Basisinformationstechnologie II

Sommersemester 2023. 4: Algorithmen der Bildverarbeitung I

Grundlagen und Kompressionsverafahren. *Basierend auf Jan Wieners' Folien*

Institut für Digital Humanities, Historisch-Kulturwissenschaftliche Informationsverarbeitung | Prof. Dr. Øyvind Eide | Slavina Stoyanova

Programm

- Betriebssysteme
  - Verknüpfung Hard- und Software, Aufgaben von Betriebssystemen, Prozesse, Multitasking, Speicher- und Dateiverwaltung
- Rechnerkommunikation / Computernetzwerke
  - Themen u.a.: Hardwareaspekte, Übertragungstechnik, Verbindungsarten, Dienste, Protokolle, Ports, Schichtenmodelle: ISO/OSI vs. TCP / IP Modell, ...
- Text, Datenmodellierung, Metadaten(standards)
  - Themen u.a.: XML, PDF, DocX & Co., Metadatenstandards im BAM-Sektor, Semantic Web, RDF, CIDOC CRM
- Bild
  - Themen u.a.: Farbmischung, Bildbearbeitungs-/verarbeitungsalgorithmen, Kompression, ...
- Künstliche Intelligenz
  - Computer Vision
  - Machinelles Lernen, Q-Learning
  - KI in Computer- und Videogames: Game Engines, Wegfindung, Schwarmverhalten, Agenten, Multiagentensysteme, ...

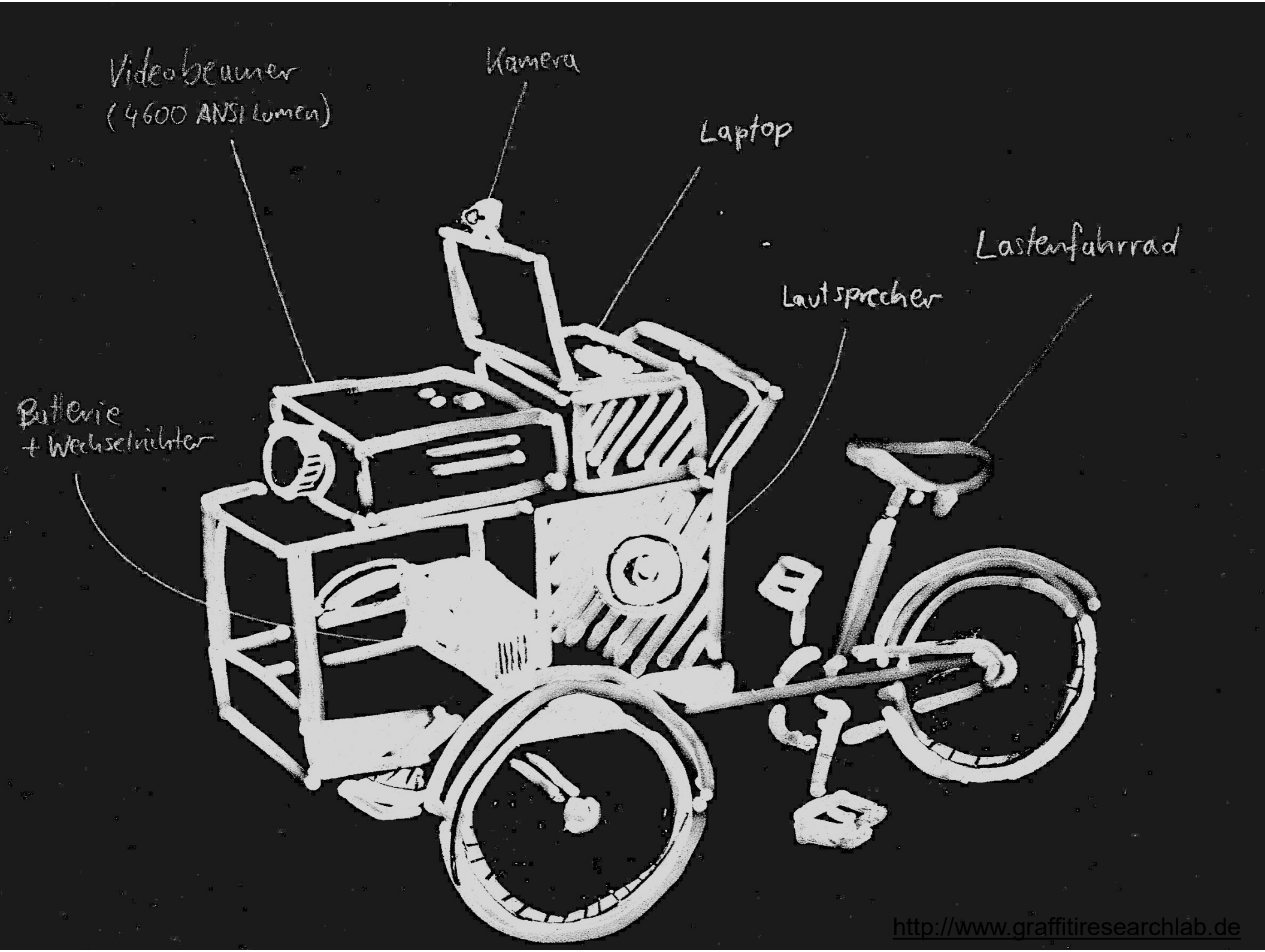
Themenüberblick

Grundbegriffe:

- Farbmischung: Additiv, Subtraktiv
- Raster- vs. Vektorgrafik
- Pixel
- Auflösung
- Farbtiefe

Kompressionsverfahren

- Nicht verlustbehaftet
  - Run Length Encoding (RLE)
  - Wörterbuch-Algorithmen, z.B. Lempel-Ziv-Welsh (LZW)
  - Huffman-Codierung
- (Verlustbehaftet)

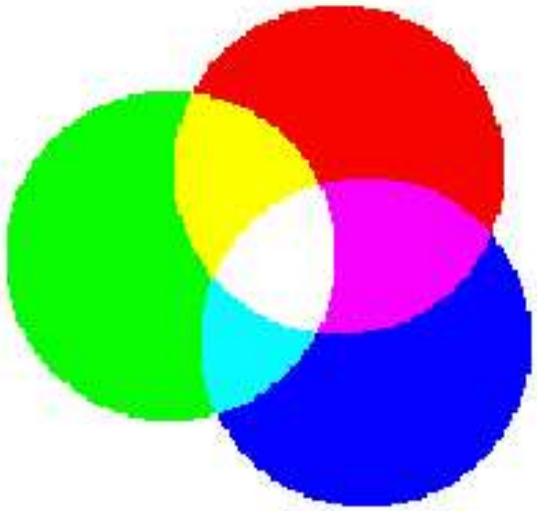


http://www.graffitiresearchlab.de


http://www.graffitiresearchlab.de

Additive Farbmischung

Primärfarbe			Grundfarbe
Blau	Grün	Orangerot	
⊗	–	–	Blau
–	⊗	–	Grün
–	–	⊗	Orangerot
⊗	⊗	–	Cyan
⊗	–	⊗	Magenta
–	⊗	⊗	Gelb
⊗	⊗	⊗	Weiß
–	–	–	Schwarz

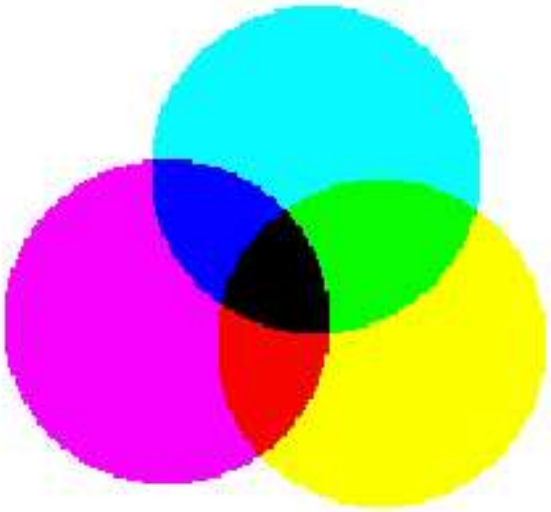


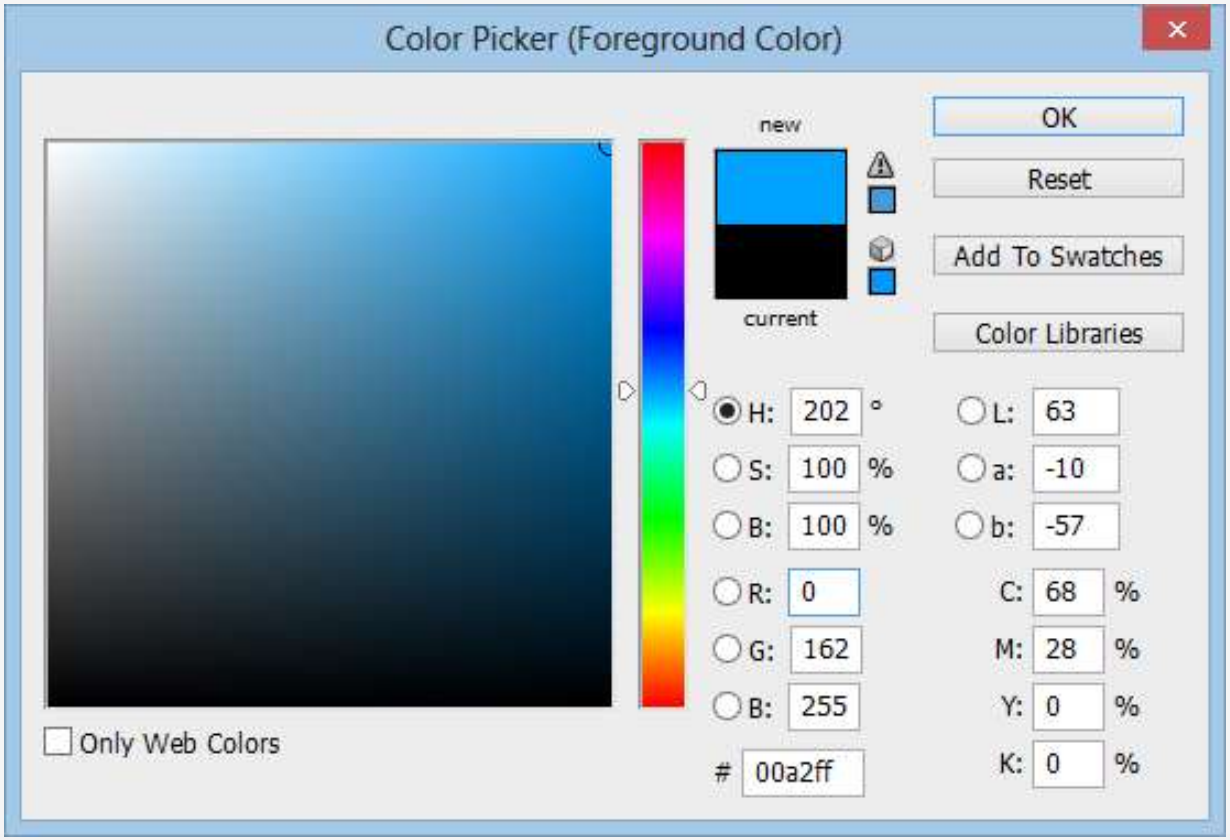

BK
M
C
PGBK
Y
Canon
526
BK
Black
CLI-526BK
Ink Tank
Made in Japan
PUSH
Canon
526
M
Magenta
CLI-526M
Ink Tank
Made in Japan
PUSH
Canon
526
C
Cyan
CLI-526C
Ink Tank
Made in Japan
PUSH
Canon
525
PGBK
Black
PGI-525PGBK
Ink Tank
Made in Japan
PUSH
Canon
526
Y
Yellow
CLI-526Y
Ink Tank
Made in Japan
PUSH

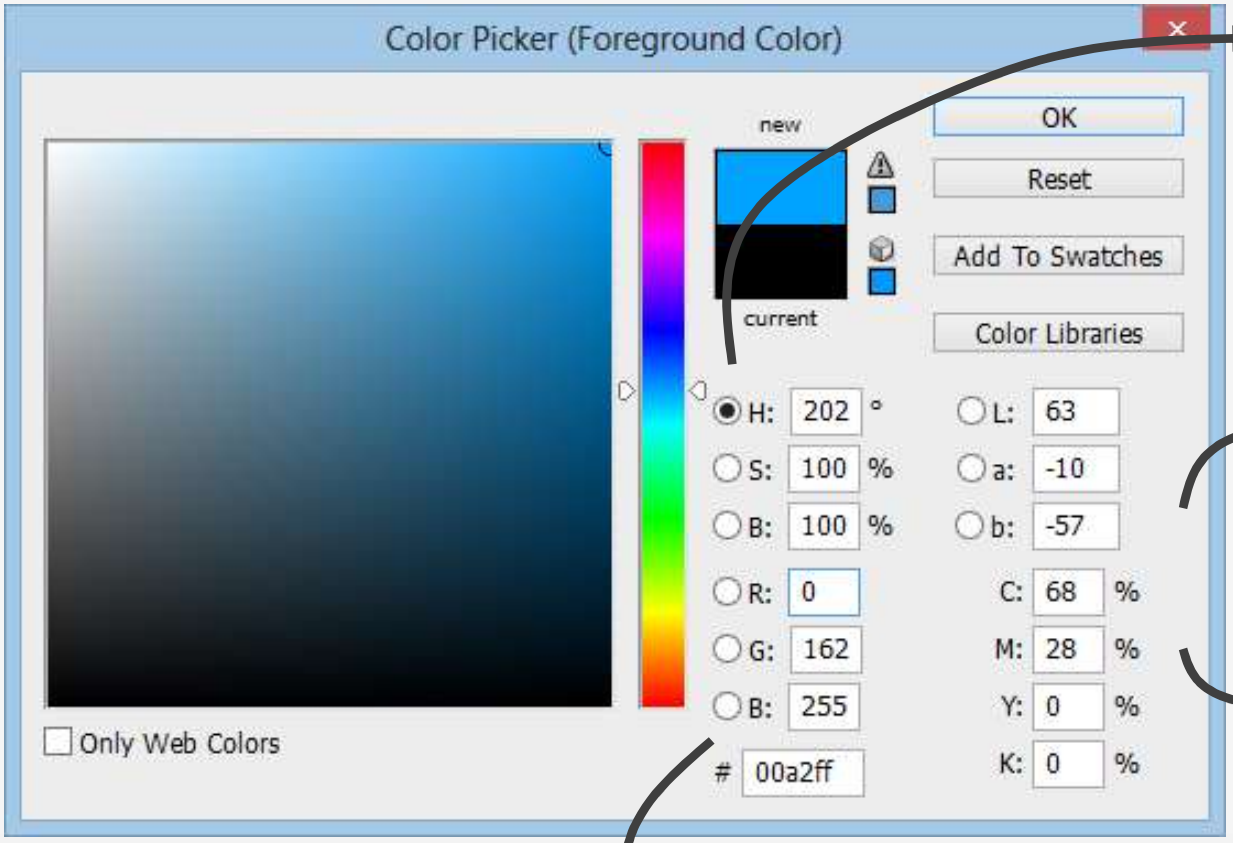
Subtraktive Farbmischung

Z.B.: Mischung von Farbpigmenten → Malen mit Deckfarbkasten

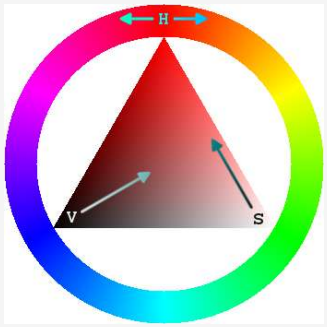
Primärfarbe			Grundfarbe
Cyan	Magenta	Gelb	
⊗	–	–	Cyan
–	⊗	–	Magenta
–	–	⊗	Gelb
⊗	⊗	–	Blau
⊗	–	⊗	Grün
–	⊗	⊗	Orangerot
⊗	⊗	⊗	Schwarz
–	–	–	Weiß




Color Picker (Foreground Color)
new
current
OK
Reset
Add To Swatches
Color Libraries
H: 202 °
S: 100 %
B: 100 %
R: 0
G: 162
B: 255
L: 63
a: -10
b: -57
C: 68 %
M: 28 %
Y: 0 %
K: 0 %
00a2ff
Only Web Colors



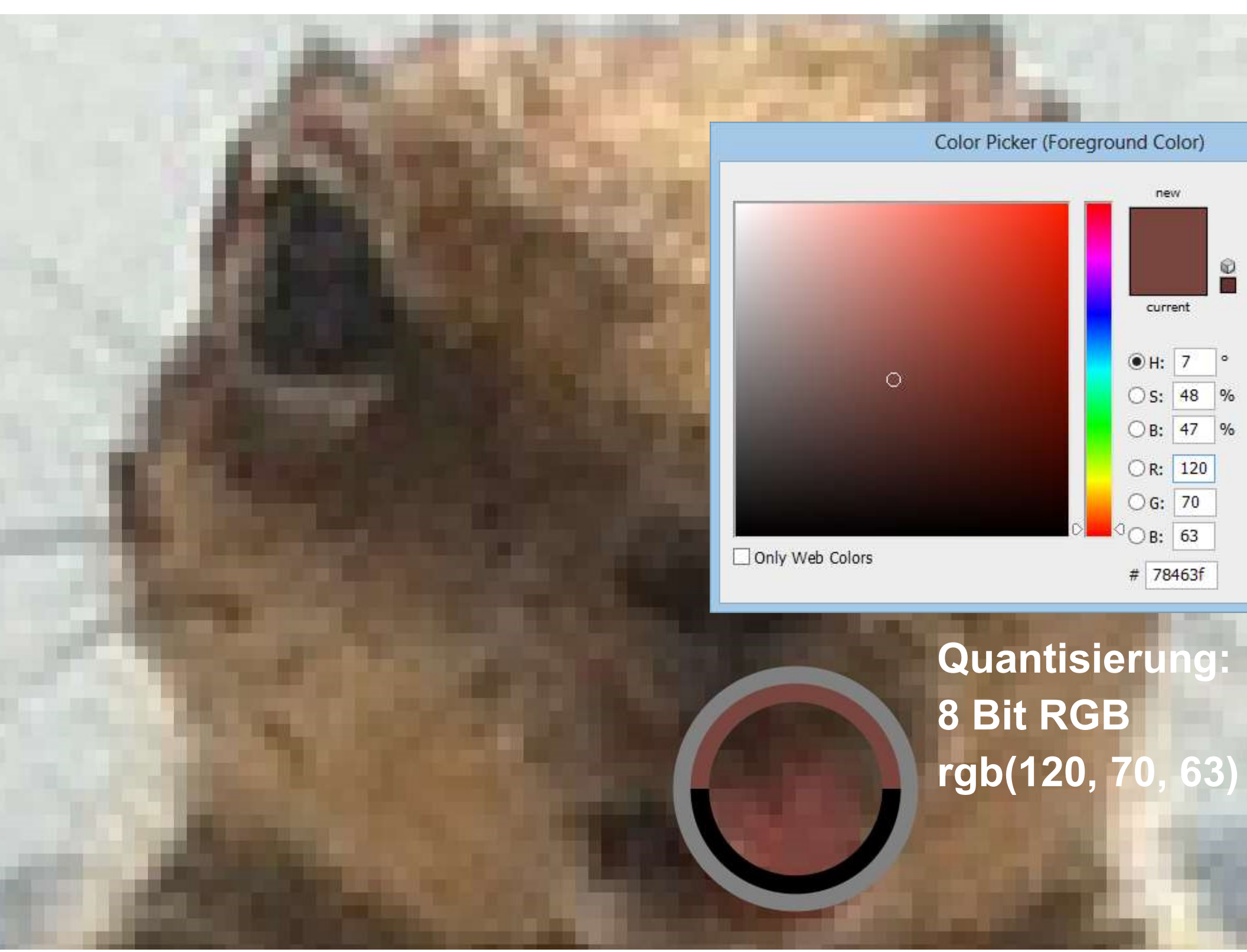
HSV: Farbwinkel, Sättigung, Hellwert bzw. absolute Helligkeit (B), Brightness



L*a*b*: Menschl. Wahrnehmung

RGB: Wenn jede der drei Primärfarben mit einer Auflösung von 256 Werten dargestellt werden kann, dann erhalten wir 256^3 = 16,7 Mio. verschiedene Farbtöne.

