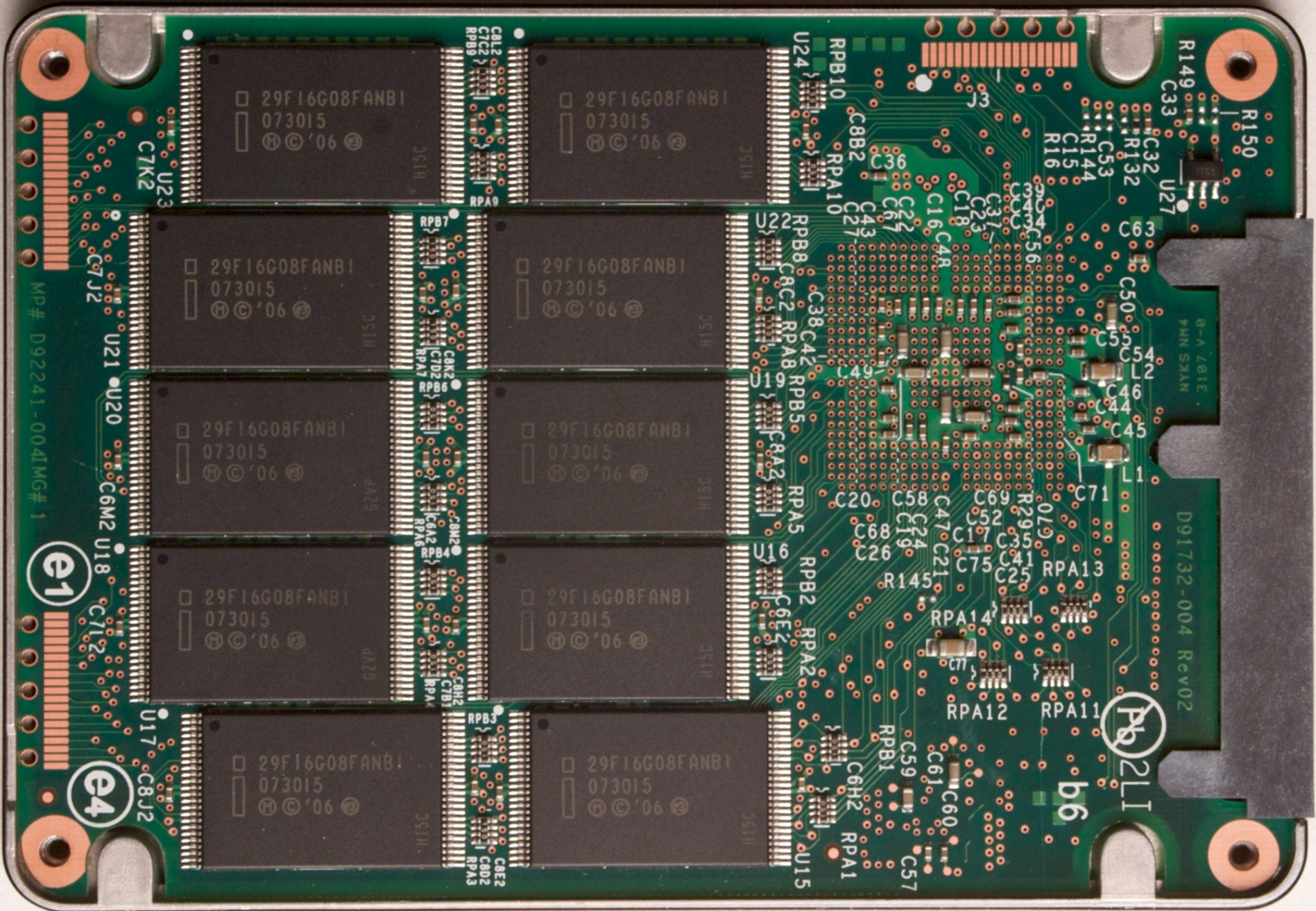


Beyond Data Structures

Huffman Encoding





R150
R149 C33 U27
R132 R133
R144 R145
R16

8-A 481E
4-NM SKAN

D91732-004 Rev02

Pb 2LI

b6

C32 C53 C154
C15 C155
C16 C17 C18 C19 C20 C21 C22 C23 C24 C25 C26 C27 C28 C29 C30 C31 C32 C33 C34 C35 C36 C37 C38 C39 C40 C41 C42 C43 C44 C45 C46 C47 C48 C49 C50 C51 C52 C53 C54 C55 C56 C57 C58 C59 C60 C61 C62 C63 C64 C65 C66 C67 C68 C69 C70 C71 C72 C73 C74 C75 C76 C77 C78 C79 C80 C81 C82 C83 C84 C85 C86 C87 C88 C89 C90 C91 C92 C93 C94 C95 C96 C97 C98 C99 C100
R145 R146 R147 R148 R149 R150 R151 R152 R153 R154 R155 R156 R157 R158 R159 R160 R161 R162 R163 R164 R165 R166 R167 R168 R169 R170 R171 R172 R173 R174 R175 R176 R177 R178 R179 R180 R181 R182 R183 R184 R185 R186 R187 R188 R189 R190 R191 R192 R193 R194 R195 R196 R197 R198 R199 R200 R201 R202 R203 R204 R205 R206 R207 R208 R209 R210 R211 R212 R213 R214 R215 R216 R217 R218 R219 R220 R221 R222 R223 R224 R225 R226 R227 R228 R229 R230 R231 R232 R233 R234 R235 R236 R237 R238 R239 R240 R241 R242 R243 R244 R245 R246 R247 R248 R249 R250 R251 R252 R253 R254 R255 R256 R257 R258 R259 R260 R261 R262 R263 R264 R265 R266 R267 R268 R269 R270 R271 R272 R273 R274 R275 R276 R277 R278 R279 R280 R281 R282 R283 R284 R285 R286 R287 R288 R289 R290 R291 R292 R293 R294 R295 R296 R297 R298 R299 R300 R301 R302 R303 R304 R305 R306 R307 R308 R309 R310 R311 R312 R313 R314 R315 R316 R317 R318 R319 R320 R321 R322 R323 R324 R325 R326 R327 R328 R329 R330 R331 R332 R333 R334 R335 R336 R337 R338 R339 R340 R341 R342 R343 R344 R345 R346 R347 R348 R349 R350 R351 R352 R353 R354 R355 R356 R357 R358 R359 R360 R361 R362 R363 R364 R365 R366 R367 R368 R369 R370 R371 R372 R373 R374 R375 R376 R377 R378 R379 R380 R381 R382 R383 R384 R385 R386 R387 R388 R389 R390 R391 R392 R393 R394 R395 R396 R397 R398 R399 R400 R401 R402 R403 R404 R405 R406 R407 R408 R409 R410 R411 R412 R413 R414 R415 R416 R417 R418 R419 R420 R421 R422 R423 R424 R425 R426 R427 R428 R429 R430 R431 R432 R433 R434 R435 R436 R437 R438 R439 R440 R441 R442 R443 R444 R445 R446 R447 R448 R449 R450 R451 R452 R453 R454 R455 R456 R457 R458 R459 R460 R461 R462 R463 R464 R465 R466 R467 R468 R469 R470 R471 R472 R473 R474 R475 R476 R477 R478 R479 R480 R481 R482 R483 R484 R485 R486 R487 R488 R489 R490 R491 R492 R493 R494 R495 R496 R497 R498 R499 R500 R501 R502 R503 R504 R505 R506 R507 R508 R509 R510 R511 R512 R513 R514 R515 R516 R517 R518 R519 R520 R521 R522 R523 R524 R525 R526 R527 R528 R529 R530 R531 R532 R533 R534 R535 R536 R537 R538 R539 R540 R541 R542 R543 R544 R545 R546 R547 R548 R549 R550 R551 R552 R553 R554 R555 R556 R557 R558 R559 R560 R561 R562 R563 R564 R565 R566 R567 R568 R569 R570 R571 R572 R573 R574 R575 R576 R577 R578 R579 R580 R581 R582 R583 R584 R585 R586 R587 R588 R589 R590 R591 R592 R593 R594 R595 R596 R597 R598 R599 R600 R601 R602 R603 R604 R605 R606 R607 R608 R609 R610 R611 R612 R613 R614 R615 R616 R617 R618 R619 R620 R621 R622 R623 R624 R625 R626 R627 R628 R629 R630 R631 R632 R633 R634 R635 R636 R637 R638 R639 R640 R641 R642 R643 R644 R645 R646 R647 R648 R649 R650 R651 R652 R653 R654 R655 R656 R657 R658 R659 R660 R661 R662 R663 R664 R665 R666 R667 R668 R669 R670 R671 R672 R673 R674 R675 R676 R677 R678 R679 R680 R681 R682 R683 R684 R685 R686 R687 R688 R689 R690 R691 R692 R693 R694 R695 R696 R697 R698 R699 R700 R701 R702 R703 R704 R705 R706 R707 R708 R709 R710 R711 R712 R713 R714 R715 R716 R717 R718 R719 R720 R721 R722 R723 R724 R725 R726 R727 R728 R729 R730 R731 R732 R733 R734 R735 R736 R737 R738 R739 R740 R741 R742 R743 R744 R745 R746 R747 R748 R749 R750 R751 R752 R753 R754 R755 R756 R757 R758 R759 R760 R761 R762 R763 R764 R765 R766 R767 R768 R769 R770 R771 R772 R773 R774 R775 R776 R777 R778 R779 R780 R781 R782 R783 R784 R785 R786 R787 R788 R789 R790 R791 R792 R793 R794 R795 R796 R797 R798 R799 R800 R801 R802 R803 R804 R805 R806 R807 R808 R809 R810 R811 R812 R813 R814 R815 R816 R817 R818 R819 R820 R821 R822 R823 R824 R825 R826 R827 R828 R829 R830 R831 R832 R833 R834 R835 R836 R837 R838 R839 R840 R841 R842 R843 R844 R845 R846 R847 R848 R849 R850 R851 R852 R853 R854 R855 R856 R857 R858 R859 R860 R861 R862 R863 R864 R865 R866 R867 R868 R869 R870 R871 R872 R873 R874 R875 R876 R877 R878 R879 R880 R881 R882 R883 R884 R885 R886 R887 R888 R889 R890 R891 R892 R893 R894 R895 R896 R897 R898 R899 R900 R901 R902 R903 R904 R905 R906 R907 R908 R909 R910 R911 R912 R913 R914 R915 R916 R917 R918 R919 R920 R921 R922 R923 R924 R925 R926 R927 R928 R929 R930 R931 R932 R933 R934 R935 R936 R937 R938 R939 R940 R941 R942 R943 R944 R945 R946 R947 R948 R949 R950 R951 R952 R953 R954 R955 R956 R957 R958 R959 R960 R961 R962 R963 R964 R965 R966 R967 R968 R969 R970 R971 R972 R973 R974 R975 R976 R977 R978 R979 R980 R981 R982 R983 R984 R985 R986 R987 R988 R989 R990 R991 R992 R993 R994 R995 R996 R997 R998 R999 R1000

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

29F16G08FANBI
073015
© '06

U23
C7K2

U21
C7J2

U20
C6M2

U18
C7L2

U17
C8J2

MP# D92241-004IMG# 1

e1

e4

It's All Bits and Bytes

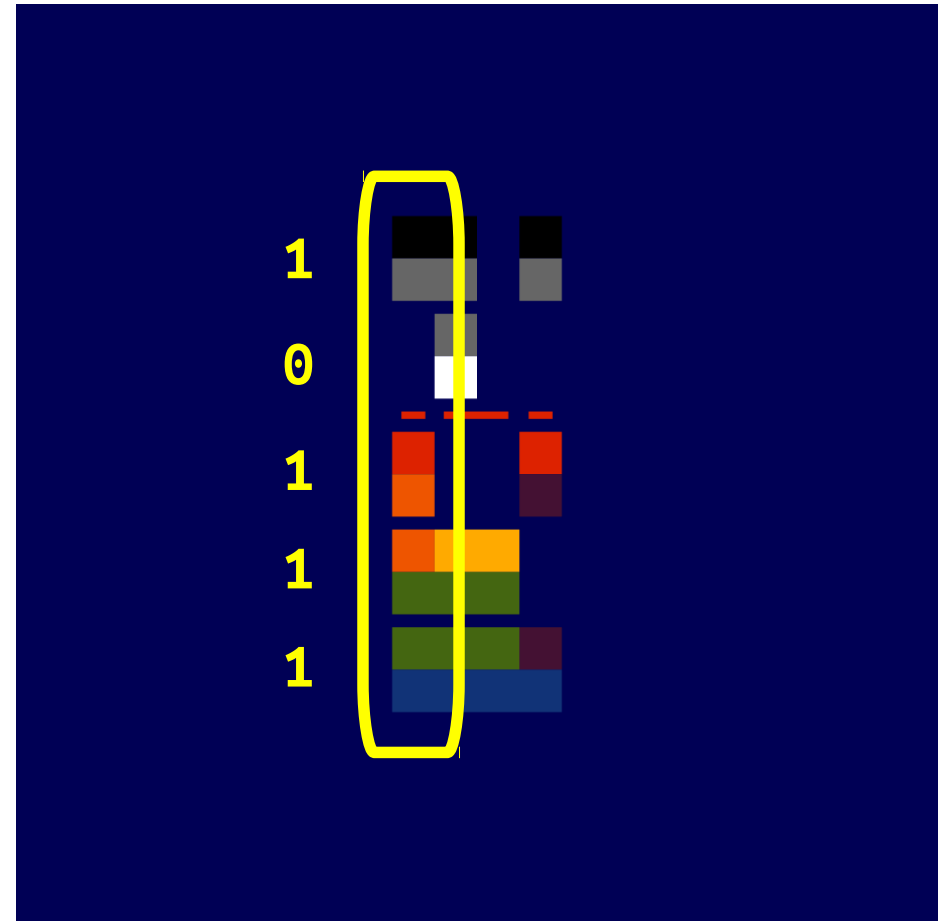
- Data stored on disk consists of 0s and 1s.
 - Usually encoded by magnetic orientation on small (10nm!) metal particles or by trapping electrons in small gates.
- A single 0 or 1 is called a **bit**.
- A group of eight bits is called a **byte**.
00000000, 00000001, 00000010, 00000011,
00000100, 00000101, 00000110, ...
- There are $2^8 = 256$ different bytes.
 - **Great recursion practice:** Write a function to list all of them!

Representing Text

- Text uses all sorts of different characters.
- Computers require everything to be written as zeros and ones.
- To represent text inside the computer, we can choose some way of mapping characters to series of zeros and ones and vice-versa.
- There are many ways to do this.

Baudot Codes

- The ***Baudot code*** was one of the first codes for representing text as 0s and 1s.
- It was used by early teleprinters and, while mostly obsoleted, is still around.
- Coldplay's album ***X&Y*** used it for their Hip and Stylish album cover.



ASCII

- Early (American) computers needed some standard way to send output to their (physical!) printers.
- Since there were fewer than 256 different characters to print (1960's America!), each character was assigned a one-byte value.
- This initial code was called **ASCII**. Surprisingly, it's still around, though in a modified form (more on that later).
- For example, the letter A is represented by the byte 01000001 (65). You can still see this in C++:

```
cout << int('A') << endl; // Prints 65
```


ASCII

- In ASCII:
 - Each character is *exactly one byte* (8 bits).
 - All computers agree *in advance* which characters have which values.
- This makes it possible to transmit text data from one computer to another by just writing out a series of bits.
- Here's what this might look like:

Transmitting the Dikdik

K	01001011
I	01001001
R	01010010
K	01001011
'	00100111
S	01010011
	00100000
D	01000100
I	01001001
K	01001011
D	01000100
I	01001001
K	01001011



Transmitting the Dikdik

K	01001011
I	01001001
R	01010010
K	01001011
'	00100111
S	01010011
	00100000
D	01000100
I	01001001
K	01001011
D	01000100
I	01001001
K	01001011

```
01001011010010010101001001001011
00100111010100110010000001000100
01001001010010110100010001001001
01001011
```

Transmitting the Dikdik

```
01001011010010010101001001001011  
00100111010100110010000001000100  
01001001010010110100010001001001  
01001011
```

Transmitting the Dikdik

01001011	01001001	01010010	01001011
00100111	01010011	00100000	01000100
01001001	01001011	01000100	01001001
01001011			

Transmitting the Dikdik

01001011

01001001

01010010

01001011

00100111

01010011

00100000

01000100

01001001

01001011

01000100

01001001

01001011

Transmitting the Dikdik

K	01001011
I	01001001
R	01010010
K	01001011
'	00100111
S	01010011
	00100000
D	01000100
I	01001001
K	01001011
D	01000100
I	01001001
K	01001011

An Observation

- In ASCII, every character has exactly the same number of bits in it.
- Any message with n characters will use up exactly $8n$ bits.
 - Space for **KIRK'S DIKDIK**: 104 bits.
 - Space for **COPYRIGHTABLE**: 104 bits.
- We say that ASCII is a ***fixed-length encoding***.

A Different Encoding

- The phrase **KIRK'S DIKDIK** has exactly 7 different characters in it.
- We can use a different encoding to represent this string using many fewer bits:

A Different Encoding

- The phrase **KIRK'S DIKDIK** has exactly 7 different characters in it.
- We can use a different encoding to represent this string using many fewer bits:

K	000	S	100
I	001		101
R	010	D	110
'	011		

A Different Encoding

- The phrase **KIRK'S DIKDIK** has exactly 7 different characters in it.
- We can use a different encoding to represent this string using many fewer bits:

000	001	010	000	011	100	101	110	001	000	110	001	000
K	I	R	K	'	S		D	I	K	D	I	K

K	000	S	100
I	001		101
R	010	D	110
'	011		

A Different Encoding

- The phrase **KIRK'S DIKDIK** has exactly 7 different characters in it.
- We can use a different encoding to represent this string using many fewer bits:

000	001	010	000	011	100	101	110	001	000	110	001	000
K	I	R	K	'	S		D	I	K	D	I	K

- Down from 104 bits to 39 bits: using 37.5% as much space as before!

K	000	S	100
I	001		101
R	010	D	110
'	011		

The Key Idea

- If we can find a way to
give all characters a bit pattern,
that both the sender and receiver know
about, and
that can be decoded uniquely,
then we can represent the same piece of
text in multiple different ways.
- **Goal:** Find a way to do this that uses *less space* than the standard ASCII representation.

Exploiting Redundancy

- Not all letters have the same frequency in “KIRK'S DIKDIK.”
- Frequency table:

K	4
I	3
D	2
R	1
'	1
S	1
	1

- What if we gave shorter encodings to more common characters?

