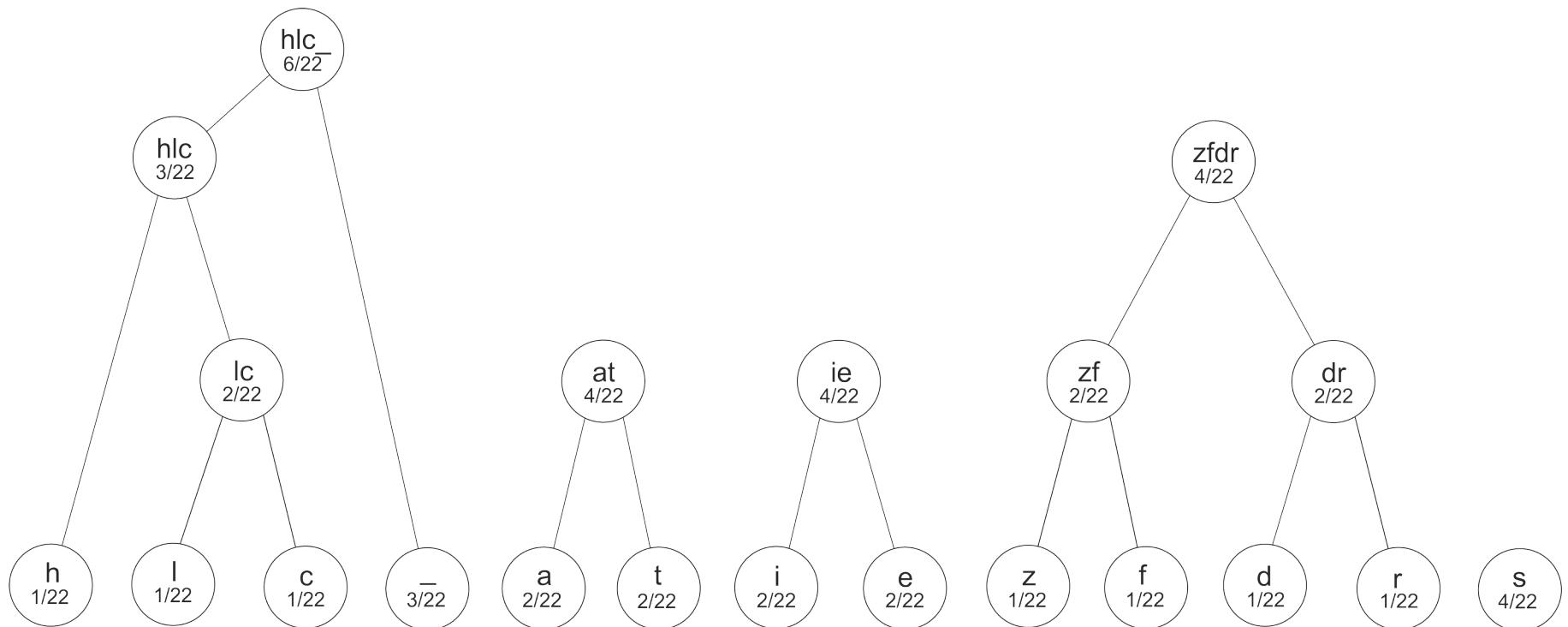
# Konstruktion



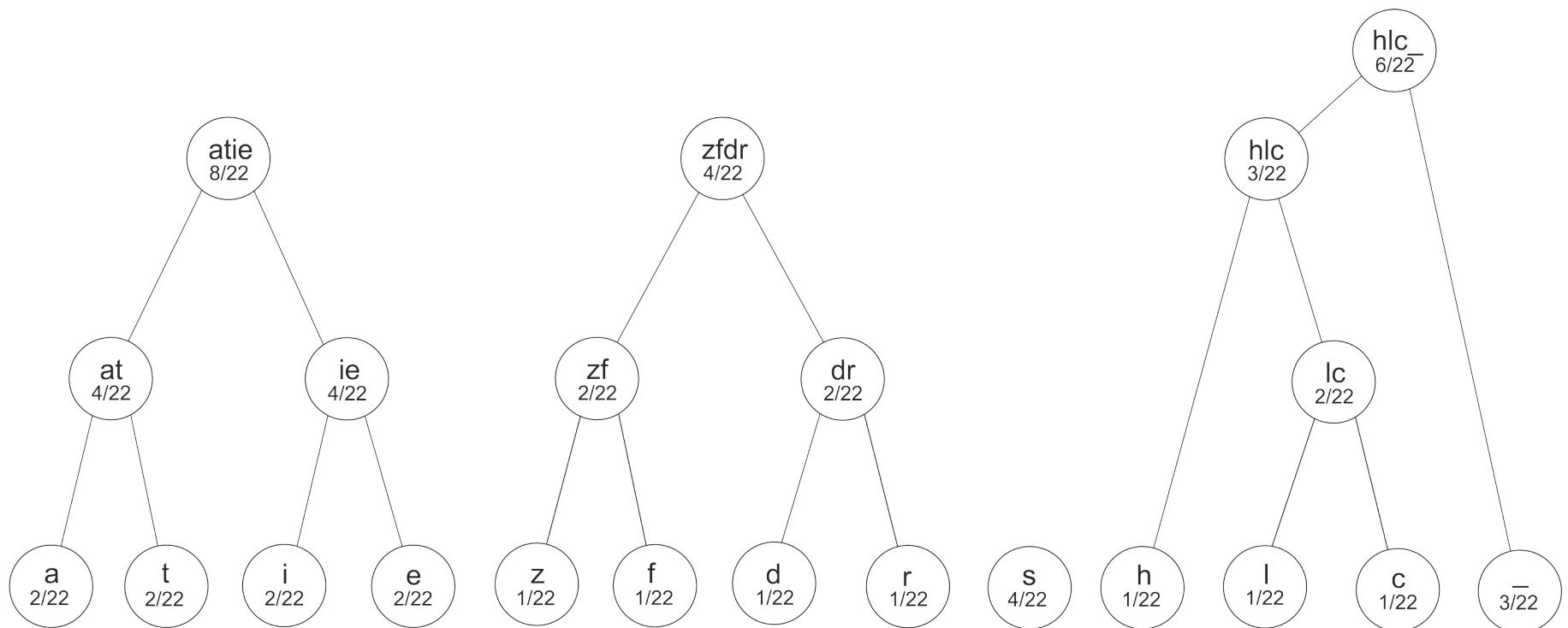
# Konstruktion



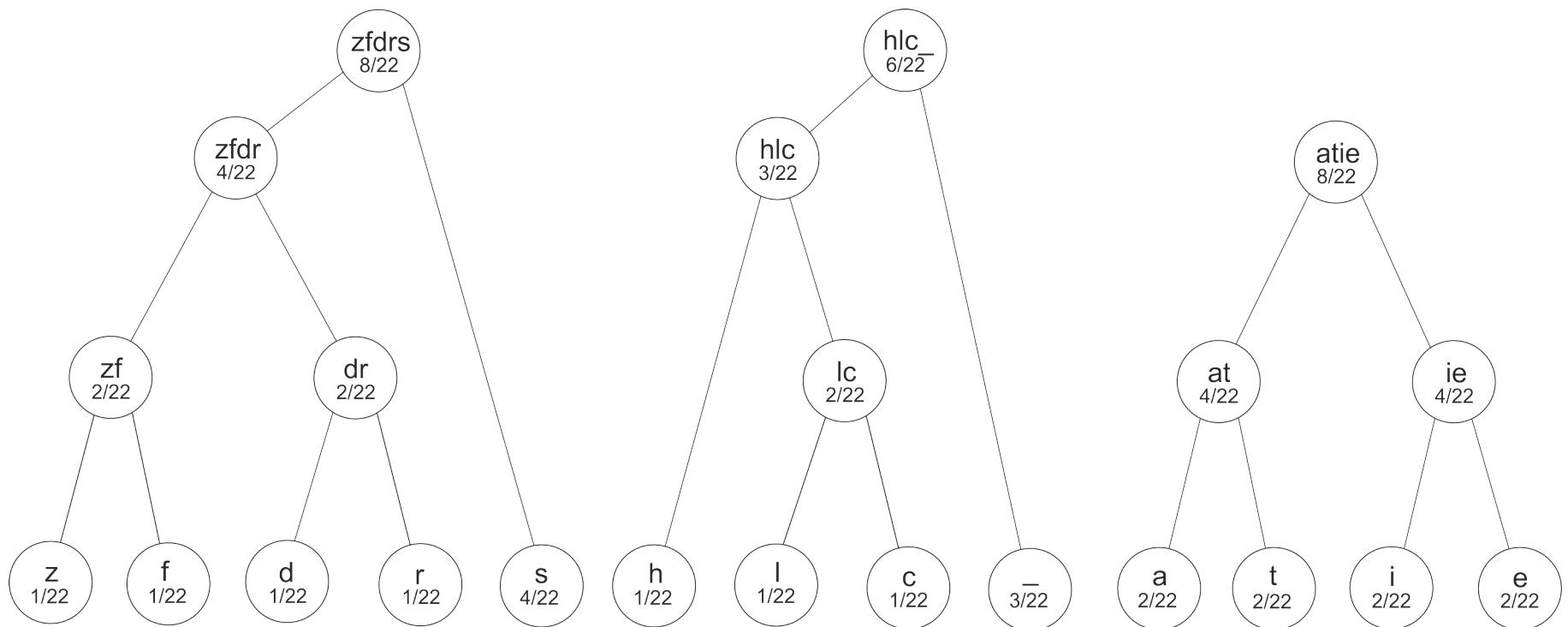
# Konstruktion



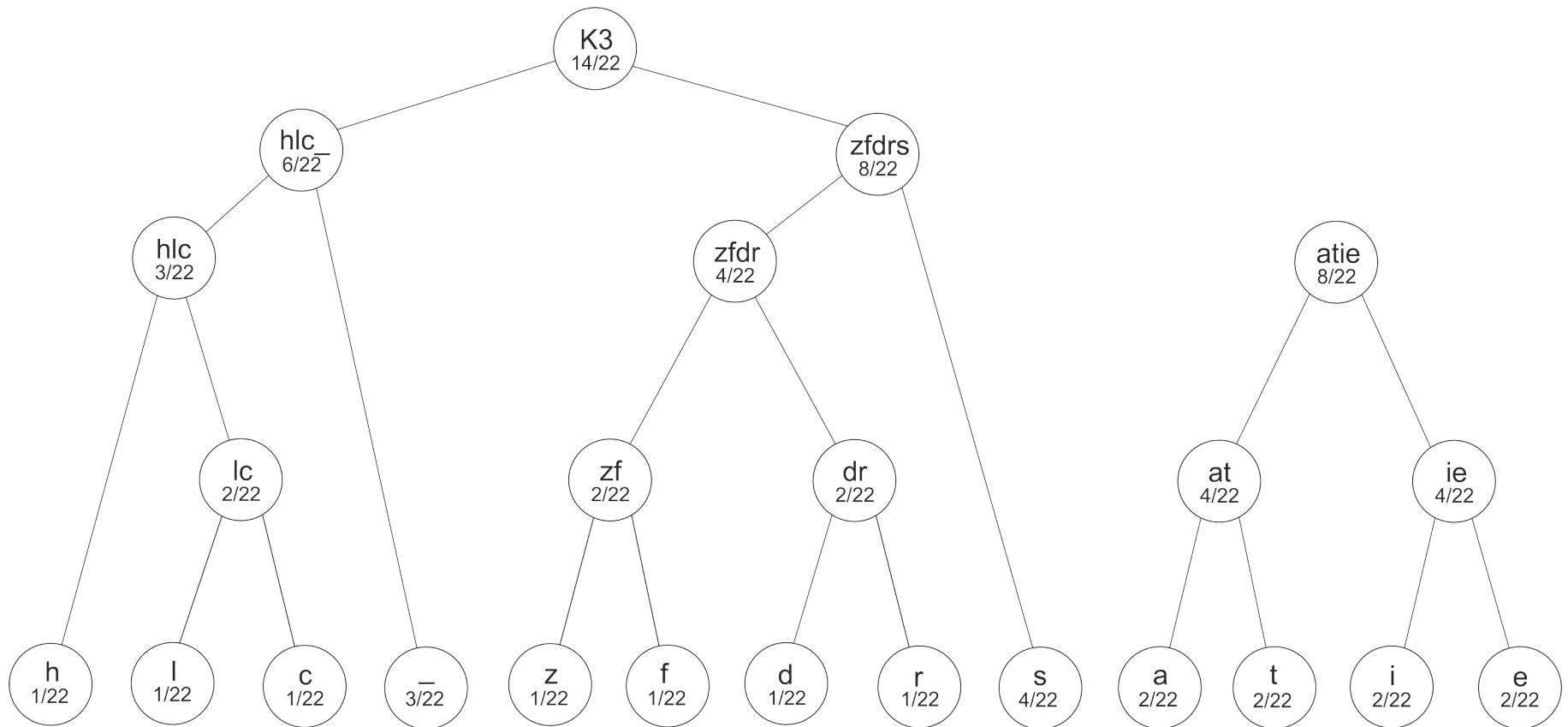
# Konstruktion



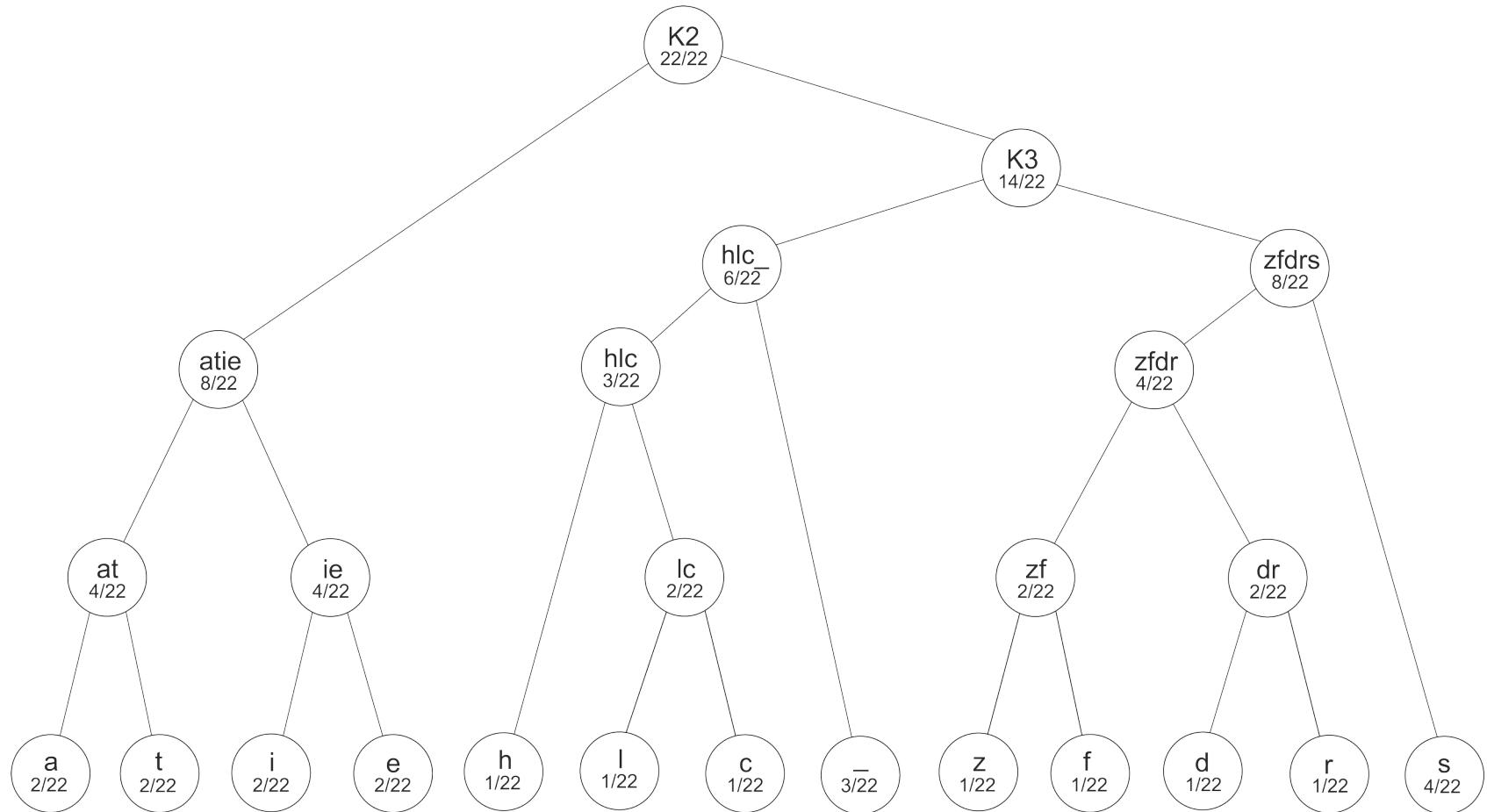