Raster- vs. Vektorgrafik

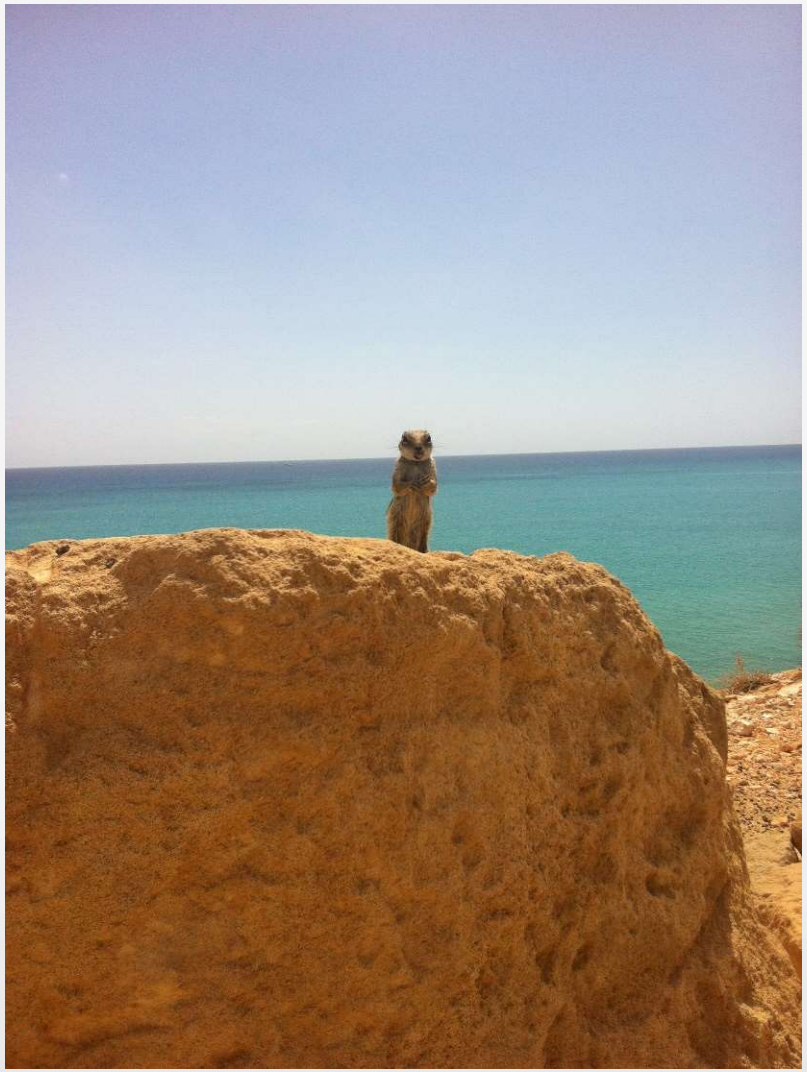

Color Picker (Foreground Color)
new
current
H: 7 °
S: 48 %
B: 47 %
R: 120
G: 70
B: 63
Only Web Colors
78463f
Quantisierung:
8 Bit RGB
rgb(120, 70, 63)

http://www.clker.com/clipart-cartoon-squirrel.html

Auflösung

Gesamtzahl der Bildpunkte eines Bildes, z.B. 2592px * 1936px = ~ 5 Mio. Pixel

2592 Pixel



1936 Pixel

Kompression


ucrit / 9GAG
Just one
MicroSD card
stores more than
the rest combined...
25 years
of storage

Format	Kompression	Unterstützte Bittiefen	Vor- und Nachteile	Größe eines 8-Bit Graustufenbildes
BMP	RLE	1, 4, 8 und 24	+ leichte Erzeugung - Größe	256 KB (lossless)
GIF (Graphics Interchange Format)	LZW	1 bis 8 (aus einer 24 Bit Tabelle)	+ Steuerblock + Schachtelung + Interlaced / Transparent - Lizenzpflicht seit 1994	240 KB (lossless)
PNG (Portabel Network Graphics Format)	LZW-Variante	8, 16, 24, 48	+ Lizenzfrei + auch 24 Bit + Interlaced / Transparent	219 KB (lossless)
JPEG	DCT und Huffman	24	+ für Fotografien + Größe + Progressive Format - schlecht für "digitale" Bilder	21 KB (lossy)
TIF (Tagged Image Format)	LZW-Variante	1 bis 48	+ Plattormunabhängig + viele Metainformationen ablegbar - viele Unterformate	264 KB (lossless)

Ansätze zur Datenkompression

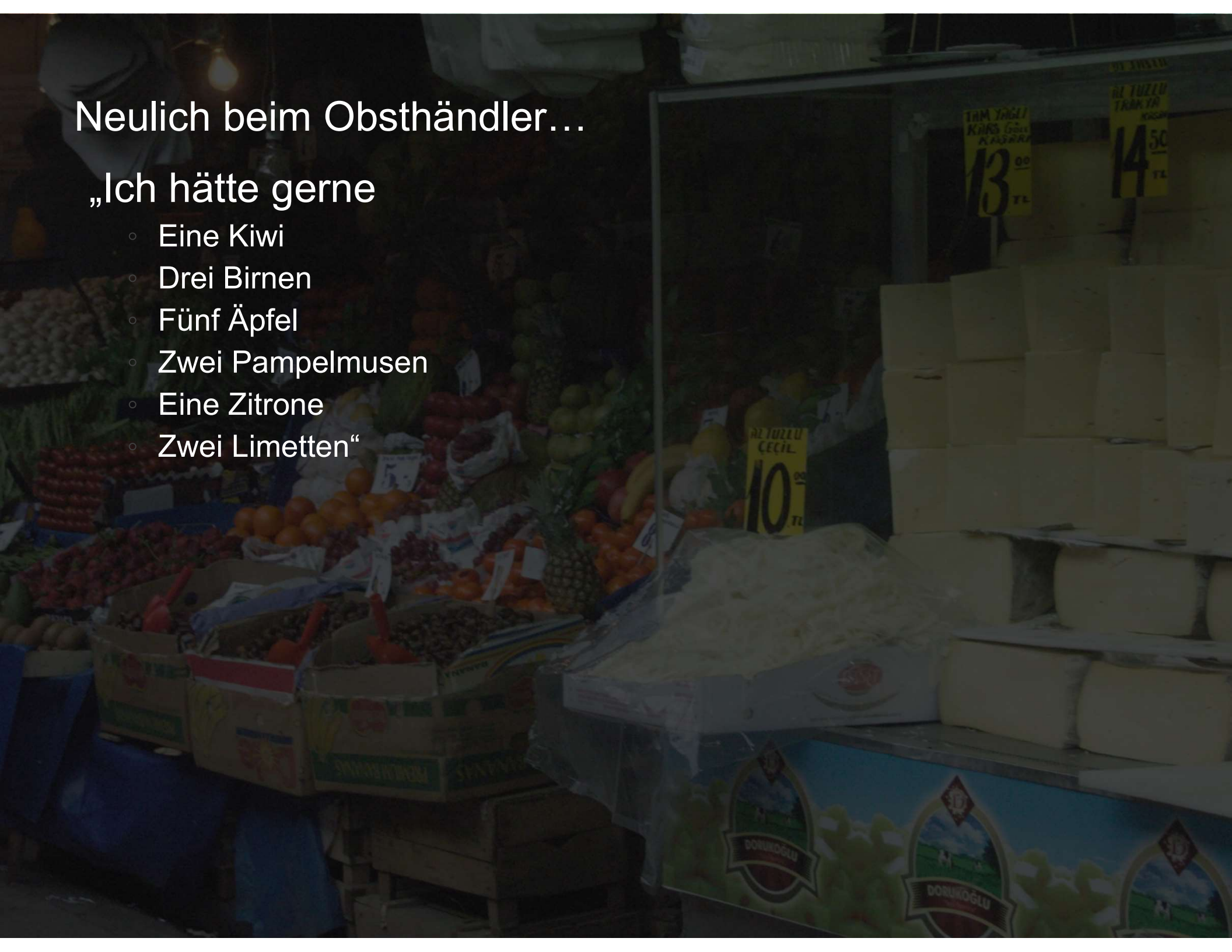
- RLE – Run Length Encoding (Lauflängenkodierung)
- Wörterbuchbasierte Kompressionsmethoden
  - Lempel-Ziv-Welch (LZW)
- Statistische Kompressionsmethoden
  - Huffman-Algorithmus

Lauflängenkodierung

Neulich beim Obsthändler…

„Ich hätte gerne

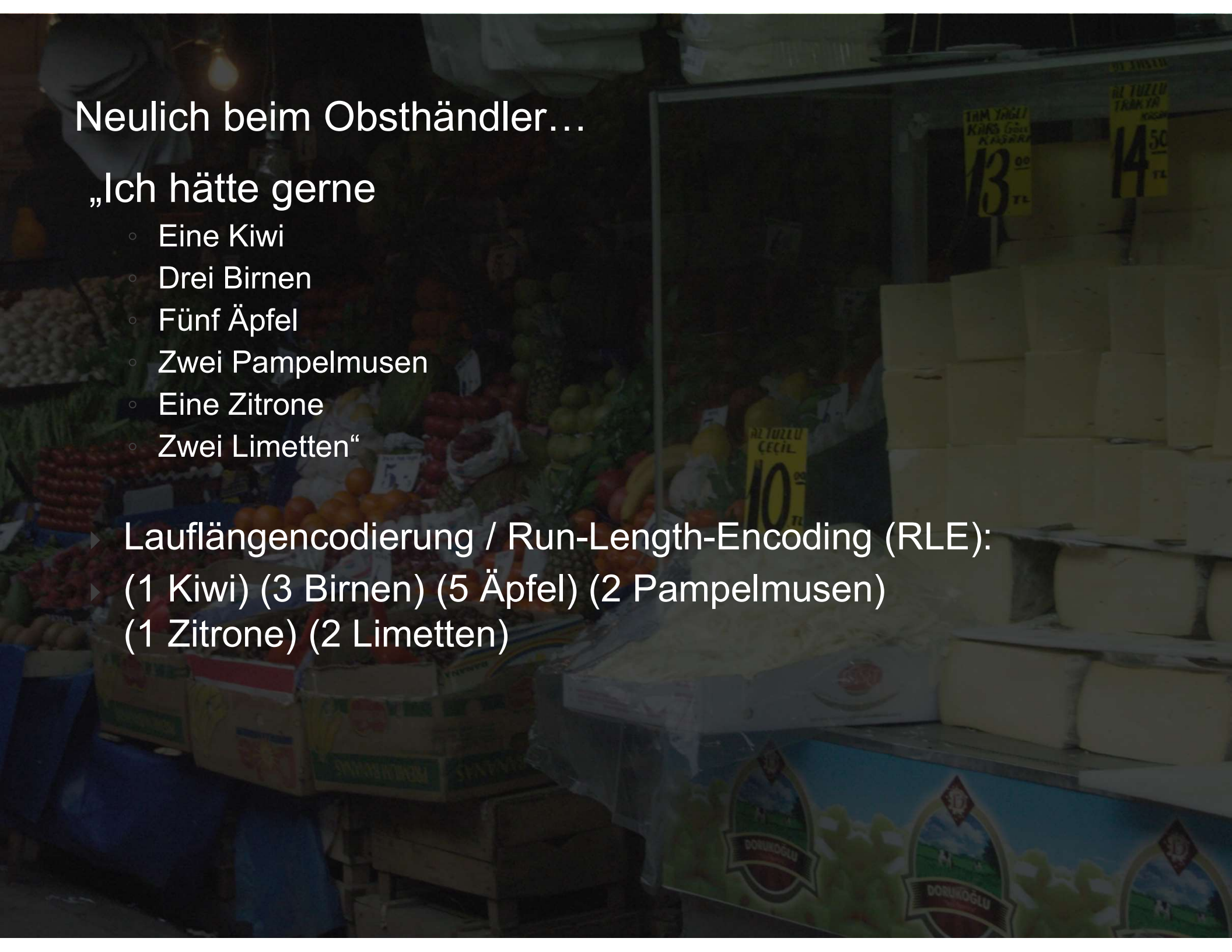
- Eine Kiwi
- Eine Birne
- Eine Birne
- Eine Birne
- Einen Apfel
- Einen Apfel
- Einen Apfel
- Einen Apfel
- Einen Apfel
- Eine Pampelmuse
- Eine Pampelmuse
- Eine Zitrone
- Eine Limette
- Eine Limette“



Neulich beim Obsthändler…

„Ich hätte gerne

- Eine Kiwi
- Drei Birnen
- Fünf Äpfel
- Zwei Pampelmusen
- Eine Zitrone
- Zwei Limetten“

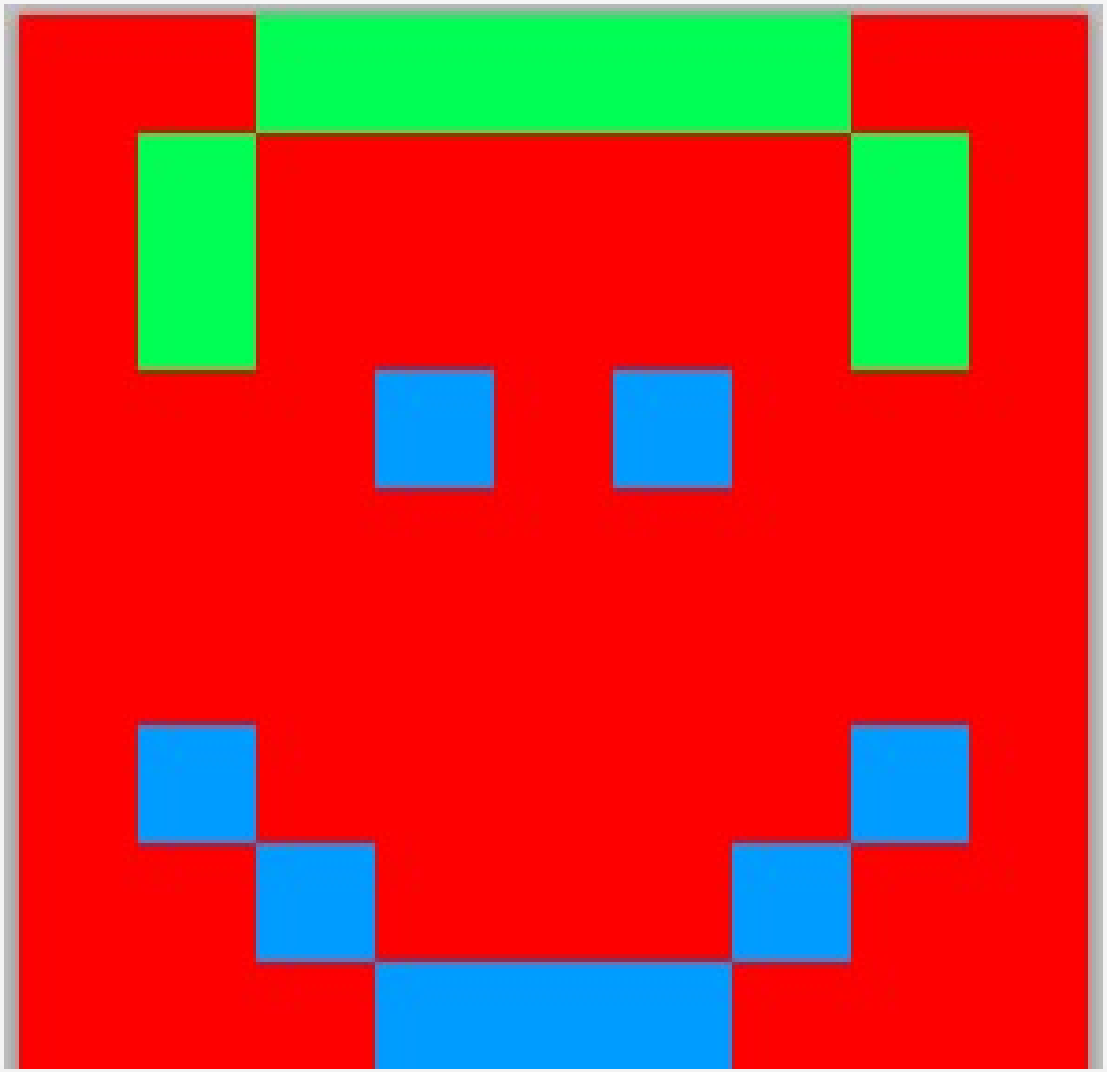


Neulich beim Obsthändler…

„Ich hätte gerne

- Eine Kiwi
- Drei Birnen
- Fünf Äpfel
- Zwei Pampelmusen
- Eine Zitrone
- Zwei Limetten“

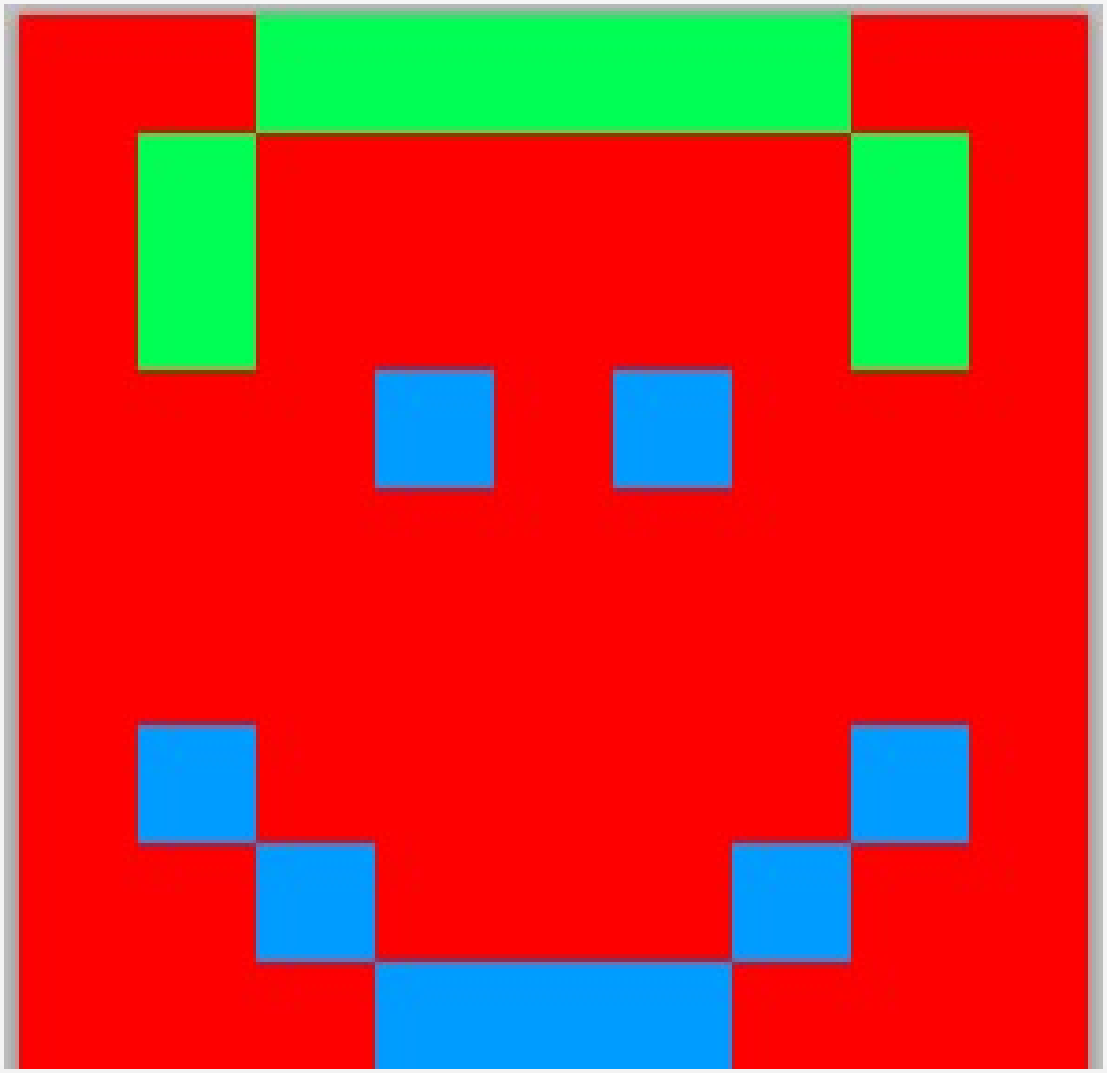
- Lauflängencodierung / Run-Length-Encoding (RLE):
- (1 Kiwi) (3 Birnen) (5 Äpfel) (2 Pampelmusen) (1 Zitrone) (2 Limetten)



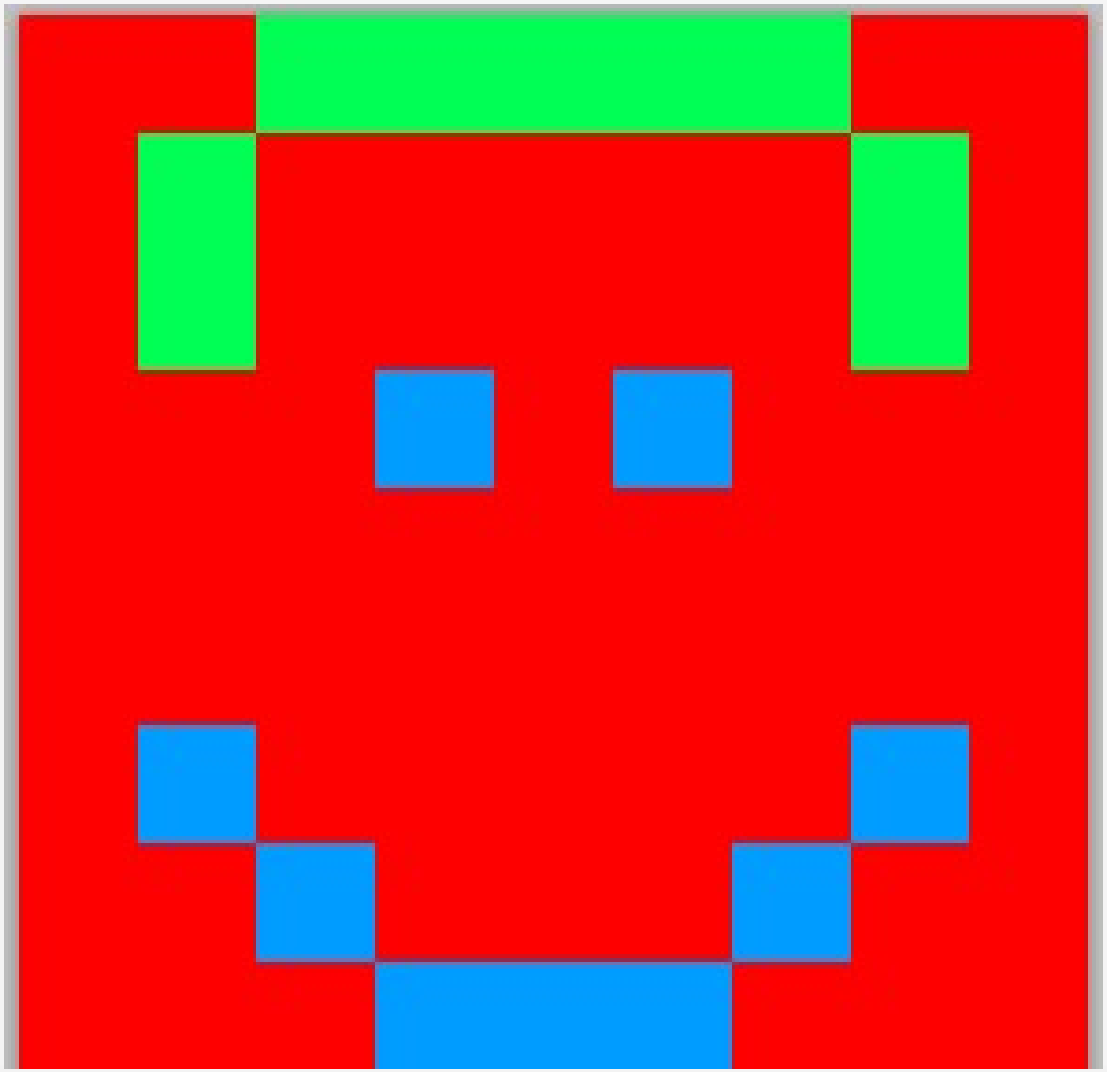
rot := rgb(255, 0, 0)

grün := rgb(0, 255, 0)

blau := rgb(0, 0, 255)



rot, rot, grün, grün, grün, grün, grün, rot, rot
rot, grün, rot, rot, rot, rot, rot, grün, rot,
rot, grün, rot, rot, rot, rot, rot, grün, rot,
rot, rot, rot, blau, rot, blau, rot, rot, rot,
rot, rot, rot, rot, rot, rot, rot, rot
[…]