A First Attempt

K	0
I	1
D	00
R	01
'	10
S	11
	100

0101010111000000100010

0	1	01	0	10	11	100	00	001	0	00	1	0
K	I	R	K	'	S		D	I	K	D	I	K

A First Attempt

K	0
I	1
D	00
R	01
'	10
S	11
	100

0101010111000000100010

A First Attempt

K	0
I	1
D	00
R	01
'	10
S	11
	100

0101010111000000100010

0	1	01	0	10	11	100	00	001	0	00	1	0
K	I	R	K	'	S		D	I	K	D	I	K

A First Attempt

K	0
I	1
D	00
R	01
'	10
S	11
	100

0101010111000000100010

01	01	01	11	00	00	00	100	01	0
R	R	R	S	D	D	D		R	K

A First Attempt

K	0
I	1
D	00
R	01
'	10
S	11
	100



1000000100010

01	01	01	11	00	00	00	100	01	0
R	R	R	S	D	D	D		R	K

The Problem

- If we use a different number of bits for each letter, we can't necessarily uniquely determine the boundaries between letters.
- We need an encoding that makes it possible to determine where one character stops and the next starts.
- Is this possible? If so, how?

Prefix Codes

- A ***prefix code*** is an encoding system in which no code is a prefix of another code.
- For example:

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01	1111	10	001	000	1110	110	01	10	110	01	10
K	I	R	K	'	S		D	I	K	D	I	K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10
K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10
K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10
K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10
K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001**1**11110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

10011111110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001**111**110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

100111110001000111011001101100110

10	01
K	I

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

100111110001000111011001101100110

10	01	1111
K	I	R

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01	1111
K	I	R

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01	1111
K	I	R

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01	1111	10
K	I	R	K

Prefix Codes

- Using this prefix code, we can represent KIRK'S DIKDIK as the sequence

1001111110001000111011001101100110

- This uses just 34 bits, compared to our initial 39.
- Remaining questions:
 - How do you generate a prefix code?
 - And what does any of this have to do with binary trees?

Time-Out for Announcements!

Assignment 5

- Assignment 5 is due on Friday.
 - Want to use a late day? Turn it in on Monday of next week.
- Have questions?
 - Stop by the LaIR!
 - Stop by our office hours!
 - Ask your section leader!
 - Ask a partner!
 - Ask on Piazza!

Ceçi n'est pas une annonce.

K	10
I	01
R	1111
'	001
S	000
	1110
D	110

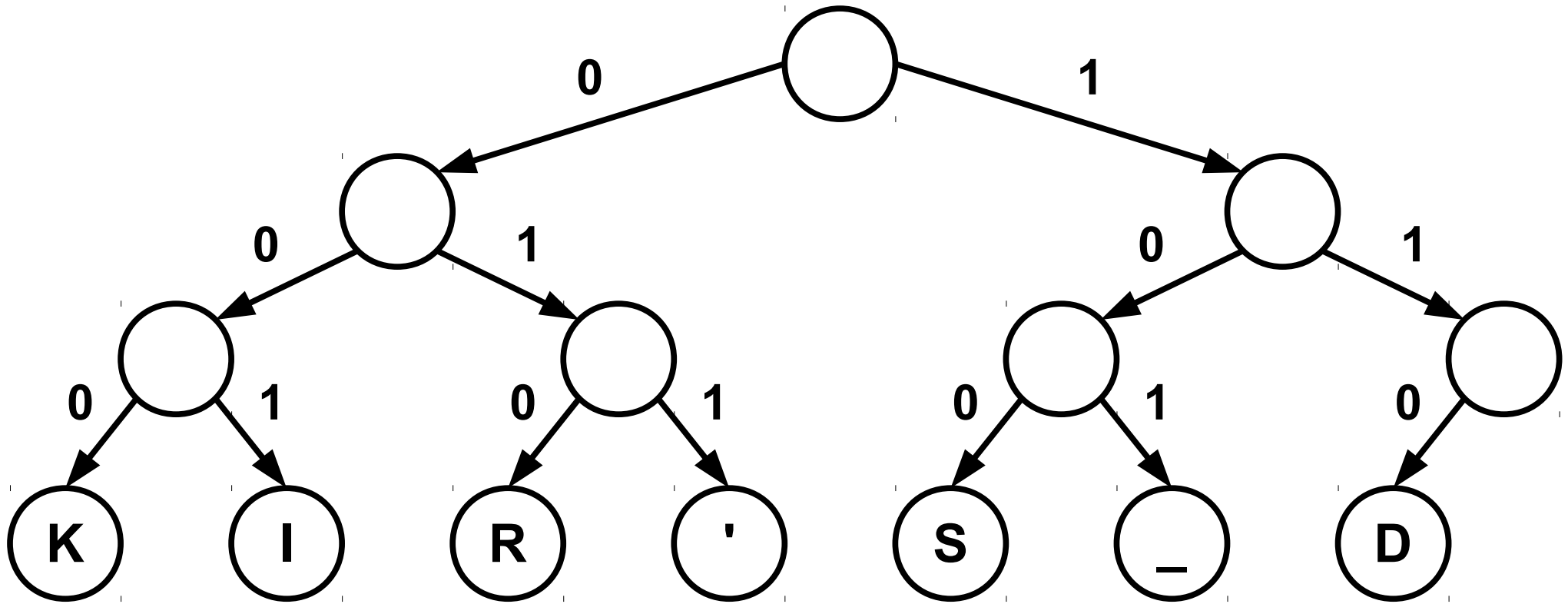
1001111110001000111011001101100110

K	1111110
I	111110
R	11110
'	1110
S	110
	10
D	0

111111011111011110111110111011010011110111111001111101111110

How do you find a “good” prefix code?

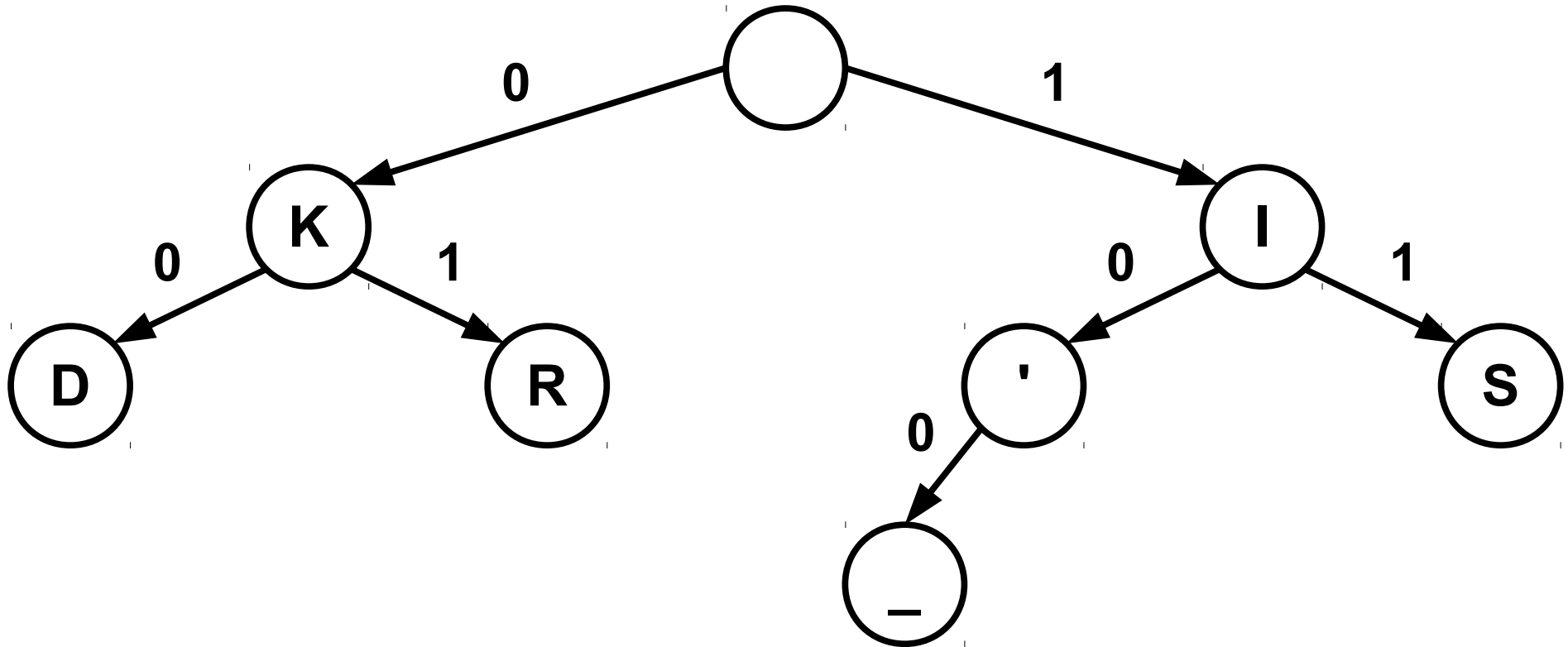
The Key Idea



K	000
I	001
R	010
'	011

S	100
	101
D	110

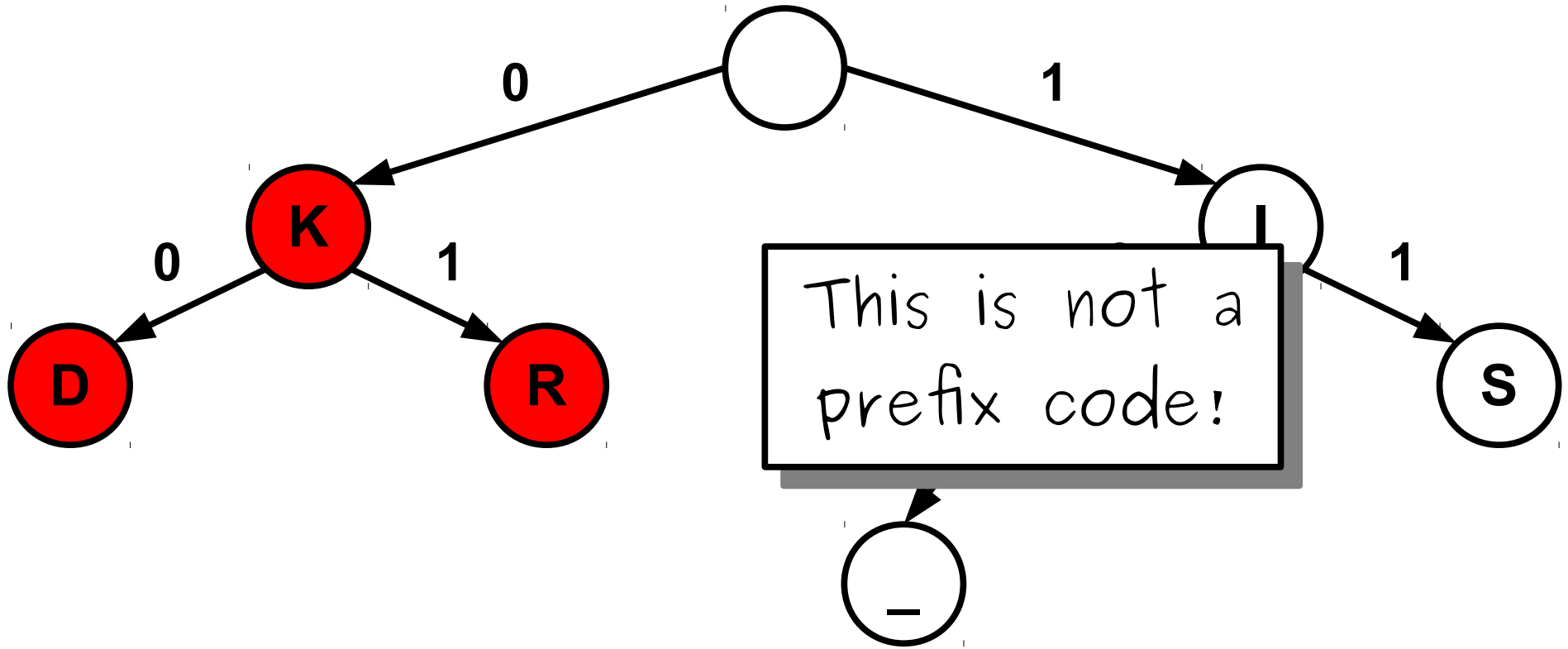
The Key Idea



K	0
I	1
D	00
R	01

'	10
S	11
	100

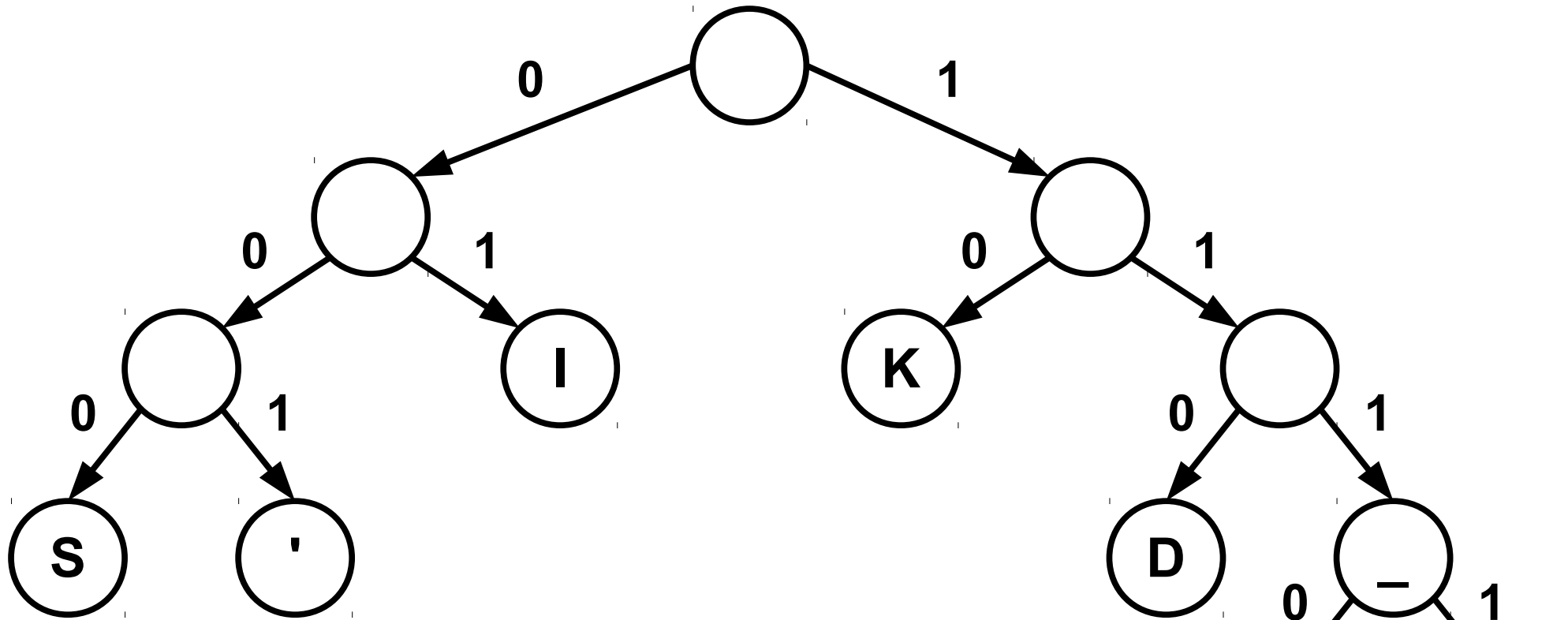
The Key Idea



K	0
I	1
D	00
R	01

'	10
S	11
	100

The Key Idea



K	10
I	01
D	110
R	1111

'	001
S	000
	1110

Finding a good prefix code is equivalent to finding a good binary tree with all values stored at the leaves.

How do we find the best binary tree with
this property?