# Konstruktion



# Konstruktion

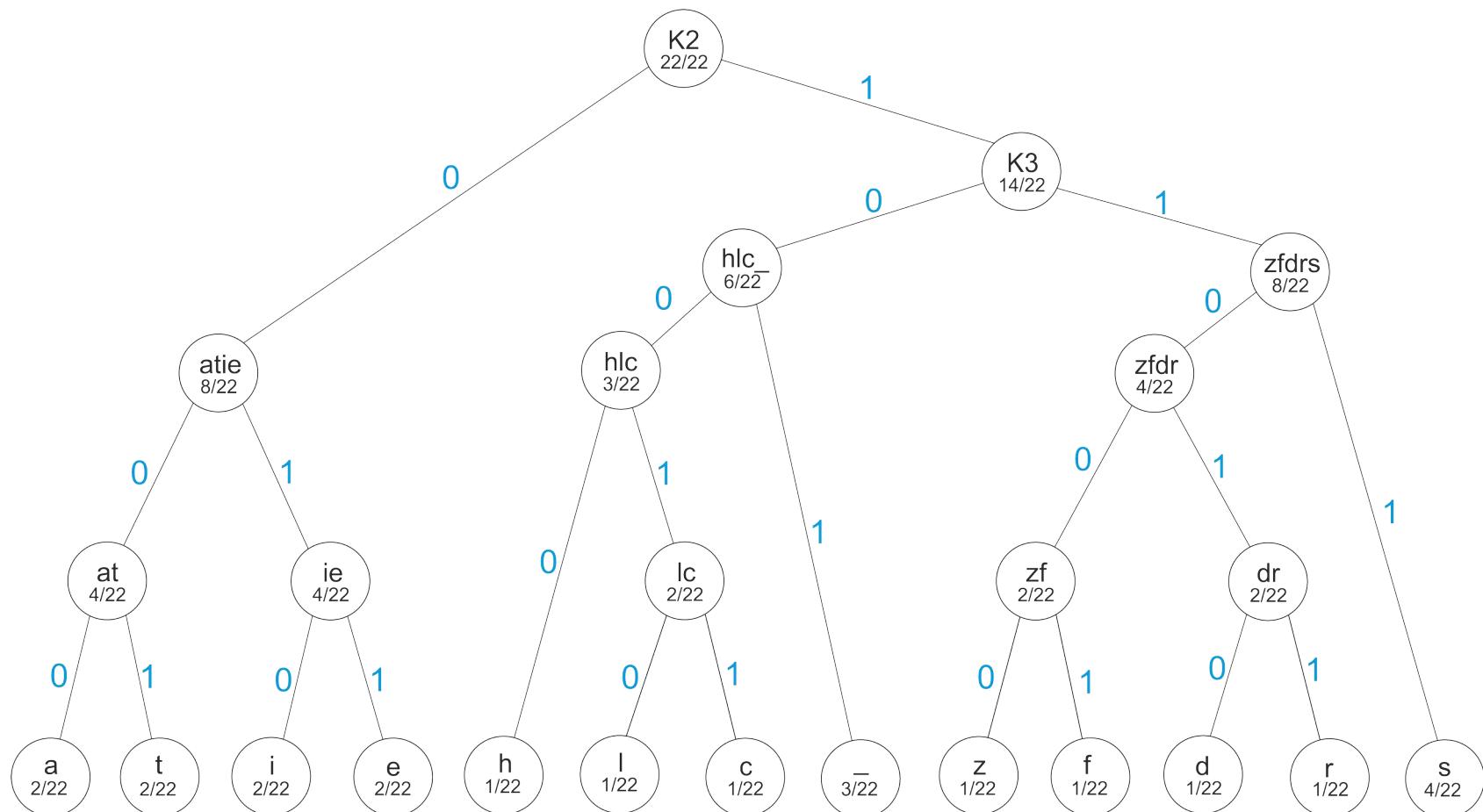


# Konstruktion



# Konstruktion

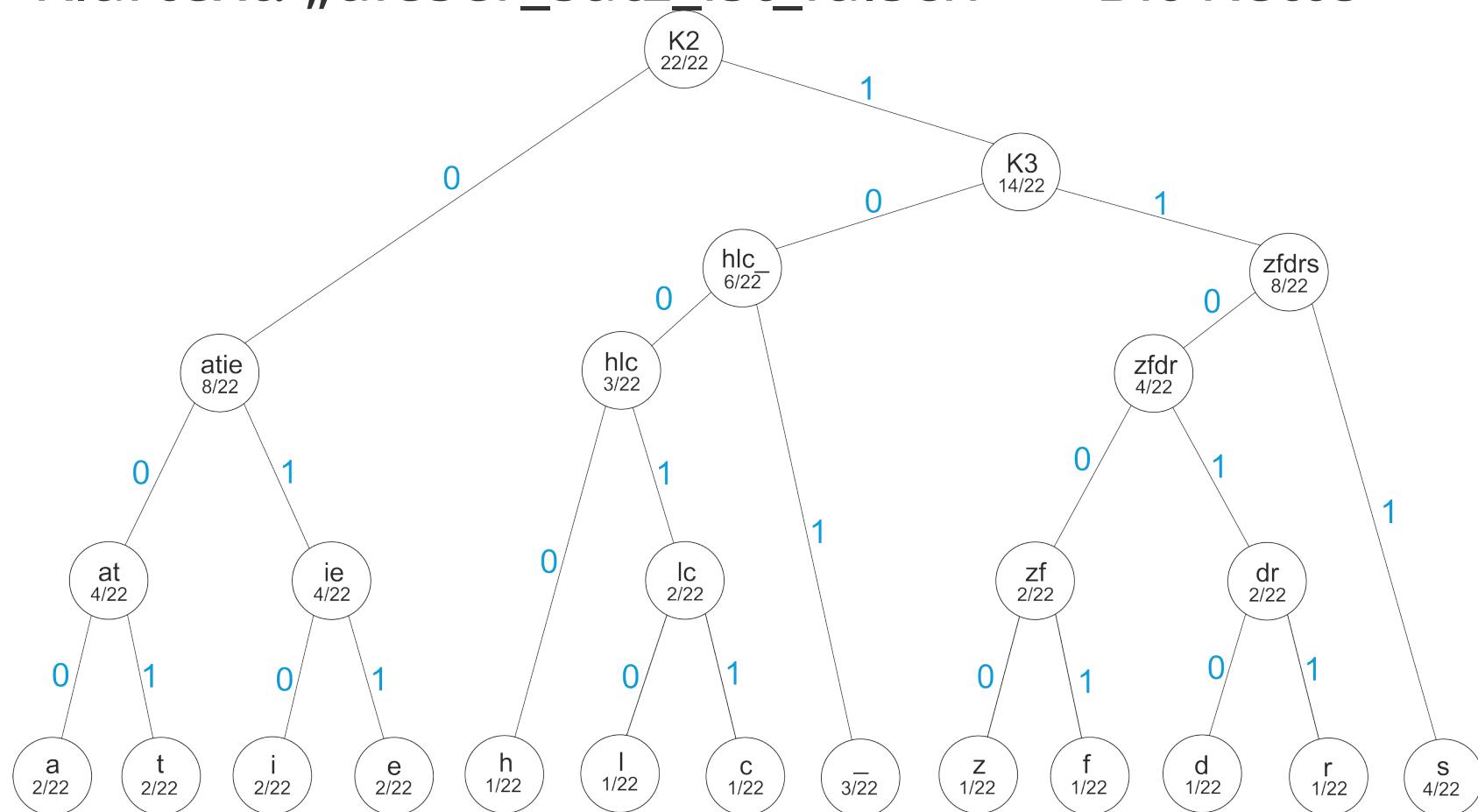
e. linke Kanten mit 0 markieren,  
rechte Kanten mit 1 markieren