Komprimiert mit RLE:
(2 rot) (5 grün) (2 rot)
(1 rot) (1 grün) (5 rot) (1 grün) (1 rot)
(1 rot) (1 grün) (5 rot) (1 grün) (1 rot)
(3 rot) (1 blau) (1 rot) (1 blau) (3 rot)
(9 rot)
[…]

Lauflängenkodierung: RLE

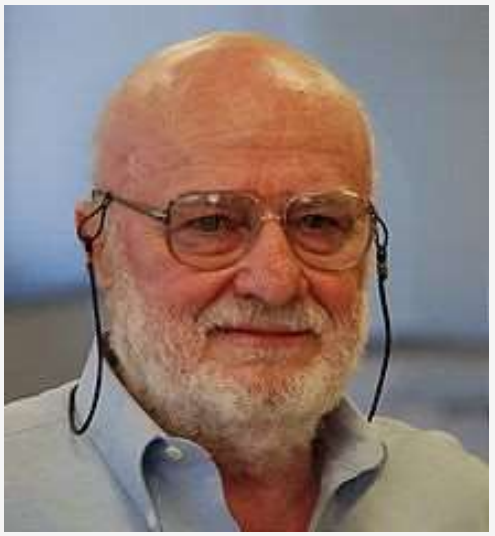
RLE - Lauflängenkodierung

- Verlustfrei
- nutzt lange Folgen sich wiederholender Zeichen oder Zeichenketten („Läufe“ / „runs“) aus:
  - AAAA BBB CCCC D EEEEE → **4A 3B 4C 1D 5E**
  - 1m 1i 2s 1i 2s 1i 2p 1i → **?**
- Funktioniert also am besten bei homogenen Eingabedaten

Wörterbuchbasierte Verfahren

LZW (Lempel-Ziv-Welch)

- 1978 von A. Lempel und J. Ziv entwickelt, 1984 von T. A. Welch verfeinert
- Verlustfrei
- Versucht, den zu komprimierenden Zeichenstrom in Teilstrings zu zerlegen und diese in einem Wörterbuch zu speichern



Zeichenkette:
„Hello World Hello“

→

Tabelle:
0: „Hello“
1: „World“

→ **Codefolge: 0 1 0**

LZW (Lempel-Ziv-Welch)

Unterschiedliche Algorithmen:

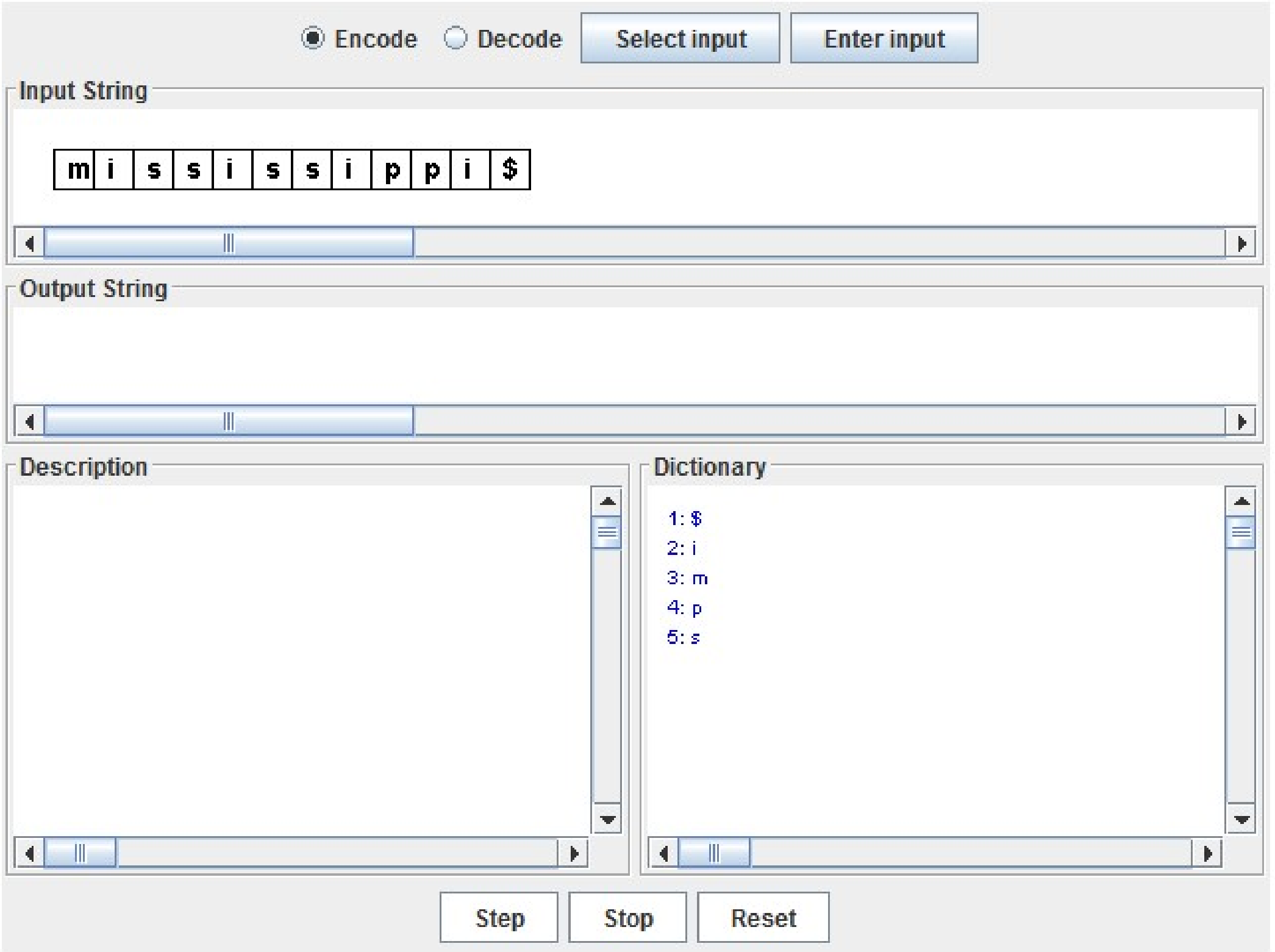
- LZSS (Lempel-Ziv-Storer-Szymanski (gzip, ZIP und andere))
- LZW (Lempel-Ziv-Welch)
- LZC (Lempel-Ziv Compress)
- LZMW (Unix Compress, GIF)

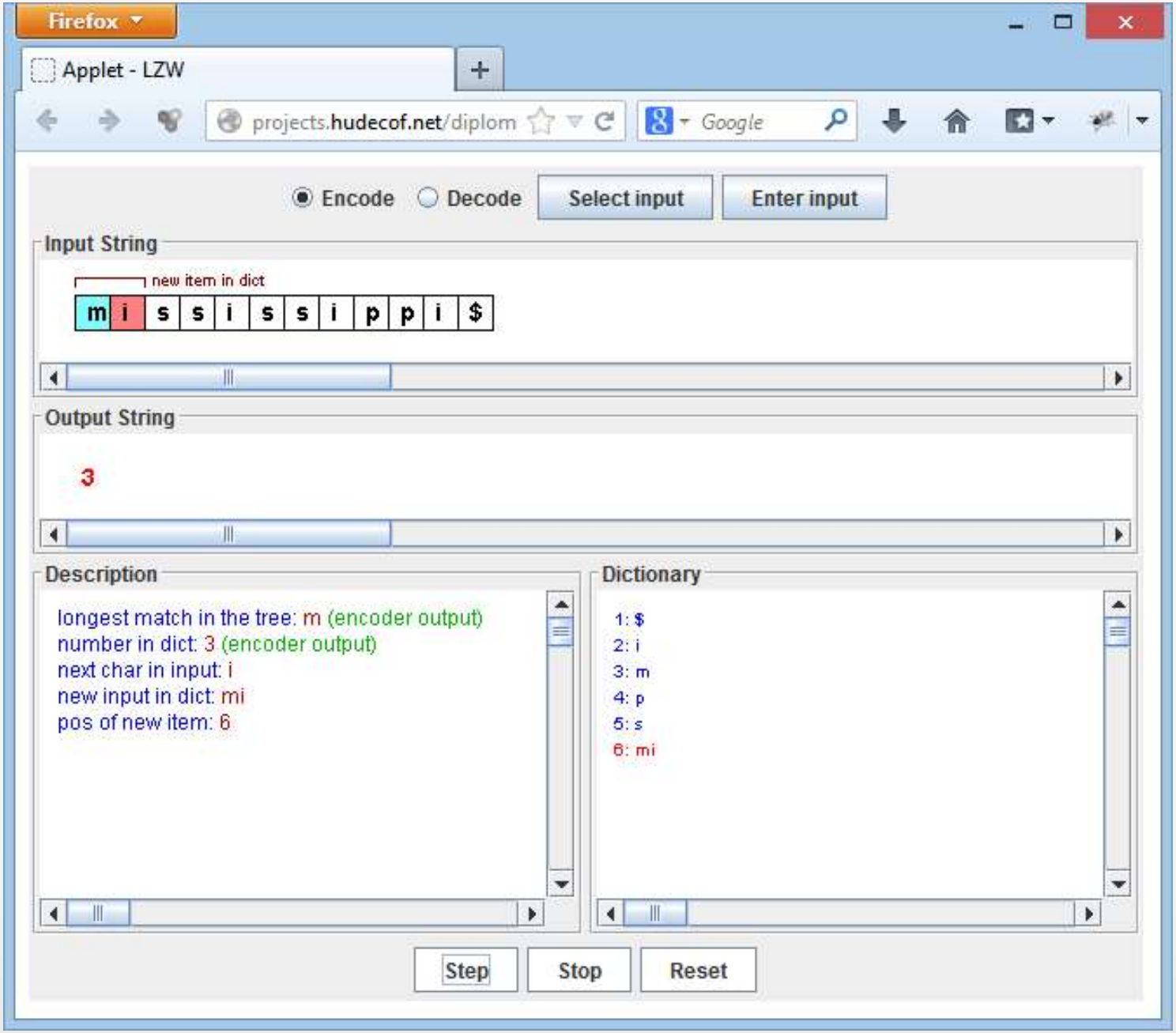
→ Vgl.: http://www-ti.informatik.uni-tuebingen.de/~reinhard/datkom/LZW_Applet.html

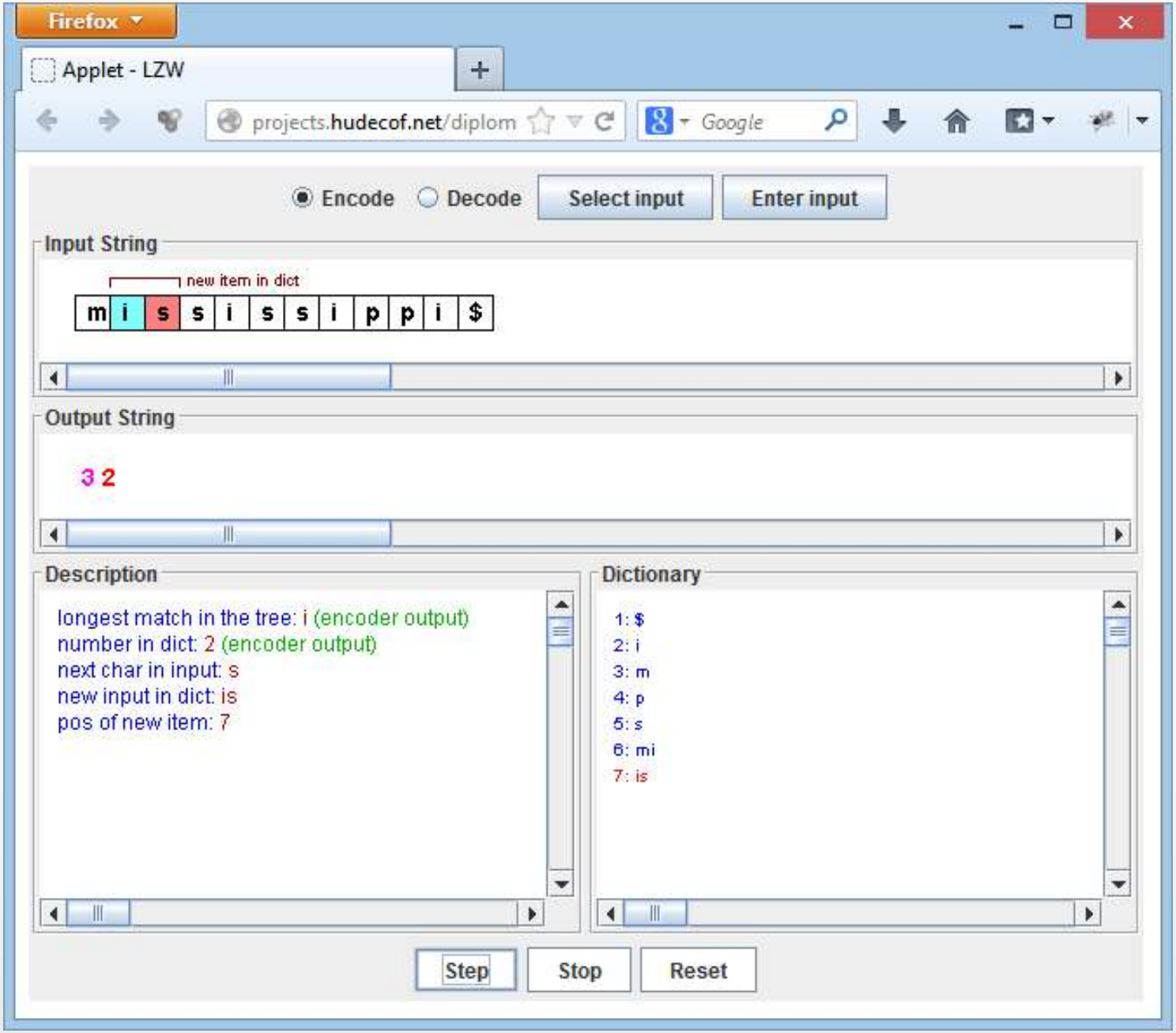
- Patente auf LZW (1980er+), darum GIF problematisch

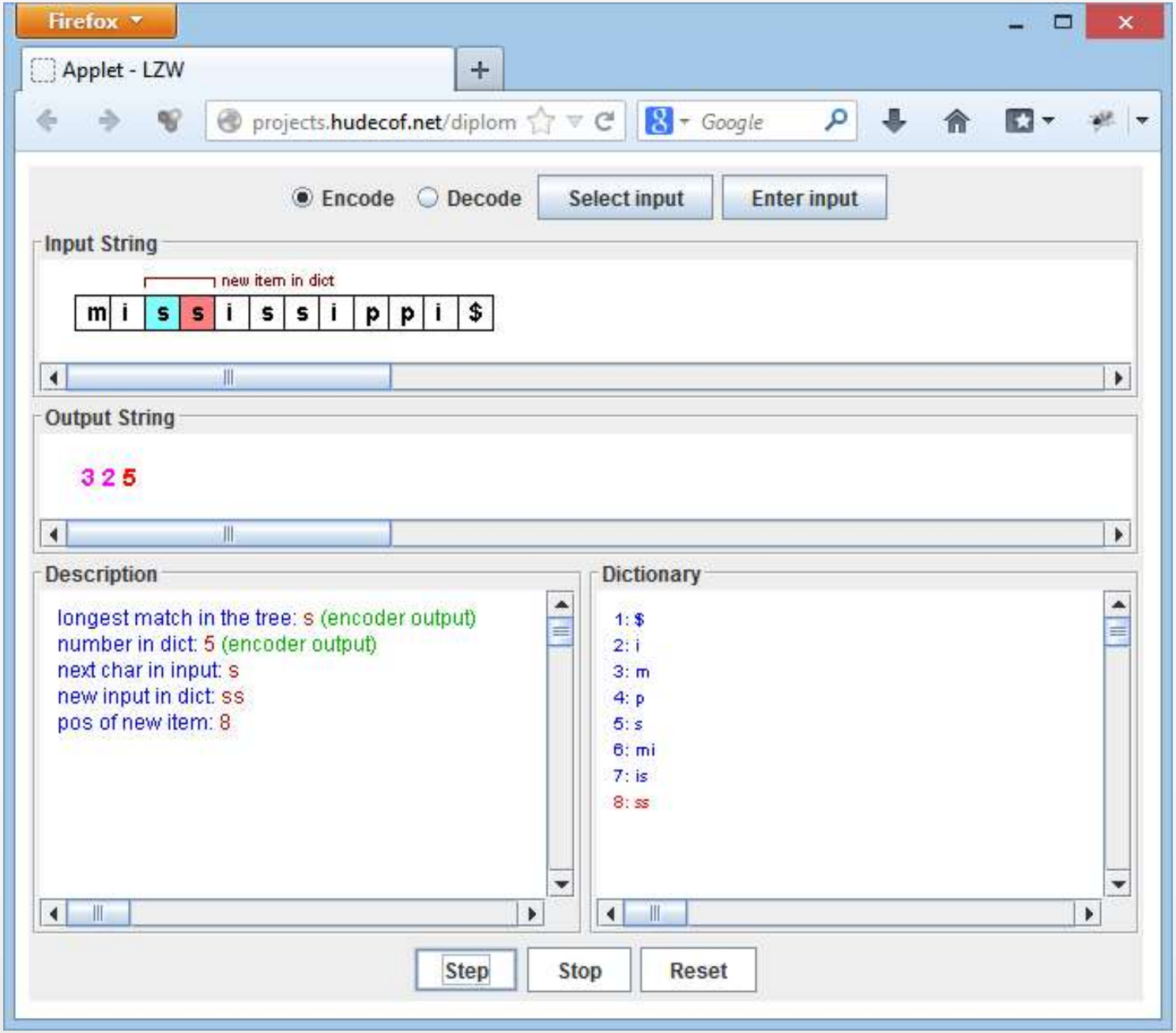
LZW – Ein Beispiel

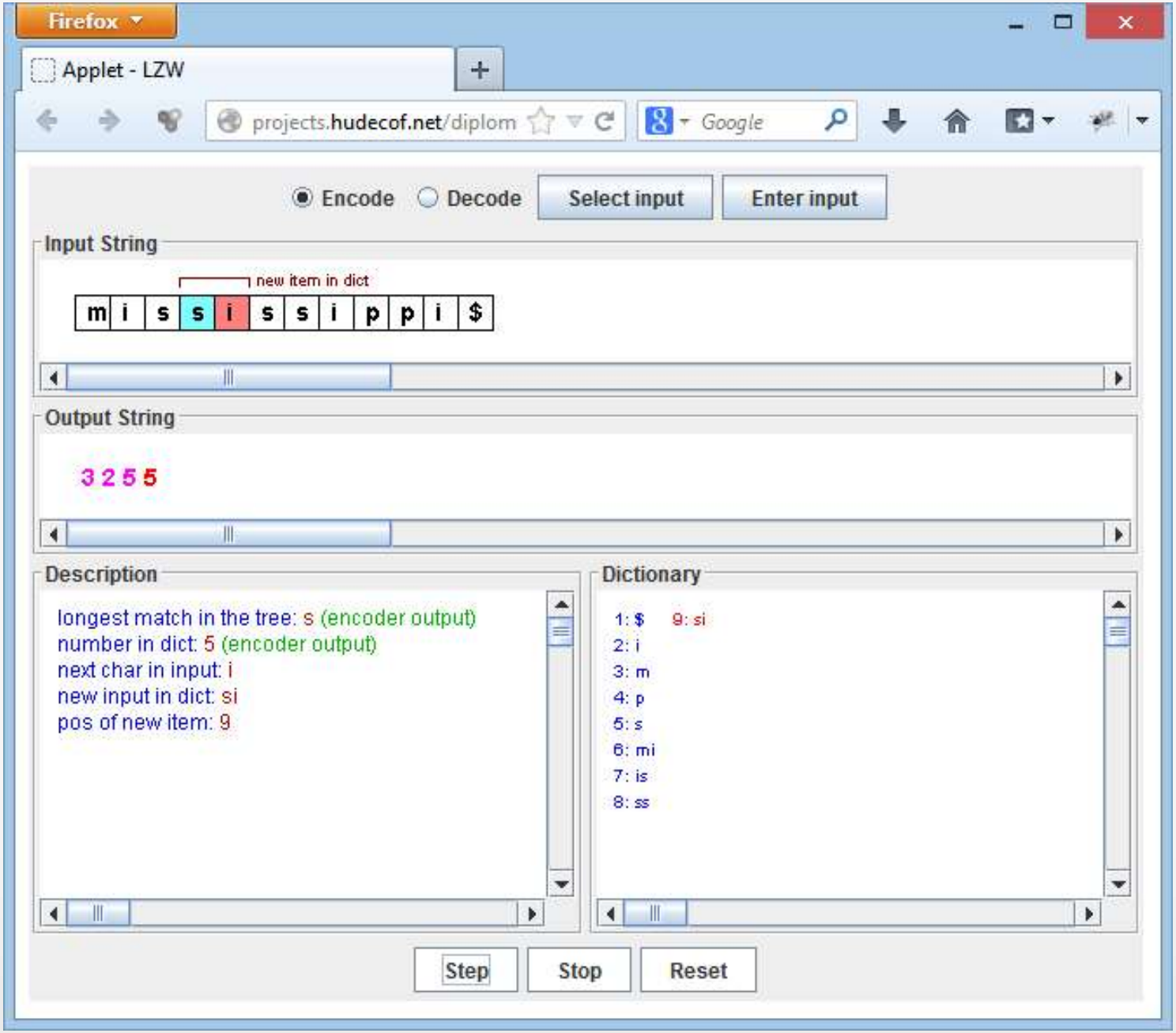
vgl. http://projects.hudecof.net/diplomovka/online/ucebnica/applets/AppletLZW.html