Huffman Coding

K
4

I
3

D
2

'
1

S
1

R
1

_
1

K	4
I	3
D	2
R	1
'	1
S	1
	1

Huffman Coding

K
4

I
3

D
2

'
1

S
1

R
1

_
1

Huffman Coding

K
4

I
3

D
2

'
1

S
1

R
1

_
1

Huffman Coding

K
4

I
3

D
2

'
1

S
1

R
1

_
1

Huffman Coding

K
4

I
3

D
2

'
1

S
1

R
1

—
1

Huffman Coding

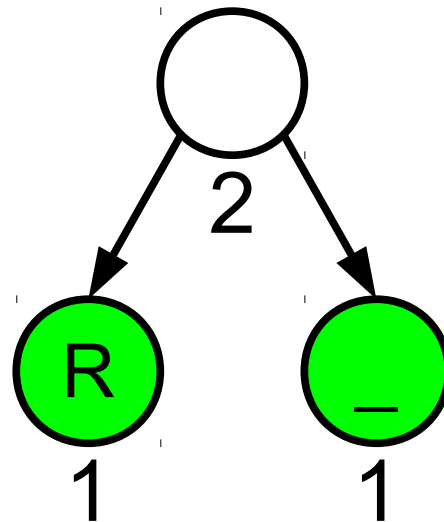
K
4

I
3

D
2

'
1

S
1



Huffman Coding

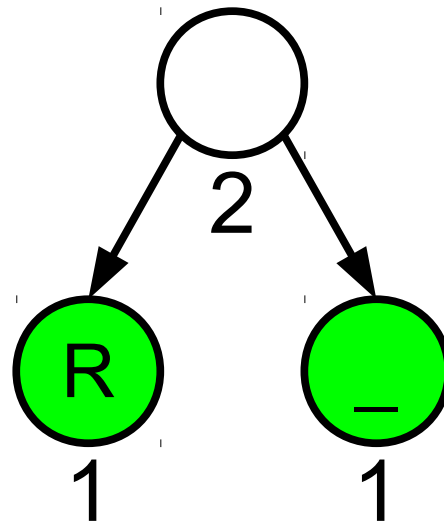
K
4

I
3

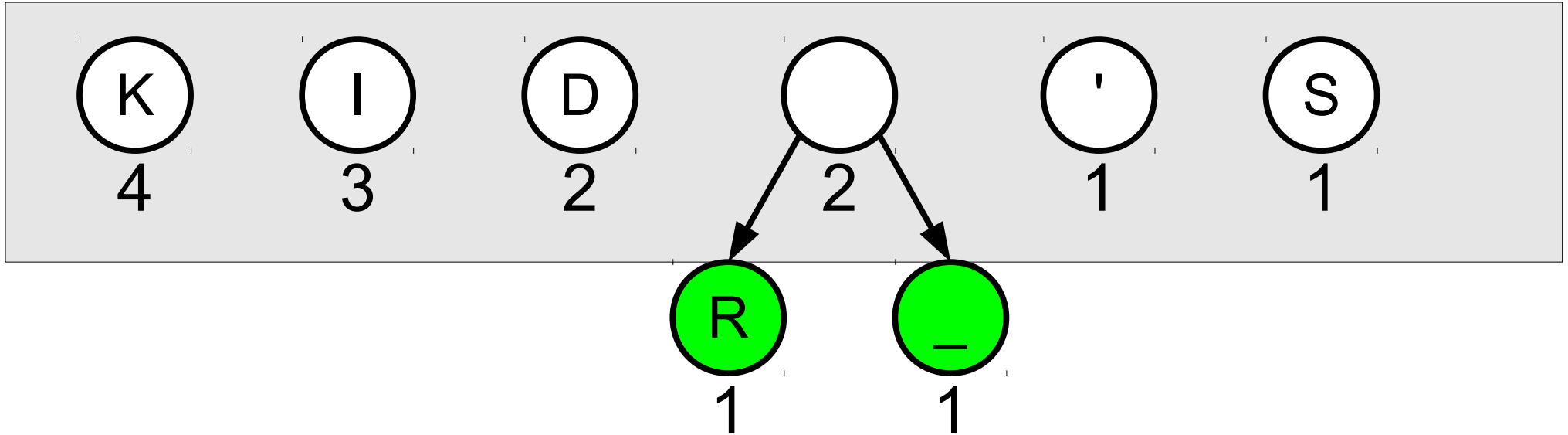
D
2

'
1

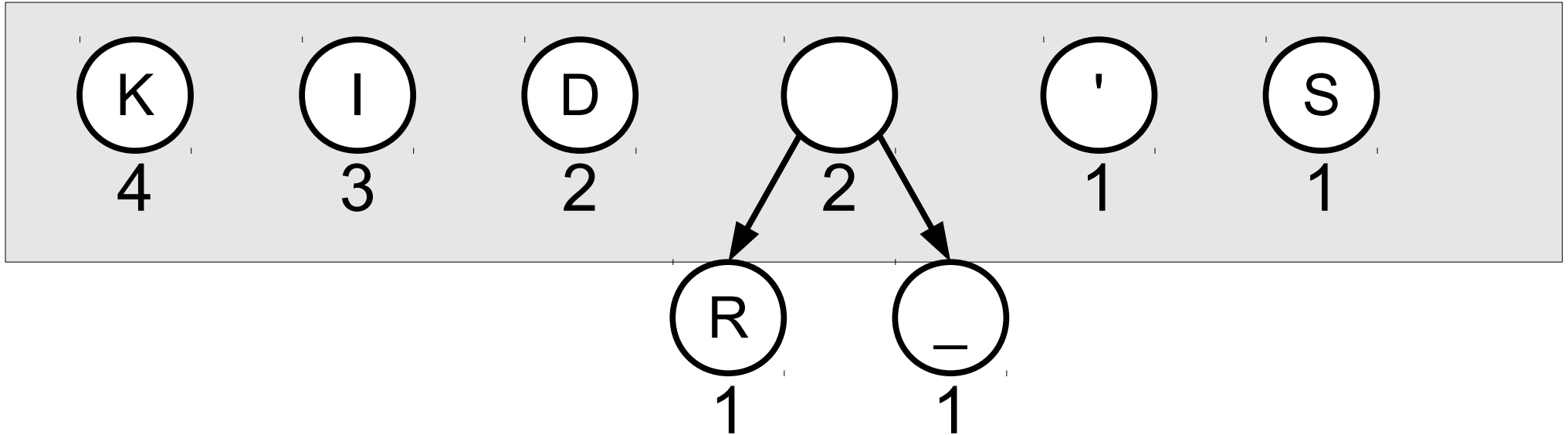
S
1



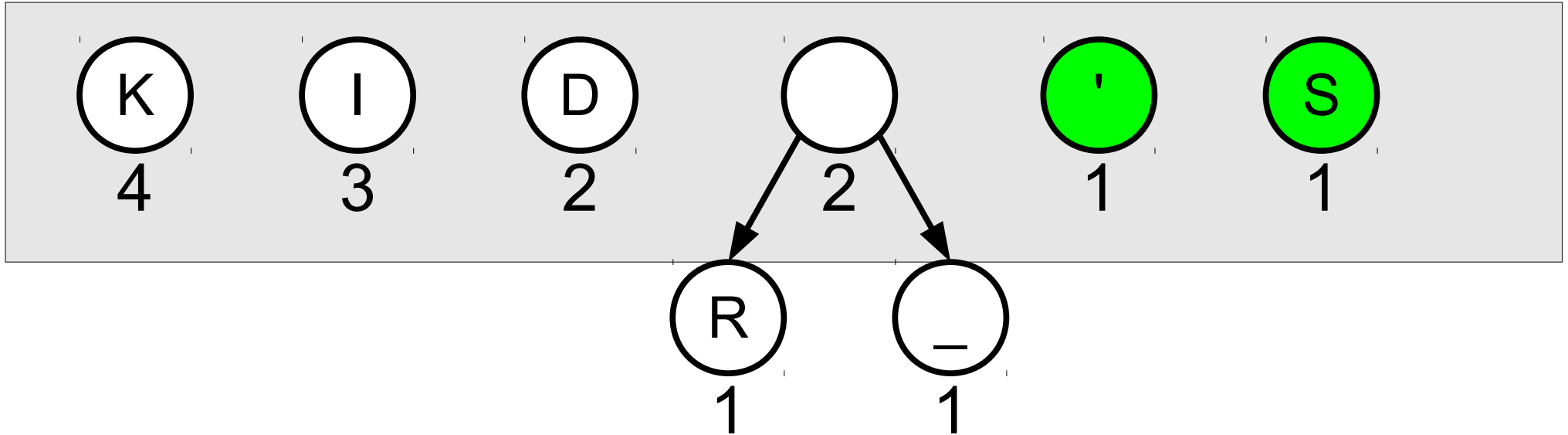
Huffman Coding



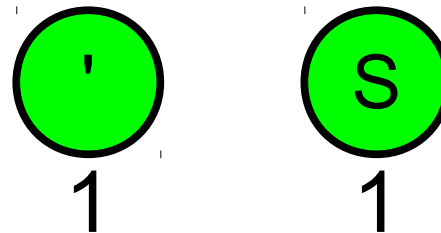
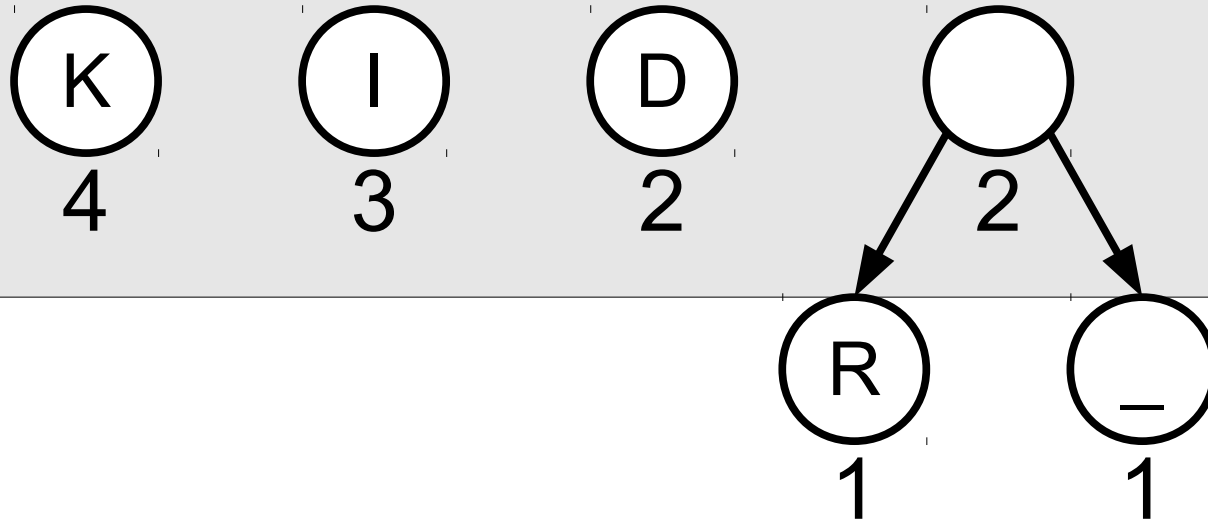
Huffman Coding



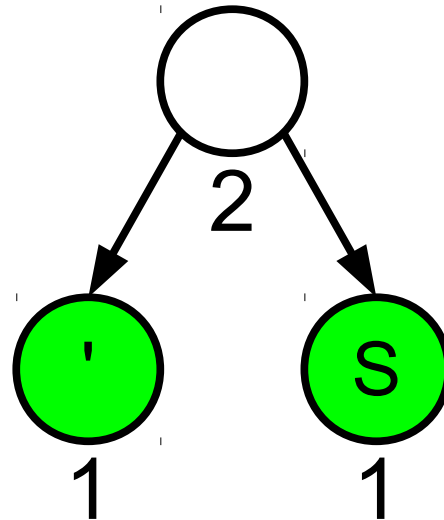
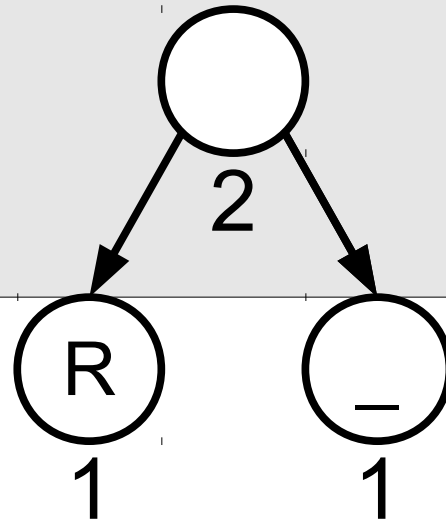
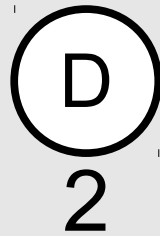
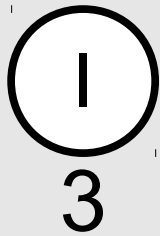
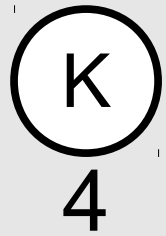
Huffman Coding



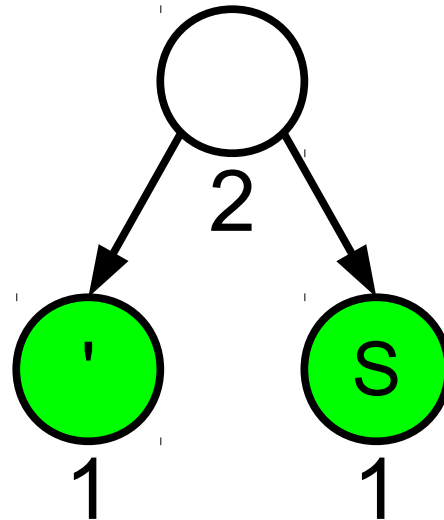
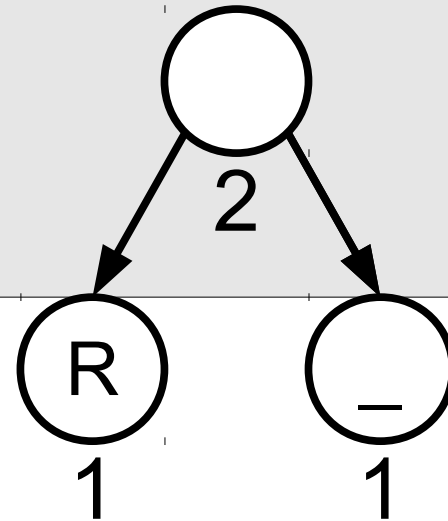
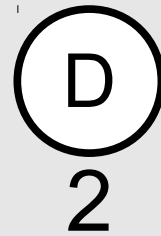
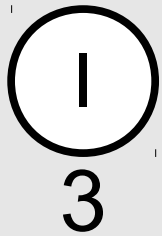
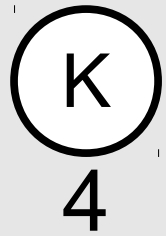
Huffman Coding



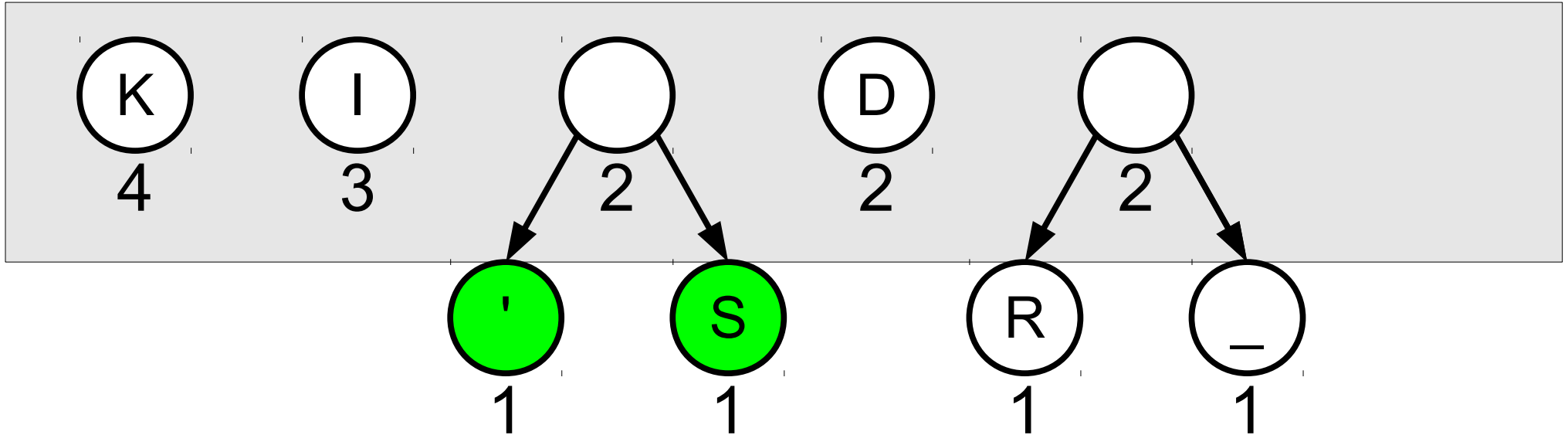
Huffman Coding



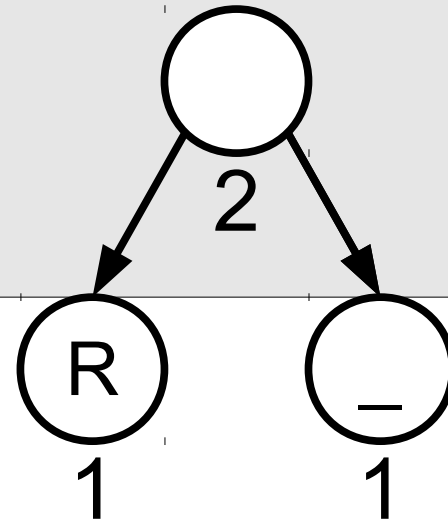
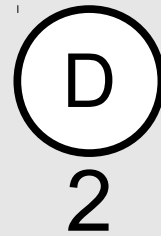
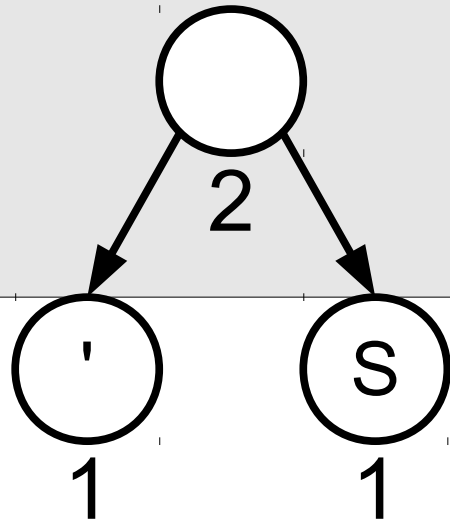
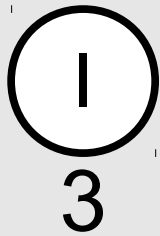
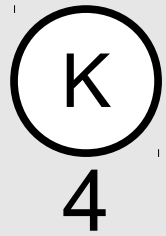
Huffman Coding



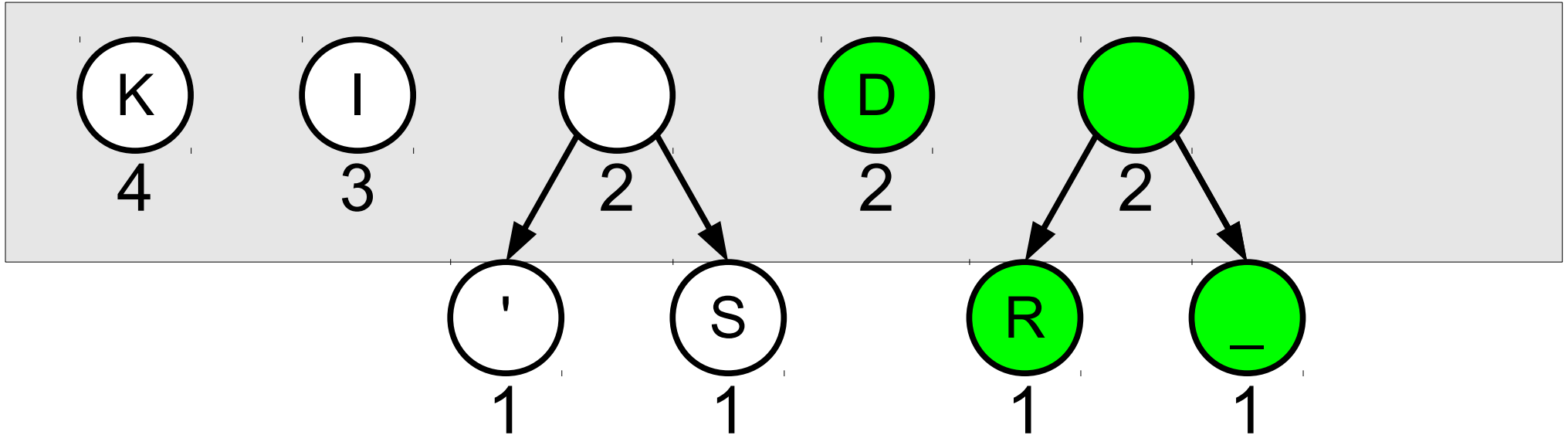
Huffman Coding



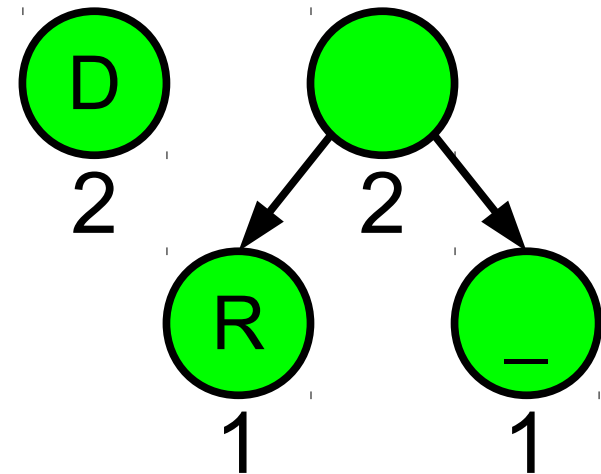
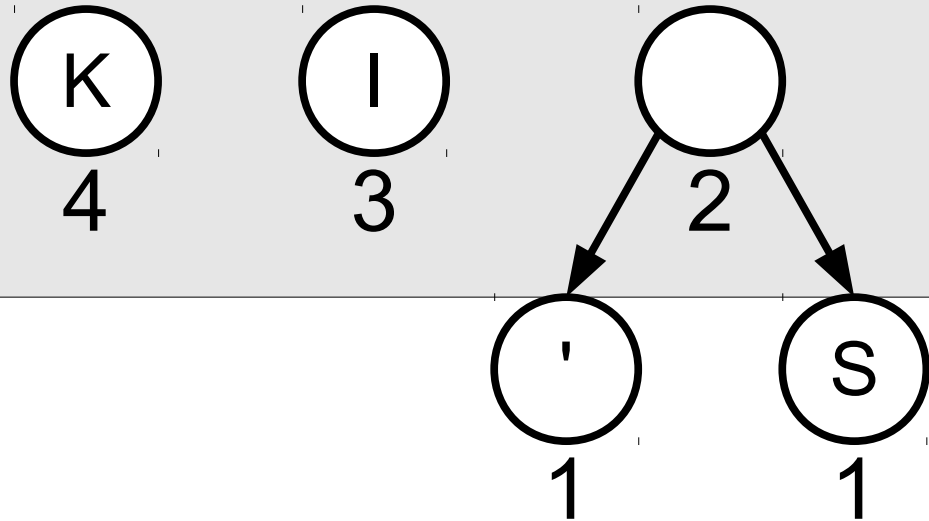
Huffman Coding



