Data Structure
Programming Project #3

郭建志

Huffman Coding

- How to encode the text efficiently?

- Consider the following short text:
  - `Eerie eyes seen near lake.`

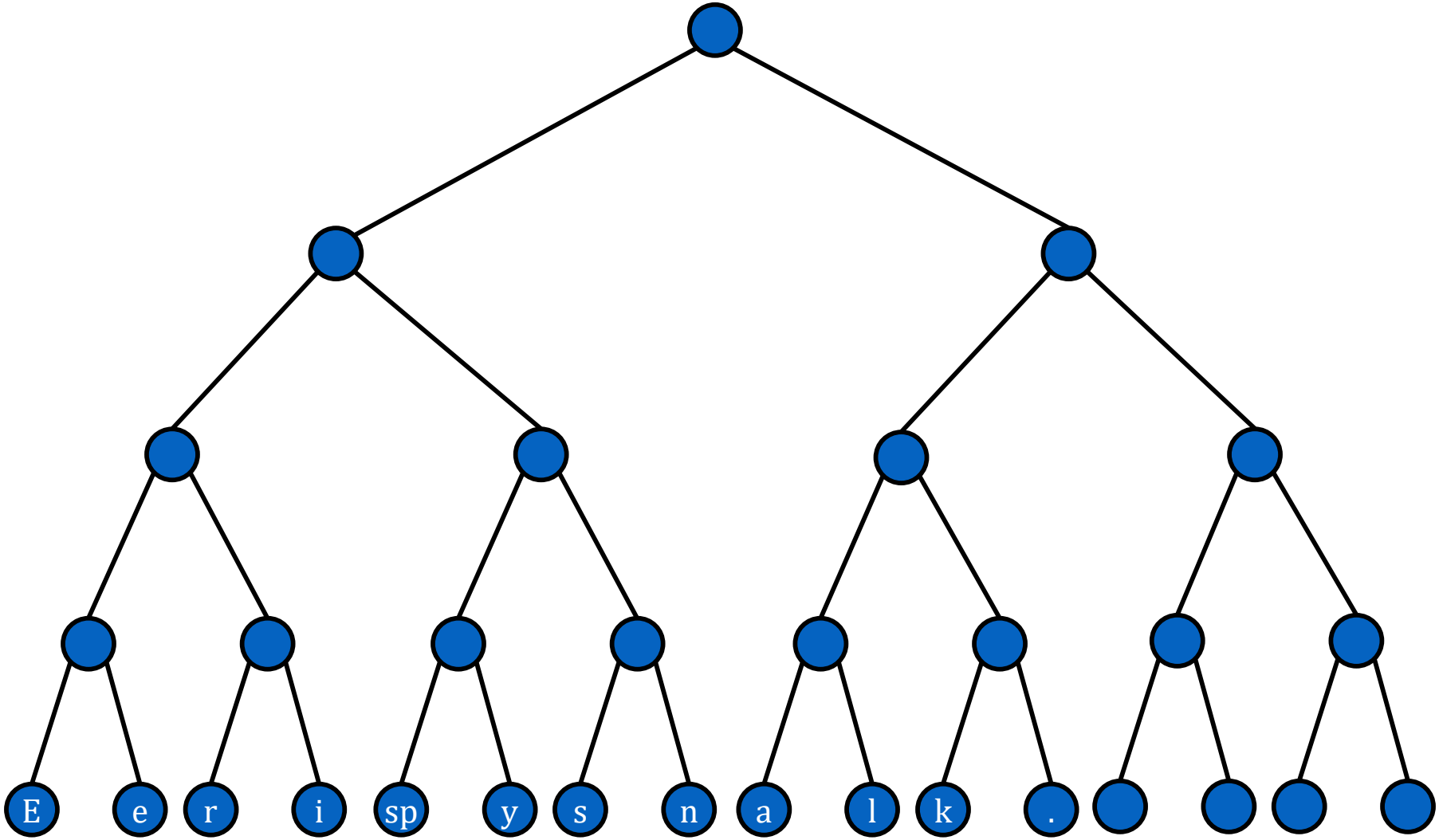
- Every character has an equal-length coding
  - 12 different characters ➔ 4-bit length

char	E	e	r	i	space	y
code	0000	0001	0010	0011	0100	0101

char	s	n	a	l	k	.
code	0110	0111	1000	1001	1010	1011

char	E	e	r	i	space	y
code	0000	0001	0010	0011	0100	0101

char	s	n	a	l	k	.
code	0110	0111	1000	1001	1010	1011



Huffman Coding

- How to encode the text efficiently?

- Consider the following short text:
  - `Eerie eyes seen near lake.`

- Every character has an equal-length coding
  - 12 different characters ➔ 4-bit length

char	E	e	r	i	sp
code	0000	0001	0010	0011	01

char	s	n	a	l	k
code	0110	0111	1000	1001	101

26 * 4 = 104 bits

Huffman Coding

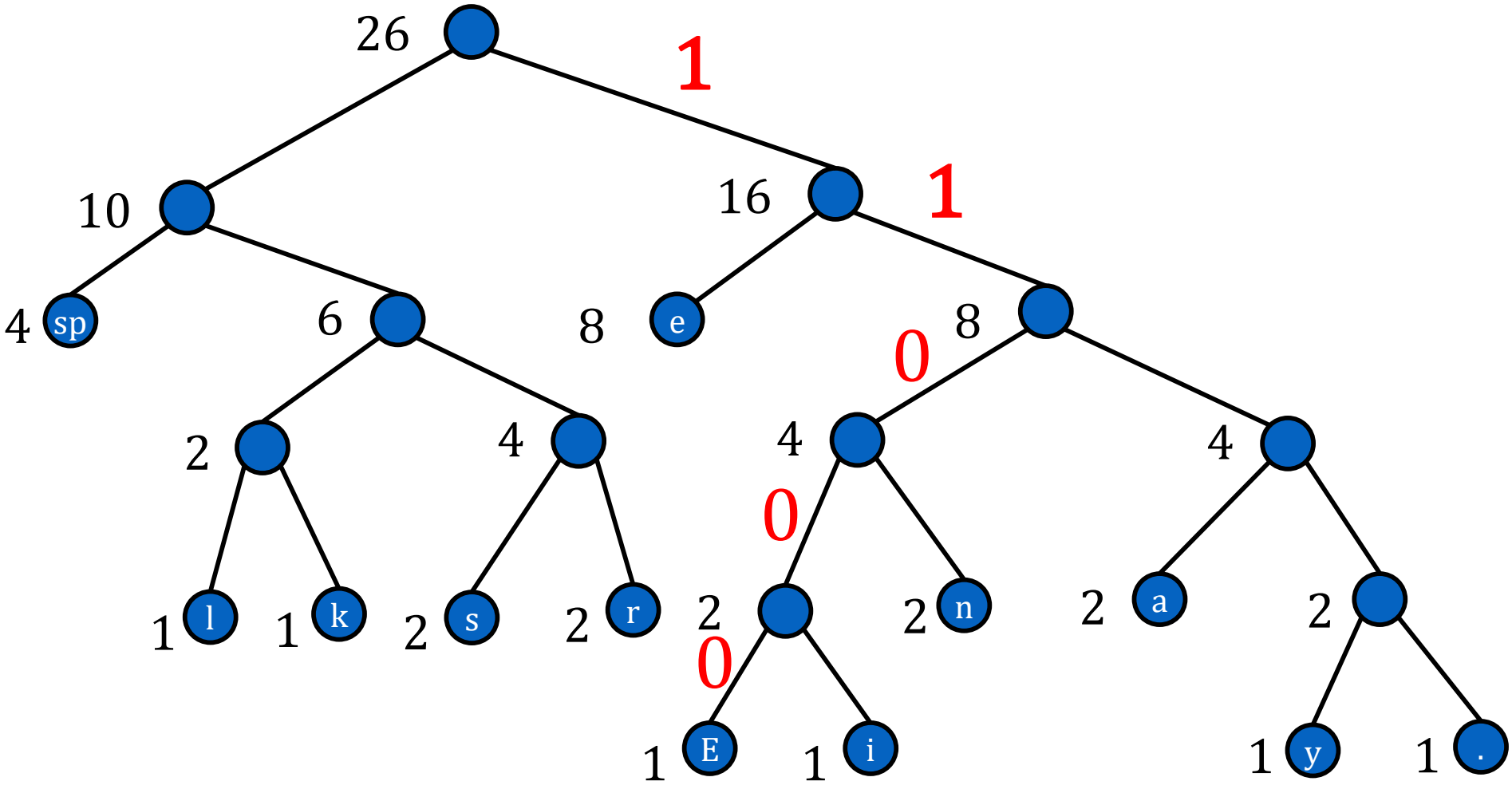
- Based on lengths of assigned codes based on frequencies

- Variable length codes
  - The char with a higher frequency → Shorter length coding

- Lossless Data Compression Algorithm
  - Definition: Lossless = without loss of information

char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

char	E	e	r	i	space	y
code	11000	10	0111	11001	00	11110
char	**s**	**n**	**a**	**l**	**k**	**.**
code	0110	1101	1110	0100	0101	11111



Huffman Coding

- Based on lengths of assigned codes based on frequ
- Varia
  - Th
  →

Before: 26 * 4 = 104 bits

After:
1 * 5 + 2 * 8 + 2 * 4 + 1 * 4 + 4 * 2 + 1 * 5 +
2 * 4 + 2 * 4 + 2 * 4 + 1 * 4 + 1 * 5 + 1 * 5 = 84 bits

char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

char	E	e	r	i	space	y
code	11000	10	0111	11001	00	11110
char	**s**	**n**	**a**	**l**	**k**	**.**
code	0110	1101	1110	0100	0101	11111

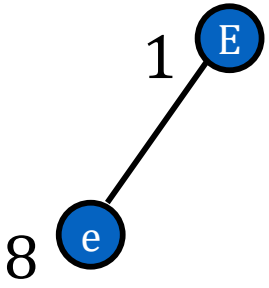
Huffman Coding

- Construct a priority queue in which:
  - The keys in nodes are the frequencies
  - The data in nodes are the characters
- Add all the characters with their frequencies into the priority queue

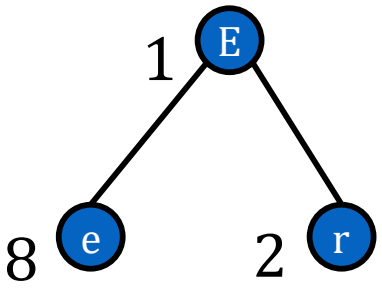
- Pop two nodes u and v in order
- Create a node w
- The left child and right child of w are set to u and v, respectively
- Push node w back to the priority queue
- Repeat the above pop and push until no node in the queue

char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

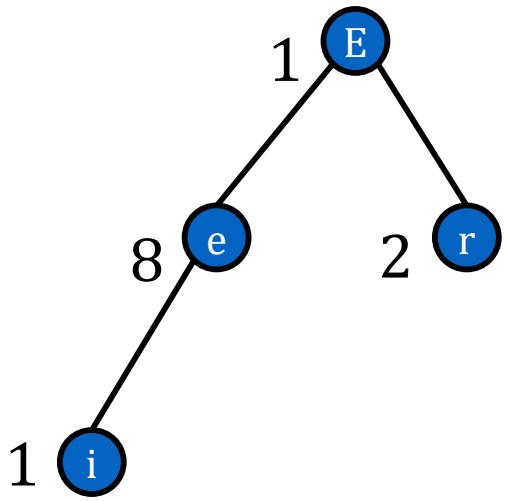
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



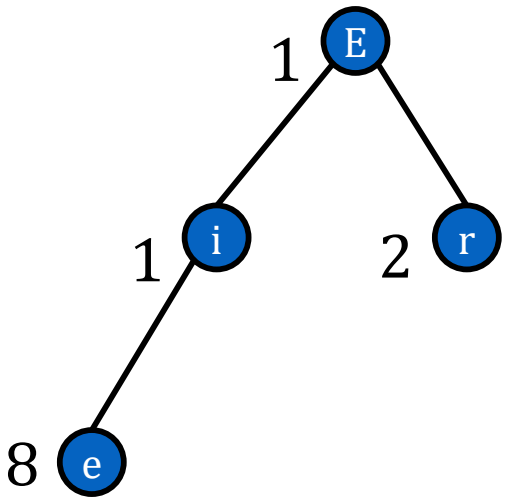
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



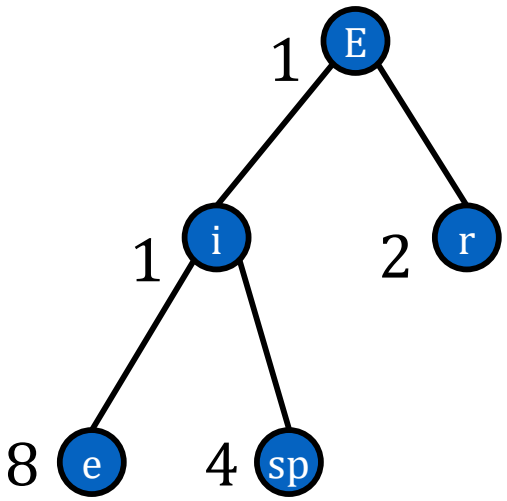
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



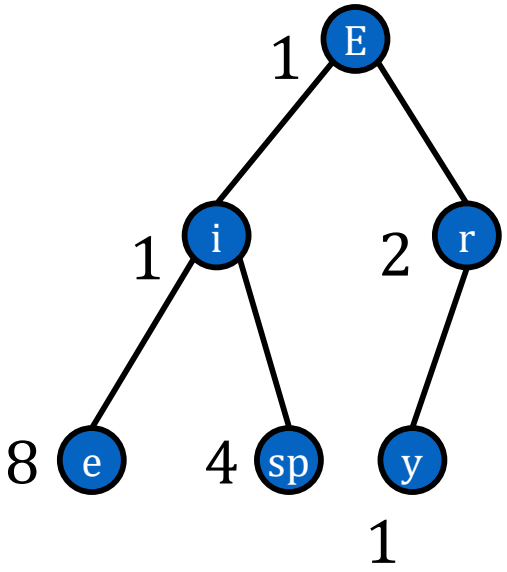
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