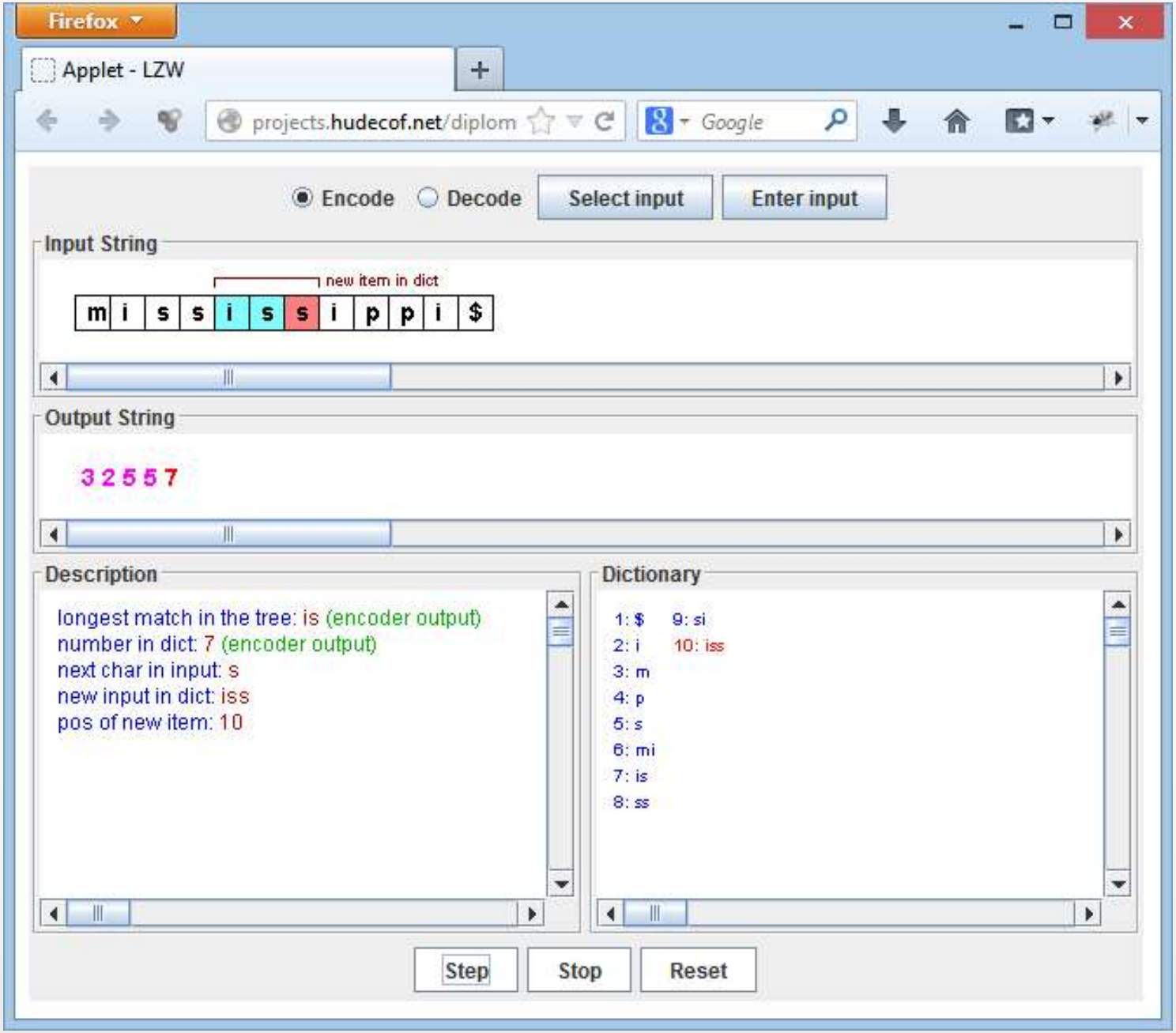

Encode
Decode
Select input
Enter input
Input String
m i s s i s s i p p i $
Output String
Description
Dictionary
1: $
2: i
3: m
4: p
5: s
Step
Stop
Reset

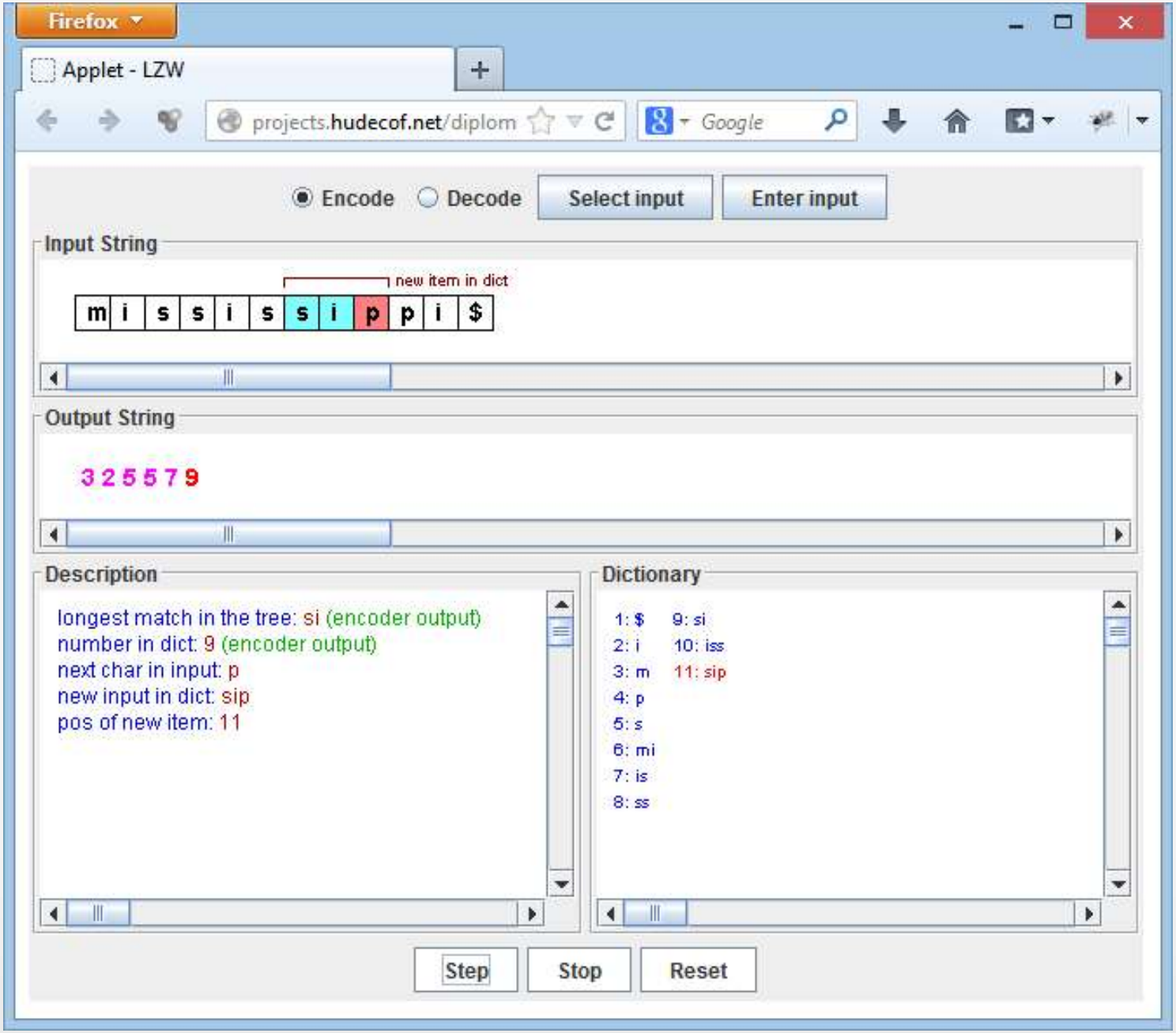

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3
Description
longest match in the tree: m (encoder output)
number in dict: 3 (encoder output)
next char in input: i
new input in dict: mi
pos of new item: 6
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
Step
Stop
Reset

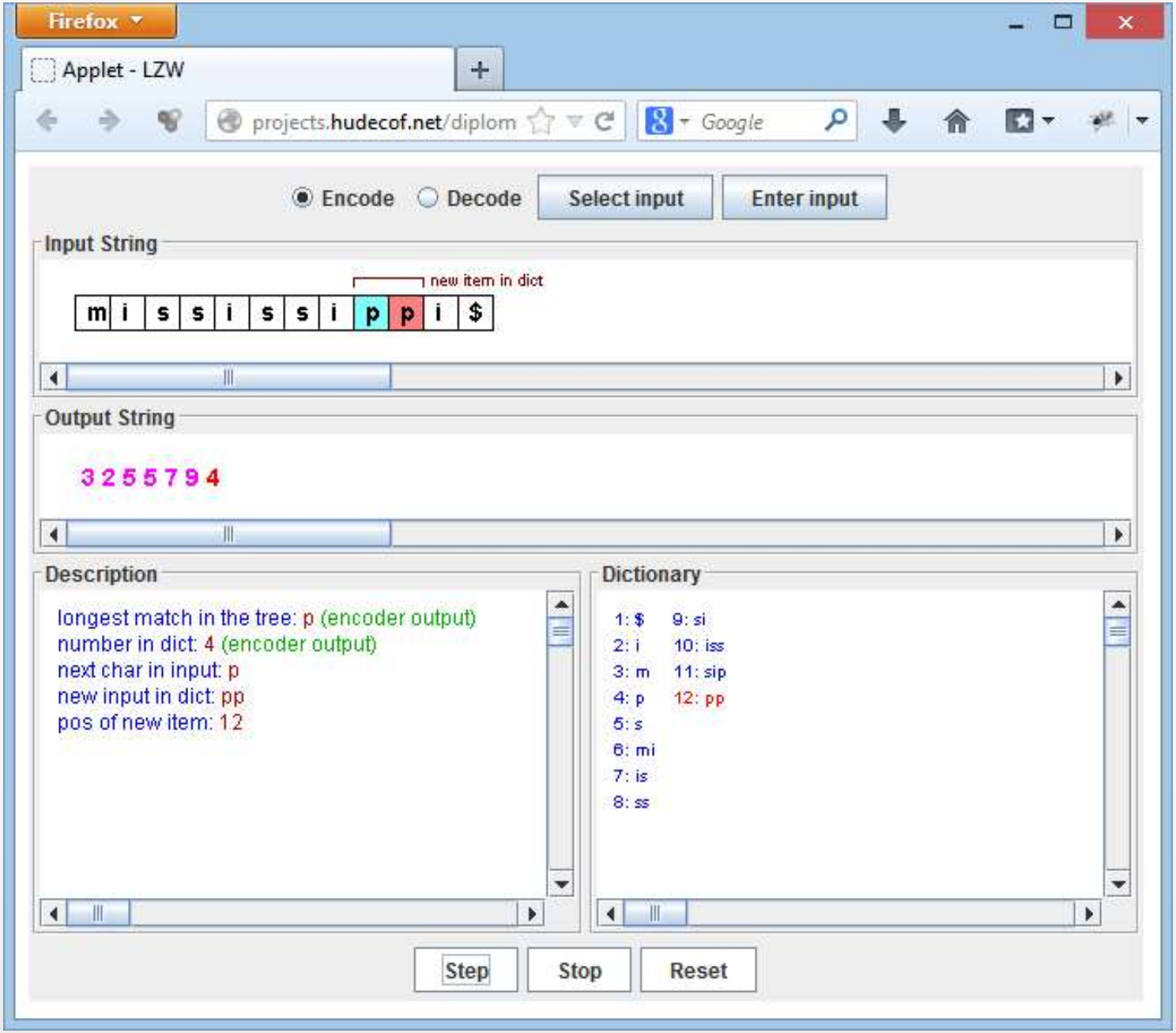

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2
Description
longest match in the tree: i (encoder output)
number in dict: 2 (encoder output)
next char in input: s
new input in dict: is
pos of new item: 7
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
Step
Stop
Reset

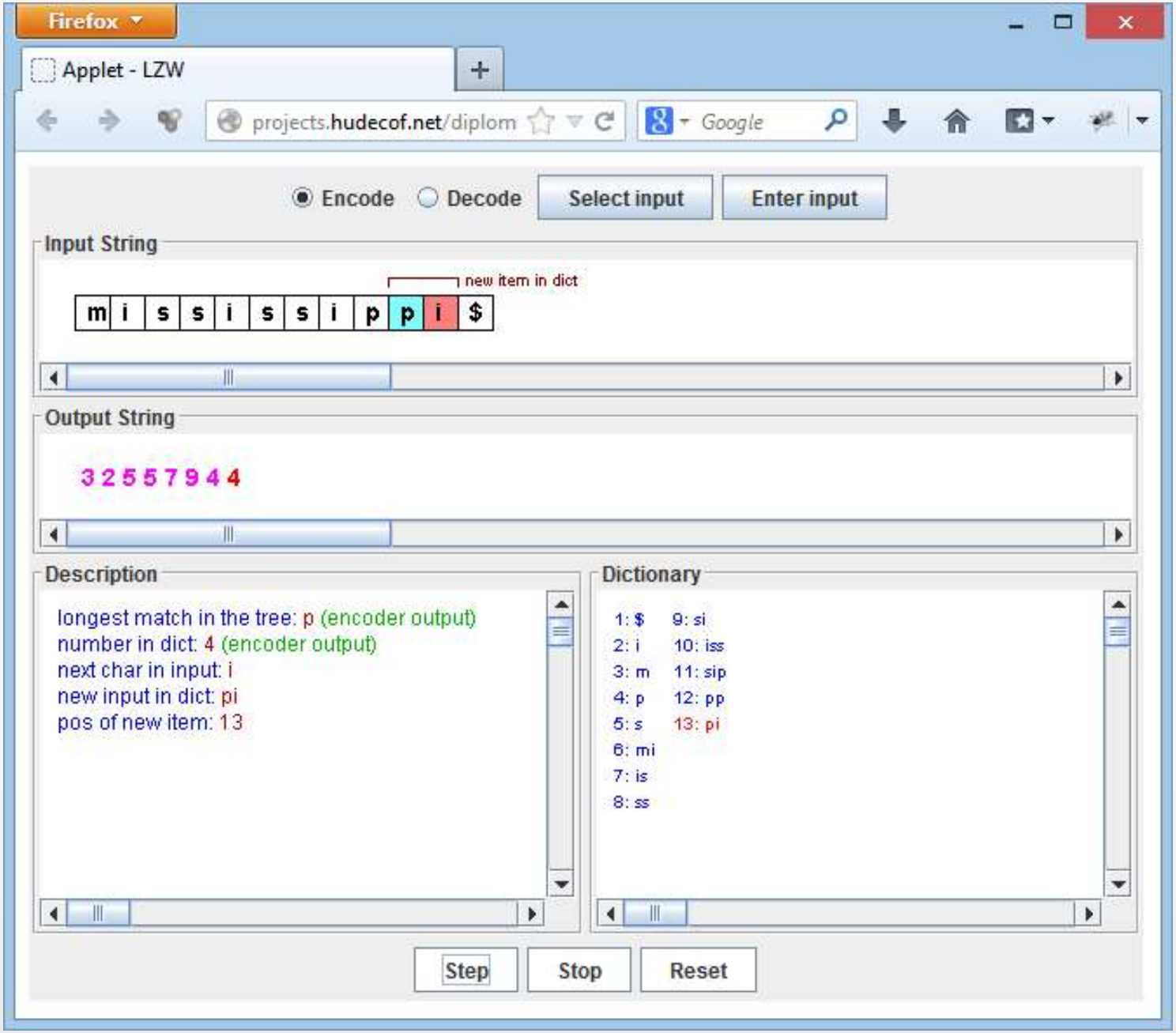

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5
Description
longest match in the tree: s (encoder output)
number in dict: 5 (encoder output)
next char in input: s
new input in dict: ss
pos of new item: 8
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
Step
Stop
Reset

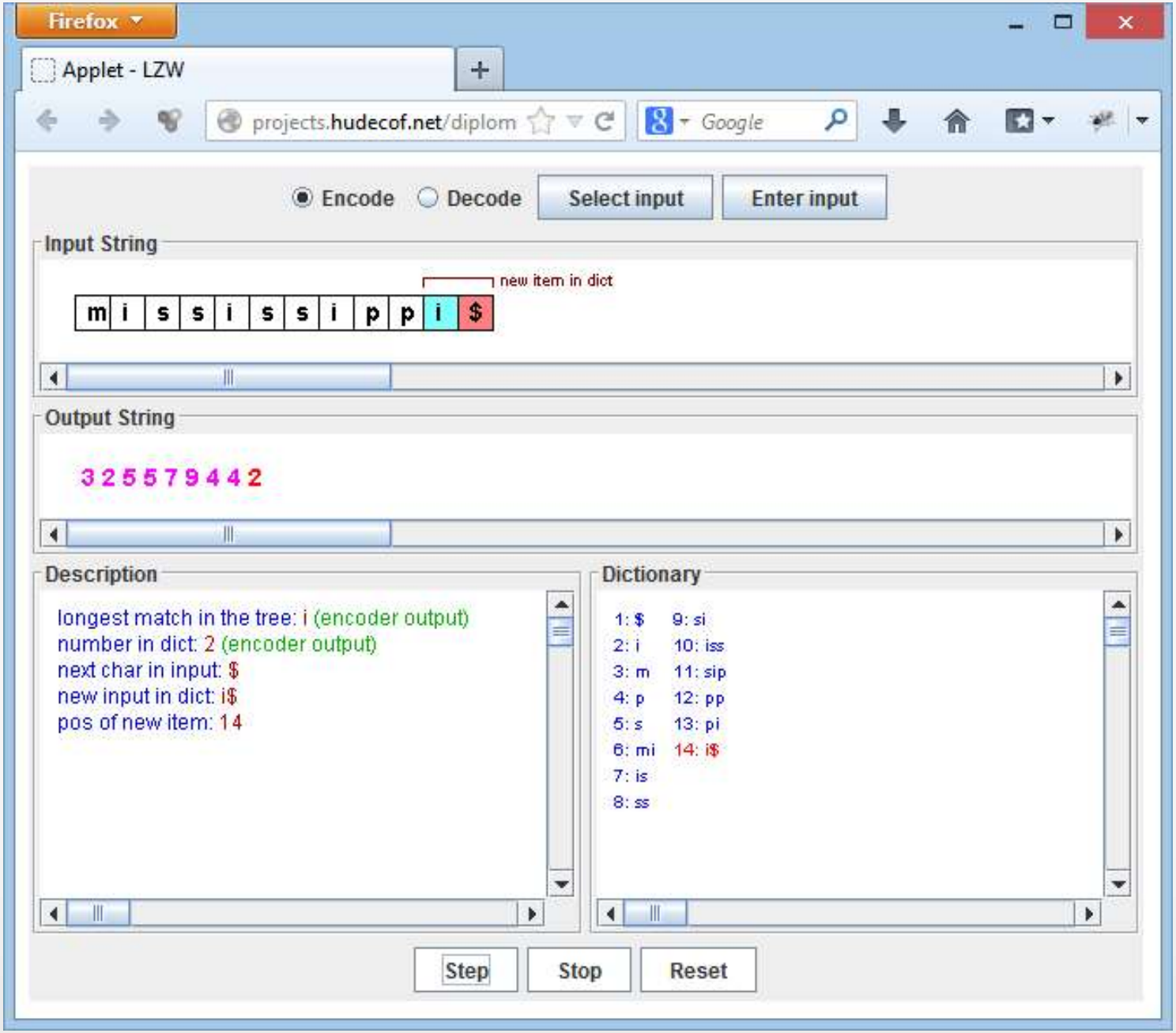

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5
Description
longest match in the tree: s (encoder output)
number in dict: 5 (encoder output)
next char in input: i
new input in dict: si
pos of new item: 9
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
Step
Stop
Reset


Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5 7
Description
longest match in the tree: is (encoder output)
number in dict: 7 (encoder output)
next char in input: s
new input in dict: iss
pos of new item: 10
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
Step
Stop
Reset

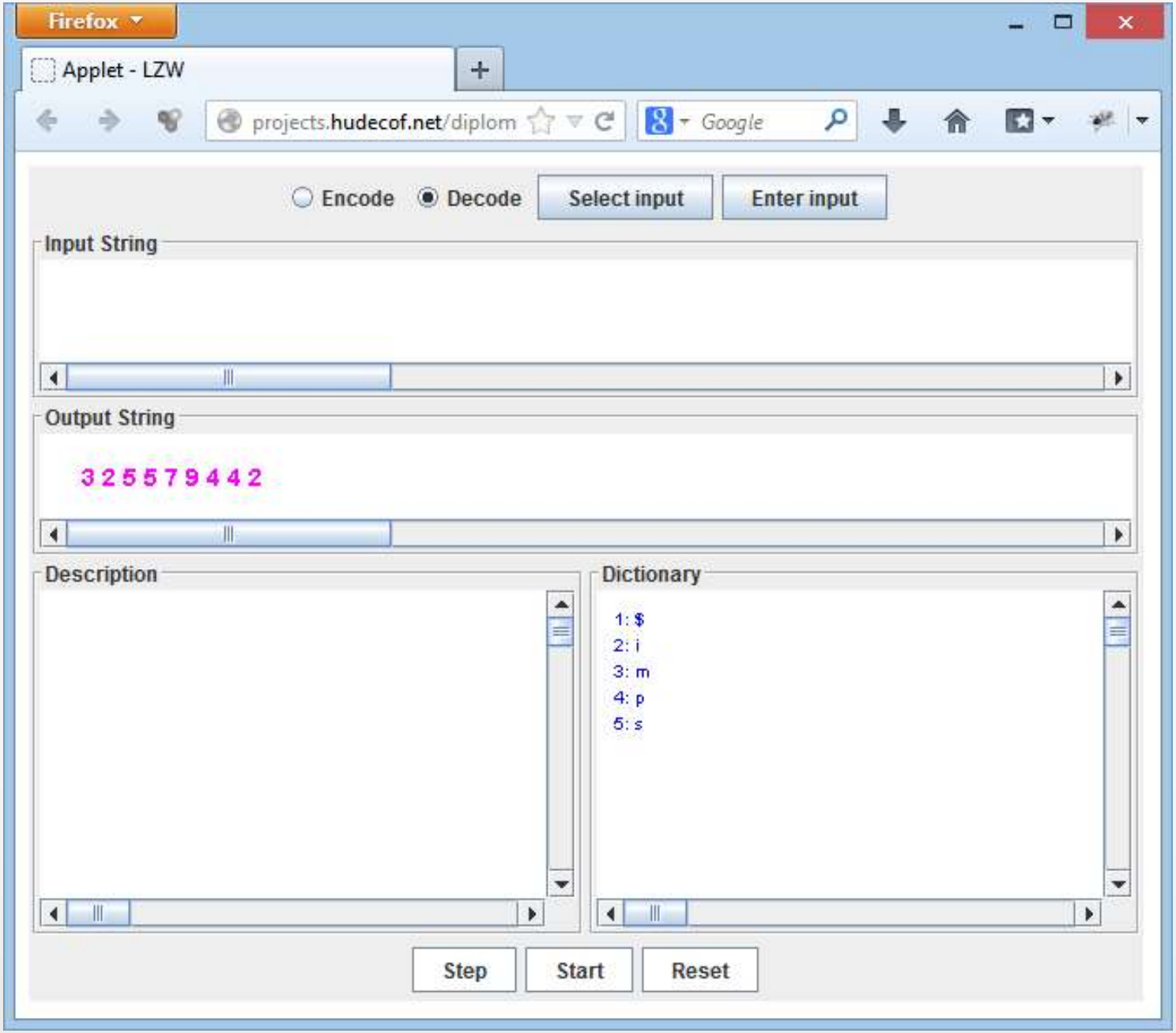

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5 7 9
Description
longest match in the tree: si (encoder output)
number in dict: 9 (encoder output)
next char in input: p
new input in dict: sip
pos of new item: 11
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
Step
Stop
Reset