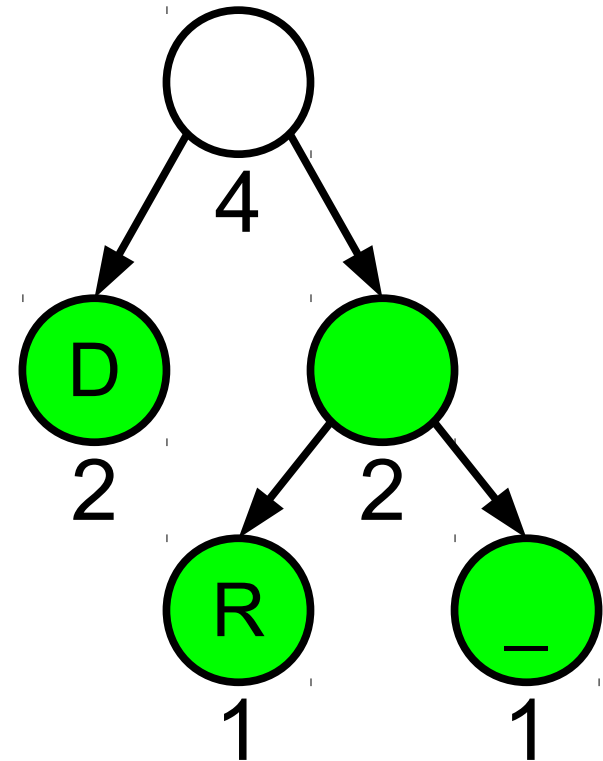
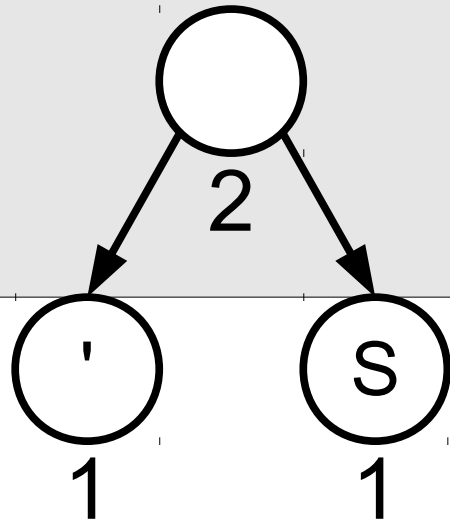
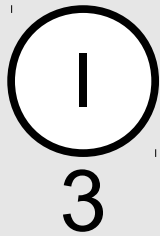
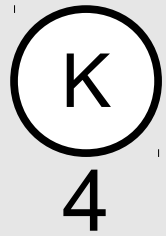
Huffman Coding



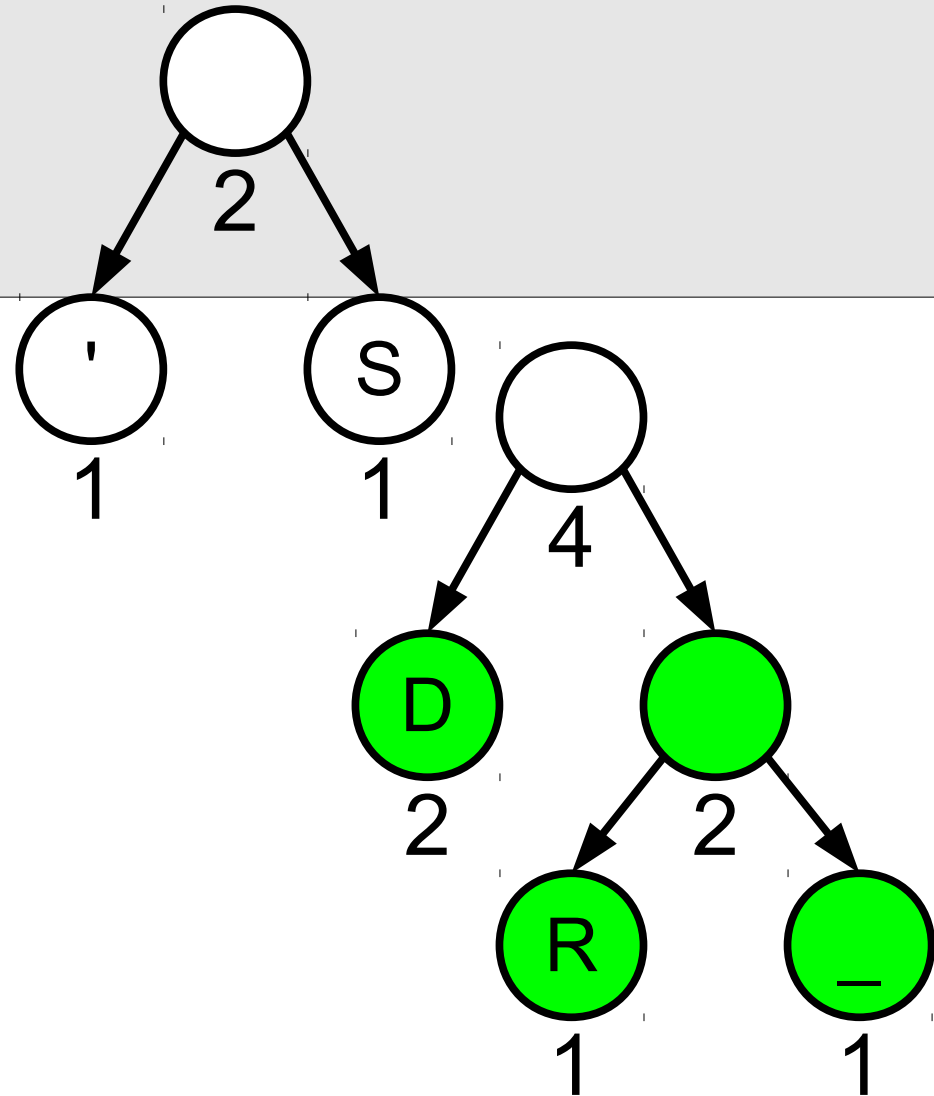
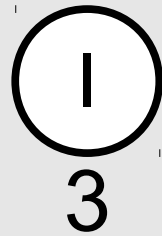
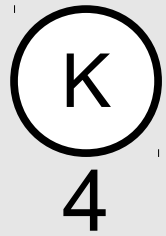
Huffman Coding



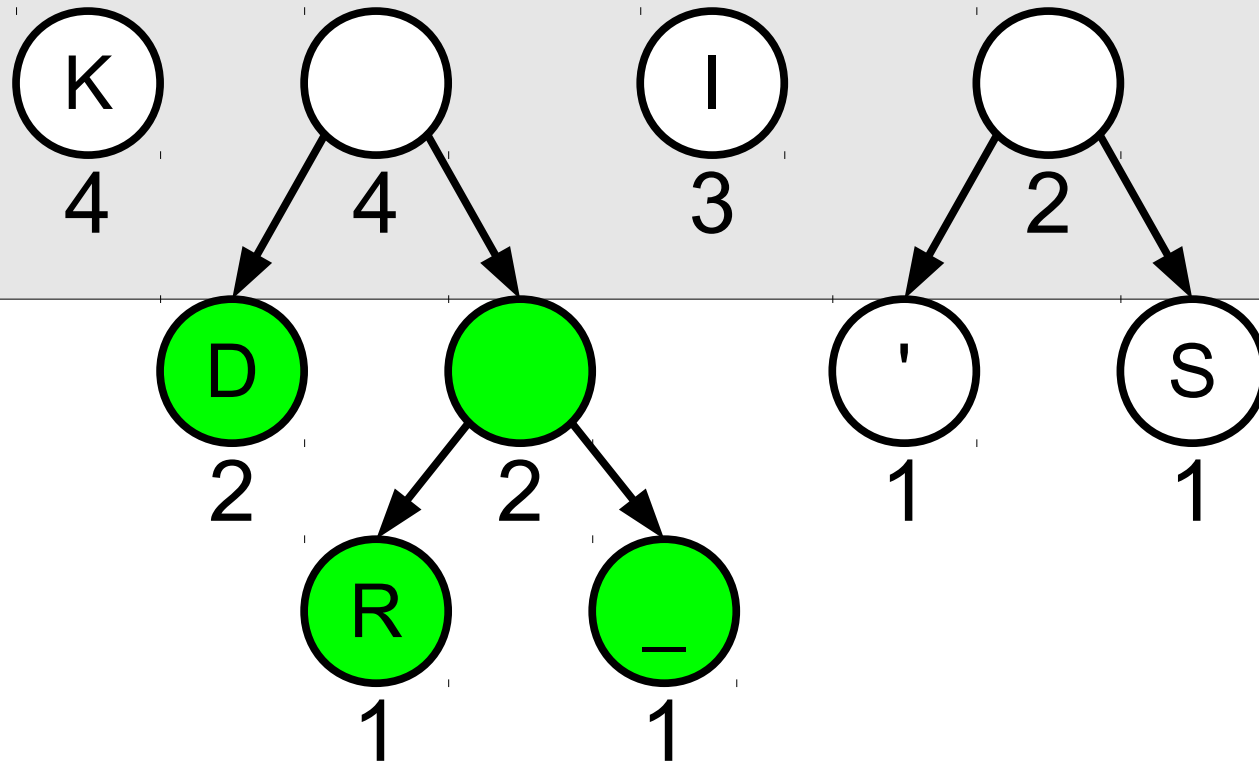
Huffman Coding



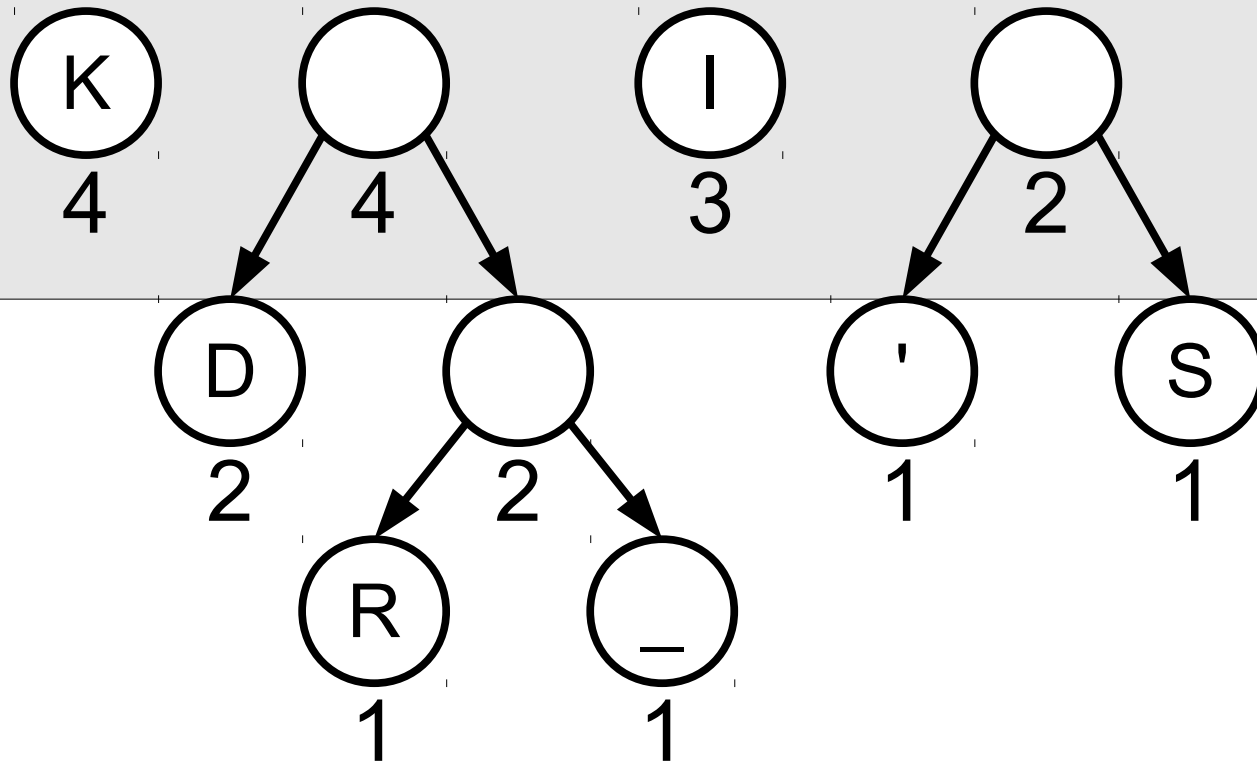
Huffman Coding



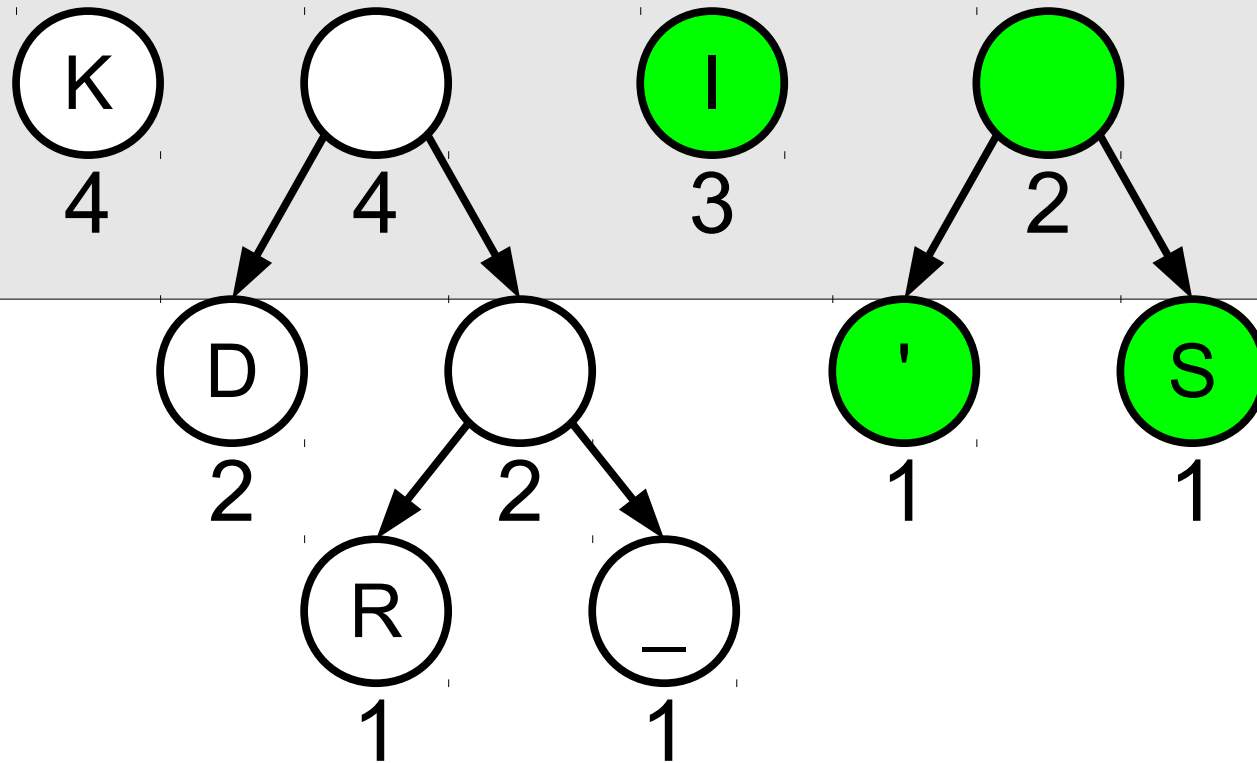
Huffman Coding



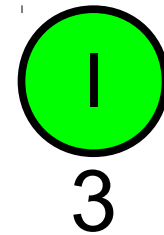
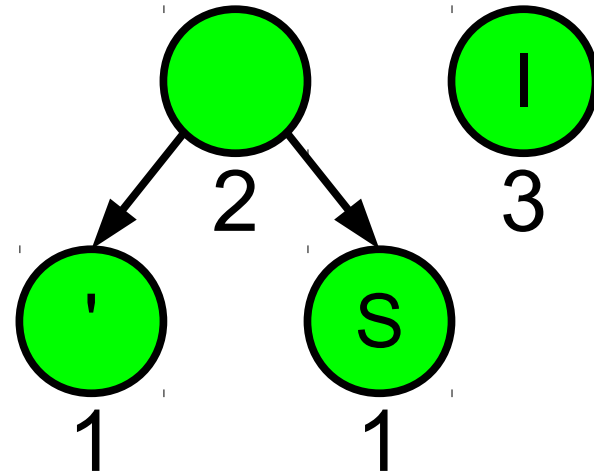
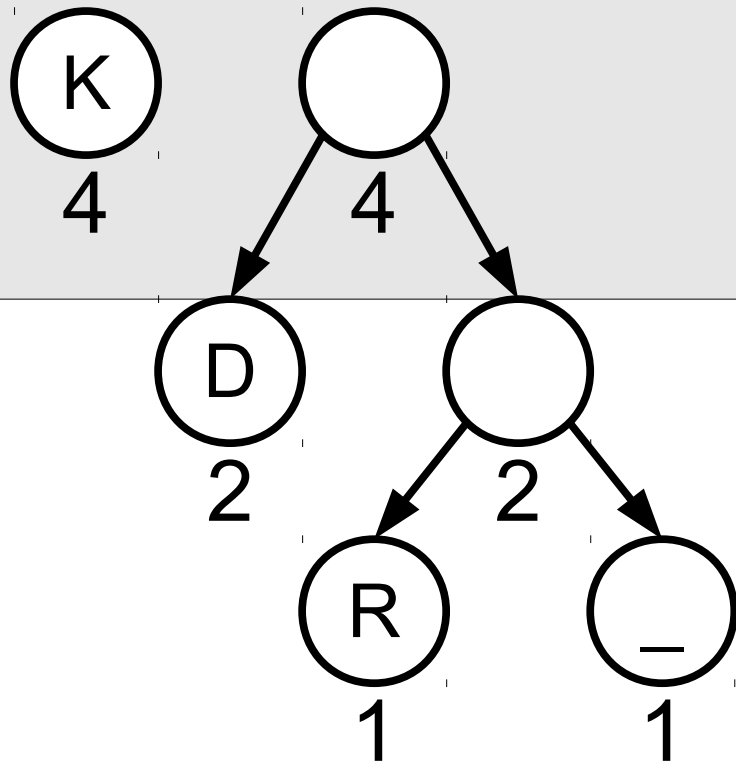
Huffman Coding