# Kodierung und Dekodierung

# Kodierung und Dekodierung

- Klartext: „dieser\_satz\_ist\_falsch“ → Bit-Kette



# Kodierung und Dekodierung

- Klartext → Bit-Kette

11010 010 011 111 011 11011 101

111 000 001 11000 101 010 111 001 101

11001 000 10010 111 10011 1000 = 79 Bit

- 97 Bit werden eingespart
- 55,114% Bit-ersparnis vor einer 8-Bit Codierung

# Kodierung und Dekodierung

- Mittlere Codewortlänge=  $\frac{79}{22} \approx 3,59$  Bit
  - kürzer als 8-Bit Codierung
  - kürzer als 4-Bit Codierung
- Zum Dekodieren:
  - Codewörter Bit für Bit den Kanten folgen

# Quellen

## Informationsquellen:

[https://www.swisseduc.ch/informatik/daten/huffmann\\_kompression/docs/huffman.pdf](https://www.swisseduc.ch/informatik/daten/huffmann_kompression/docs/huffman.pdf)  
[https://www.lntwww.de/Informationstheorie/Entropiecodierung\\_nach\\_Huffman](https://www.lntwww.de/Informationstheorie/Entropiecodierung_nach_Huffman)  
<https://www.youtube.com/watch?v=qE4mEwHL62c>  
<https://einstein.informatik.uni-oldenburg.de/rechnernetze/huffmann.htm>

## Bildquellen:

[https://d1g9li960vagp7.cloudfront.net/wp-content/uploads/2018/10/Huffman\\_Codierung\\_Bild1-1024x576.jpg](https://d1g9li960vagp7.cloudfront.net/wp-content/uploads/2018/10/Huffman_Codierung_Bild1-1024x576.jpg)  
<https://informatik.mygymer.ch/ef2021/assets/img/huffman-german.aa0dab54.svg>