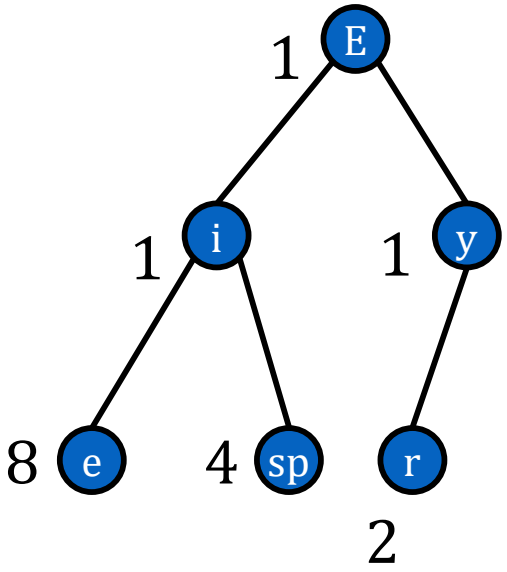
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



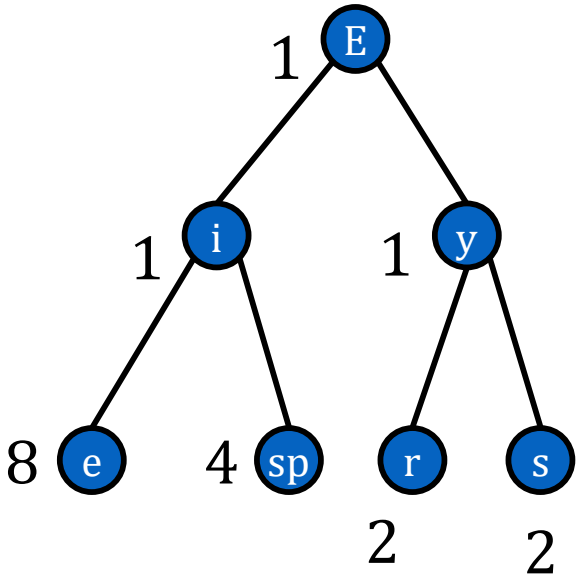
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



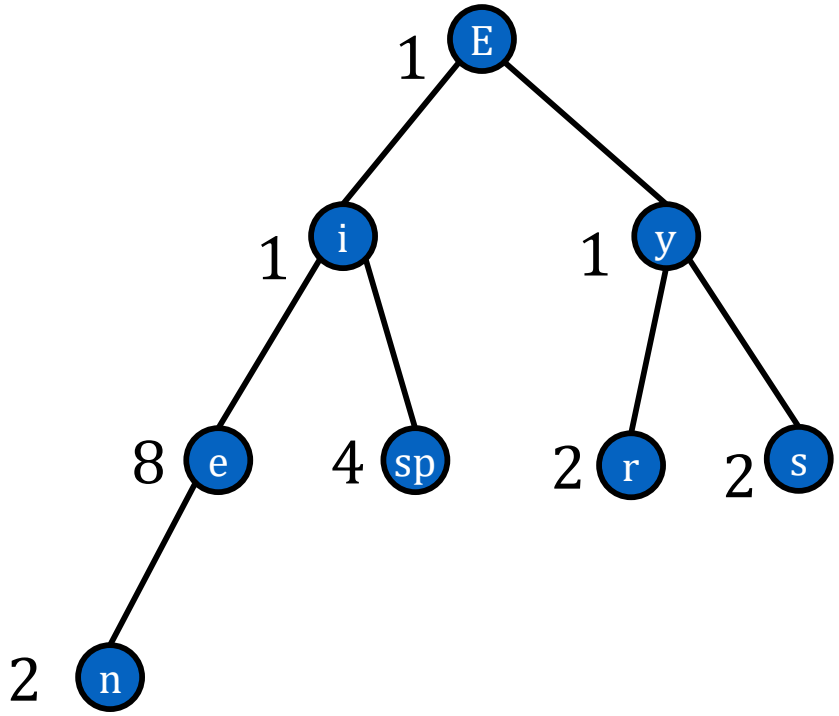
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



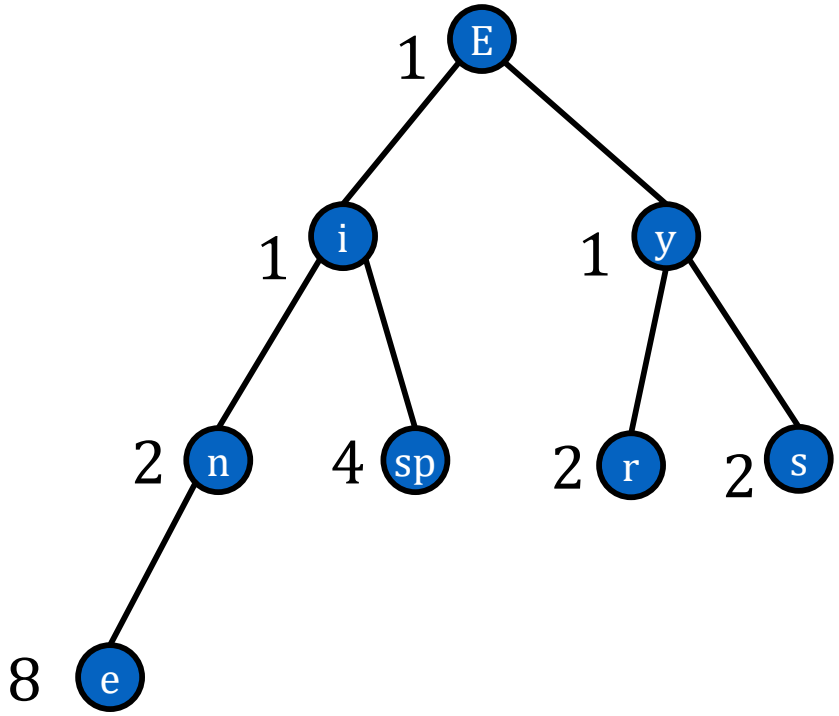
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



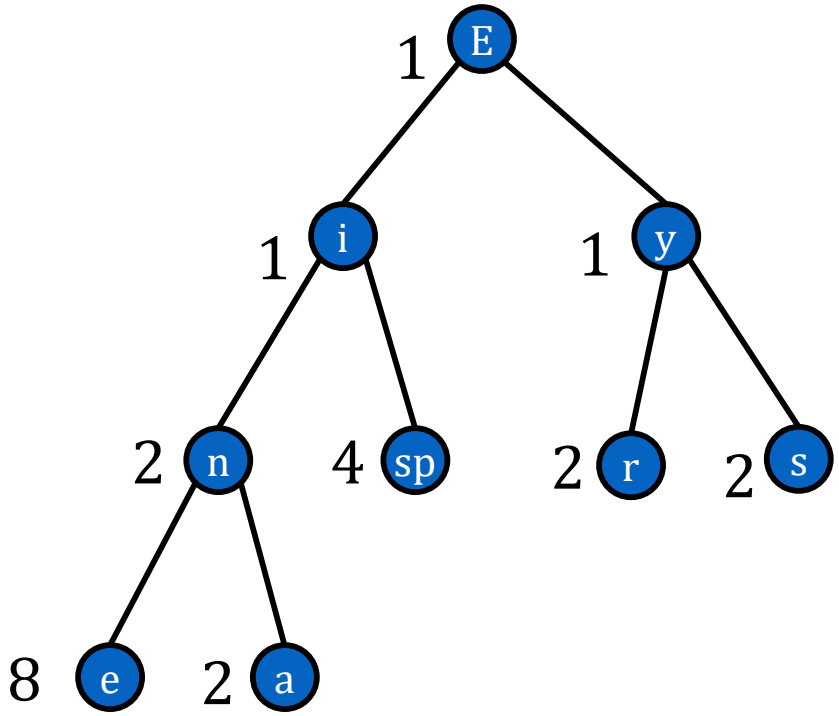
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



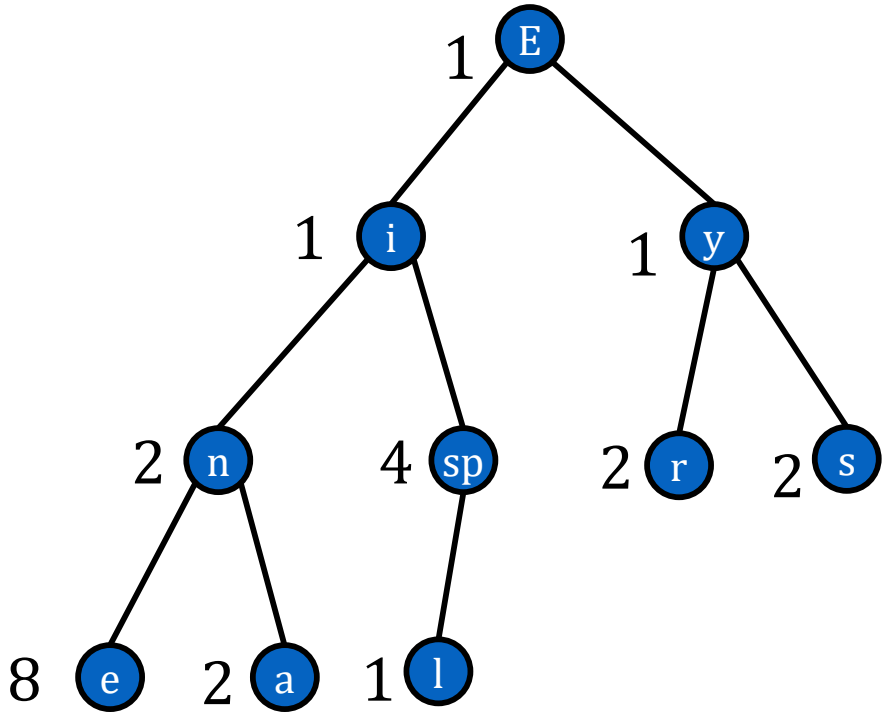
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



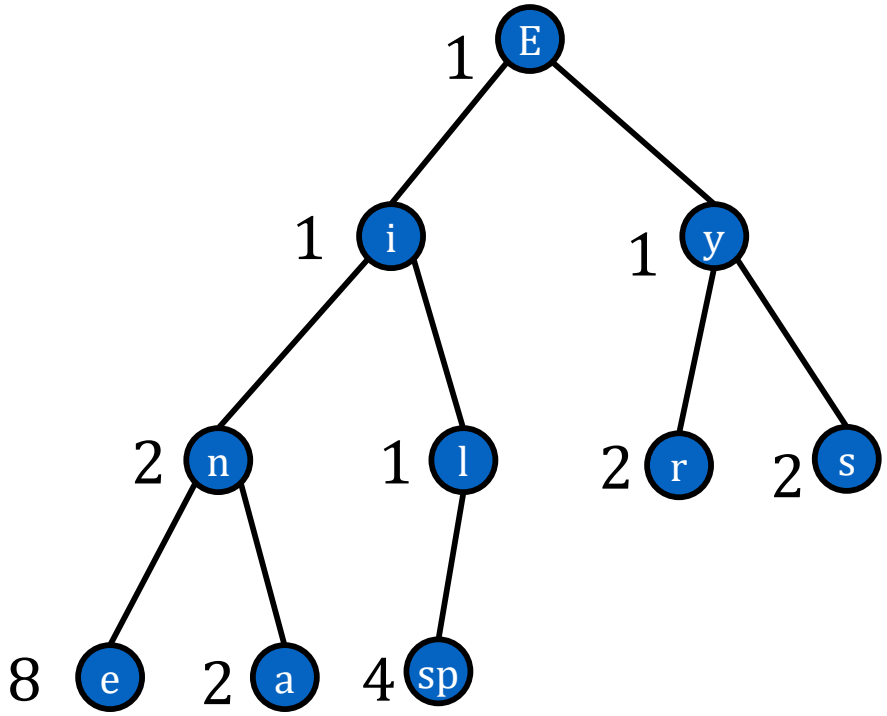
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



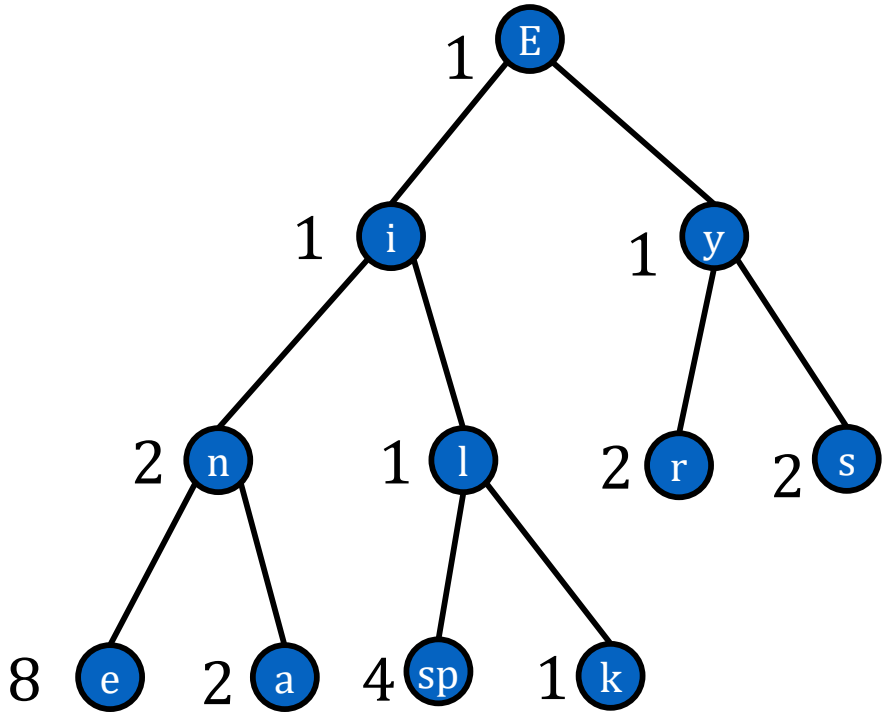
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