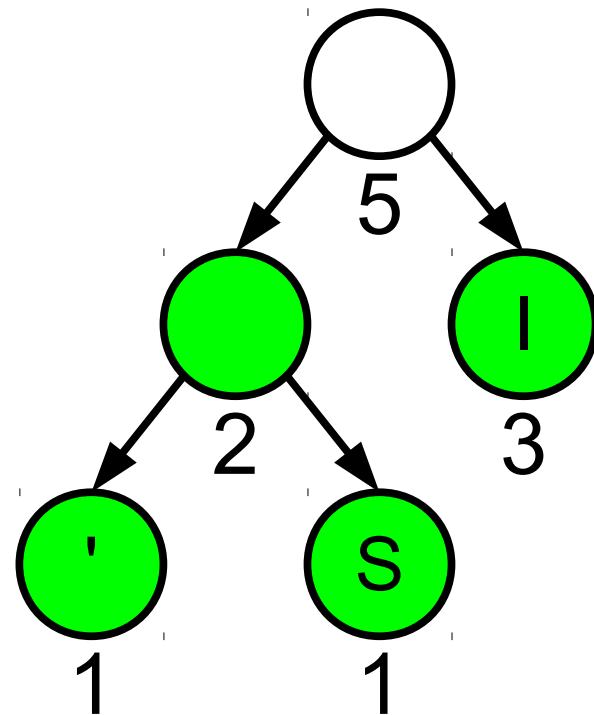
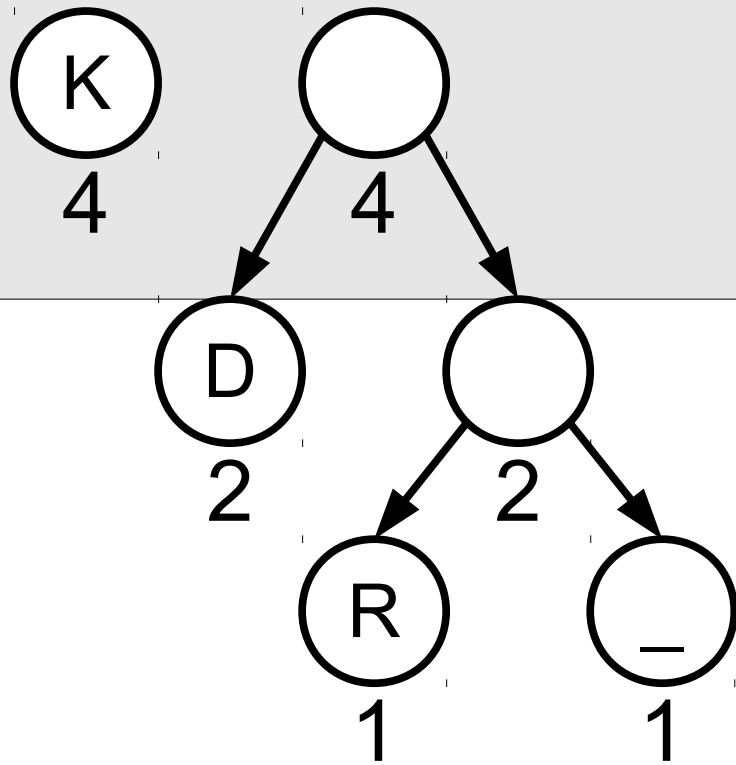
Huffman Coding



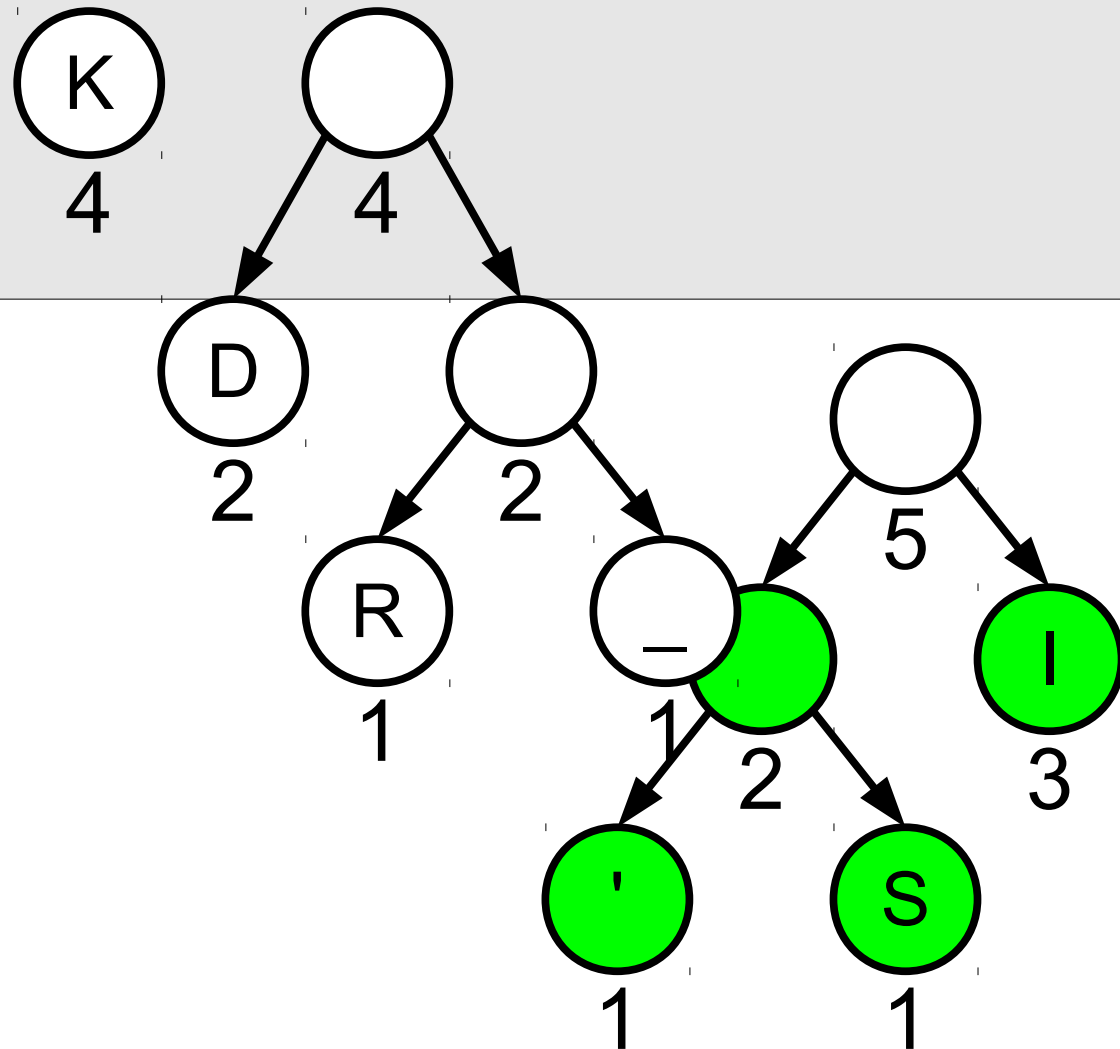
Huffman Coding



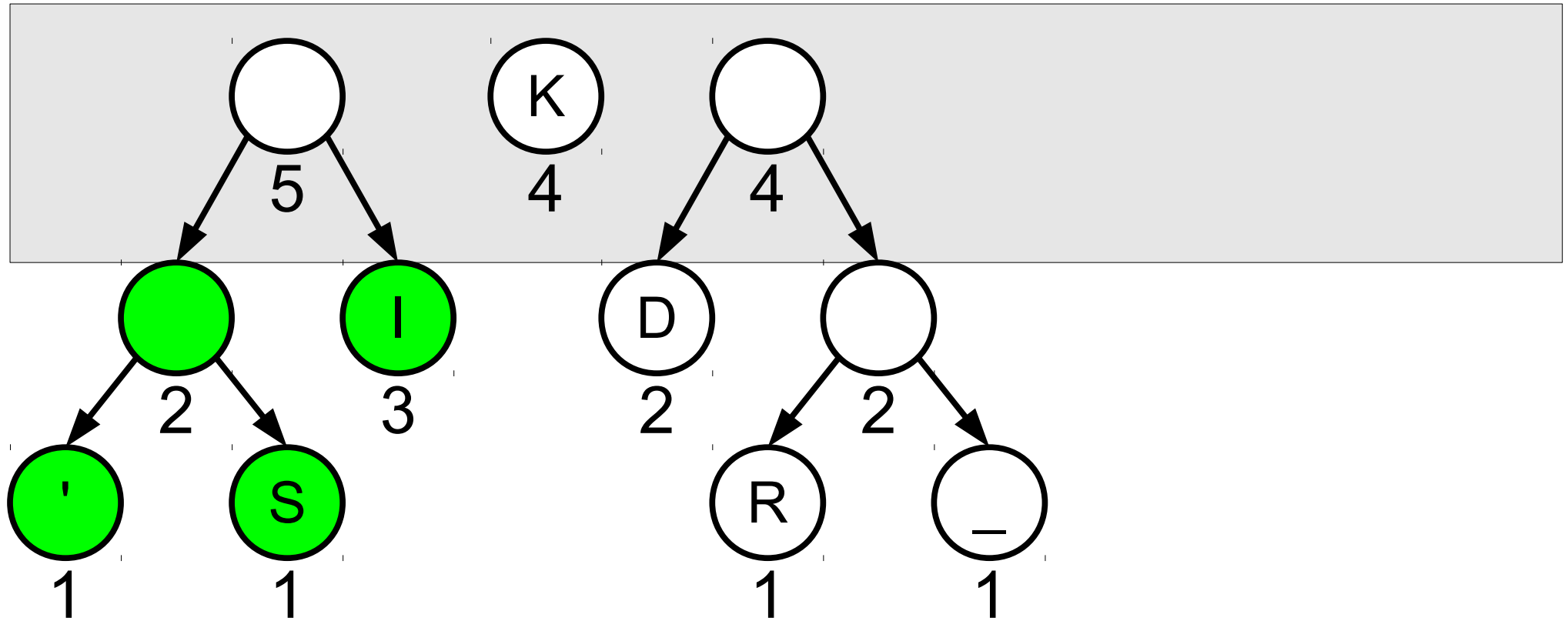
Huffman Coding



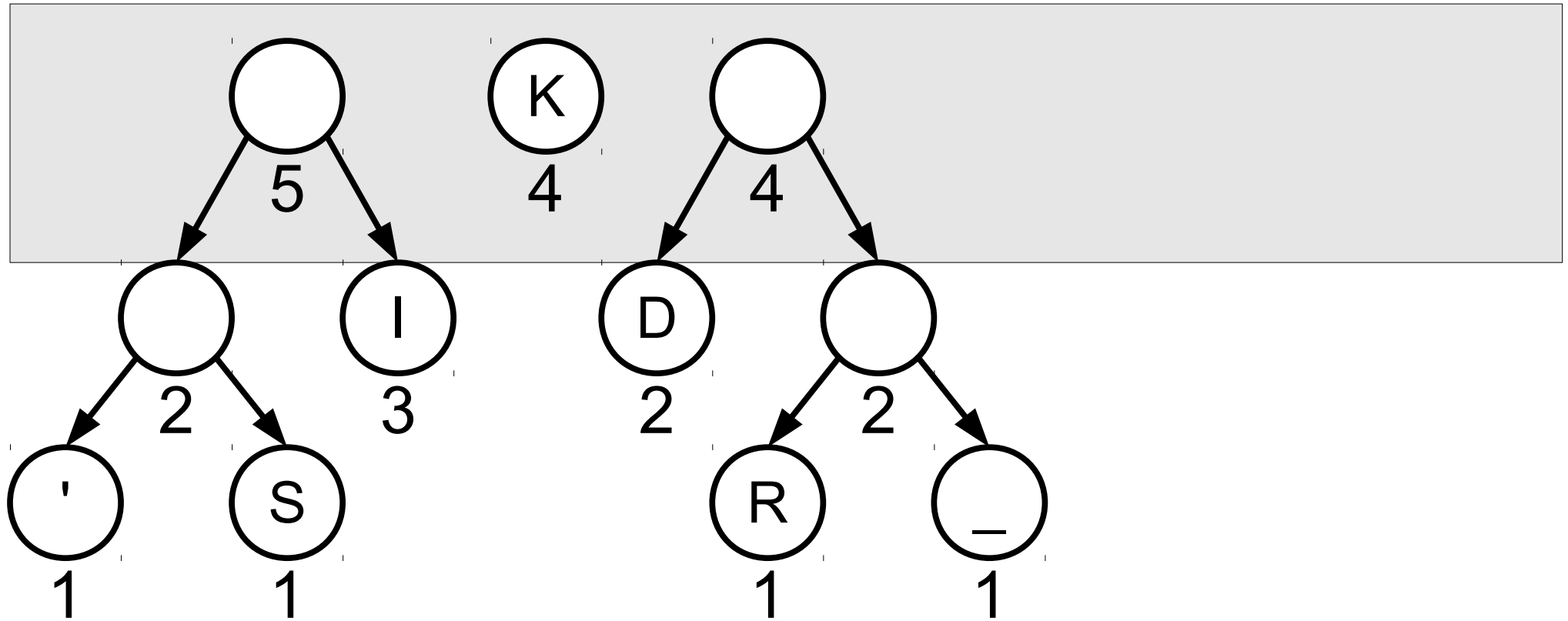
Huffman Coding



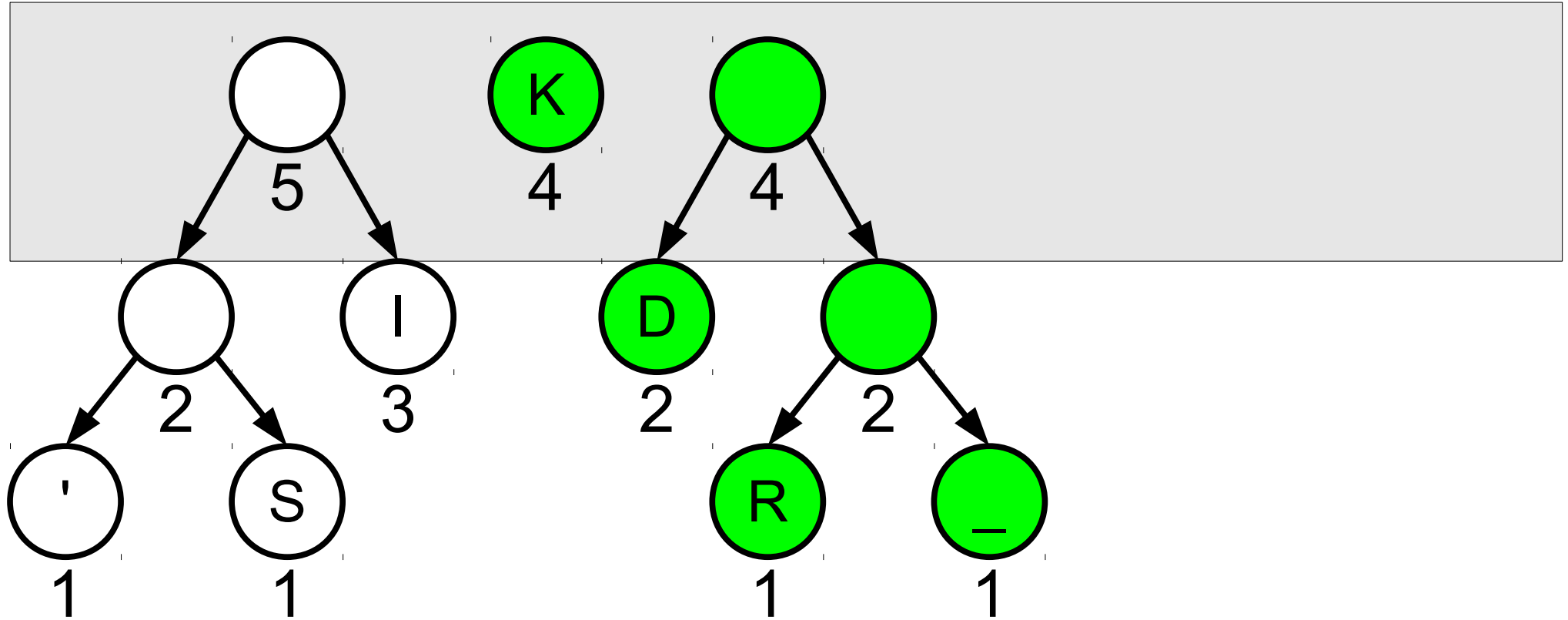
Huffman Coding



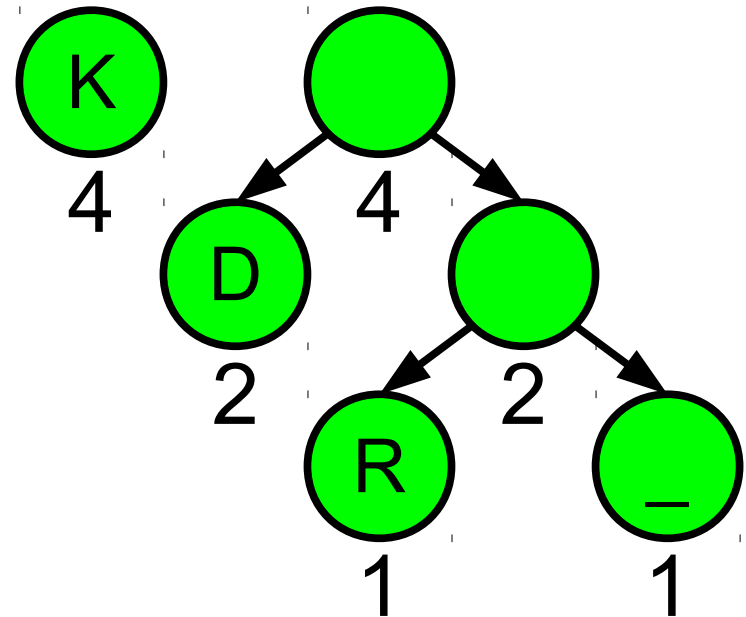
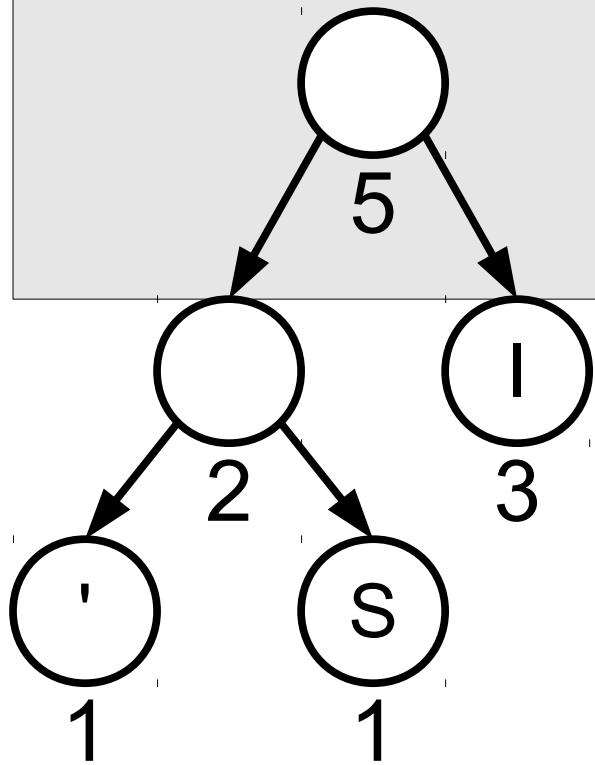
Huffman Coding