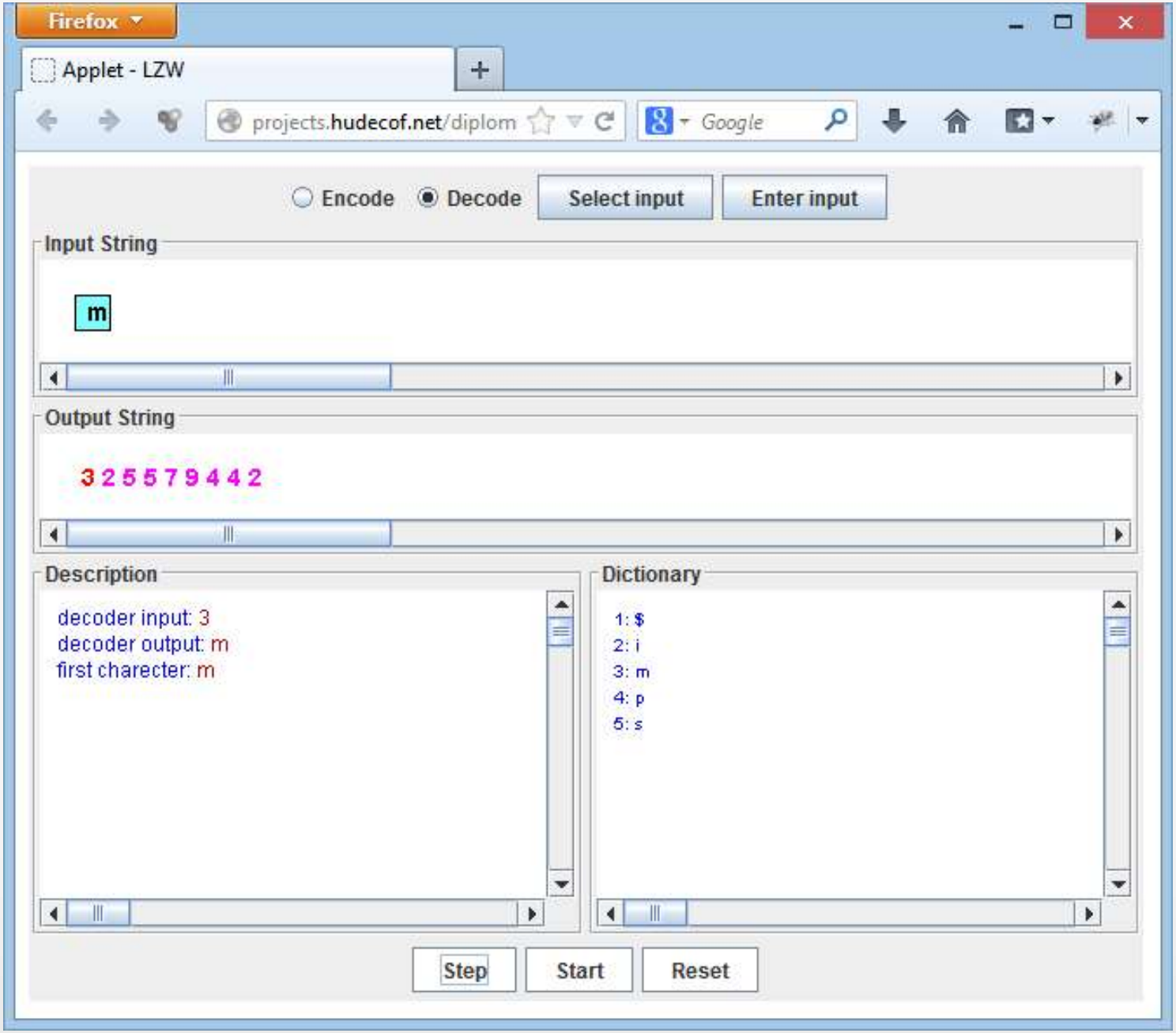

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5 7 9 4
Description
longest match in the tree: p (encoder output)
number in dict: 4 (encoder output)
next char in input: p
new input in dict: pp
pos of new item: 12
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
12: pp
Step
Stop
Reset

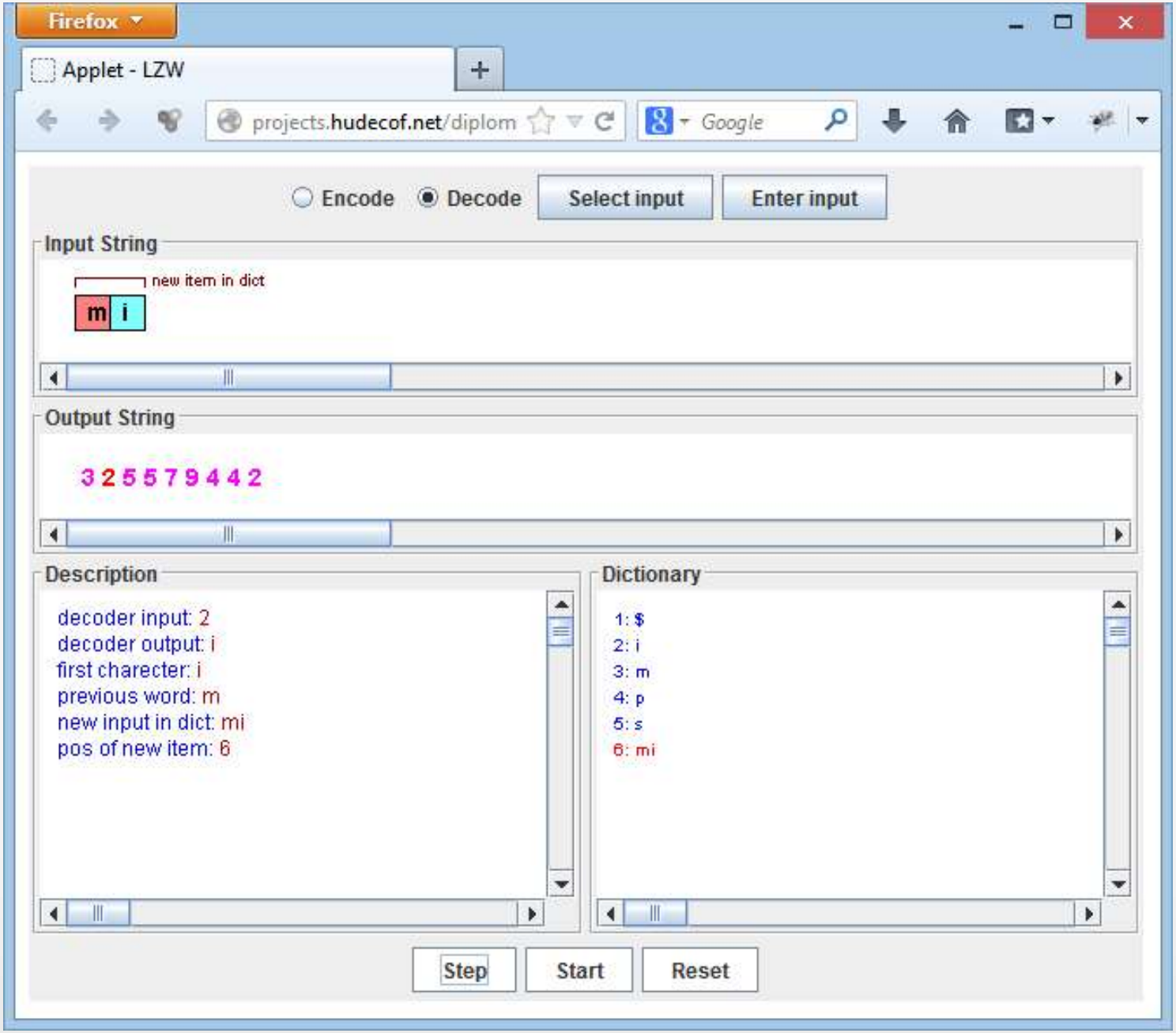

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5 7 9 4 4
Description
longest match in the tree: p (encoder output)
number in dict: 4 (encoder output)
next char in input: i
new input in dict: pi
pos of new item: 13
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
12: pp
13: pi
Step
Stop
Reset

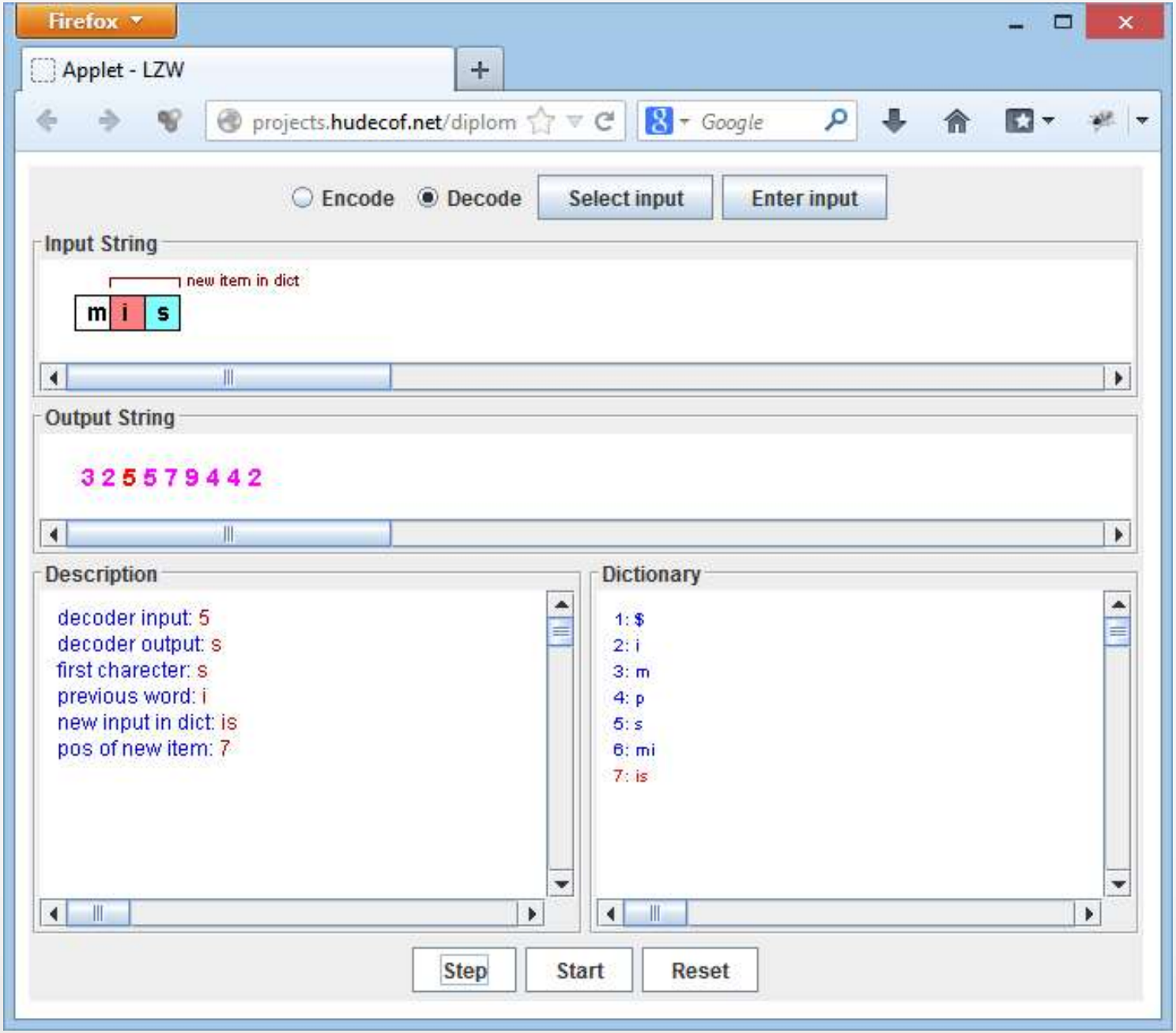

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i $
Output String
3 2 5 5 7 9 4 4 2
Description
longest match in the tree: i (encoder output)
number in dict: 2 (encoder output)
next char in input: $
new input in dict: i$
pos of new item: 14
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
12: pp
13: pi
14: i$
Step
Stop
Reset

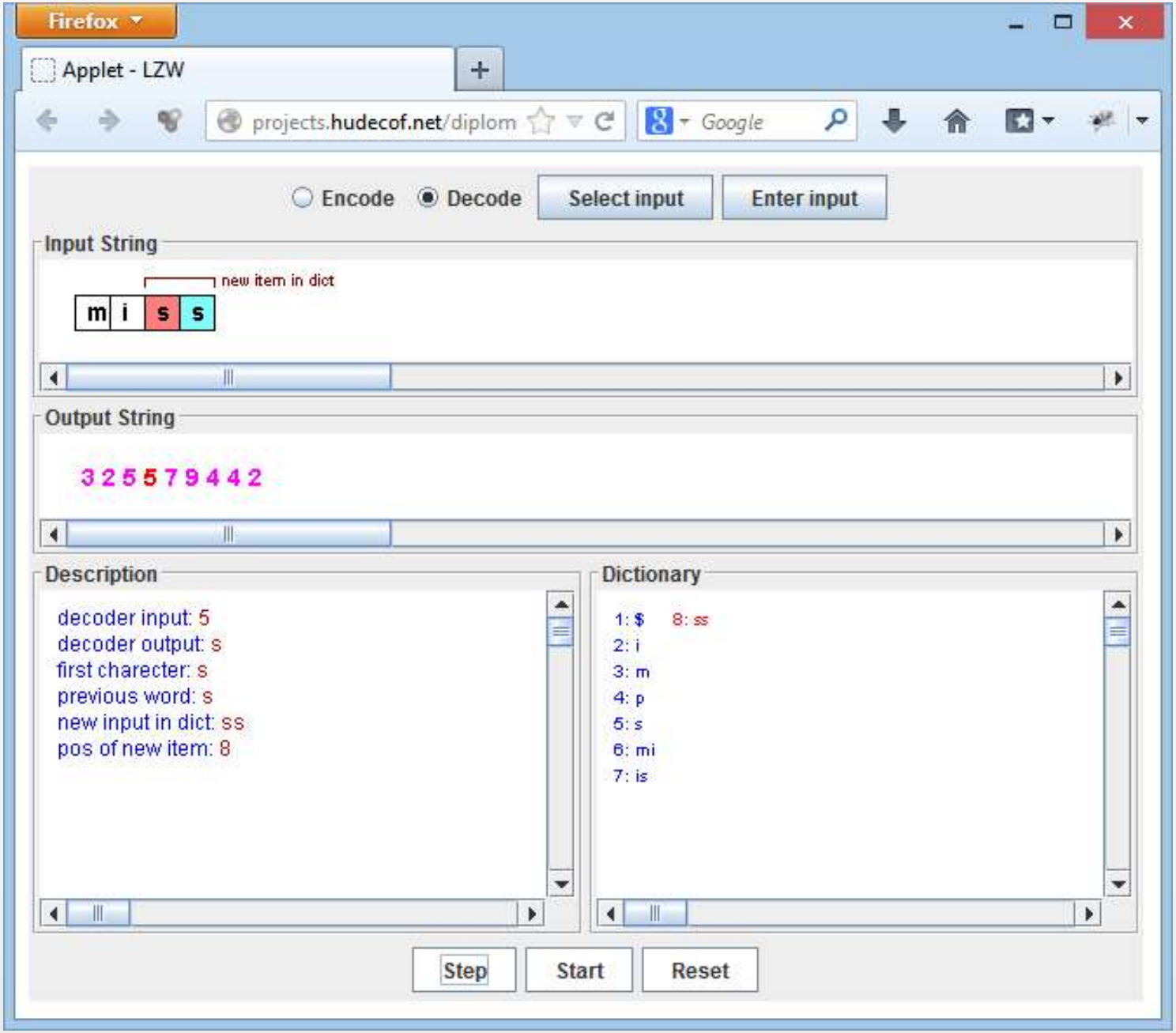
LZW - Decodierung

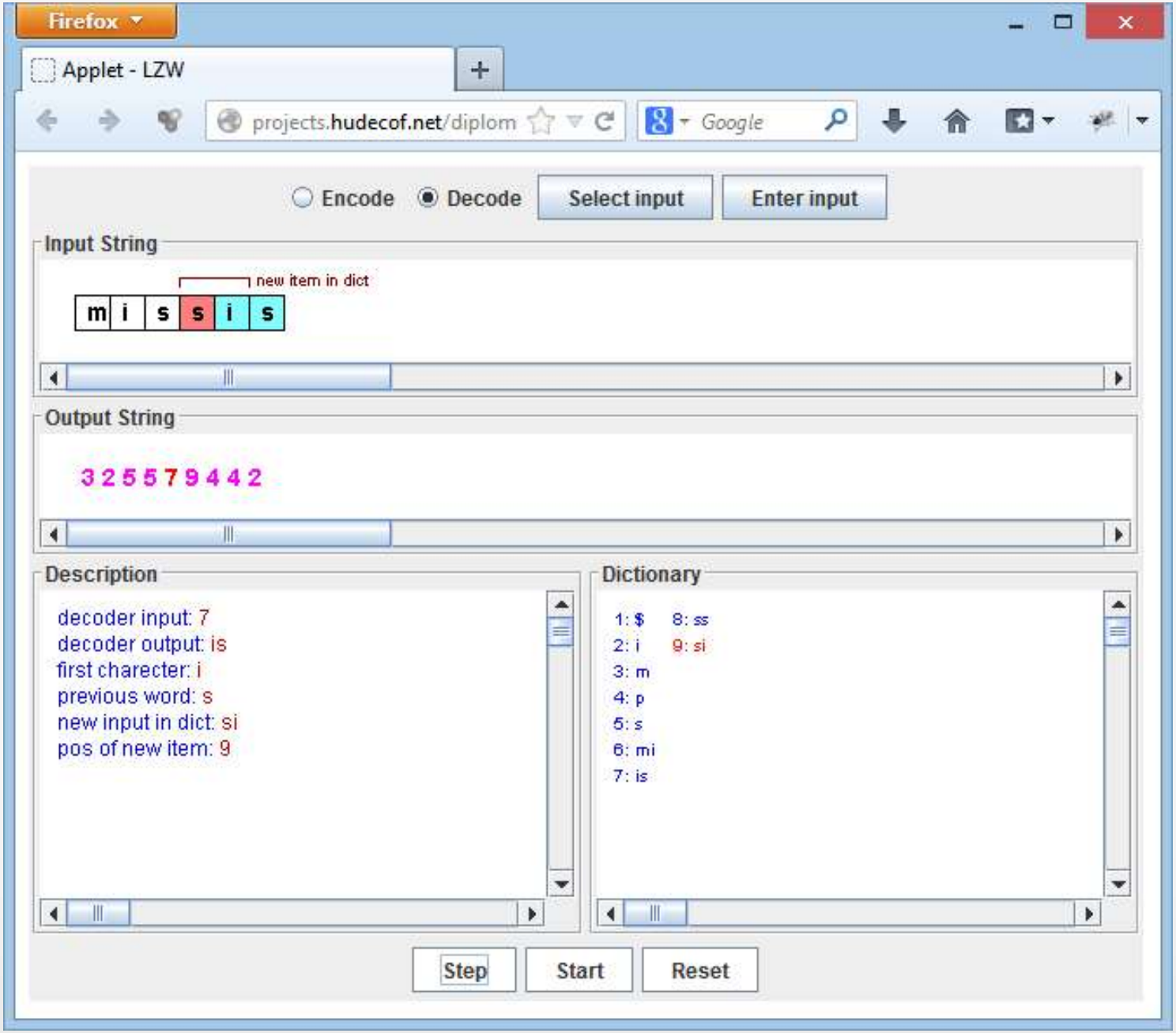

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
Output String
3 2 5 5 7 9 4 4 2
Description
Dictionary
1: $
2: i
3: m
4: p
5: s
Step
Start
Reset