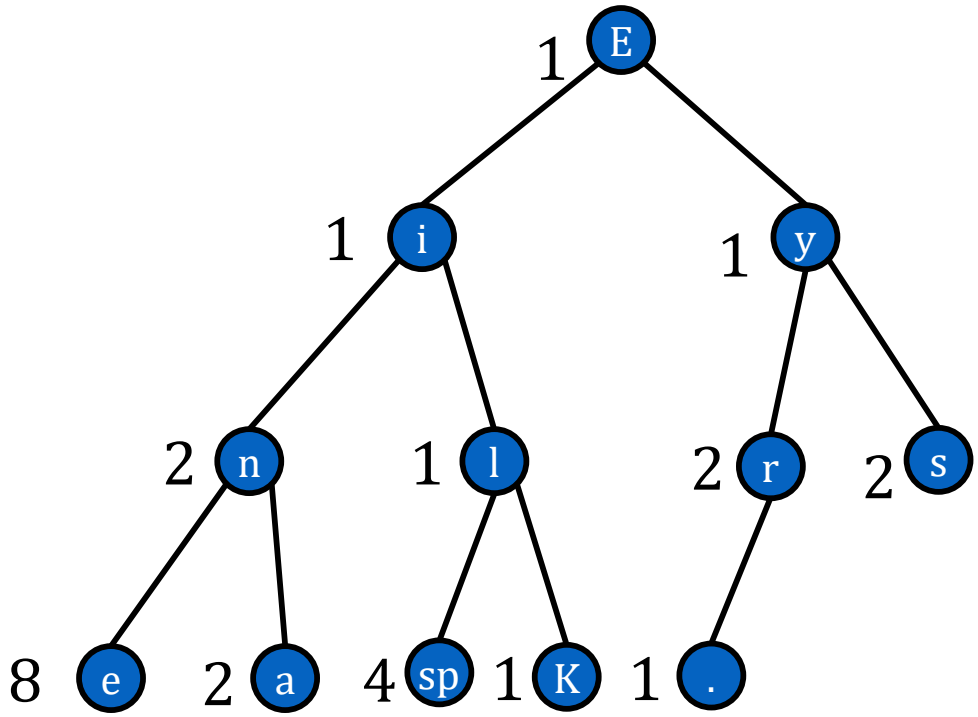
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1



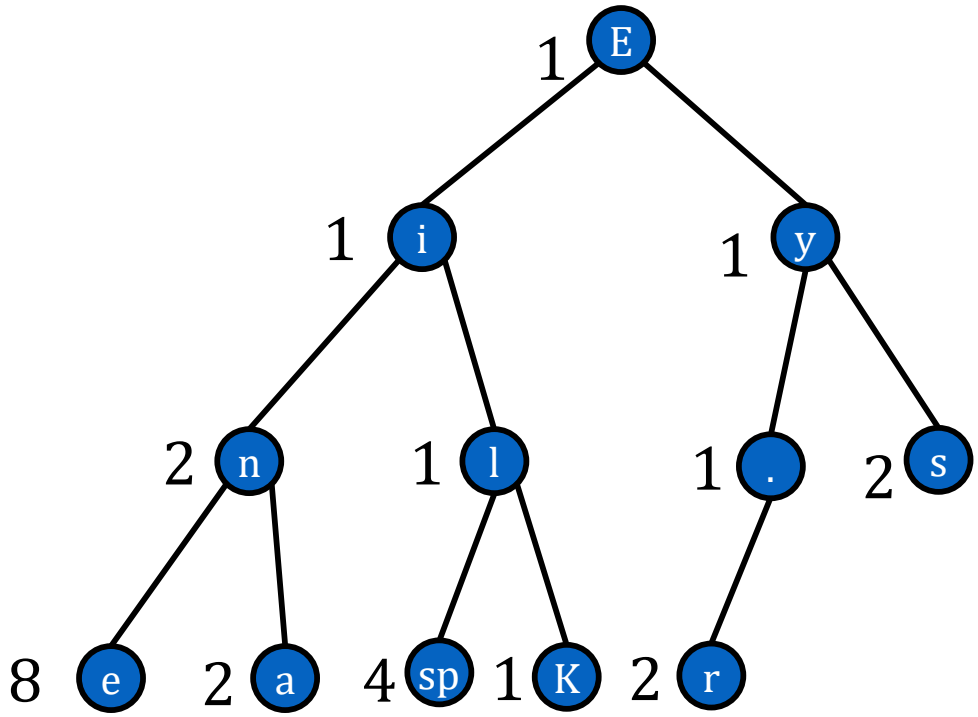
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

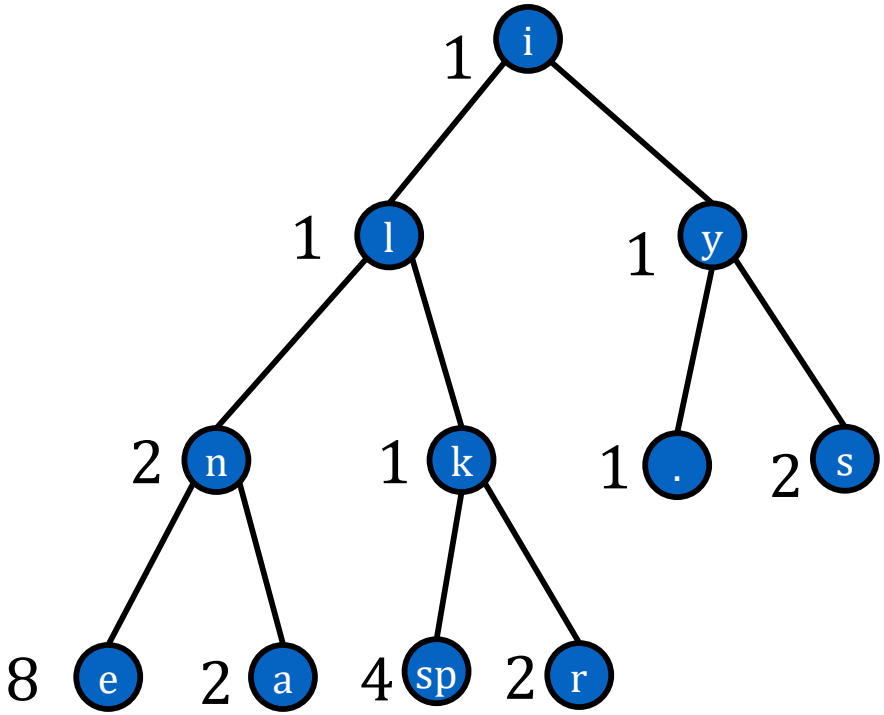


char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

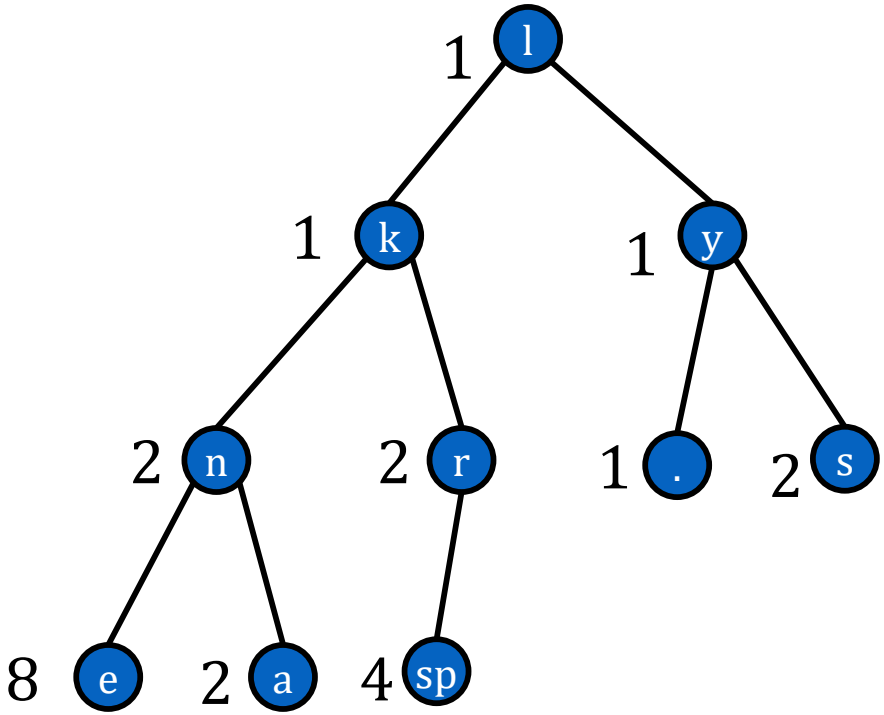


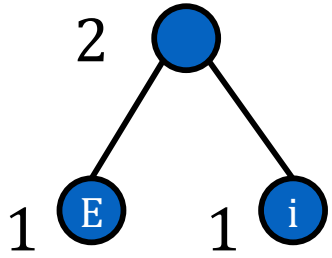
char	E	e	r	i	space	y	s	n	a	l	k	.
Freq.	1	8	2	1	4	1	2	2	2	1	1	1