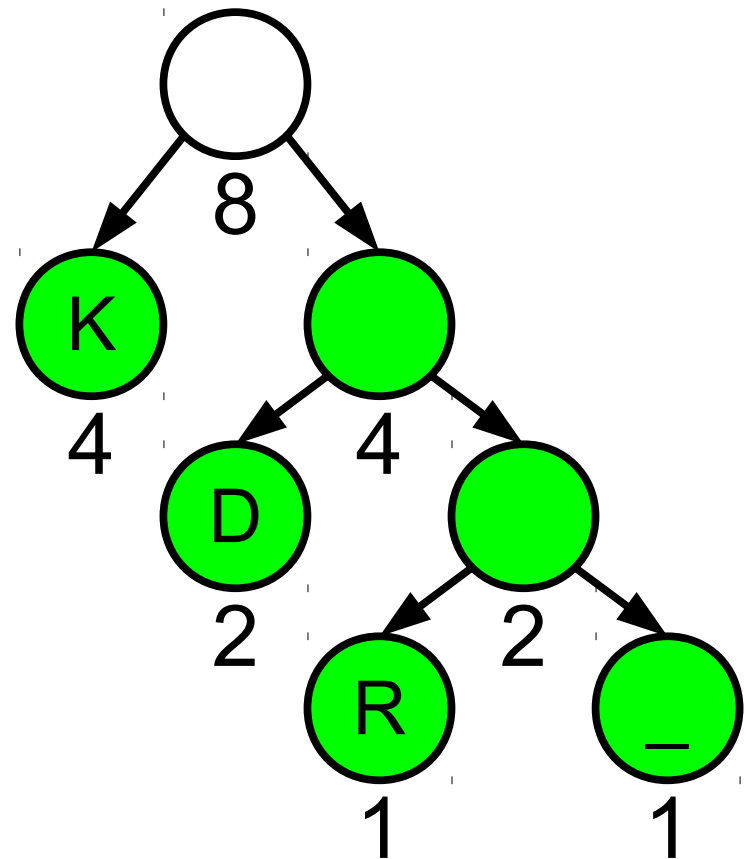
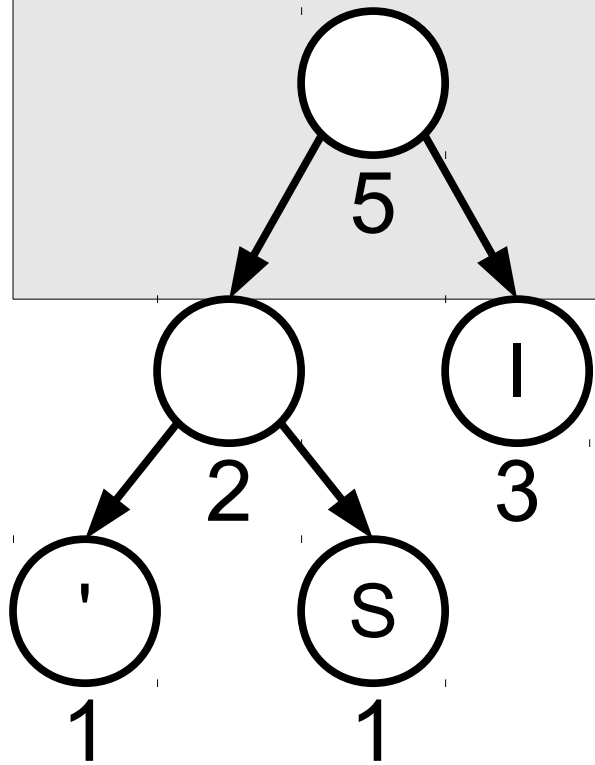
Huffman Coding