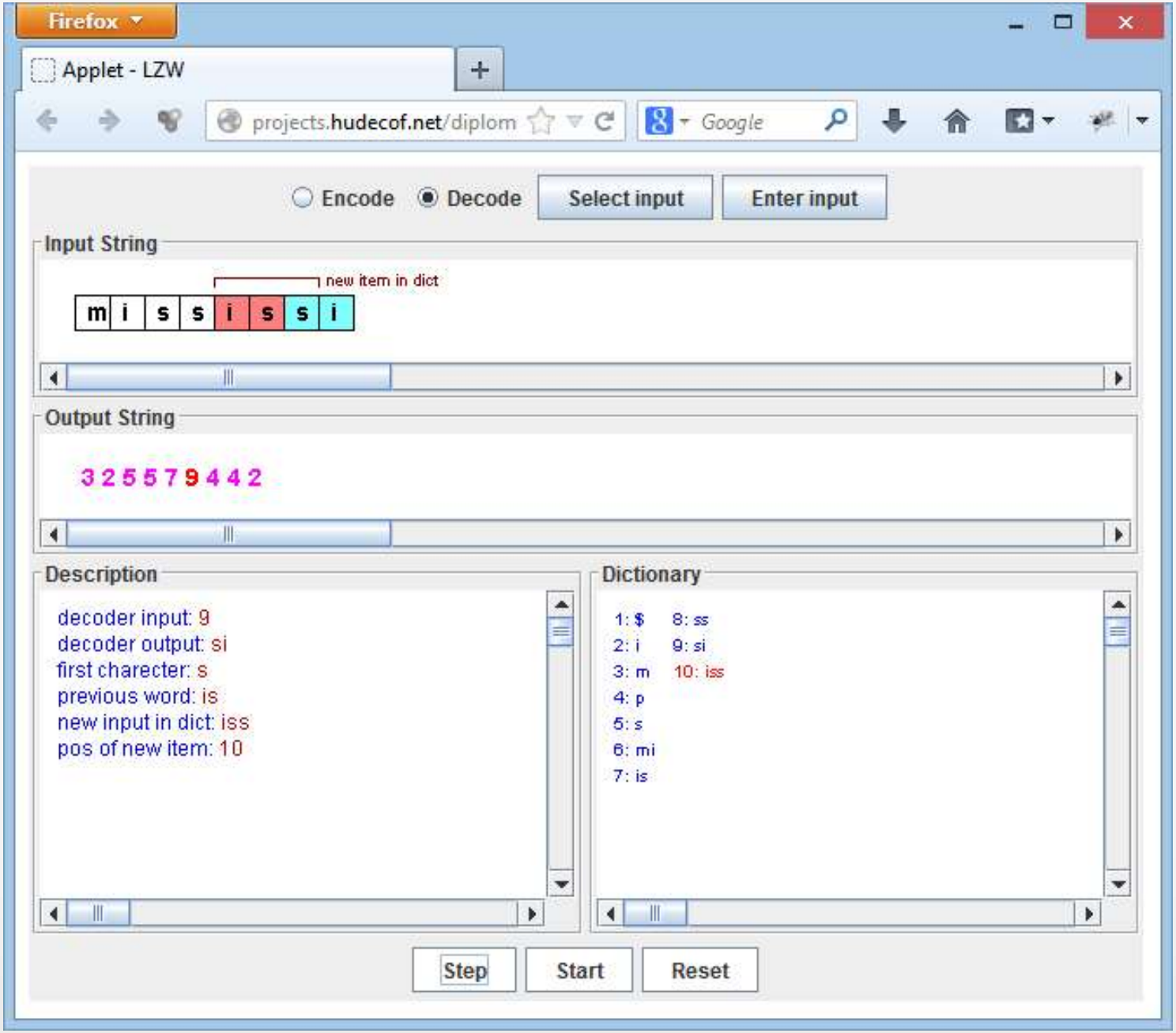

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
m
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 3
decoder output: m
first charecter: m
Dictionary
1: $
2: i
3: m
4: p
5: s
Step
Start
Reset

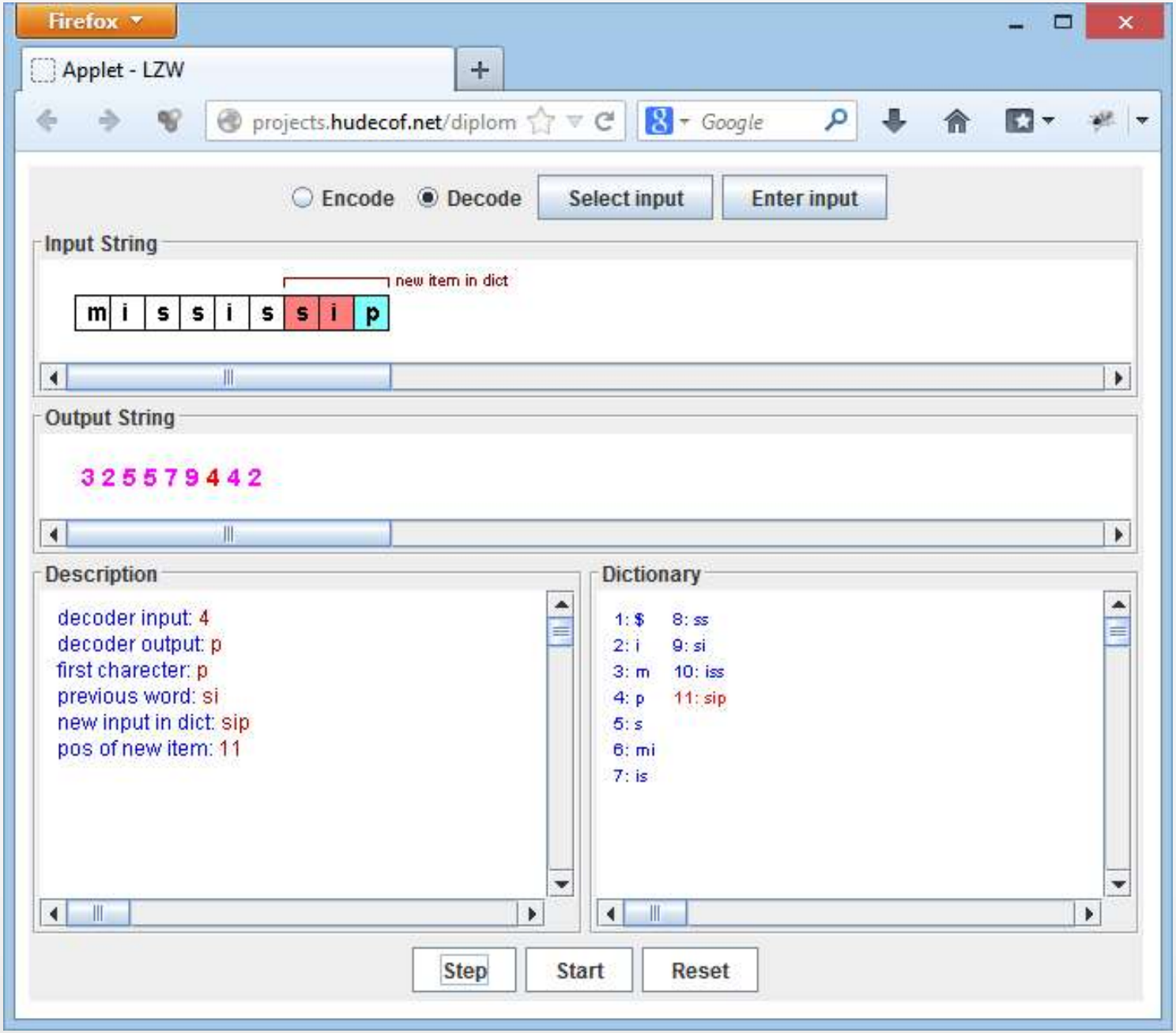

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m
i
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 2
decoder output: i
first charecter: i
previous word: m
new input in dict: mi
pos of new item: 6
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
Step
Start
Reset

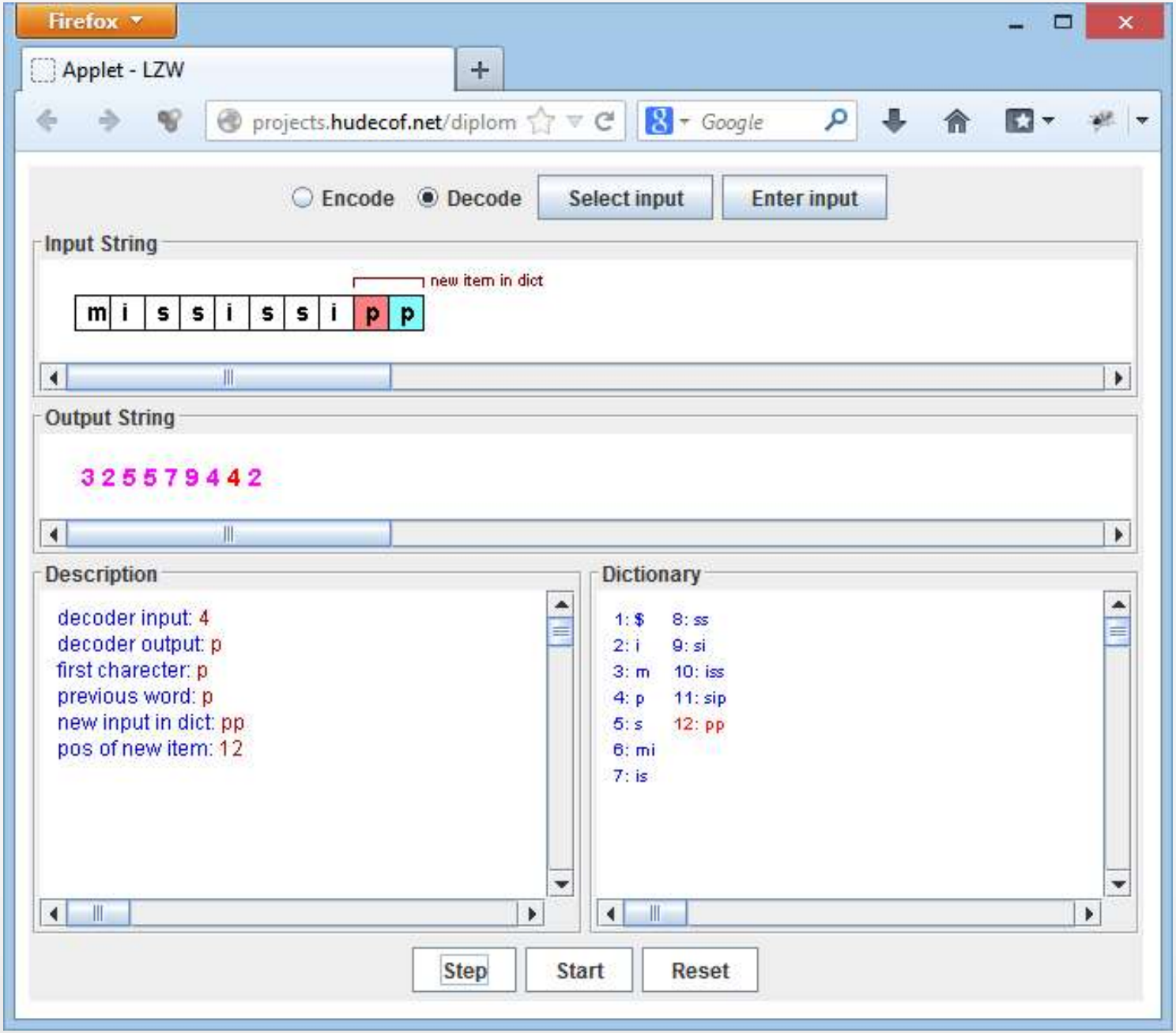

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 5
decoder output: s
first charecter: s
previous word: i
new input in dict: is
pos of new item: 7
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
Step
Start
Reset

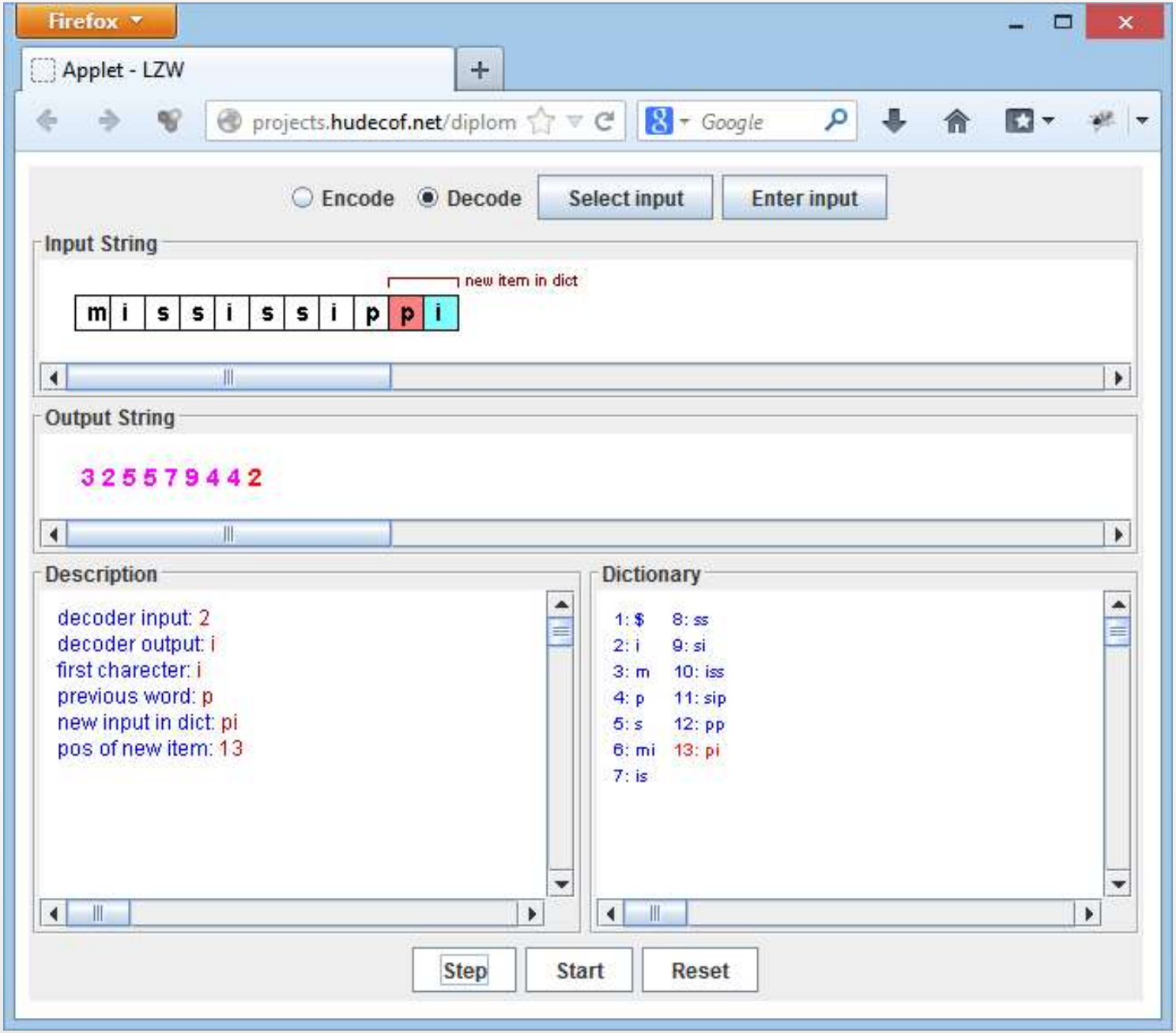

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 5
decoder output: s
first charecter: s
previous word: s
new input in dict: ss
pos of new item: 8
Dictionary
1: $ 8: ss
2: i
3: m
4: p
5: s
6: mi
7: is
Step
Start
Reset


Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 7
decoder output: is
first charecter: i
previous word: s
new input in dict: si
pos of new item: 9
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
Step
Start
Reset


Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 9
decoder output: si
first charecter: s
previous word: is
new input in dict: iss
pos of new item: 10
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
Step
Start
Reset


Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 4
decoder output: p
first charecter: p
previous word: si
new input in dict: sip
pos of new item: 11
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
Step
Start
Reset

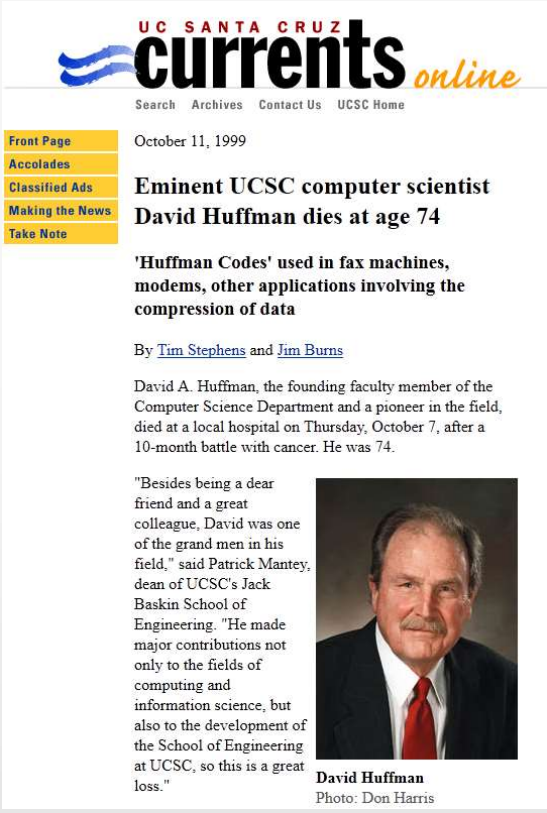

Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 4
decoder output: p
first charecter: p
previous word: p
new input in dict: pp
pos of new item: 12
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
12: pp
Step
Start
Reset


Firefox
Applet - LZW
projects.hudecof.net/diplom
Google
Encode
Decode
Select input
Enter input
Input String
new item in dict
m i s s i s s i p p i
Output String
3 2 5 5 7 9 4 4 2
Description
decoder input: 2
decoder output: i
first charecter: i
previous word: p
new input in dict: pi
pos of new item: 13
Dictionary
1: $
2: i
3: m
4: p
5: s
6: mi
7: is
8: ss
9: si
10: iss
11: sip
12: pp
13: pi
Step
Start
Reset

Statistische Kompressionsverfahren

Huffman Kodierung

- 1952 von David A. Huffman (1925-1999) vorgestellt
- Verlustfrei
- Kompression über **Binärbaum**
- Huffman-Algorithmus verfolgt das Ziel, weniger häufigen Symbolen längere Codewörter zuzuweisen



UC SANTA CRUZ currents online

Search Archives Contact Us UCSC Home

Front Page
Accolades
Classified Ads
Making the News
Take Note

October 11, 1999

Eminent UCSC computer scientist David Huffman dies at age 74

'Huffman Codes' used in fax machines, modems, other applications involving the compression of data

By Tim Stephens and Jim Burns

David A. Huffman, the founding faculty member of the Computer Science Department and a pioneer in the field, died at a local hospital on Thursday, October 7, after a 10-month battle with cancer. He was 74.

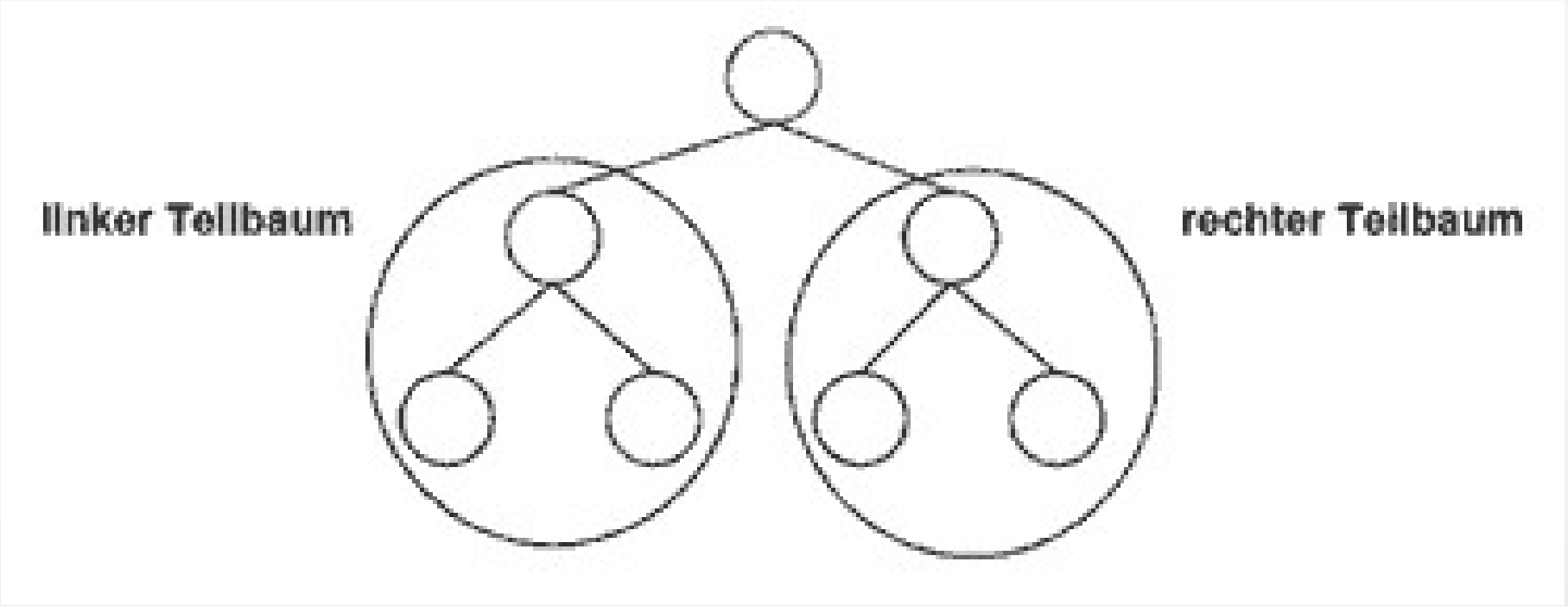
"Besides being a dear friend and a great colleague, David was one of the grand men in his field," said Patrick Mantey, dean of UCSC's Jack Baskin School of Engineering. "He made major contributions not only to the fields of computing and information science, but also to the development of the School of Engineering at UCSC, so this is a great loss."