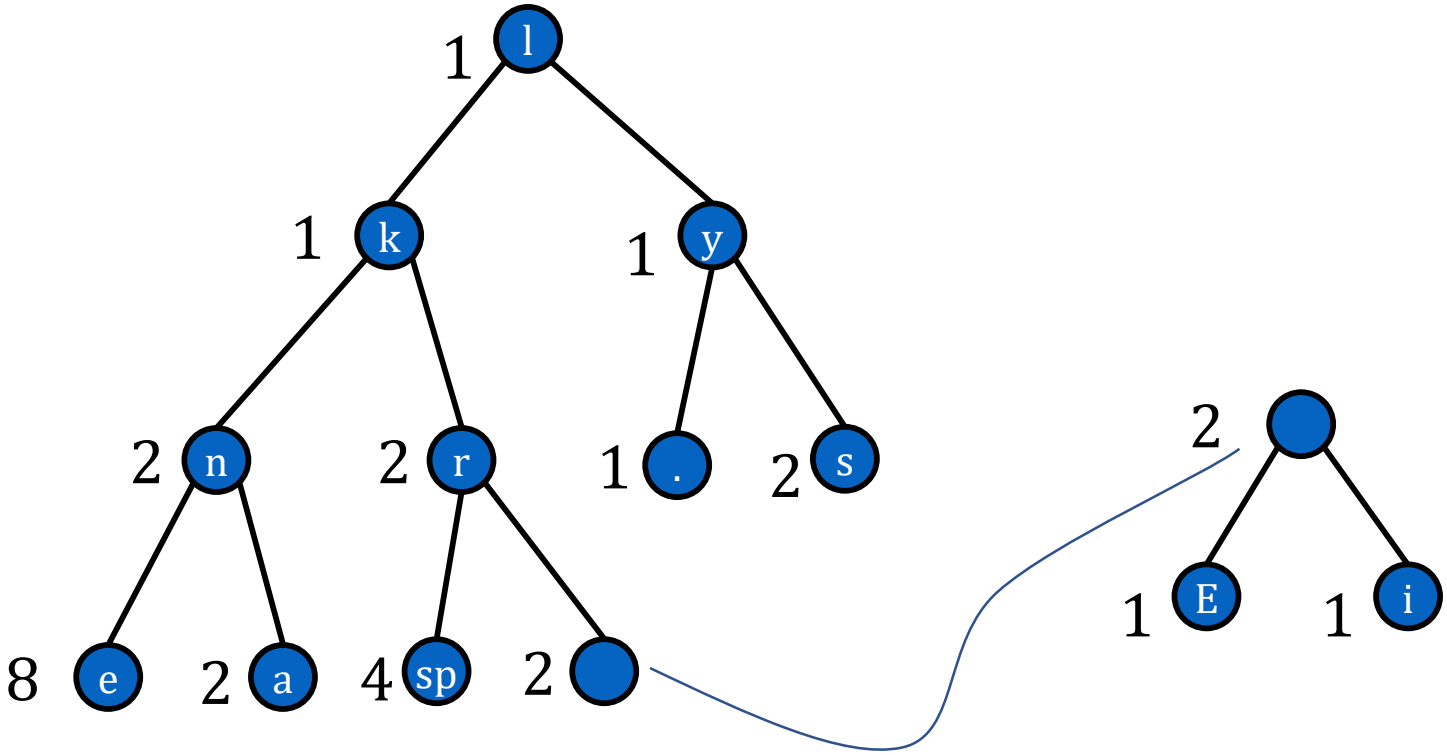


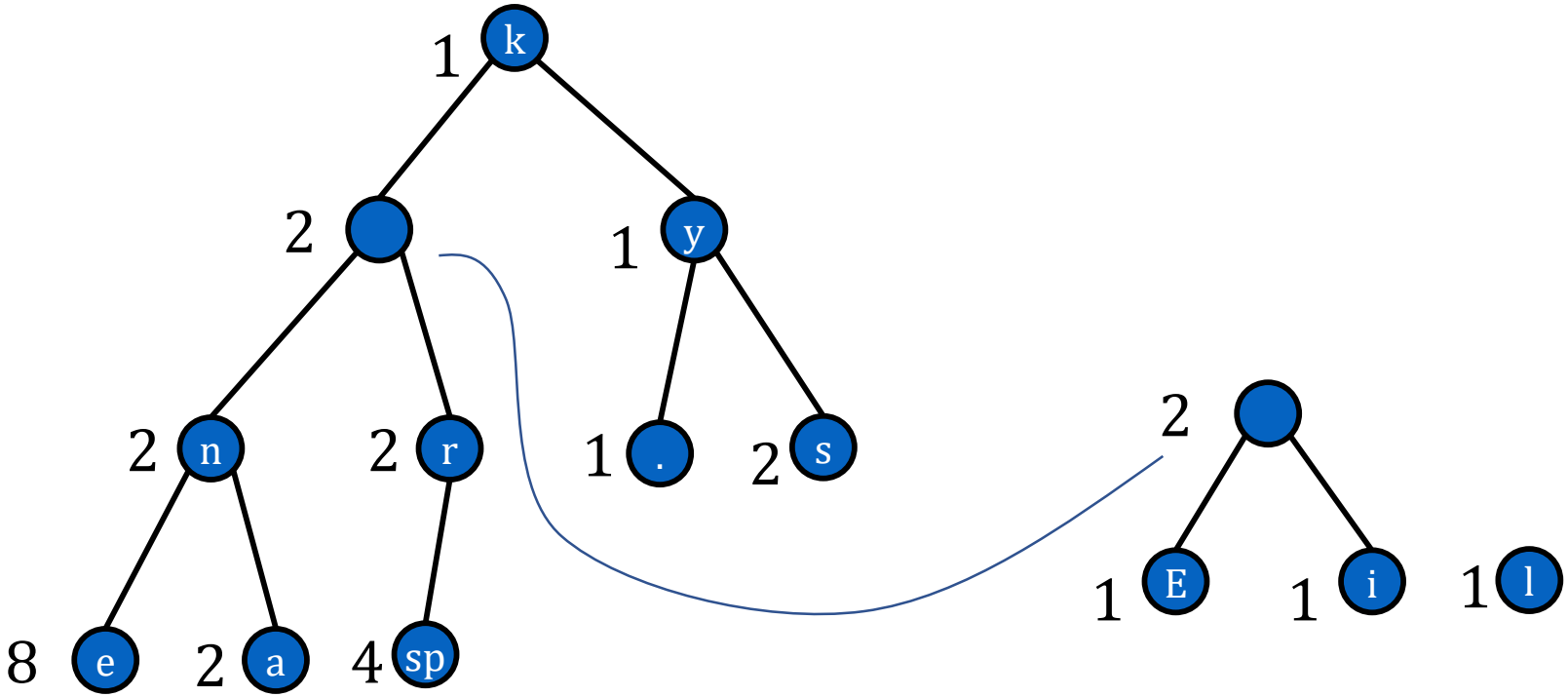

1
i
1
l
1
y
2
n
1
k
1
.
2
s
8
e
2
a
4
sp
2
r

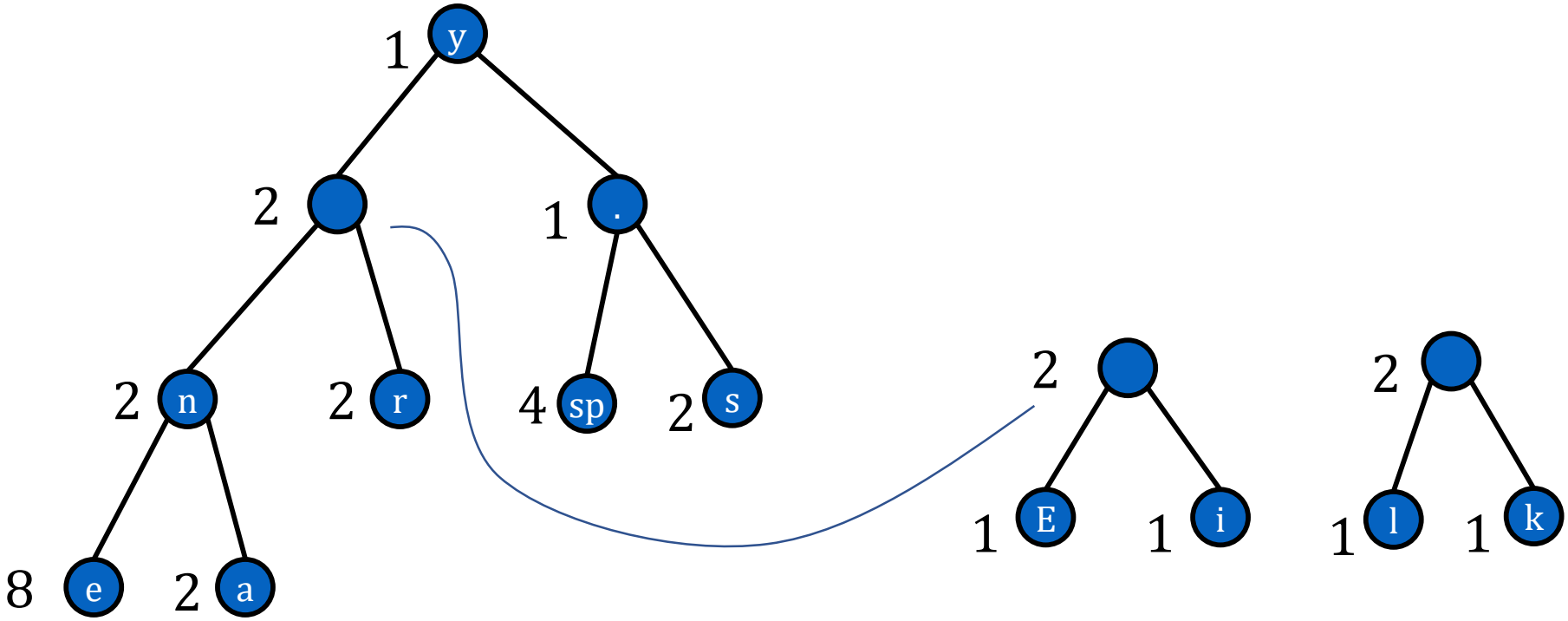

1
E

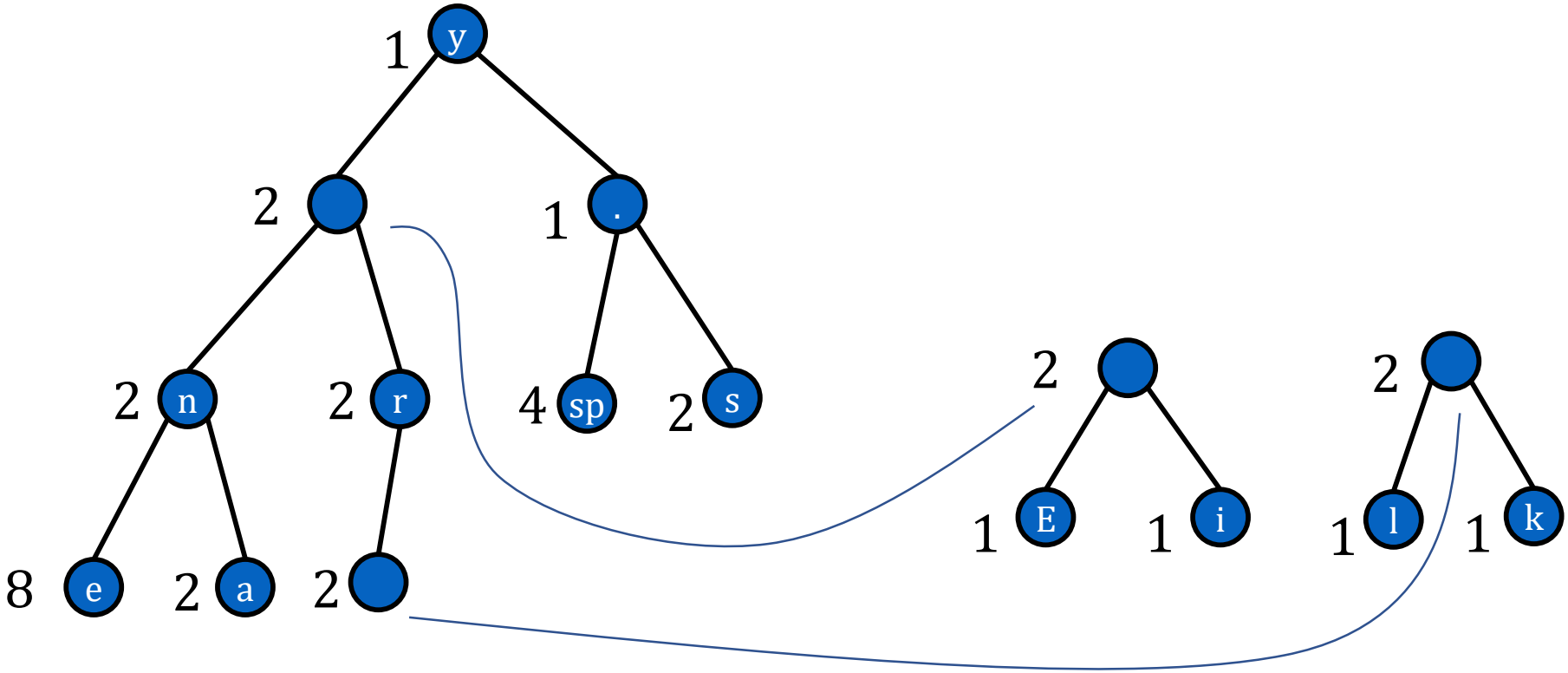

1
l
1
k
1
y
2
n
2
r
1
.
2
s
8
e
2
a
4
sp

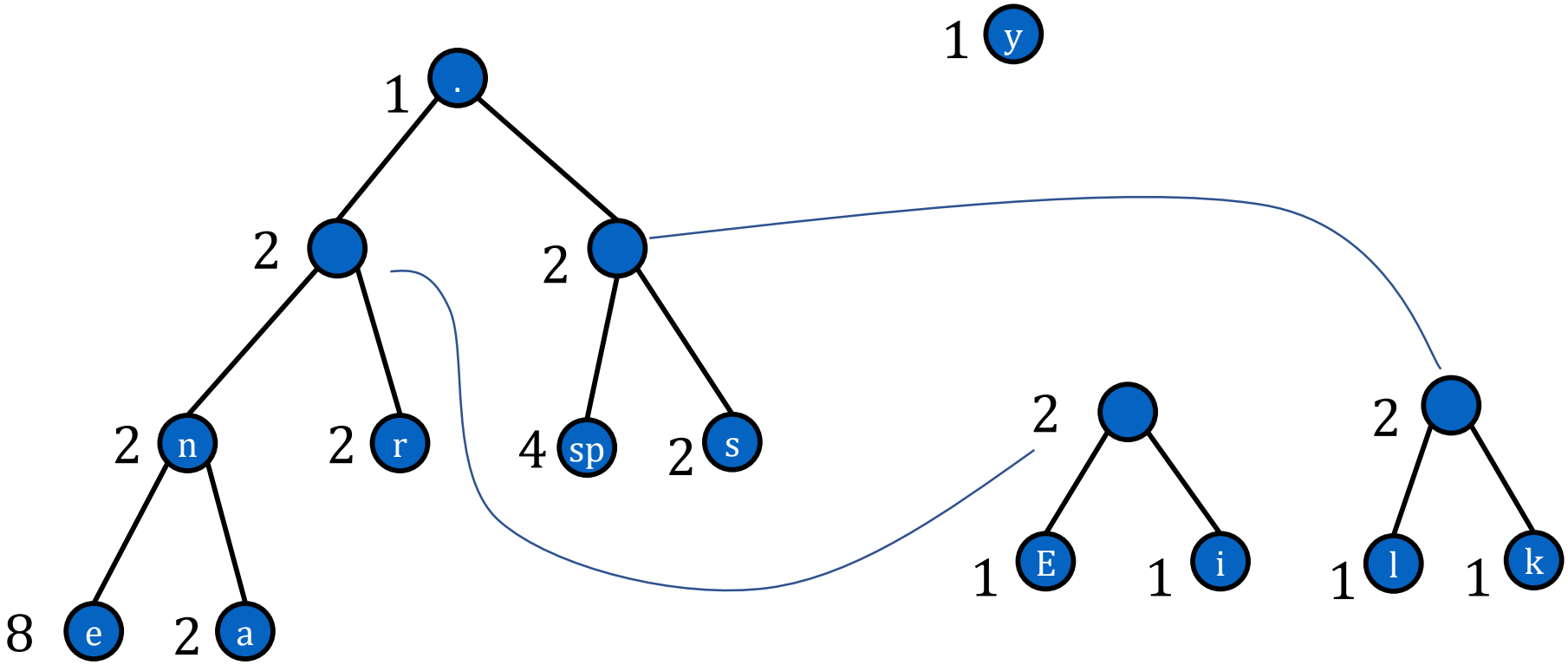

2
1
E
1
i

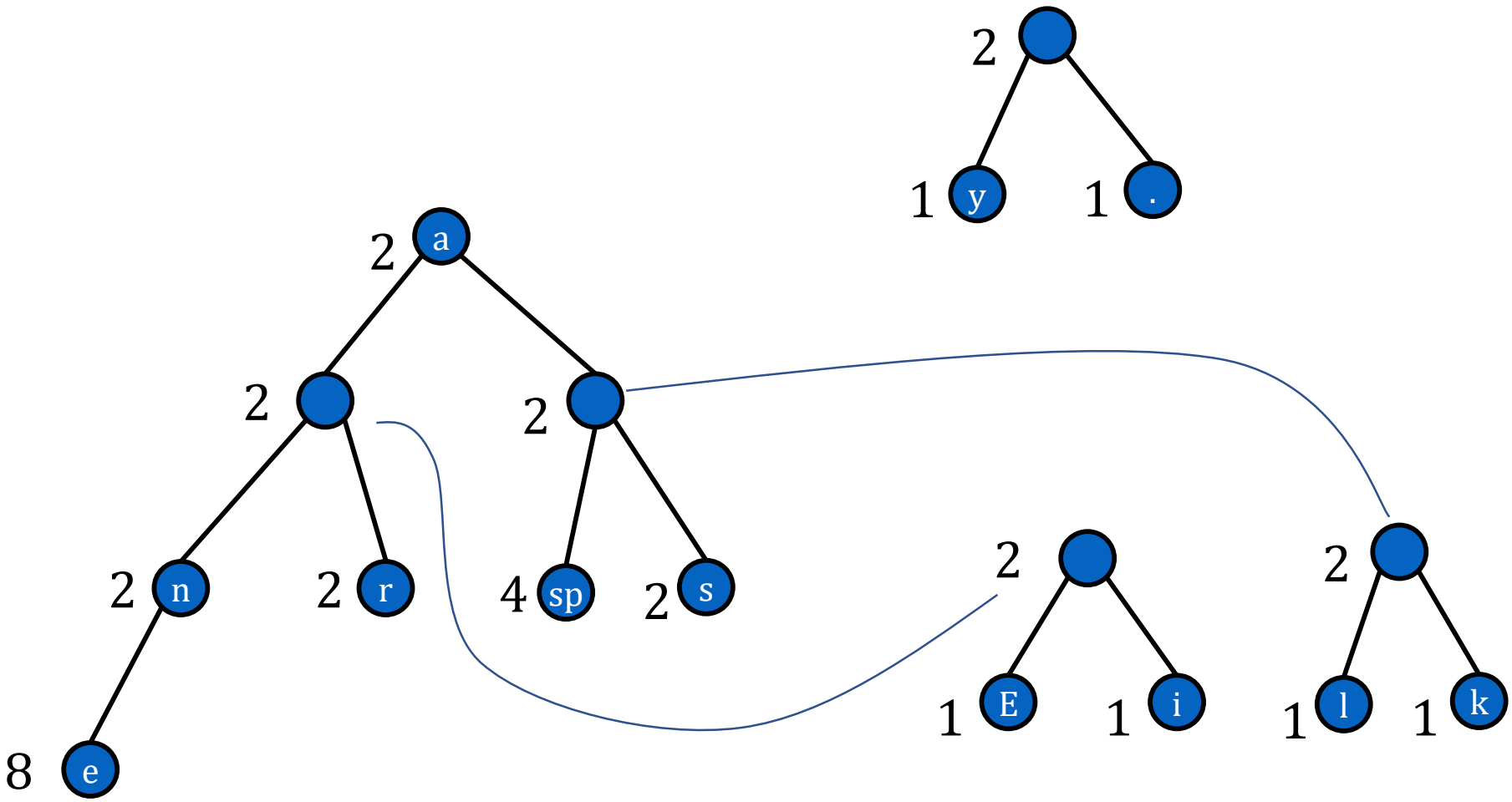

1
l
1
k
1
y
2
n