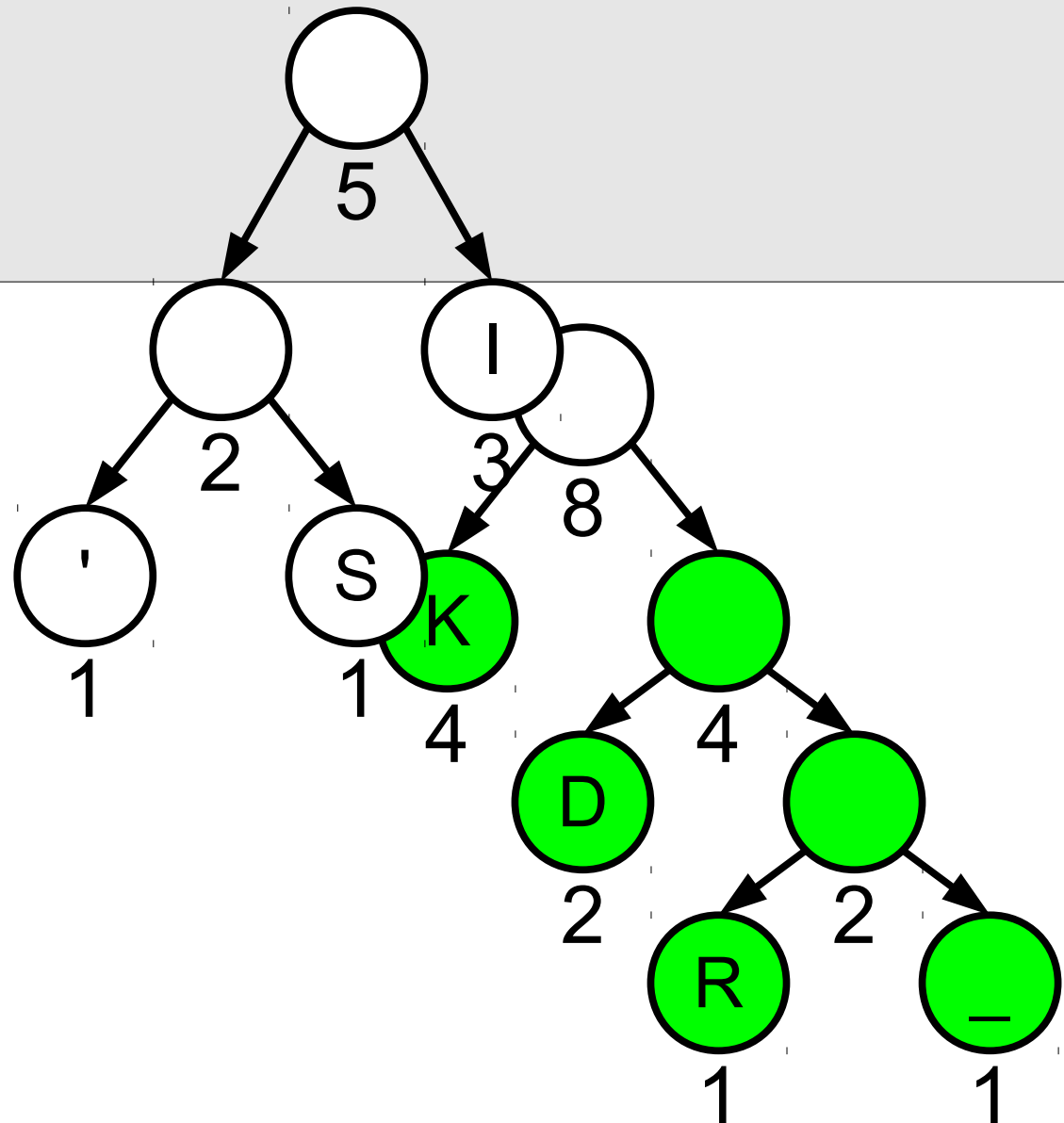
Huffman Coding



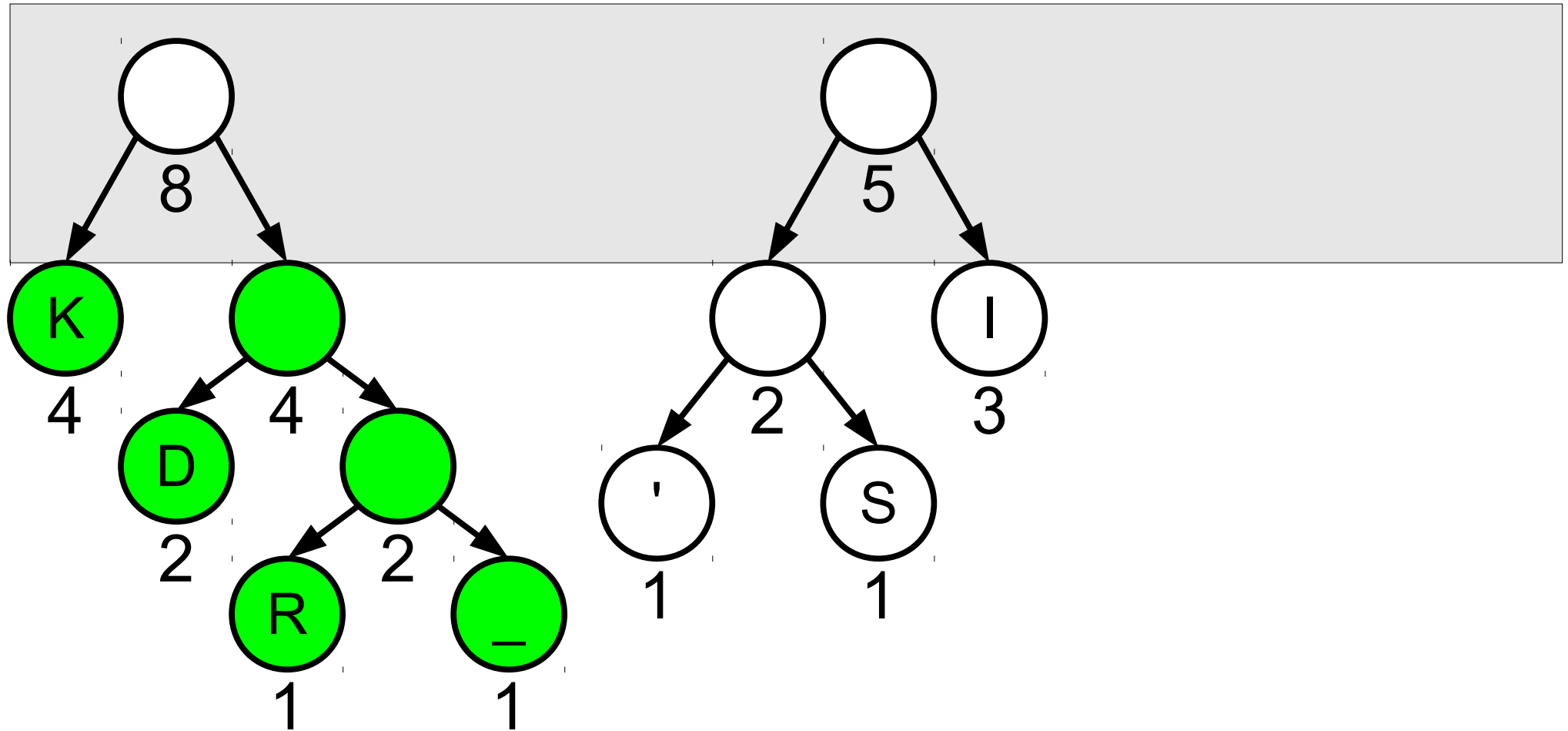
Huffman Coding



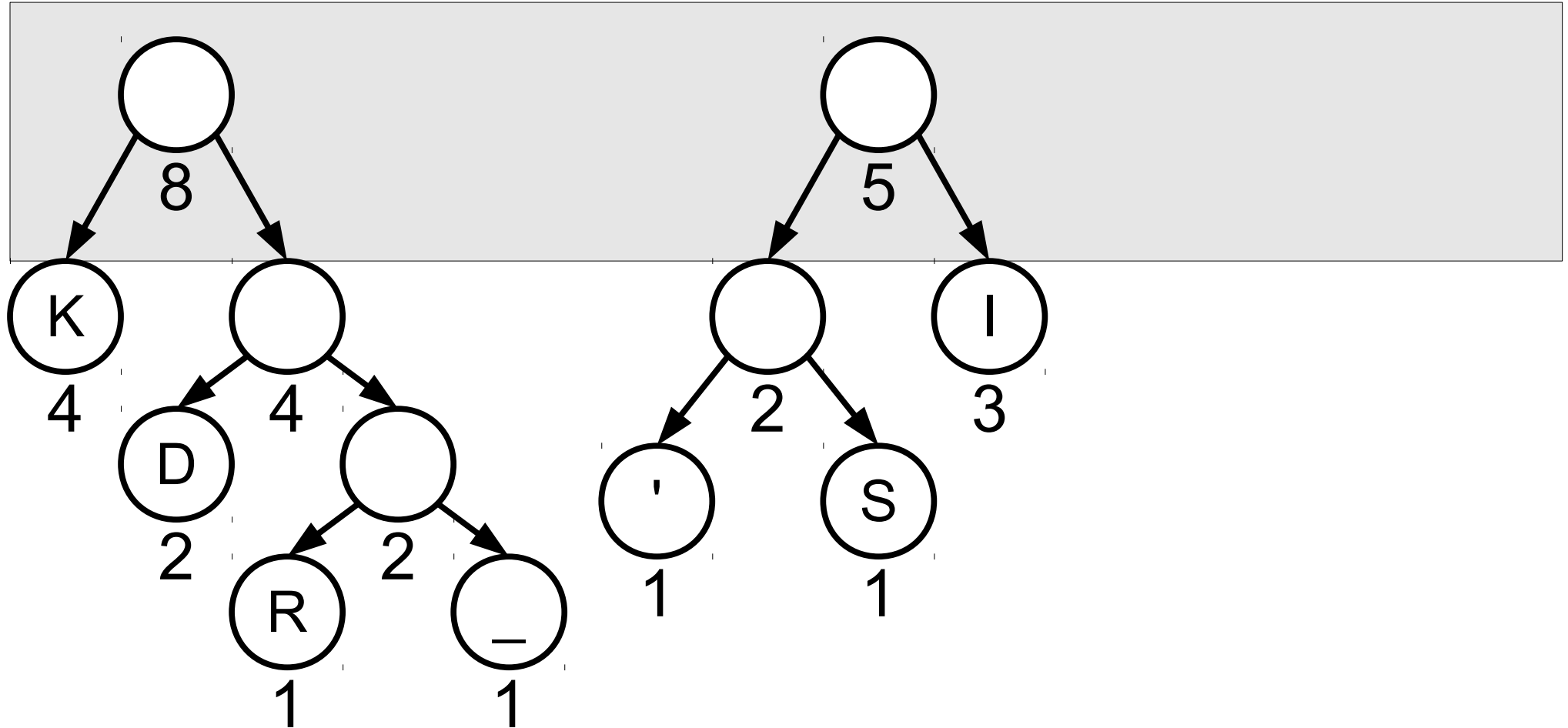
Huffman Coding



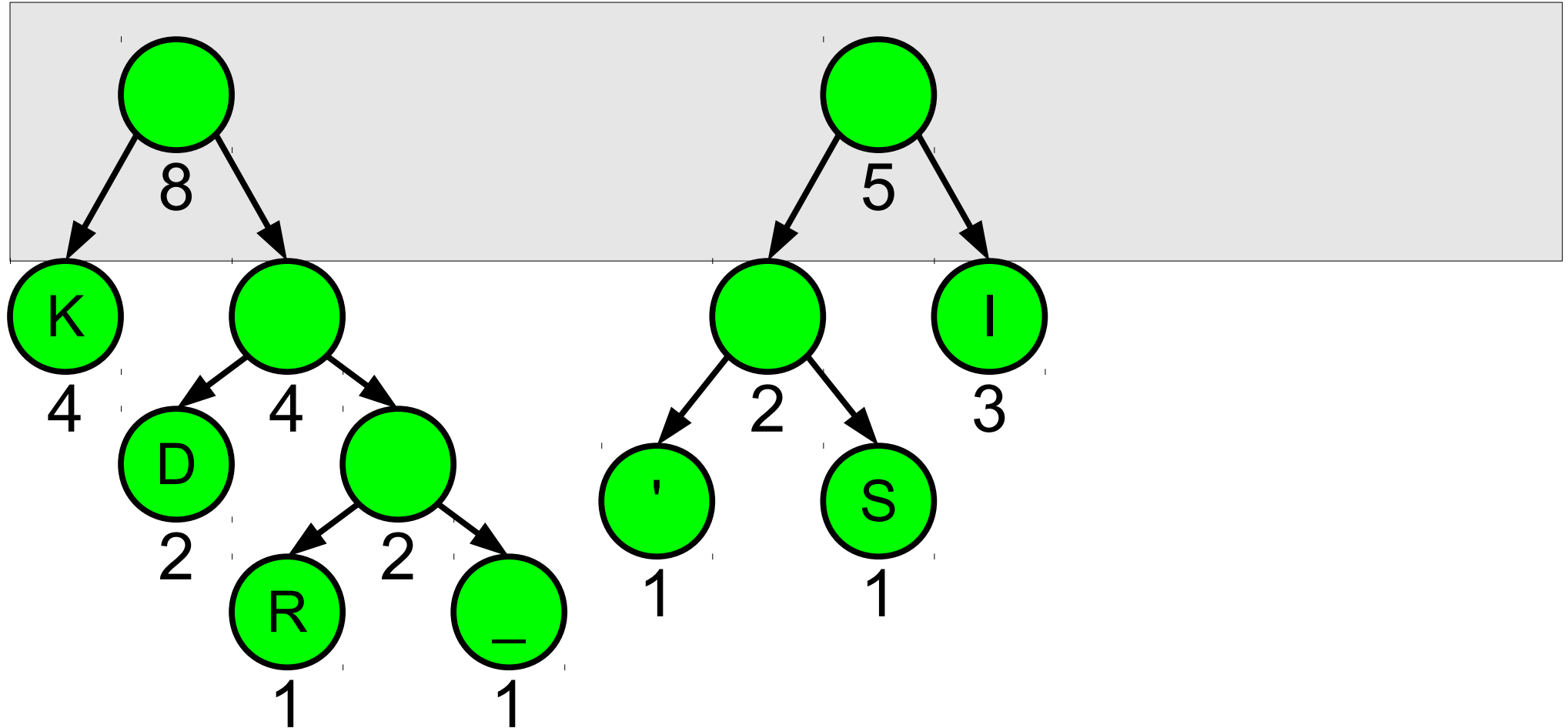
Huffman Coding



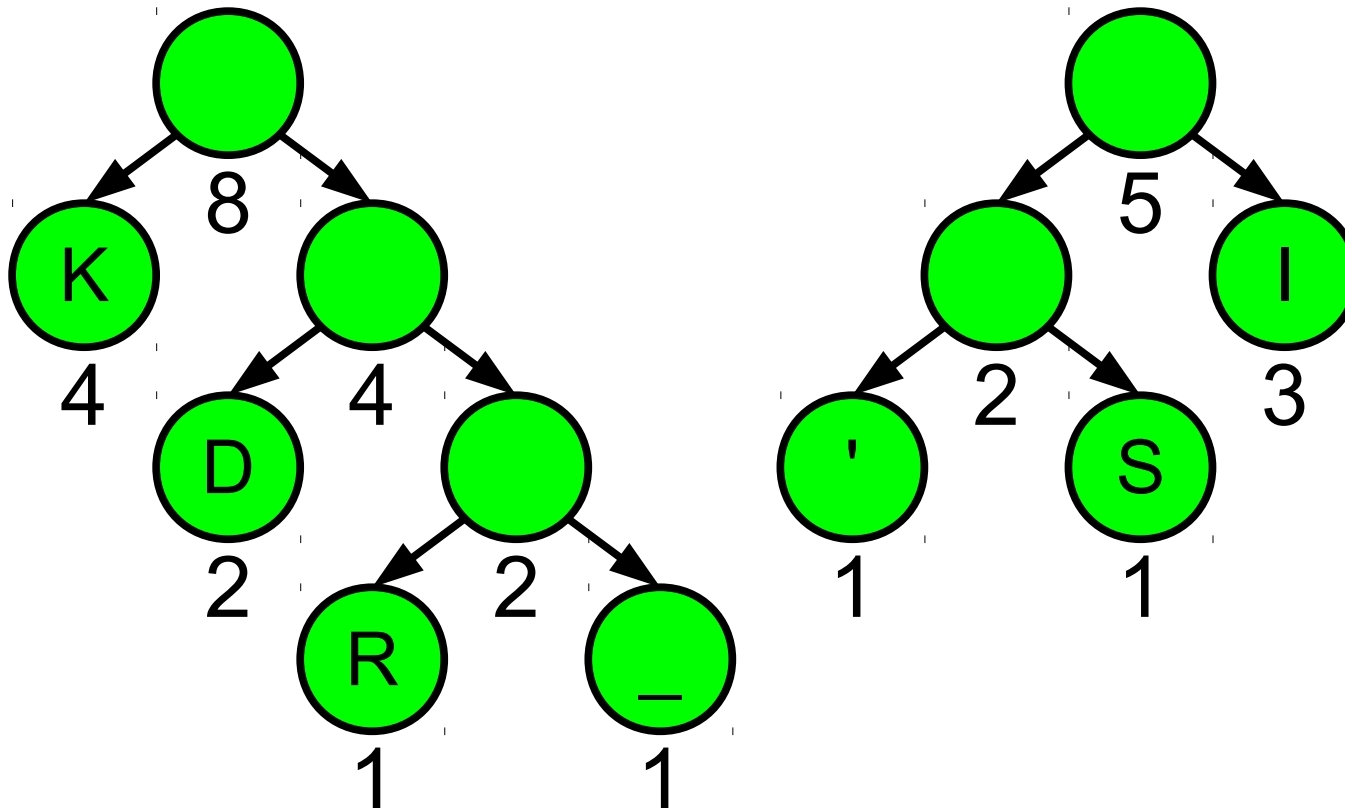
Huffman Coding



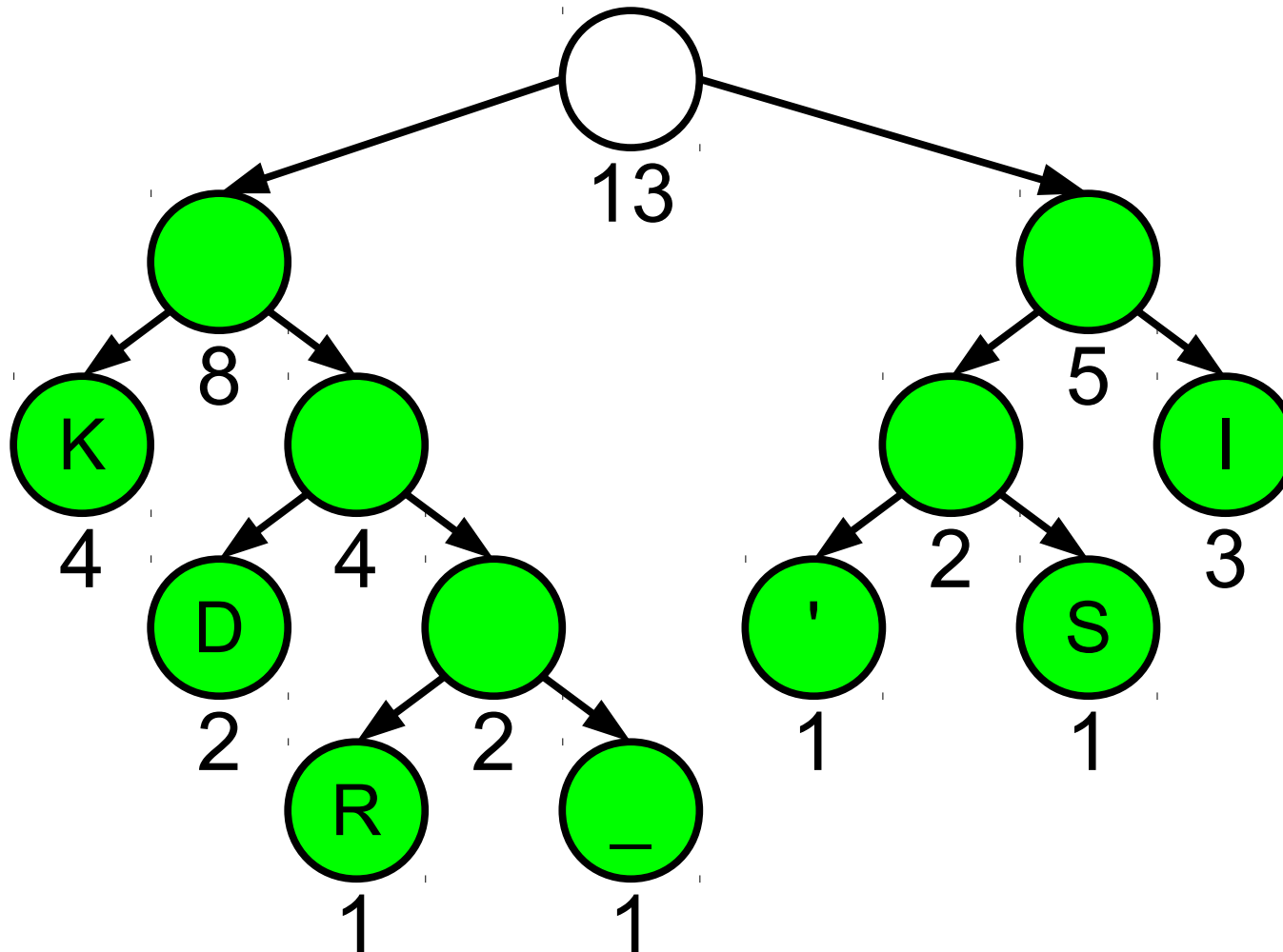
Huffman Coding



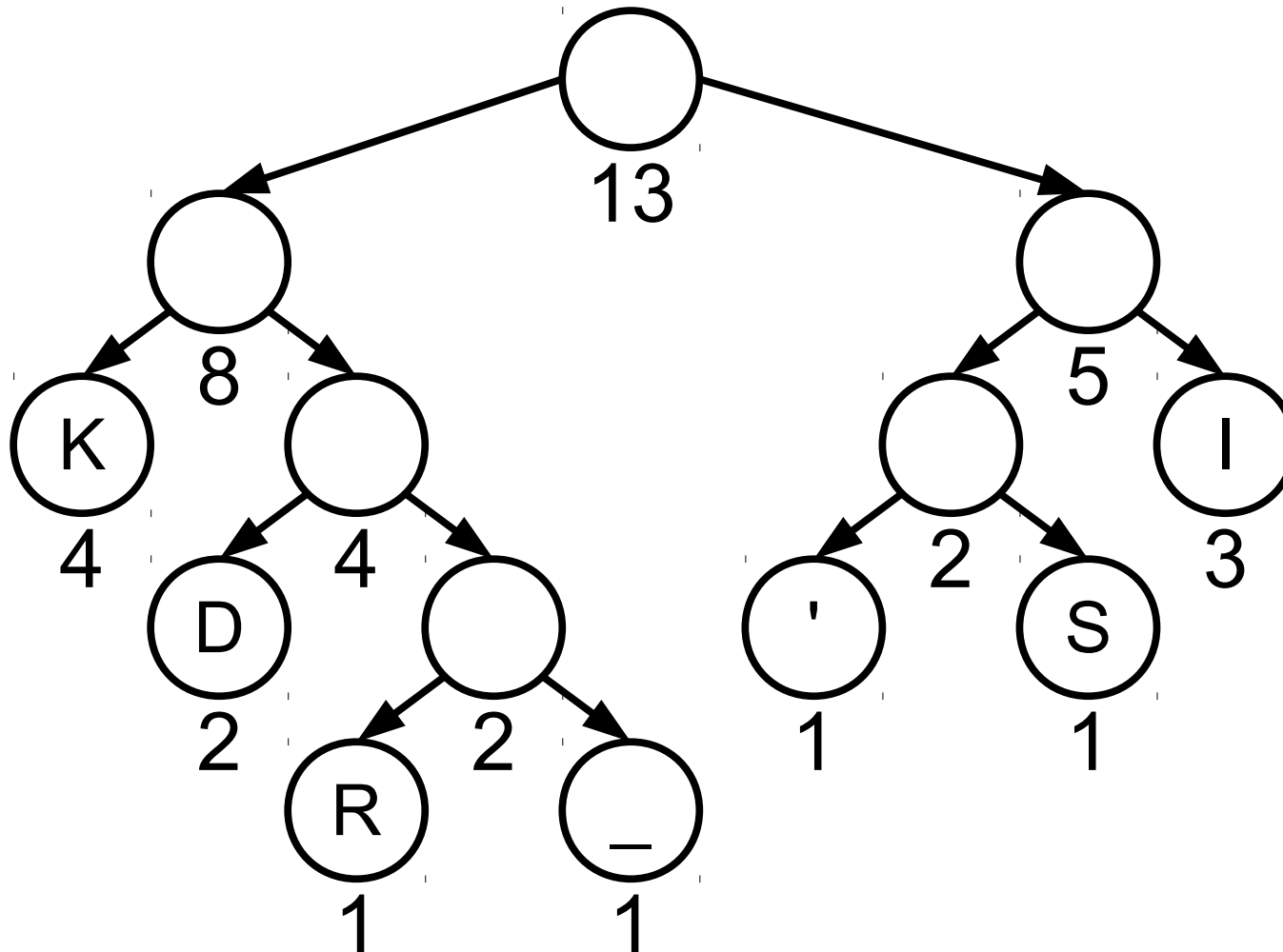
Huffman Coding



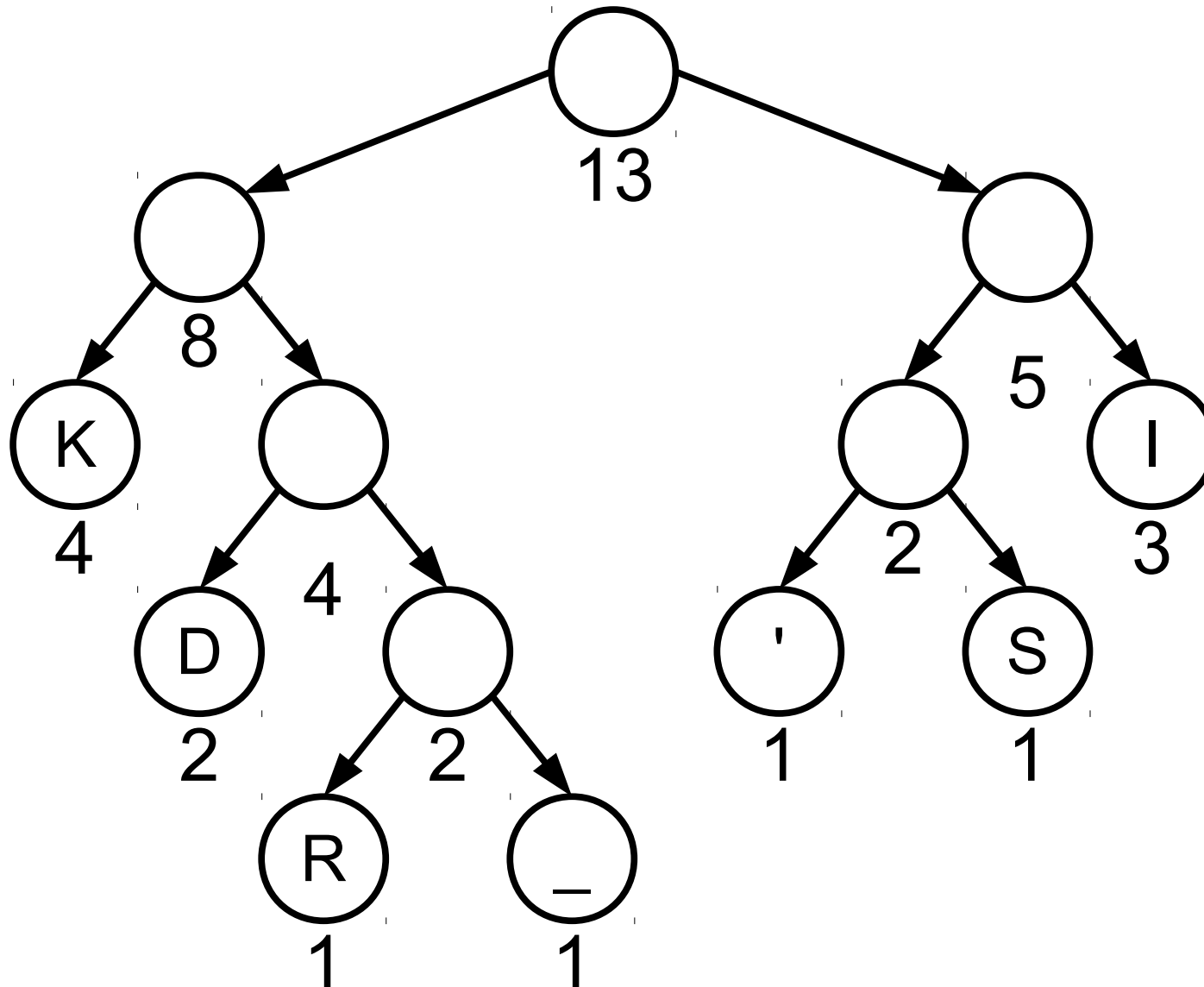
Huffman Coding



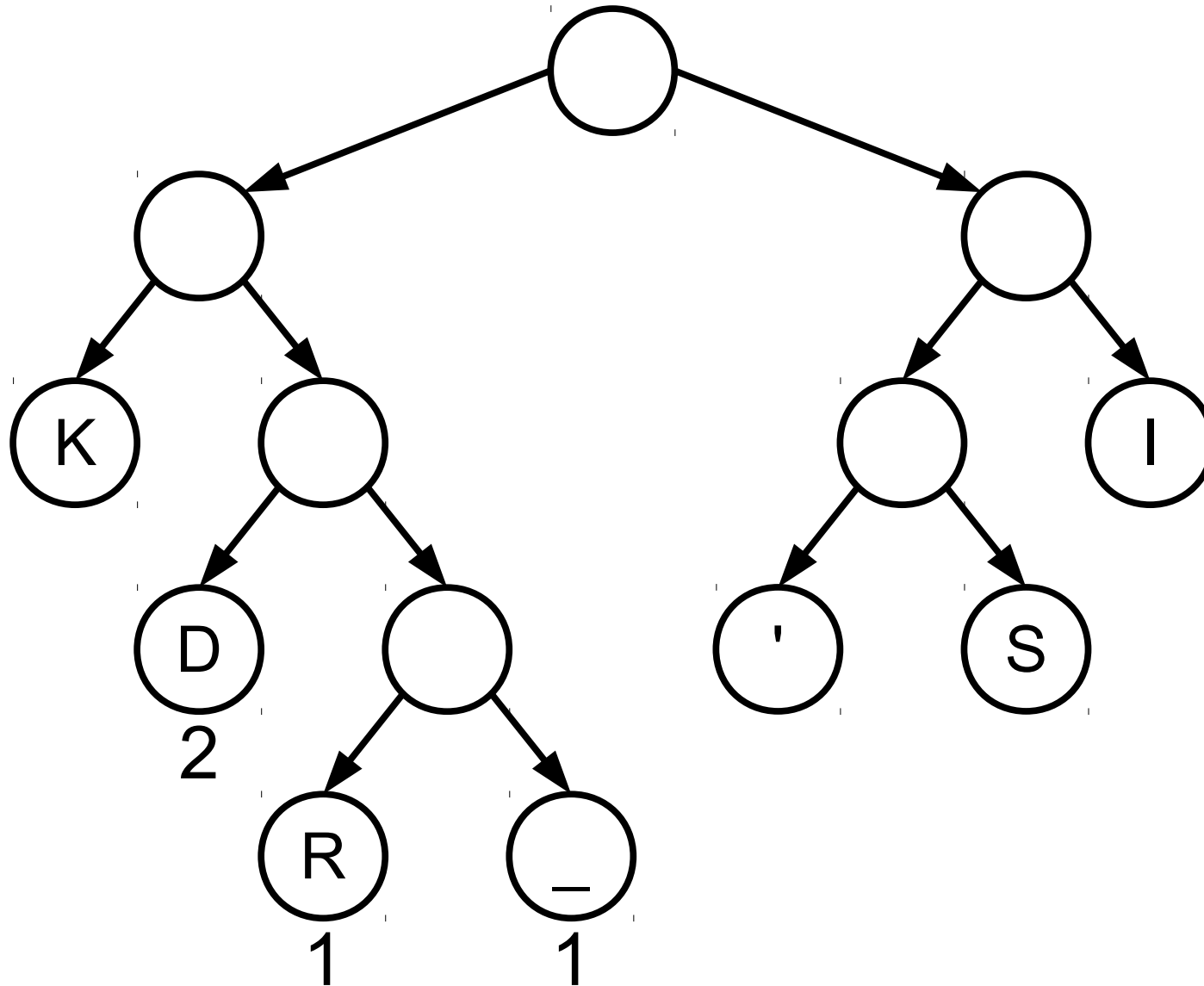
Huffman Coding



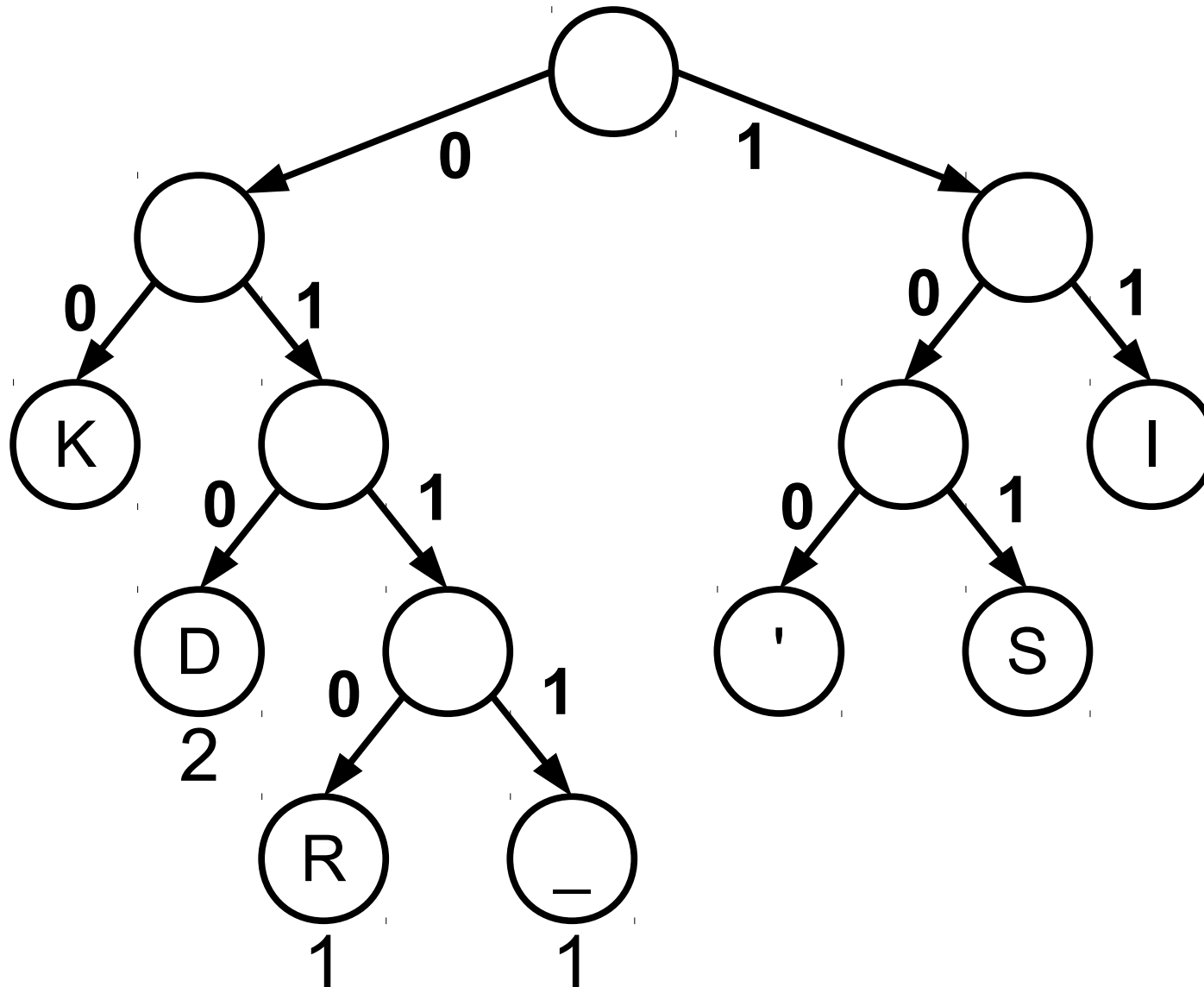
Huffman Coding



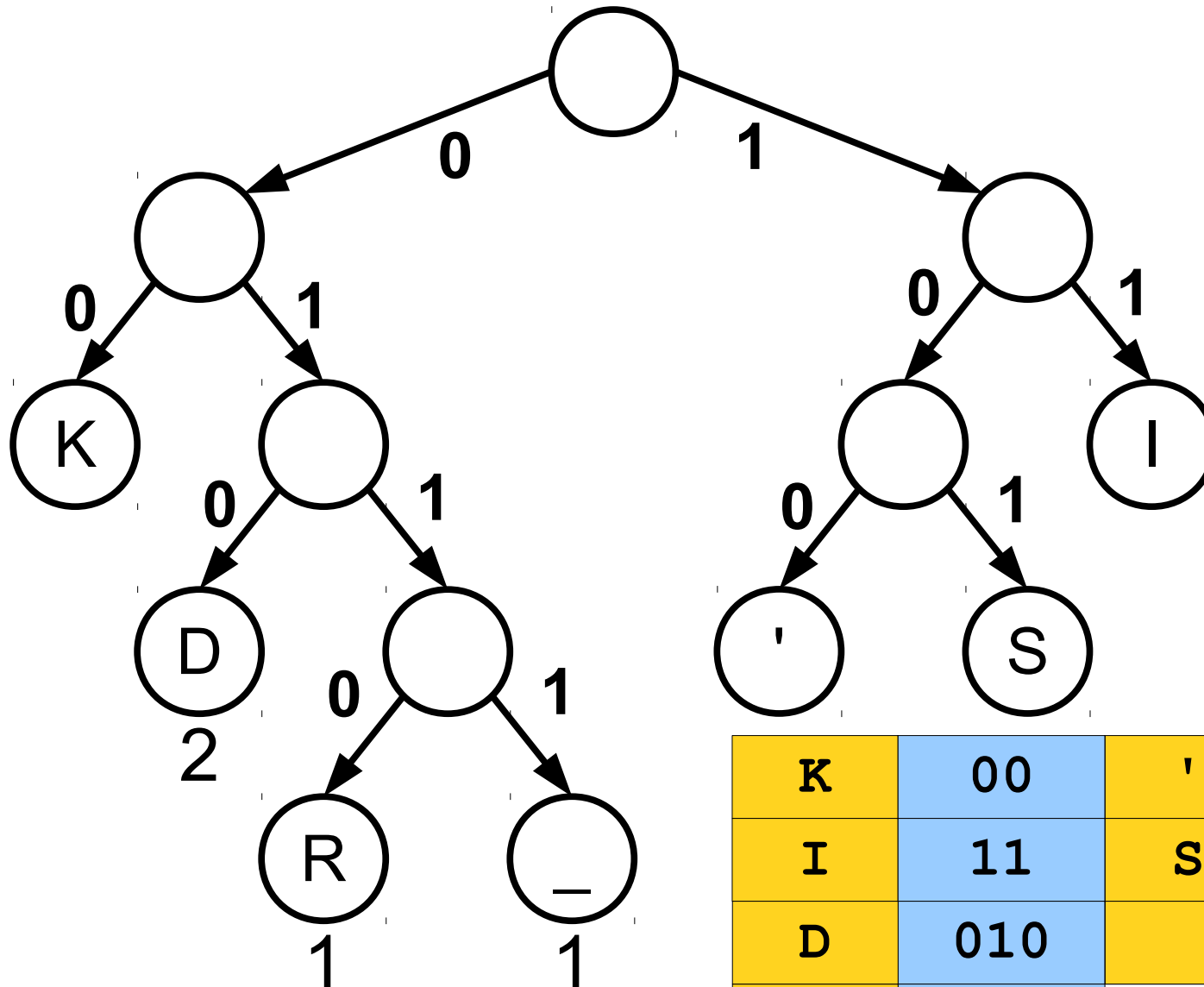
Huffman Coding



Huffman Coding



Huffman Coding



K	00	'	100
I	11	S	101
D	010		0111
R	0110		

Two Important Details

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

10	01	1111	10	001	000	1110	110	01	10	110	01	10
K	I	R	K	'	S		D	I	K	D	I	K

Prefix Codes

K	10
I	01
D	110
R	1111
'	001
S	000
	1110

1001111110001000111011001101100110

Prefix Codes

1001111110001000111011001101100110

Prefix Codes



100110

Transmitting the Tree

- In order to decompress the text, we have to remember what encoding we used!
- Idea: Prefix the compressed data with a **header** containing enough information to rebuild the table.

Encoding information

1101110010111011110001001101010111100

- This might increase the total file size!
- **Theorem**: There is no compression algorithm that can always compress all inputs.
 - **Proof**: Take CS103!

One Last Thing...

Bitten by Bytes

1001111110001000111011001101100110

Bitten by Bytes

10011111 10001000 11101100 11011001 10				
10011111	10001000	11101100	11011001	10??????

Bitten by Bytes

10011111 10001000 11101100 11011001 10				
10011111	10001000	11101100	11011001	10000001

Bitten by Bytes

10011111 10001000 11101100 11011001 10000001

K	10
I	01
D	110
R	1111

'	001
S	000
	1110

10	01	1111	10	001	000	1110	110	01	10	110	01	10	000	001
K	I	R	K	'	S		D	I	K	D	I	K	S	'

Spare Bits

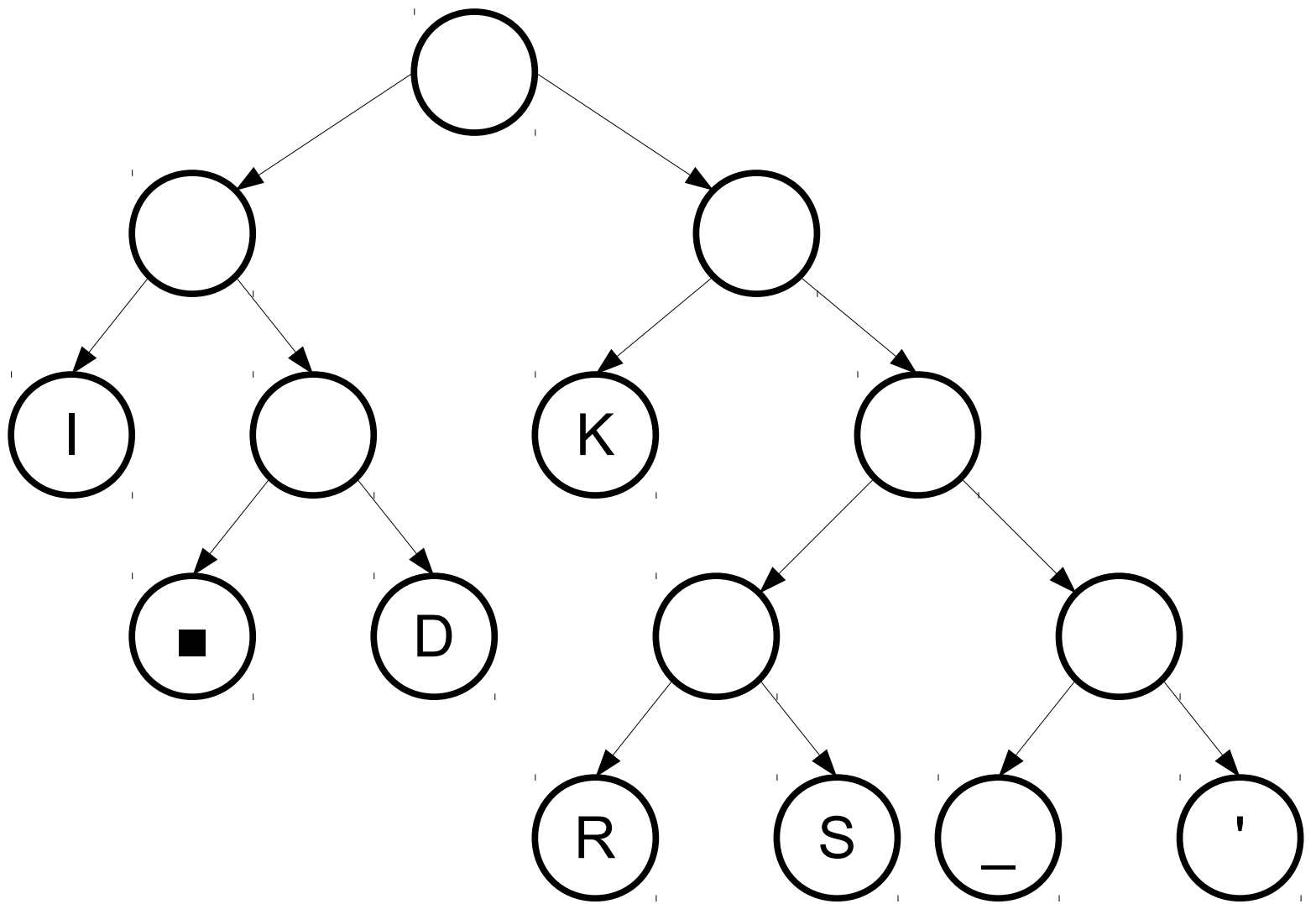
- The encoded message might not actually use all 8 bits in its final byte.
- All files are stored as bytes, so those last bits will be filled in with garbage.
- If we don't know when to stop, we might decode extra garbage data when decompressing.



STOP

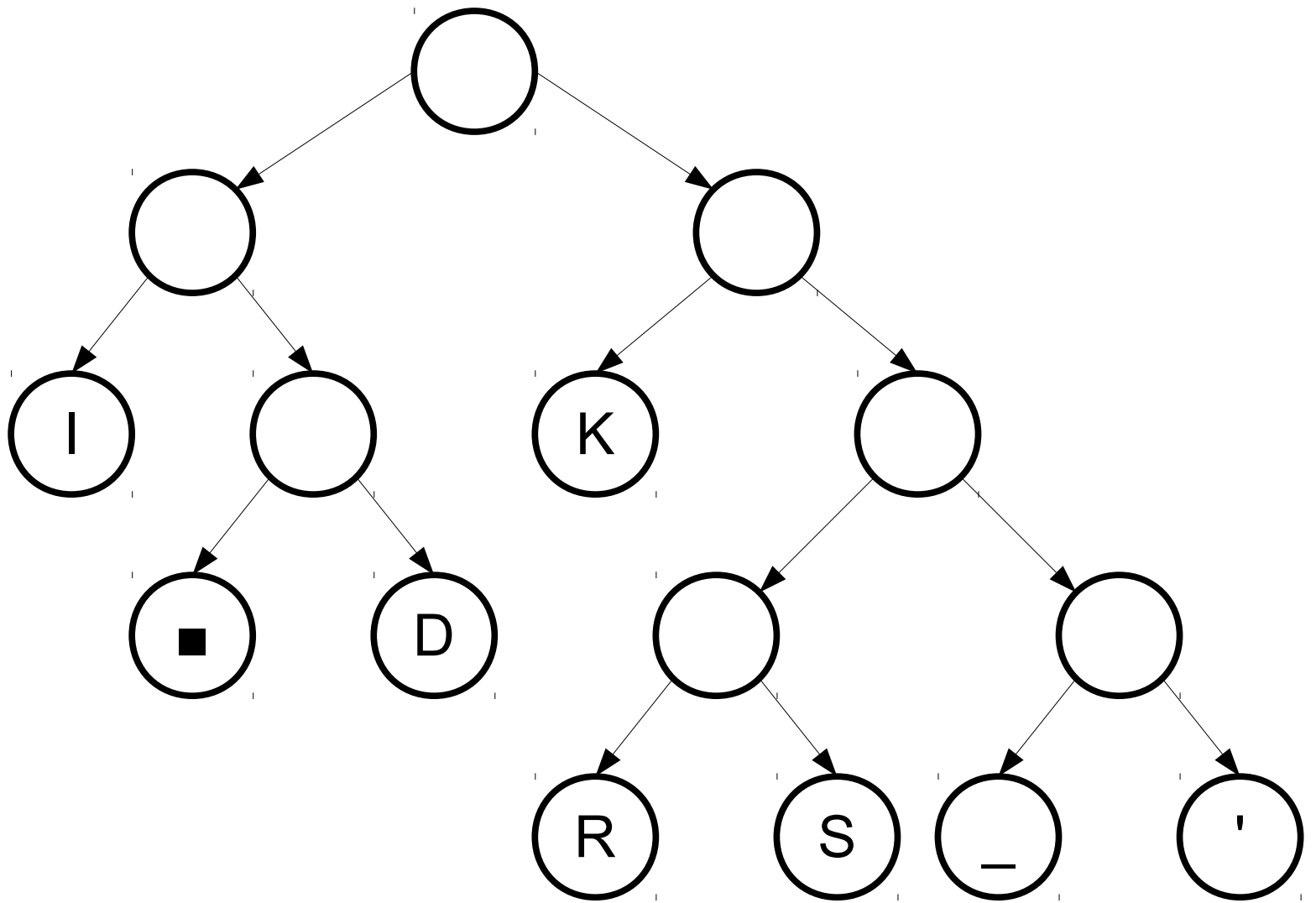
K
I
R
K
'
S
D
I
K
D
I
K
■

K	4
I	3
D	2
R	1
'	1
S	1
	1
■	1



K	4
I	3
D	2
R	1

'	1
S	1
	1
■	1



K	10
I	00
D	011
R	1100

'	1111
S	1101
	1110
■	010

Once More, With Stops