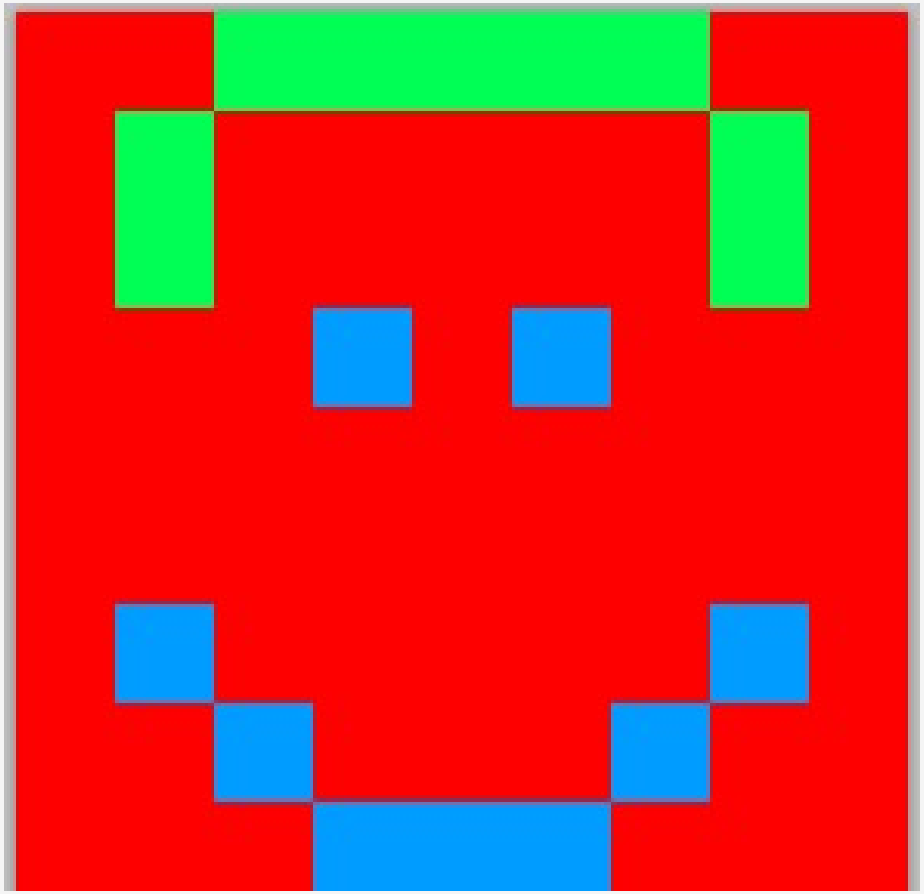
David Huffman
Photo: Don Harris

Nachruf: http://www1.ucsc.edu/currents/99-00/10-11/huffman.html

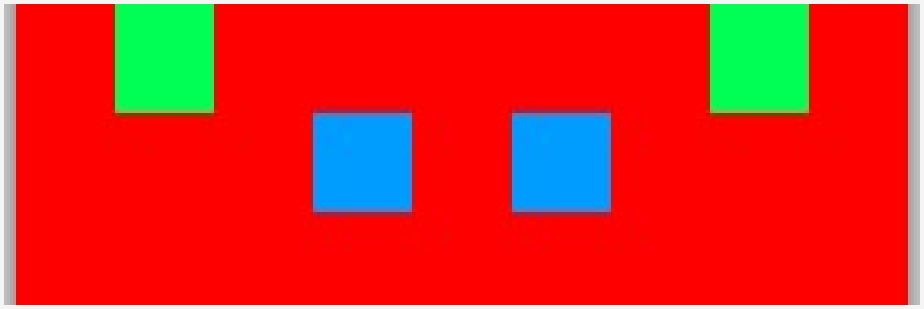
Erinnerung: Binärbäume

Ein **Binärbaum** ist definiert als ein Baum, dessen Knoten über maximal zwei Kindknoten verfügen dürfen:

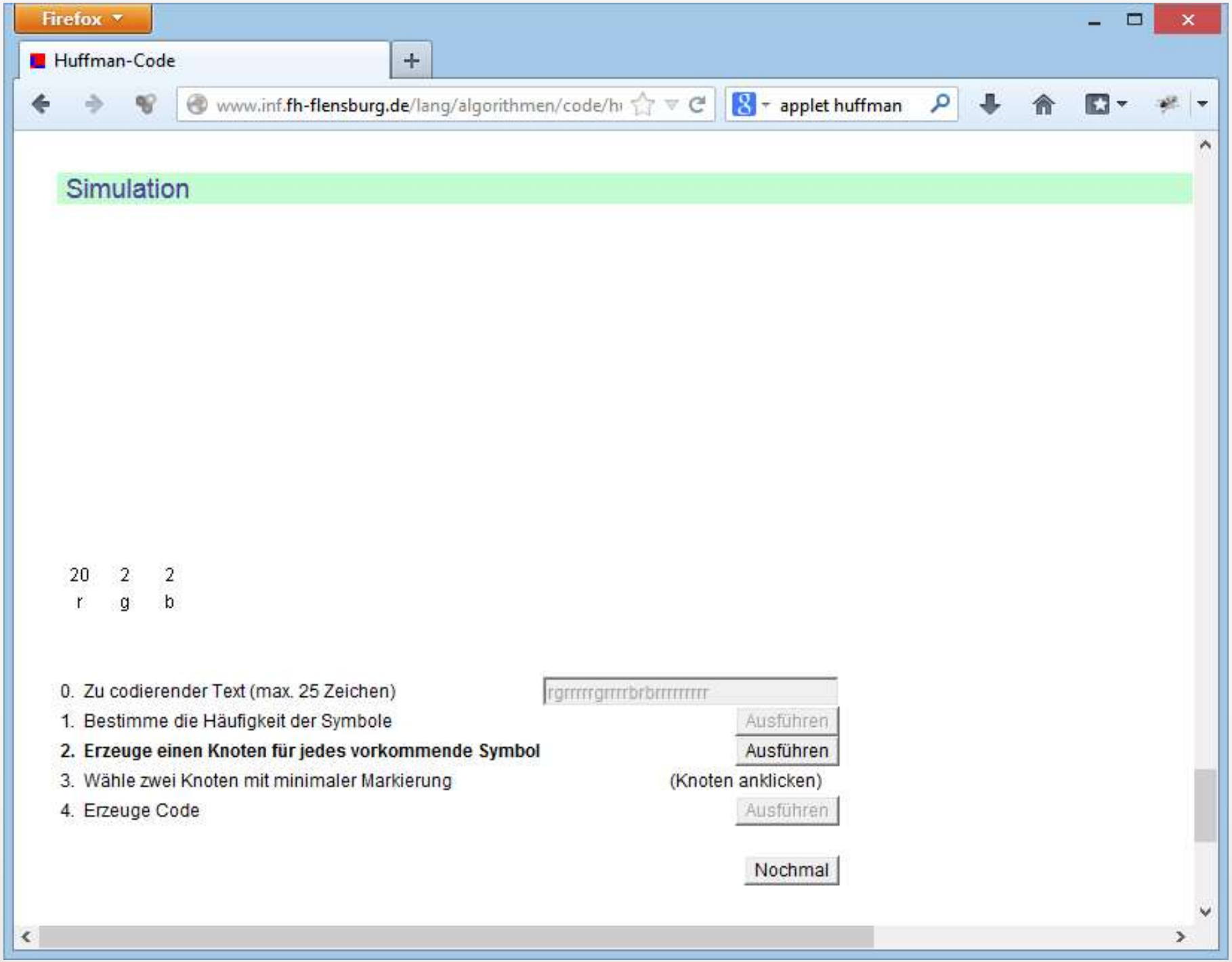




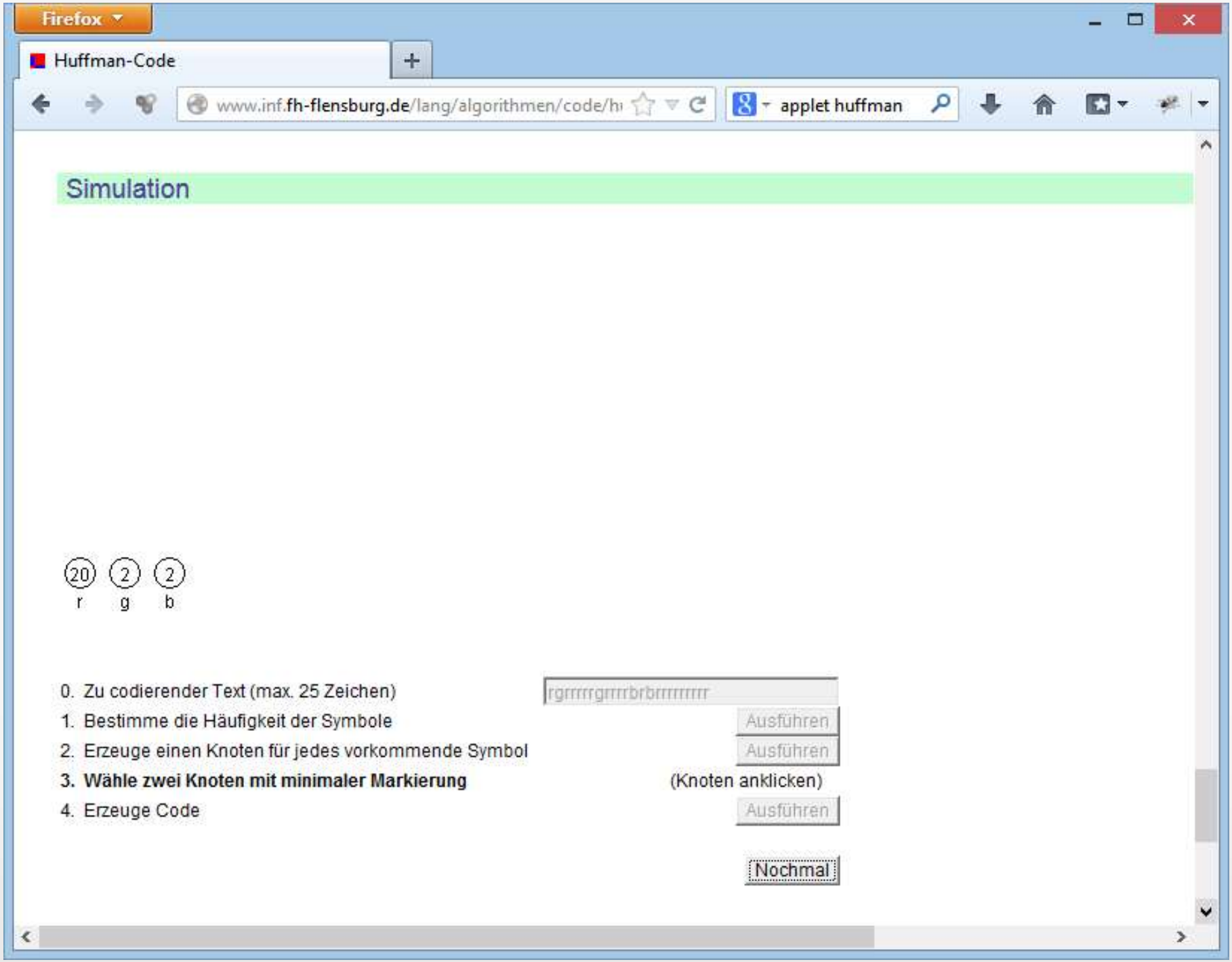
rot, rot, grün, grün, grün, grün, grün, rot, rot
rot, grün, rot, rot, rot, rot, rot, grün, rot,
rot, grün, rot, rot, rot, rot, rot, grün, rot,
rot, rot, rot, blau, rot, blau, rot, rot, rot,
rot, rot, rot, rot, rot, rot, rot, rot,
rot, rot, rot, rot, rot, rot, rot, rot
rot, blau, rot, rot, rot, rot, rot, blau, rot,
rot, rot, blau, rot, rot, rot, blau, rot, rot
rot, rot, rot, blau, blau, blau, rot, rot, rot
|
v
rrgggggrrrgrrrrrgrrgrrrrrgrrrrbrbrrrrrrrrrrrrrrrrrrrrrbr
rrrrbrrrbrrrbrrrrrbbbrrr



rgrrrrrgrrrrbrbrrrrrrrrrrrr



http://www.iti.fh-flensburg.de/lang/algorithmen/code/huffman/huffman.htm


Firefox
Huffman-Code
www.inf.fh-flensburg.de/lang/algorithmen/code/h
applet huffman
Simulation
20
2
2
r
g
b
0. Zu codierender Text (max. 25 Zeichen)
rgrrrrrgrrrrbrbrrrrrrrrr
1. Bestimme die Häufigkeit der Symbole
Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol
Ausführen
3. Wähle zwei Knoten mit minimaler Markierung
(Knoten anklicken)
4. Erzeuge Code
Ausführen
Nochmal

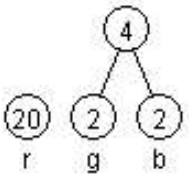
Firefox

Huffman-Code

www.inf.fh-flensburg.de/lang/algorithmen/code/h

applet huffman

Simulation



0. Zu codierender Text (max. 25 Zeichen) rgrrrrrgrrrrbrbrrrrrrrrr
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

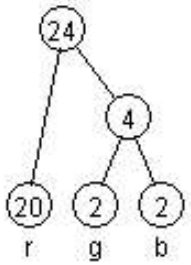
Firefox

Huffman-Code

www.inf.fh-flensburg.de/lang/algorithmen/code/h

applet huffman

Simulation



0. Zu codierender Text (max. 25 Zeichen) rgrrrrrgrrrrbrbrrrrrrrrr
1. Bestimme die Häufigkeit der Symbole — Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol — Ausführen
3. Wähle zwei Knoten mit minimaler Markierung — (Knoten anklicken)
4. **Erzeuge Code** — Ausführen

Nochmal

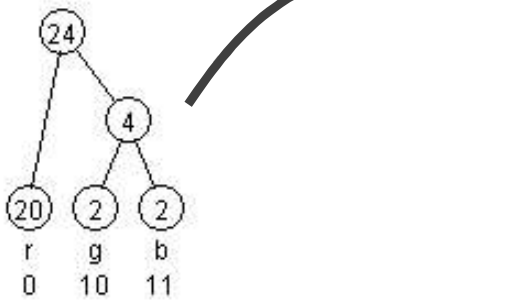
Firefox

Huffman-Code

www.inf.fh-flensburg.de/lang/algorithmen/code/h

applet huffman

Simulation



Codegenerierung nach folgender Logik:

- Start beim Wurzelknoten
- An jedem Knoten (einschl. Wurzelkn.):
  - Links abbiegen: 0 notieren
  - Rechts abbiegen: 1 notieren

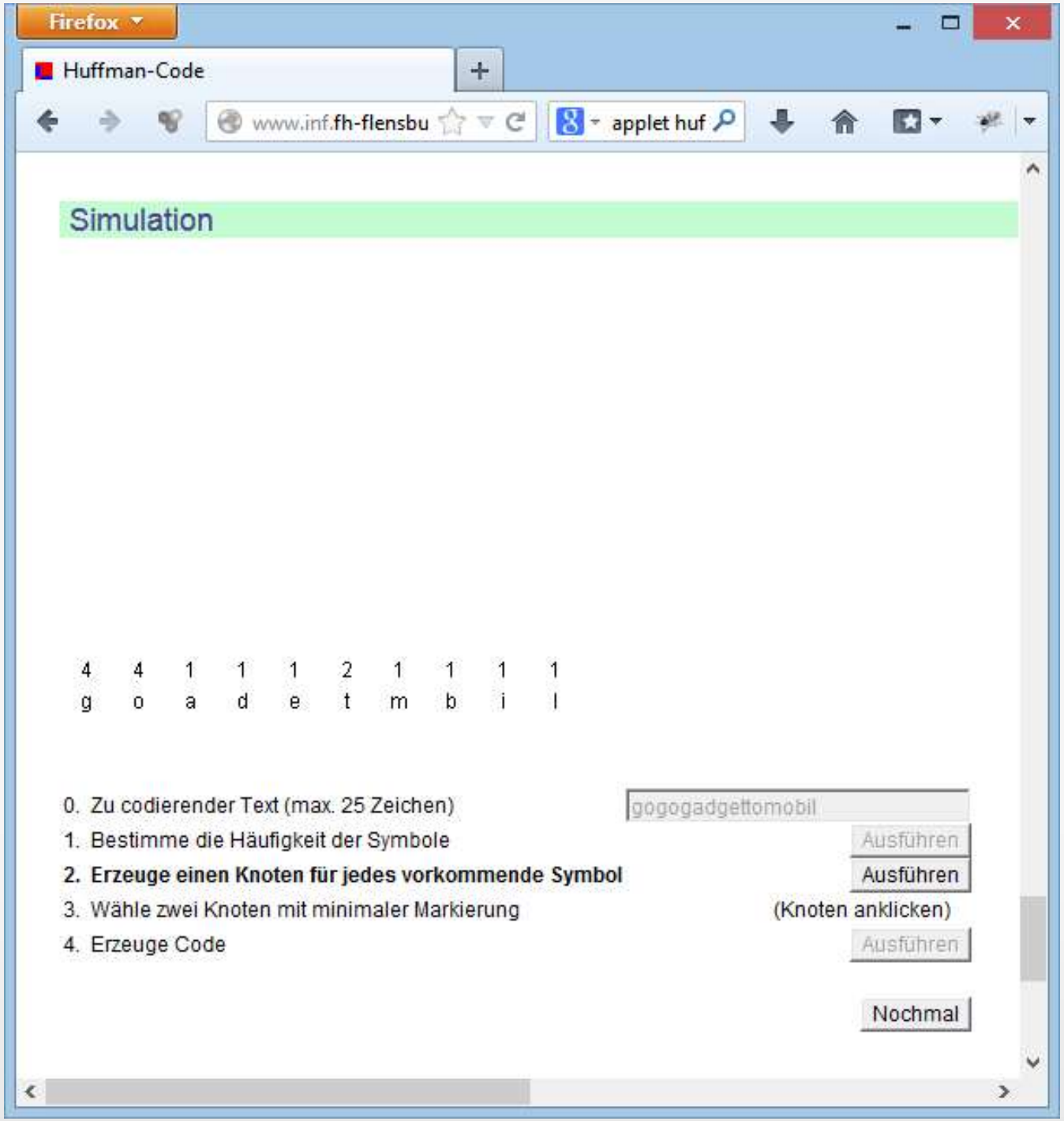
- Beispiel „g“: Vom Wurzelknoten aus rechten Kindknoten besucht (1 notiert); anschließend linken Kindknoten besucht (0 notiert) → g := 10

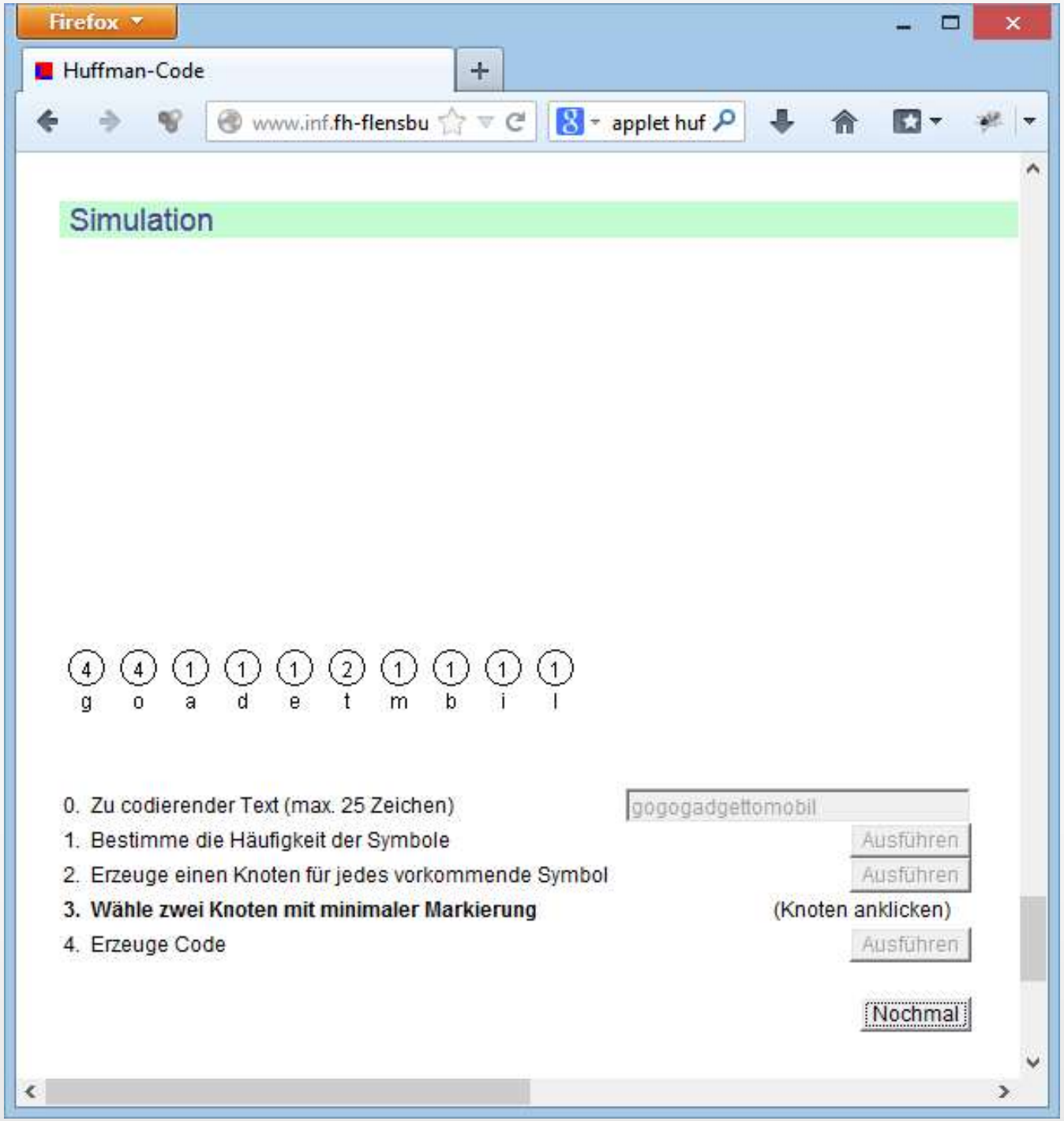
0. Zu codierender Text (max. 25 Zeichen) rgrrrrrgrrrrbrbrrrrrrrrr
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)
4. Erzeuge Code Ausführen

Codierung: 01000000100000110110000000000

4-Bit-Blockcode: 96 Huffman: 28 Informationsgehalt: 19.6 Kompression: 71.0% Nochmal

Noch ein Beispiel


Firefox
Huffman-Code
www.inf.fh-flensbu
applet huf
Simulation
4 4 1 1 1 2 1 1 1 1
g o a d e t m b i l
0. Zu codierender Text (max. 25 Zeichen)
gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole
Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol
Ausführen
3. Wähle zwei Knoten mit minimaler Markierung
(Knoten anklicken)
4. Erzeuge Code
Ausführen
Nochmal