2
r
1
.
2
s
8
e
2
a
4
sp
2
2
1
E
1
i

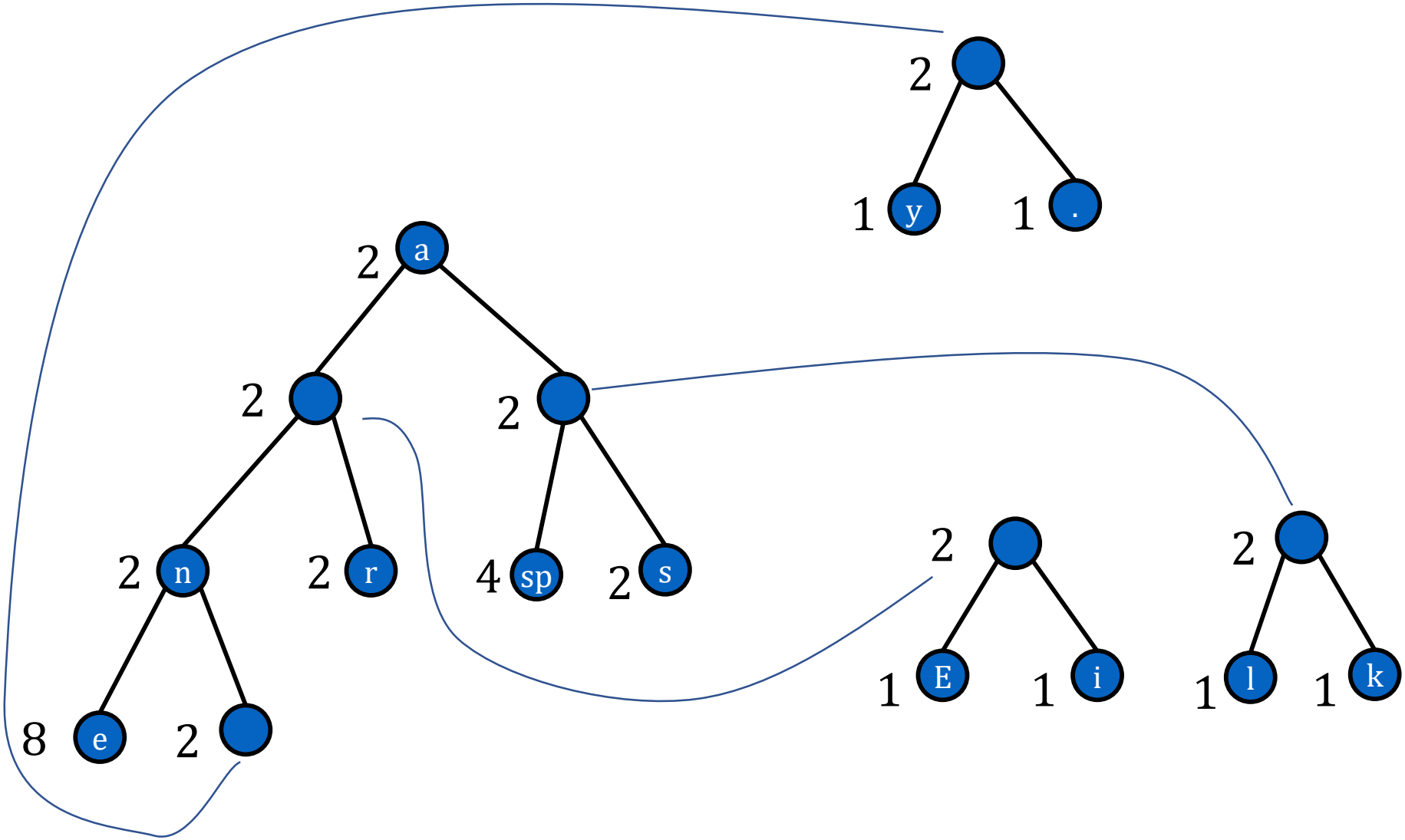

1
k
2
1
y
2
n
2
r
1
.
2
s
8
e
2
a
4
sp
2
1
E
1
i
1
l

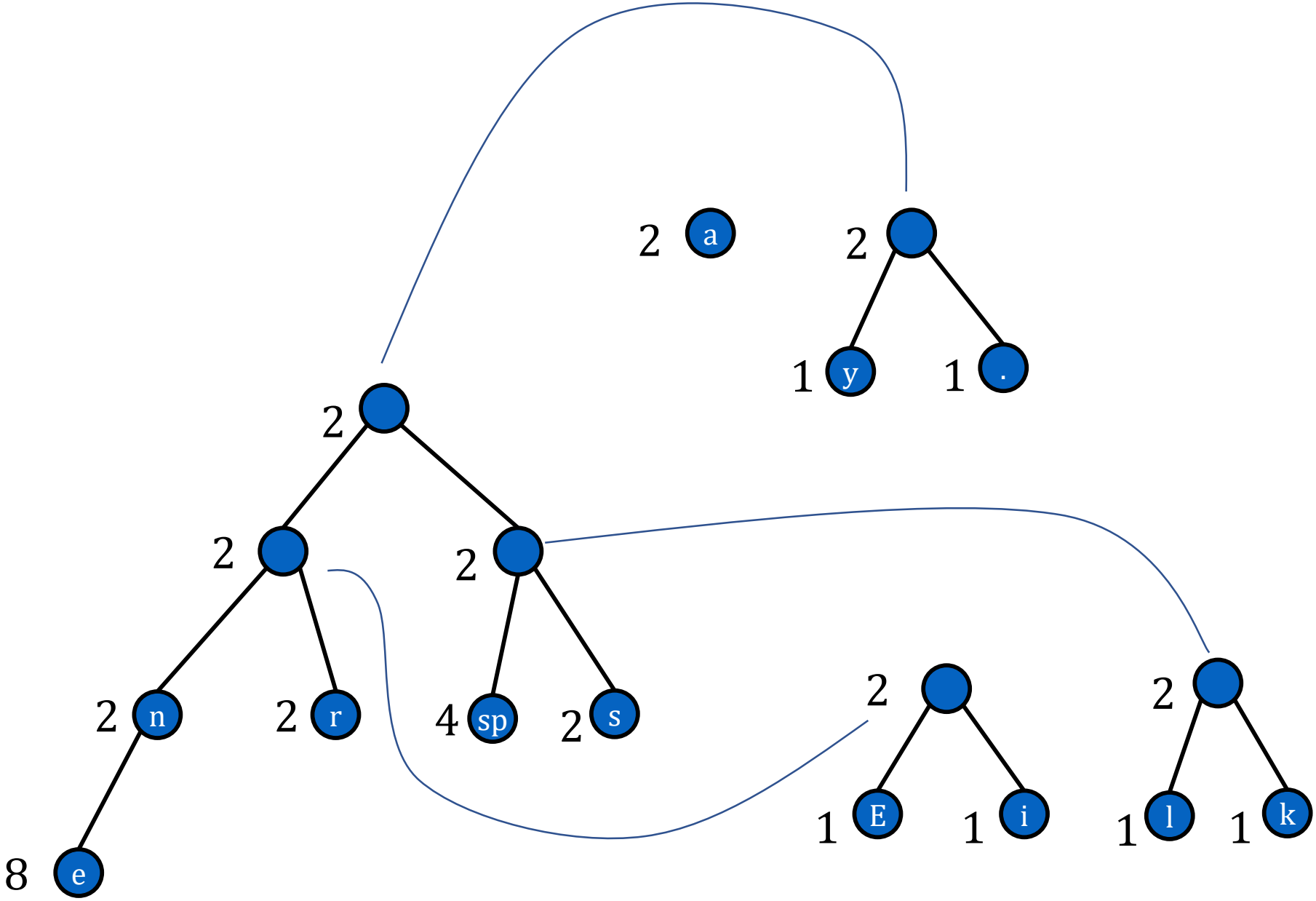

1
y
2
1
.
2
n
2
r
4
sp
2
s
8
e
2
a
2
1
E
1
i
2
1
l
1
k

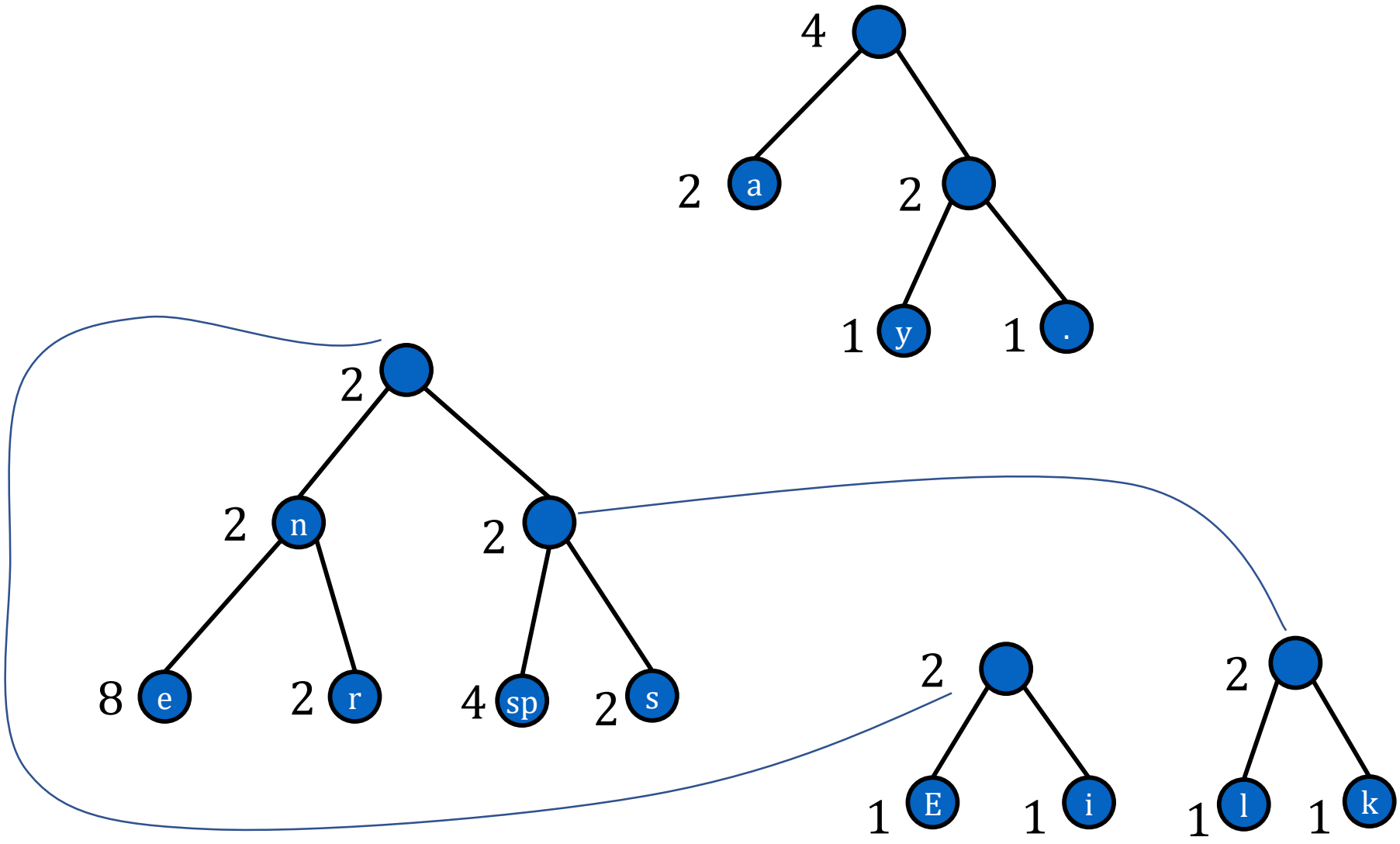

1
y
2
1
.
2
n
2
r
4
sp
2
s
8
e
2
a
2
2
1
E
1
i
2
1
l
1
k
2

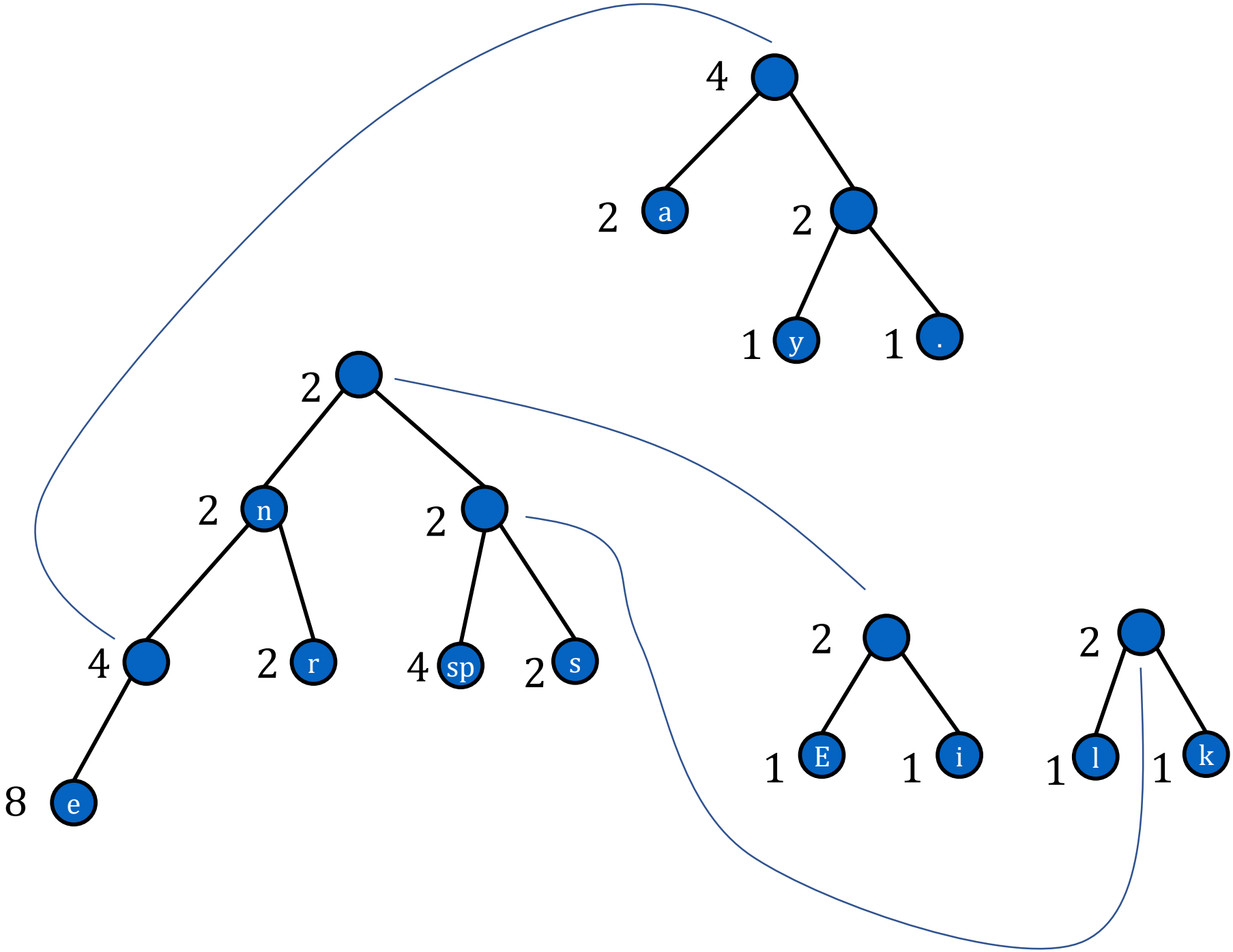