10	00	1100	10	1111	1101	1110	011	00	10	011	00	10	010
K	I	R	K	'	S		D	I	K	D	I	K	■

10001100 10111111 01111001 10010011 0010010?

K	10
I	00
D	011
R	1100

'	1111
S	1101
	1110
■	010

Pseudo-EOFs

- The marker ■ we inserted is called a ***pseudo-end-of-file marker*** (or ***pseudo-EOF***).
- Indicates where the encoding stops.
- Similar to how RNA and DNA encode proteins – certain codons are reserved for “stop here.”

Summary of Huffman Encoding

- Prefix-free encodings can be modeled as binary trees.
- Huffman encoding uses a greedy algorithm to construct encodings.
- We need to send the encoding table with the compressed message.
- We use a pseudo-EOF as a marker that the end of the bits has been reached.

Beyond ASCII

- If you just want to store ASCII text (English characters, digits, etc.), then one byte per character suffices.
- What if you want to store non-English characters or more general symbols?

Beyond ASCII

- If you just want to store ASCII text (English characters, digits, etc.), then one byte per character suffices.
- What if you want to store non-English characters or more general symbols?
- For example:
 - ¿Cómo estás?
 - السلام عليكم
 - (° □ °) ∪ ∩ ———

Unicode

- **Unicode** is a system for representing glyphs and symbols from all languages and disciplines.
- Uses a two-level encoding system:
 - Each glyph has a **code point** (a number) associated with it.
 - The code points are then represented using one of several variable-length encodings.

UTF-8

Option 1

0ddddddd

Option 2

110dddd 10dddddd

Option 3

1110ddd 10dddddd 10dddddd

Option 4

11110ddd 10dddddd 10dddddd 10dddddd

UTF-8

1110000010011111001010110001100

UTF-8

11100000 10011111 10010101 10001100

UTF-8

11100000	10011111	10010101	10001100
11100000	10011111	10010101	10001100

UTF-8

11100000	10011111	10010101	10001100
<u>11100000</u>	<u>10011111</u>	<u>10010101</u>	<u>10001100</u>

UTF-8

11100000	10011111	10010101	10001100
<u>11100000</u>	<u>10011111</u>	<u>10010101</u>	<u>10001100</u>

0000011111010101001100

UTF-8

11100000	10011111	10010101	10001100
<u>11100000</u>	<u>10011111</u>	<u>10010101</u>	<u>10001100</u>

0000011111010101001100



Further Topics

More to Explore

- ***Kolmogorov Complexity***
 - What's the theoretical limit to compression techniques?
- ***Adaptive Coding Techniques***
 - Can you change your encoding system as you go?
- ***Shannon Entropy***
 - A mathematical bound on Huffman coding.
- ***Binary Tries***
 - Other applications of trees like these!