Firefox
Huffman-Code
www.inf.fh-flensbu
applet huf
Simulation
4 4 1 1 1 2 1 1 1 1
g o a d e t m b i l
0. Zu codierender Text (max. 25 Zeichen)
gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole
Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol
Ausführen
3. Wähle zwei Knoten mit minimaler Markierung
(Knoten anklicken)
4. Erzeuge Code
Ausführen
Nochmal

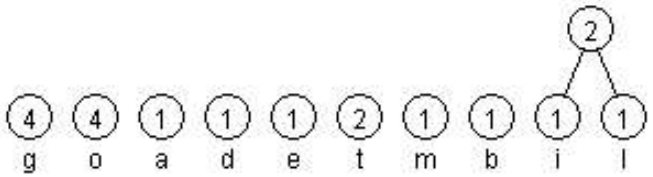
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

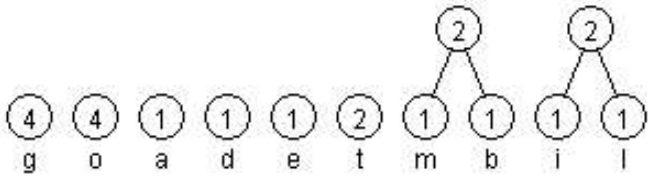
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil

1. Bestimme die Häufigkeit der Symbole Ausführen

2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen

3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)

4. Erzeuge Code Ausführen

Nochmal

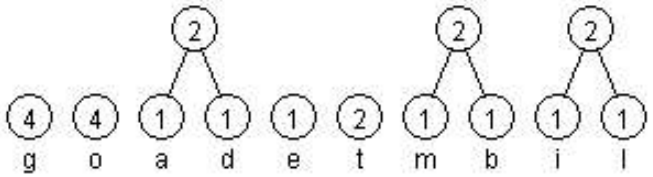
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

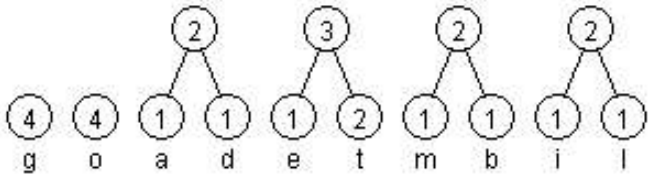
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

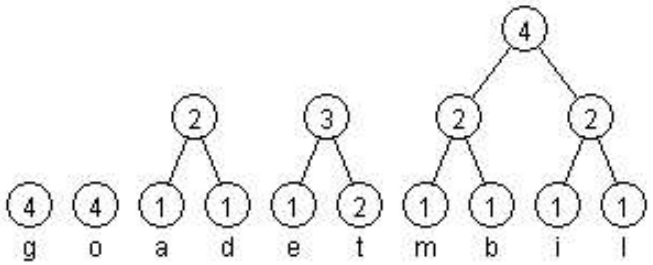
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil

1. Bestimme die Häufigkeit der Symbole Ausführen

2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen

3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)

4. Erzeuge Code Ausführen

Nochmal

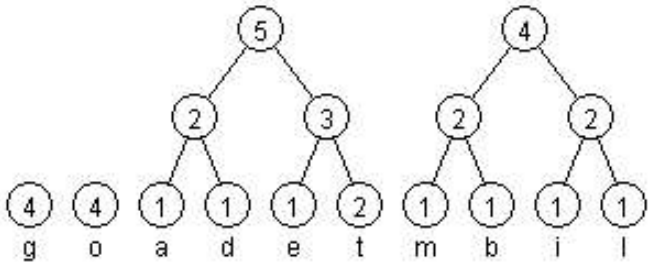
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

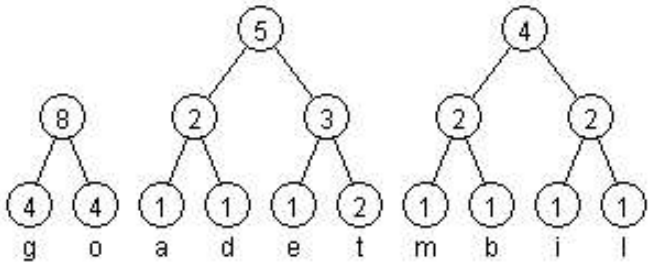
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil

1. Bestimme die Häufigkeit der Symbole Ausführen

2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen

3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)

4. Erzeuge Code Ausführen

Nochmal

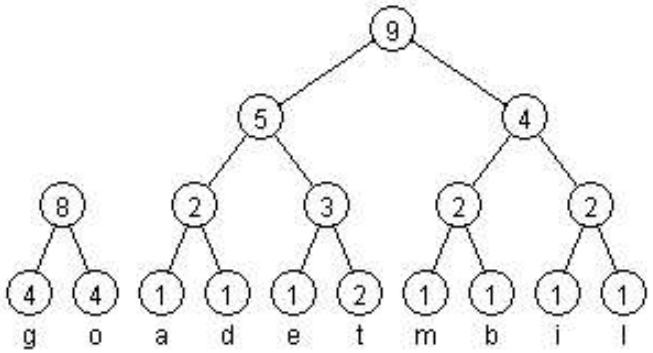
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

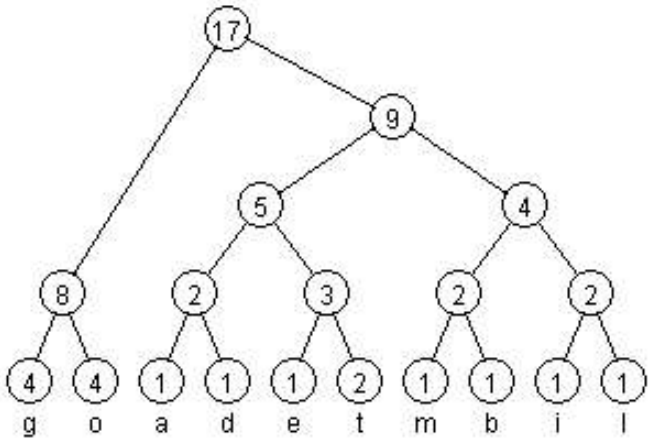
Firefox

Huffman-Code

www.inf.fh-flensbu

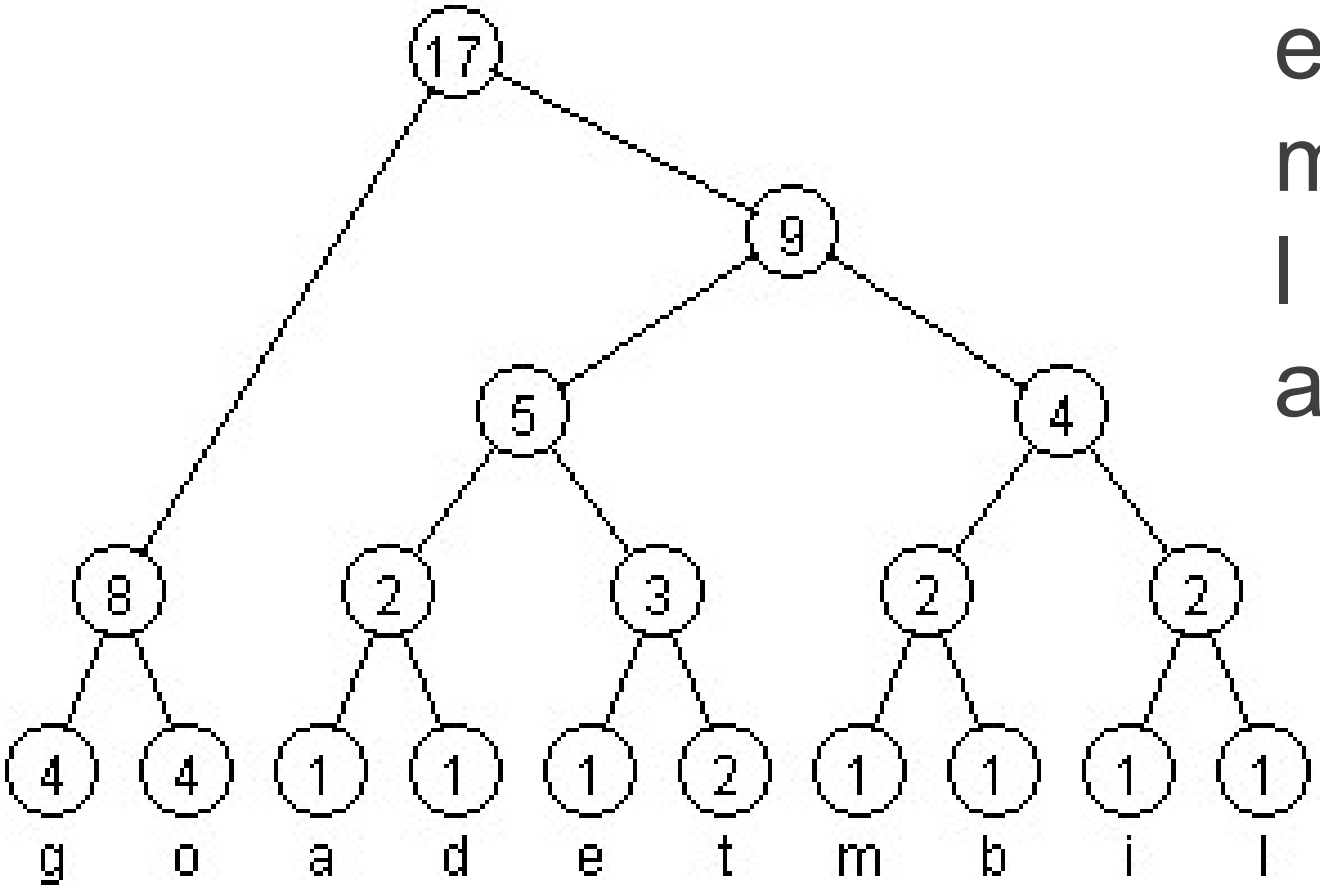
applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil
1. Bestimme die Häufigkeit der Symbole — Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol — Ausführen
3. Wähle zwei Knoten mit minimaler Markierung — (Knoten anklicken)
4. **Erzeuge Code** — Ausführen

Nochmal



e := ?

m := ?

l := ?

a := ?

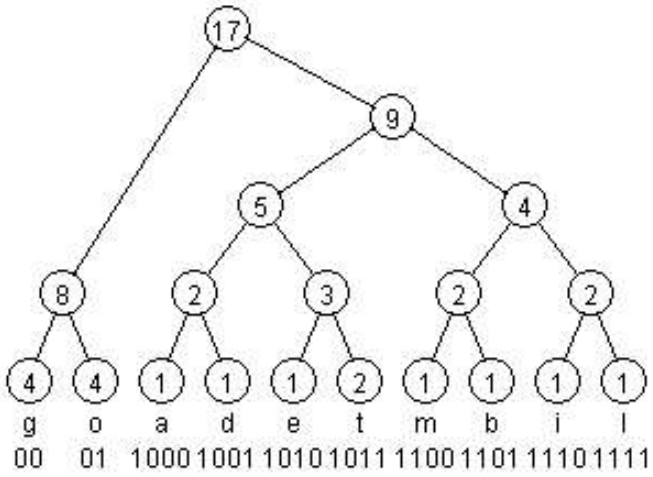
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) gogogadgettomobil

1. Bestimme die Häufigkeit der Symbole Ausführen

2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen

3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)

4. Erzeuge Code Ausführen

Codierung: 000100010010001001001010101110110111000111011111011111

4-Bit-Blockcode: 68 Huffman: 52 Informationsgehalt: 51.5 Kompression: 24.0% Nochmal

Übung

Komprimieren Sie die Zeichenkette „bananarama“ (ohne Anführungszeichen) unter Verwendung des Huffman-Algorithmus.

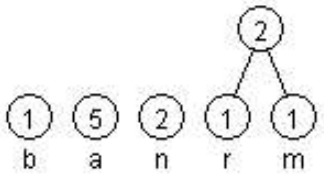
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) bananarama
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

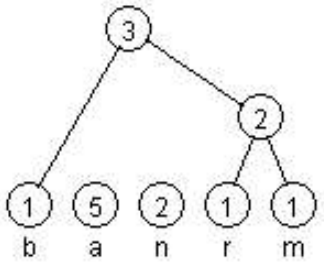
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) bananarama
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

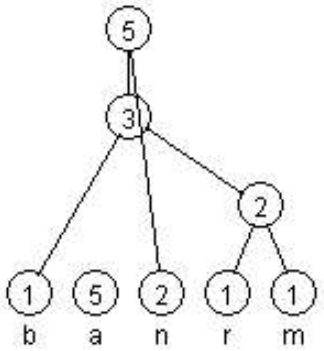
Firefox

Huffman-Code

www.inf.fh-flensbu

applet huf

Simulation



0. Zu codierender Text (max. 25 Zeichen) bananarama
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. **Wähle zwei Knoten mit minimaler Markierung** (Knoten anklicken)
4. Erzeuge Code Ausführen

Nochmal

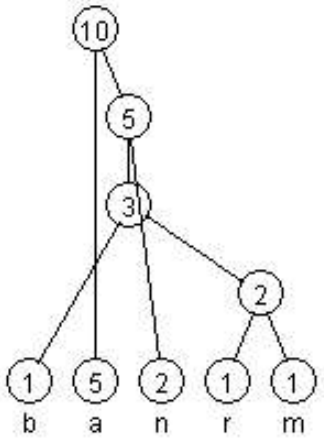
Firefox

Huffman-Code

www.inf.fh-flensbu

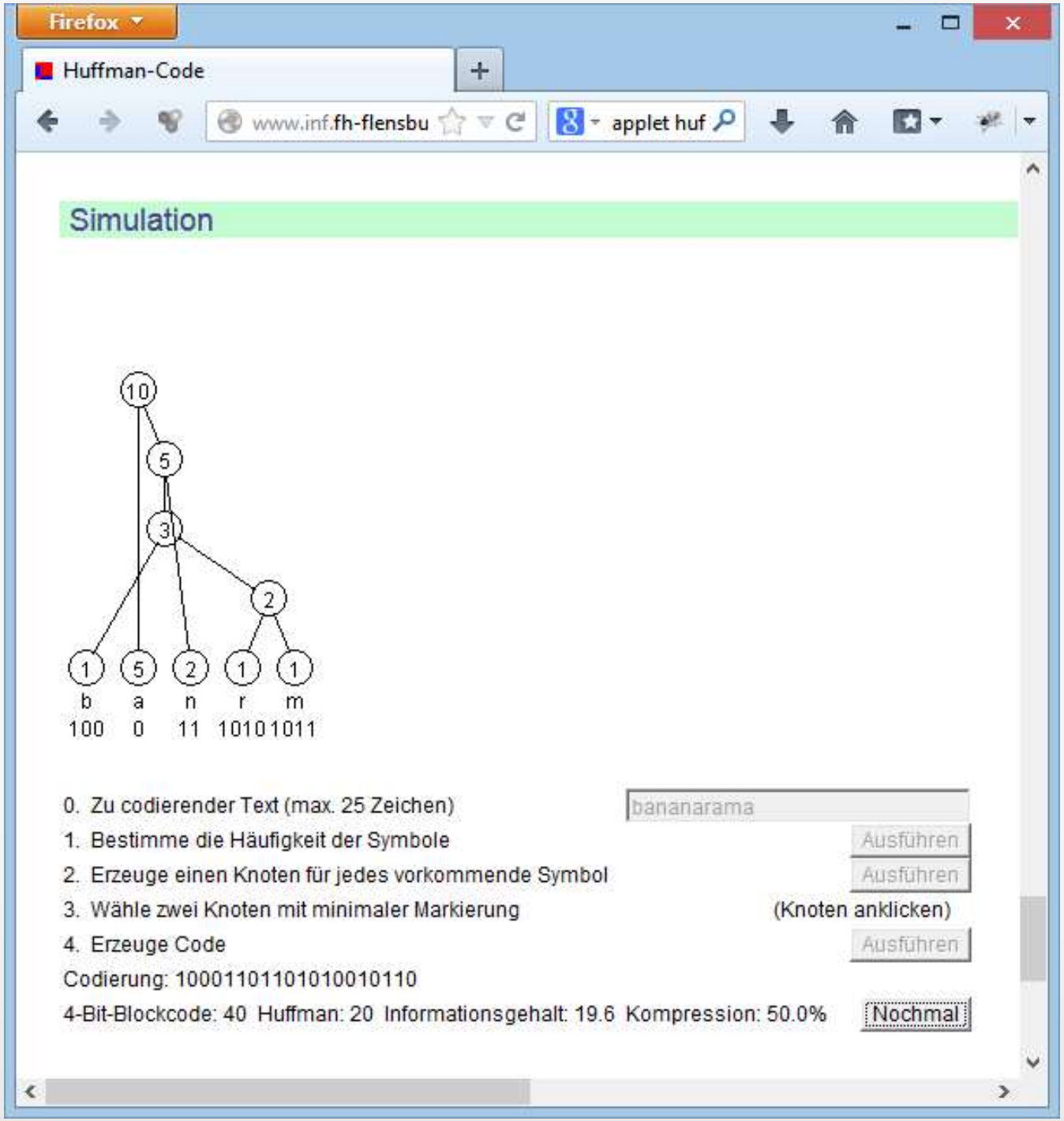
applet huf

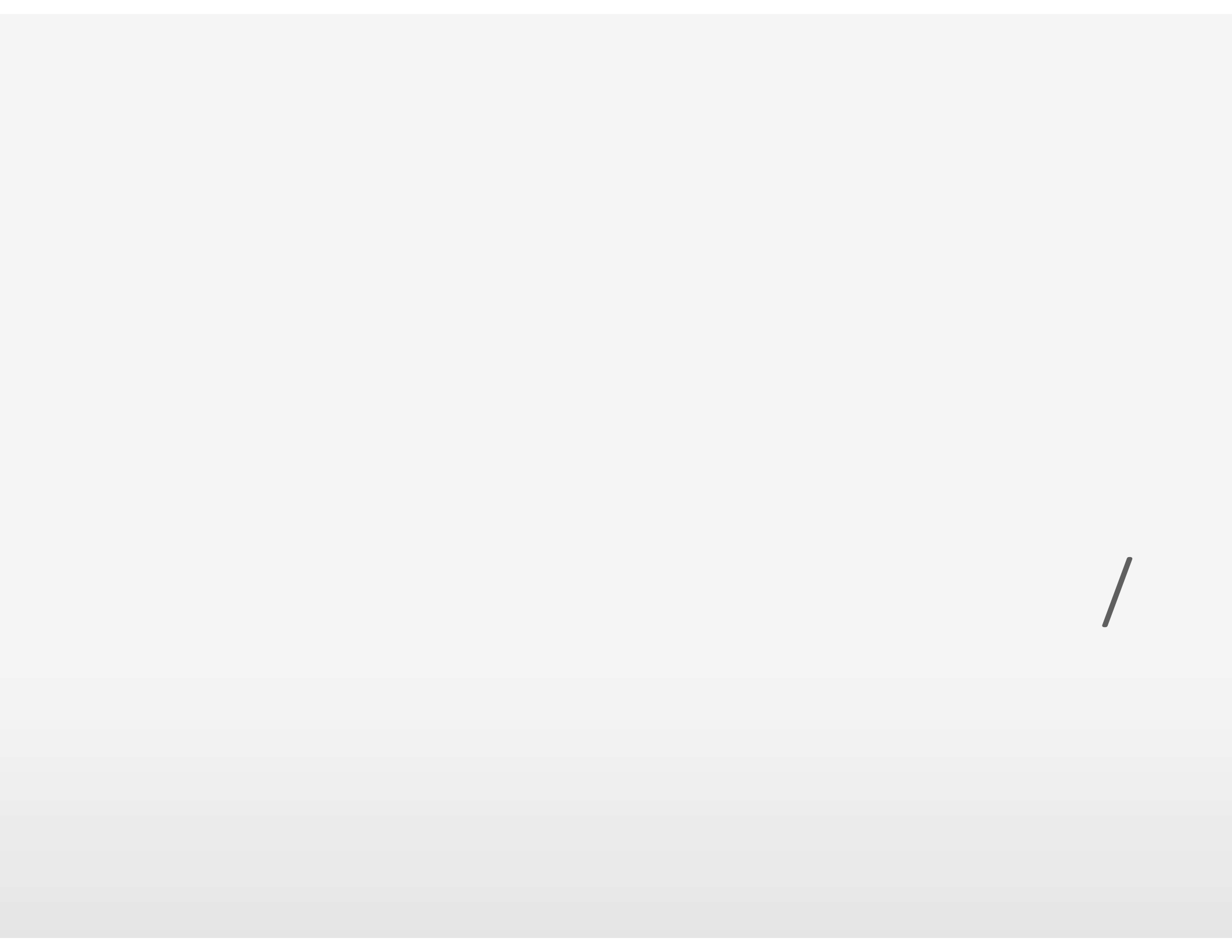
Simulation



0. Zu codierender Text (max. 25 Zeichen) bananarama
1. Bestimme die Häufigkeit der Symbole Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol Ausführen
3. Wähle zwei Knoten mit minimaler Markierung (Knoten anklicken)
4. **Erzeuge Code** Ausführen

Nochmal


Firefox
Huffman-Code
www.inf.fh-flensbu
applet huf
Simulation
10
5
3
2
1
5
2
1
1
b
a
n
r
m
100
0
11
1010
1011
0. Zu codierender Text (max. 25 Zeichen)
bananarama
1. Bestimme die Häufigkeit der Symbole
Ausführen
2. Erzeuge einen Knoten für jedes vorkommende Symbol
Ausführen
3. Wähle zwei Knoten mit minimaler Markierung
(Knoten anklicken)
4. Erzeuge Code
Ausführen
Codierung: 10001101101010010110
4-Bit-Blockcode: 40 Huffman: 20 Informationsgehalt: 19.6 Kompression: 50.0%
Nochmal



Bildnachweise

- https://commons.wikimedia.org/wiki/File:Universitat_zu_Koln_Hauptgebaude_ost.jpg
- http://causeitsallaboutthepayno.tumblr.com/post/131746453874/im-currently-listening-to-adeles-new
- www.giphy.com