1 y
1 .
2
2
2 n
2 r
4 sp
2 s
2
2
8 e
2 a
1 E
1 i
1 l
1 k

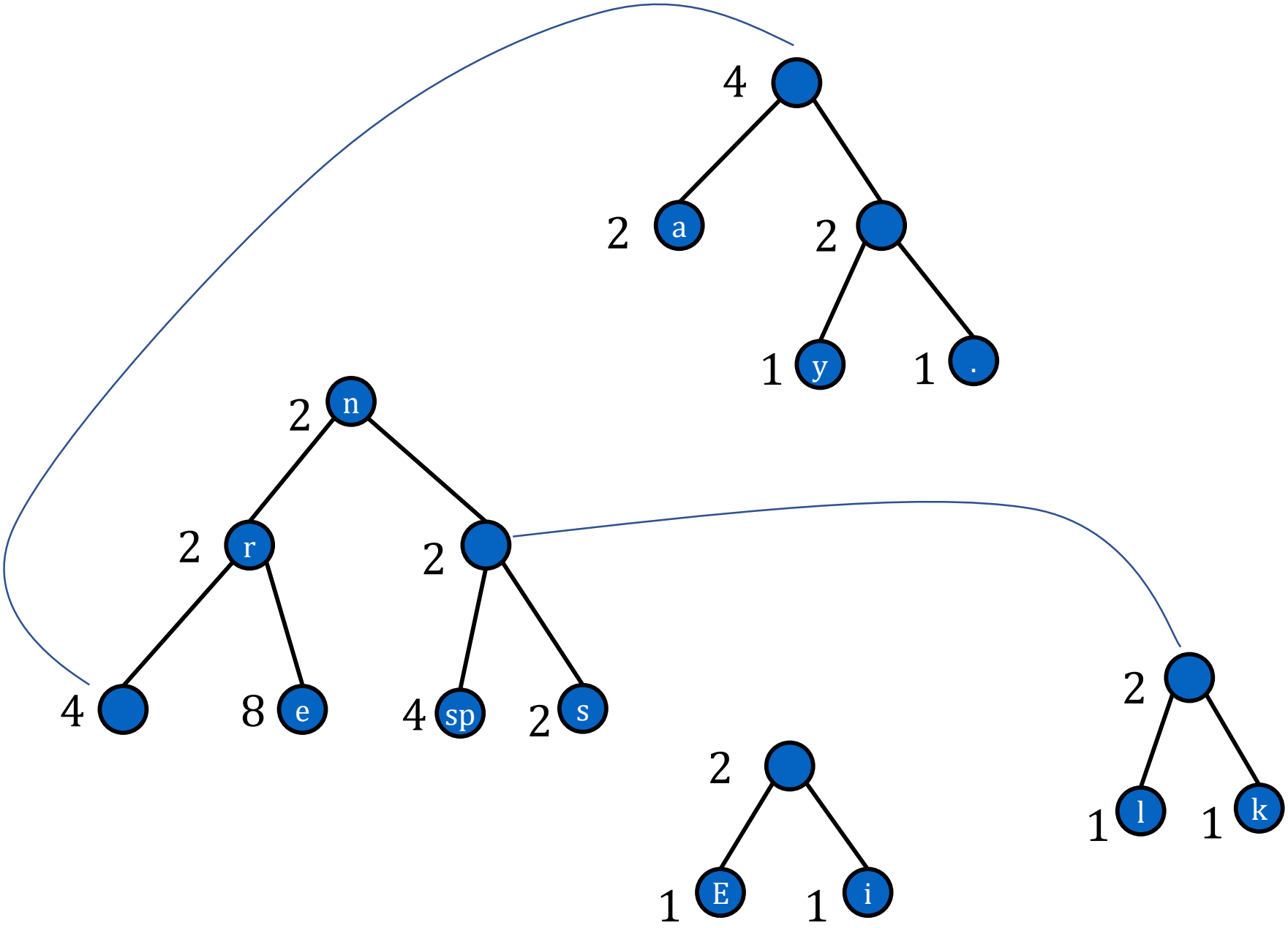

2
1 y
1 .
2 a
2
2
2 n
2 r
4 sp
2 s
8 e
2
1 E
1 i
2
1 l
1 k

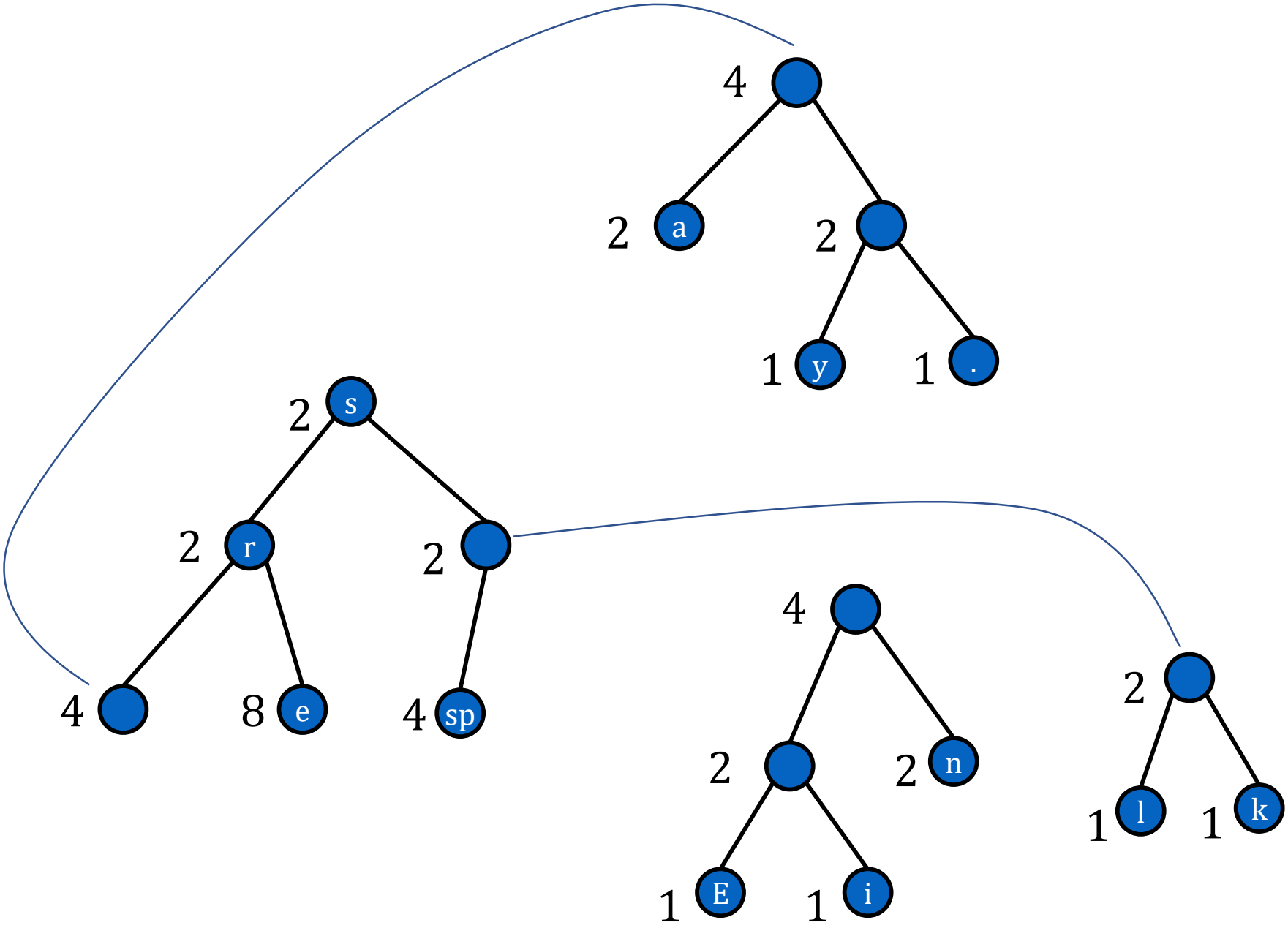

2
1 y
1 .
2 a
2
2
2 n
2 r
4 sp
2 s
8 e
2
2
1 E
1 i
2
1 l
1 k

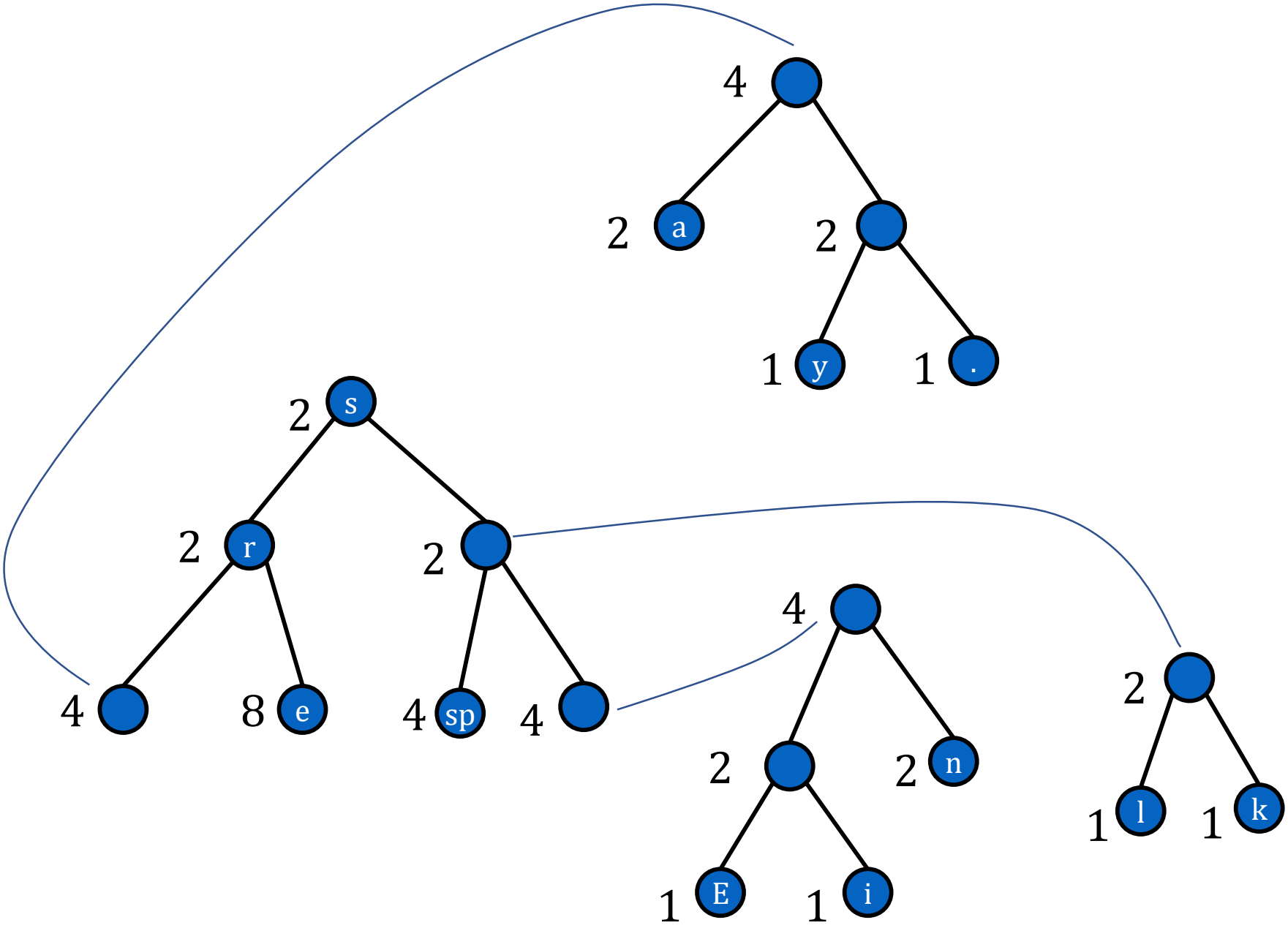

2
a
2
1
y
1
.
2
2
2
2
n
2
r
4
sp
2
s
8
e
2
1
E
1
i
2
1
l
1
k

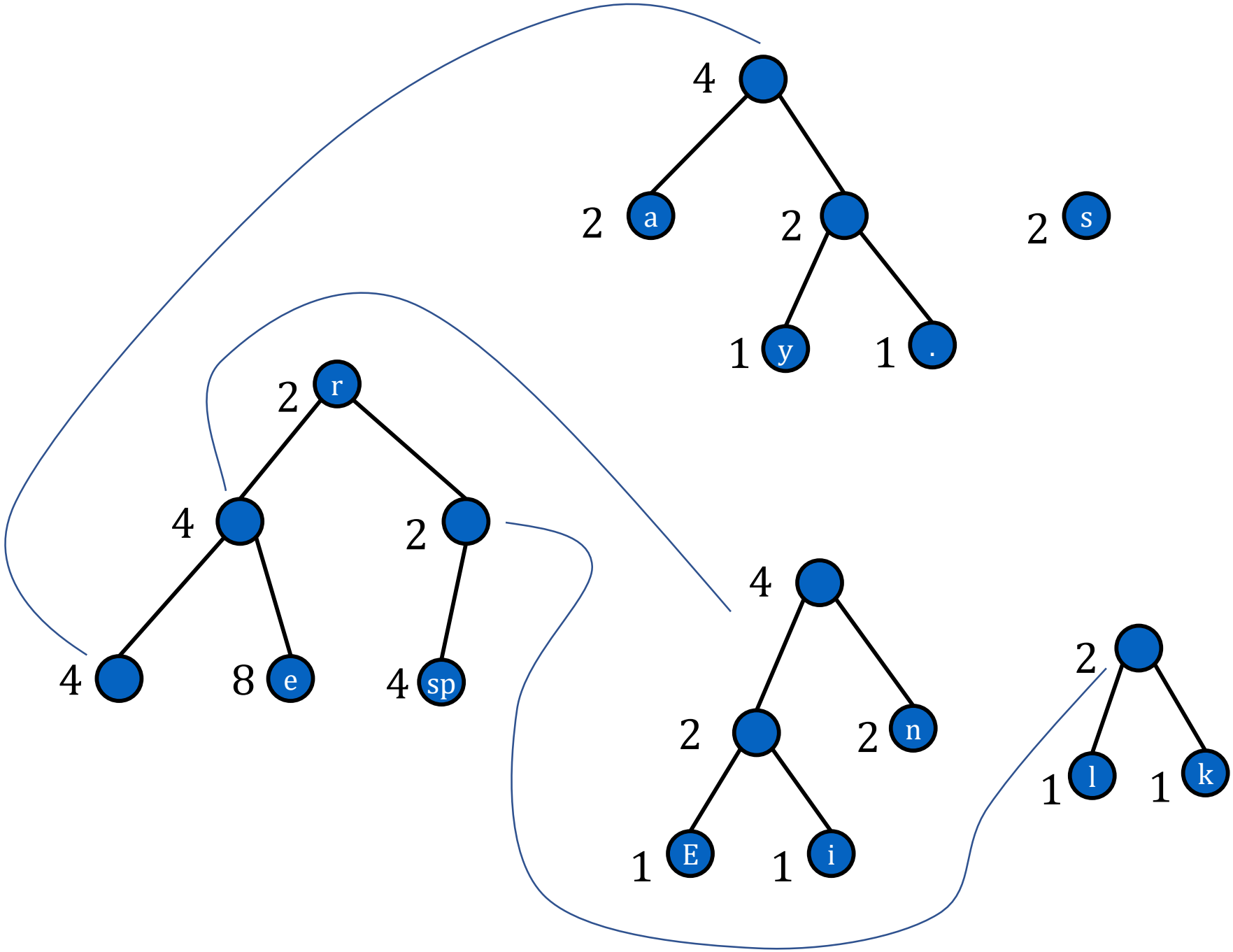

4
2
a
2
1
y
1
.
2
2
n
2
8
e
2
r
4
sp
2
s
2
1
E
1
i
2
1
l
1
k

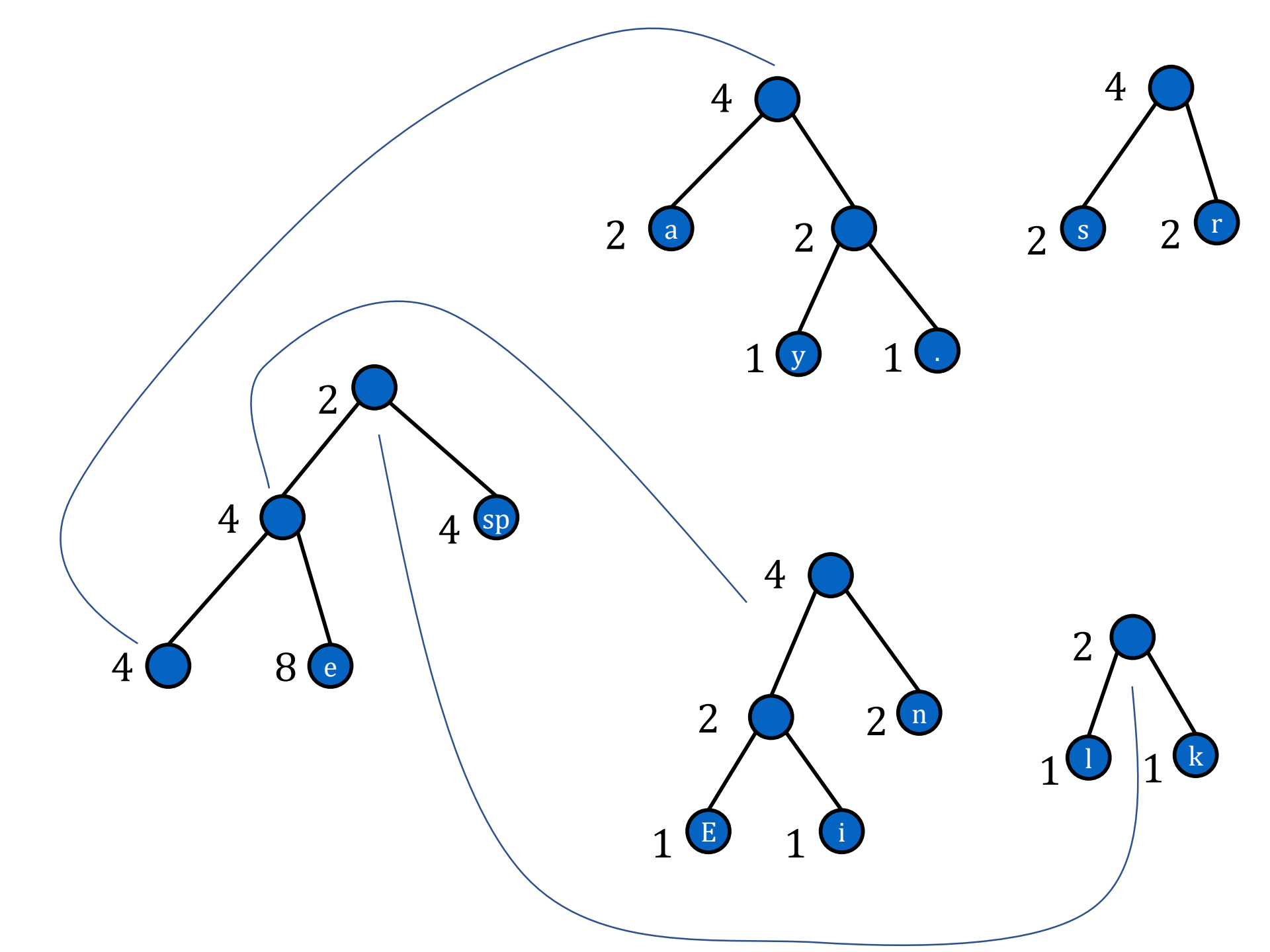

4
2
a
2
1
y
1
.
2
2
n
2
4
2
r
4
sp
2
s
8
e
2
1
E
1
i
2
1
l
1
k

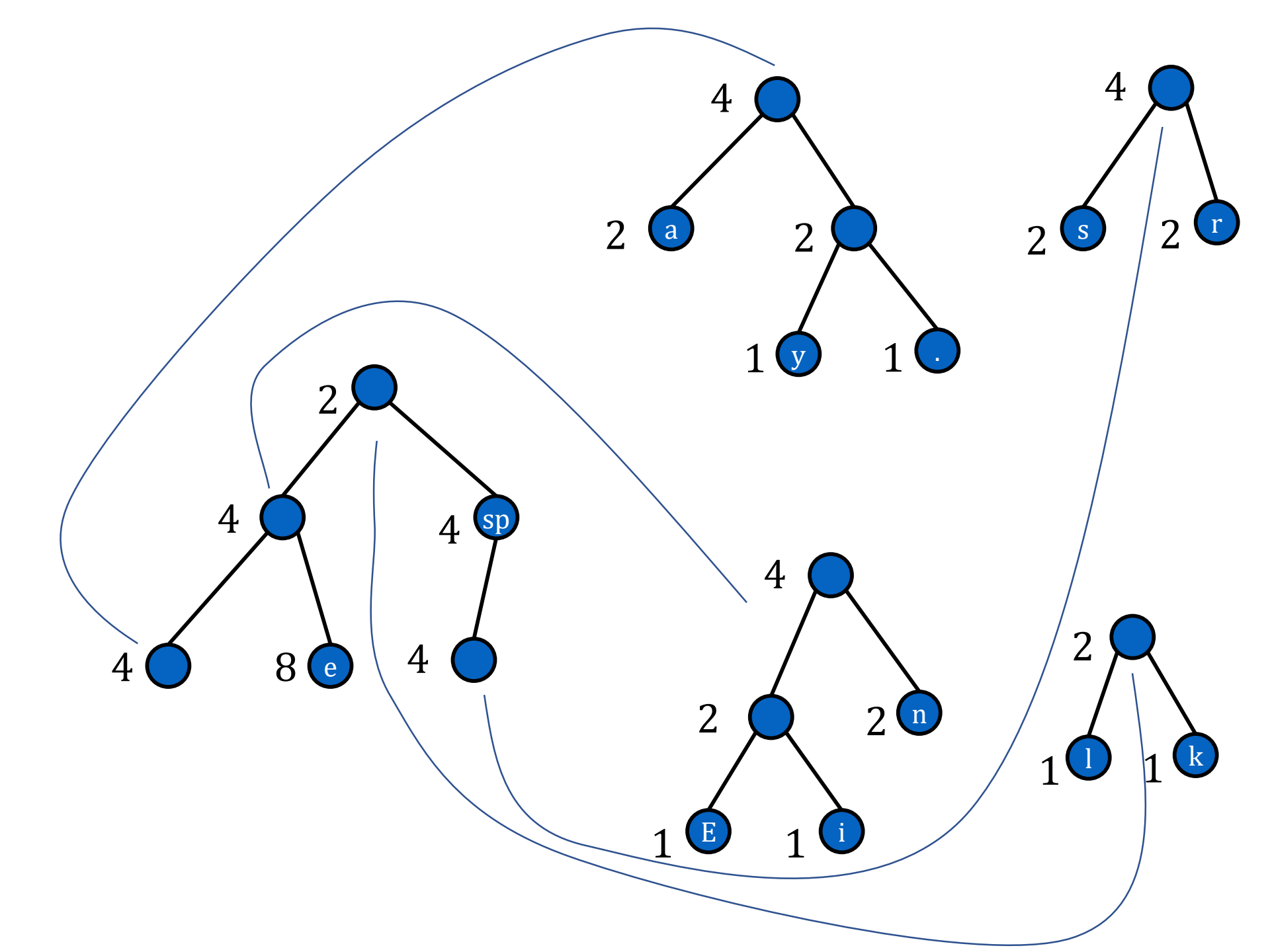

4
2
a
2
1
y
1
.
2
n
2
r
2
4
8
e
4
sp
2
s
2
1
E
1
i
2
1
l
1
k

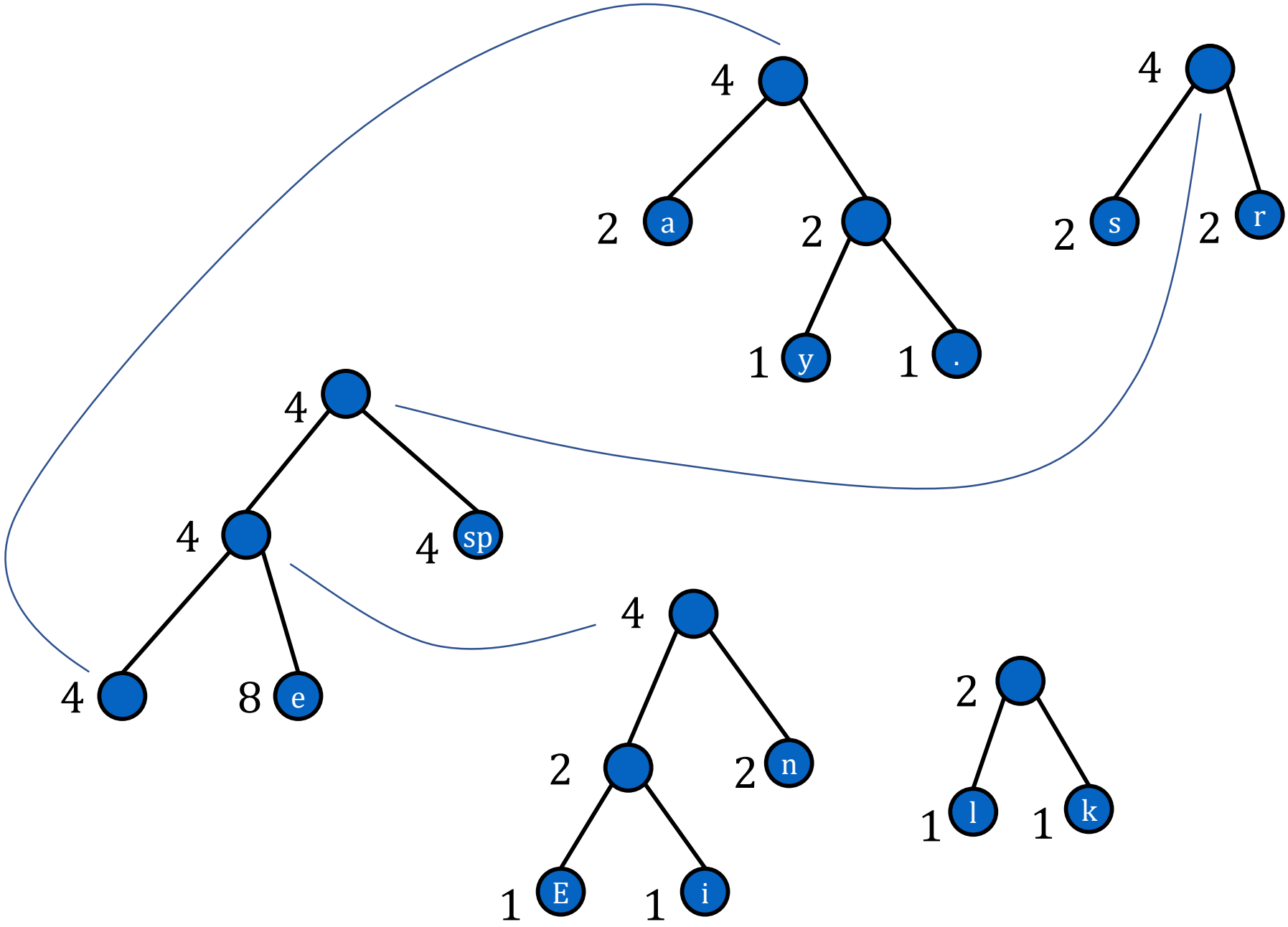

4
2
a
2
1
y
1
.
2
s
2
r
2
4
8
e
4
sp
4
2
n
1
E
1
i
2
1
l
1
k

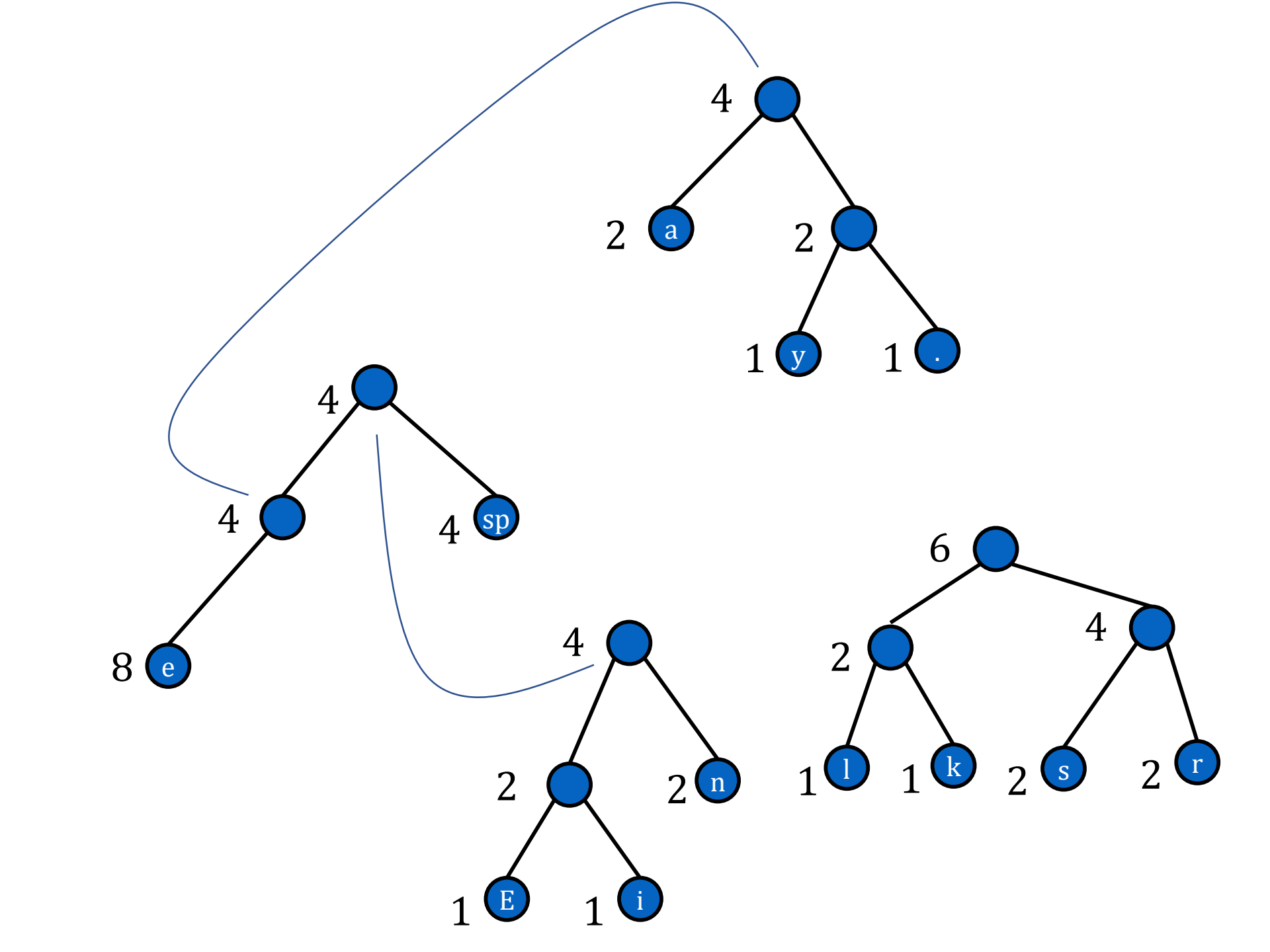

4
2
a
2
1
y
1
.
2
s
2
r
2
4
8
e
4
sp
4
4
2
2
n
1
E
1
i
2
1
l
1
k

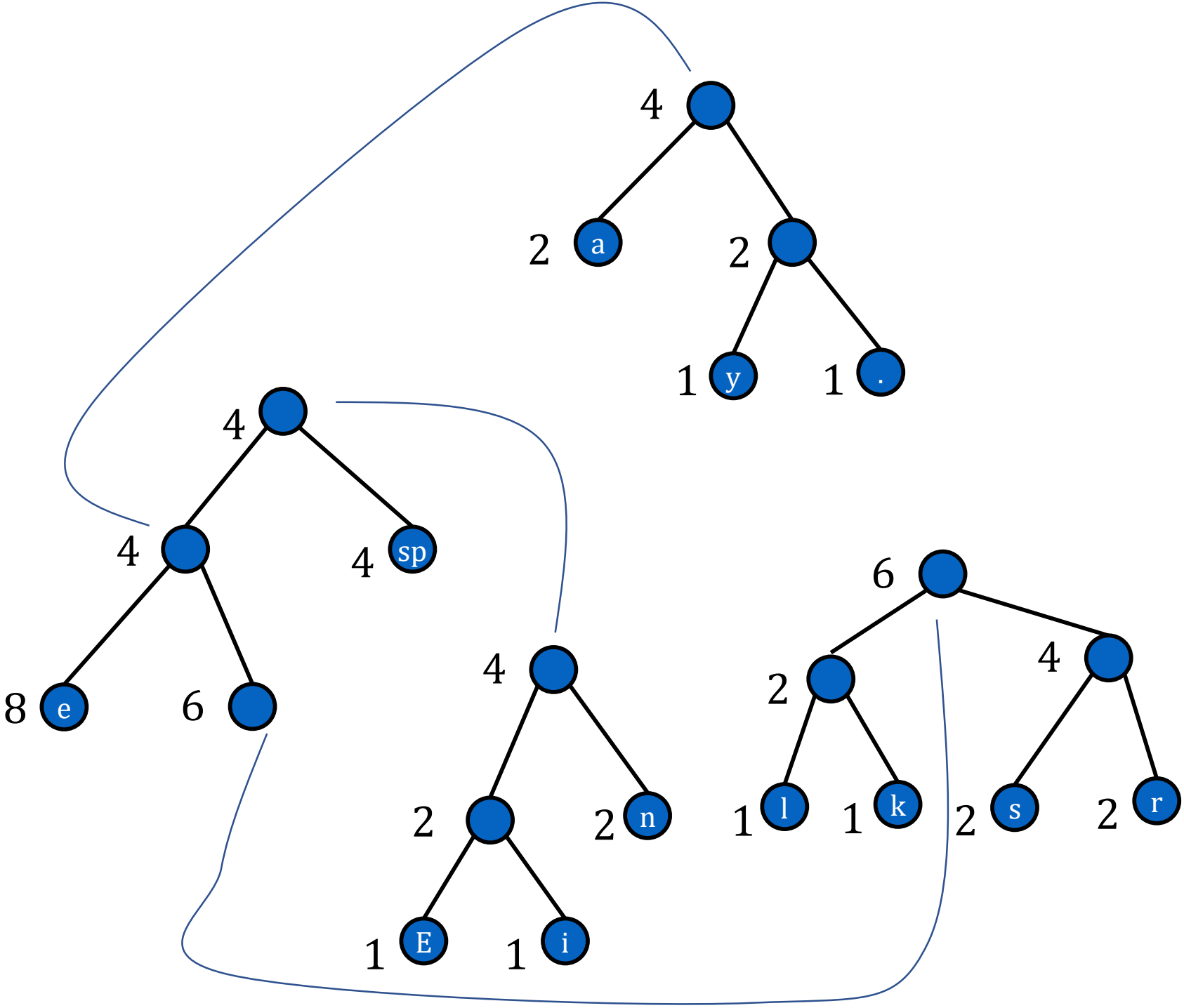

4
2
a
2
2
s
1
y
1
.
2
r
4
2
4
8
e
4
sp
4
2
2
n
1
E
1
i
2
1
l
1
k

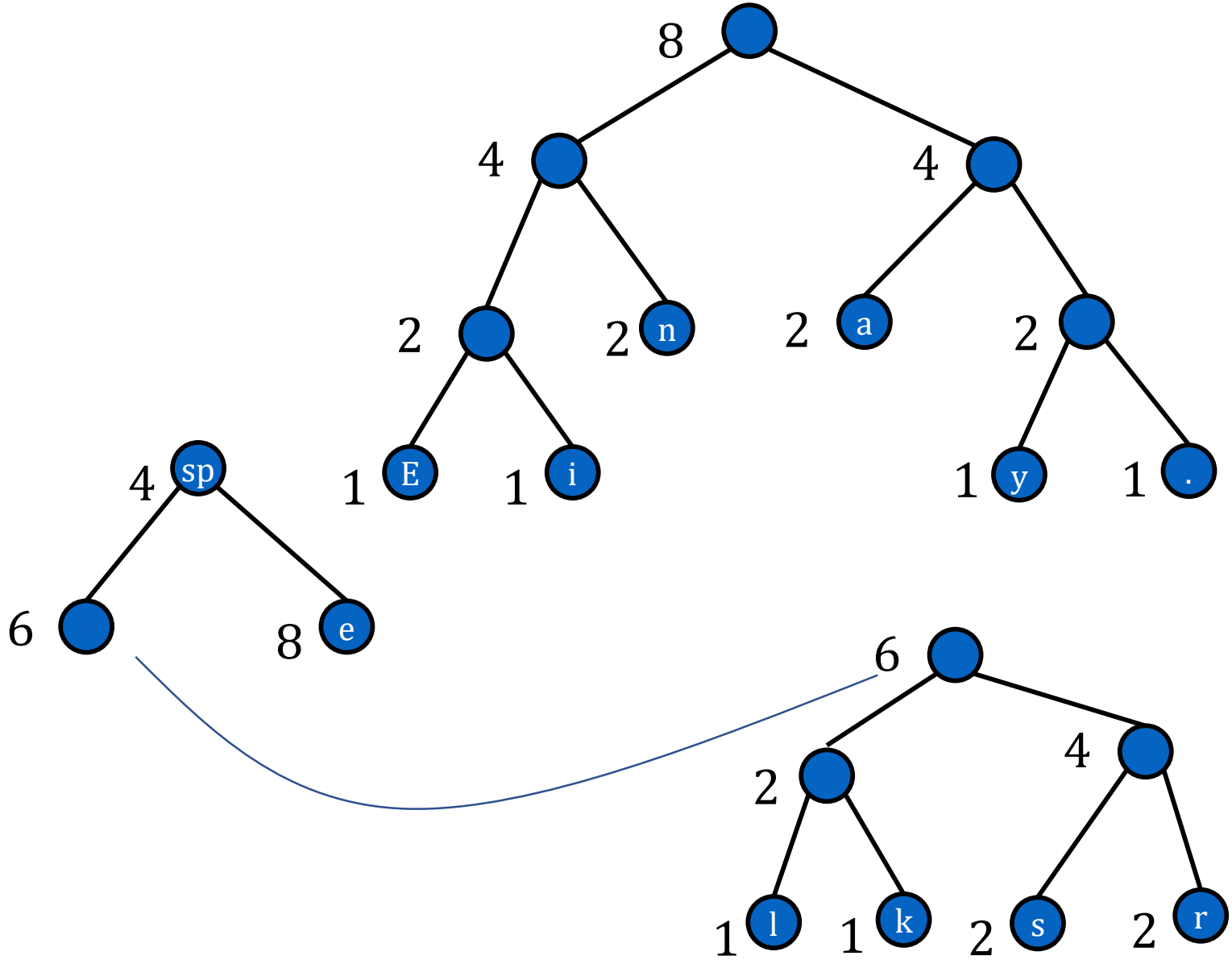

4
2
a
2
1
y
1
.
4
2
s
2
r
2
4
sp
4
4
8
e
4
2
n
2
1
E
1
i
2
1
l
1
k

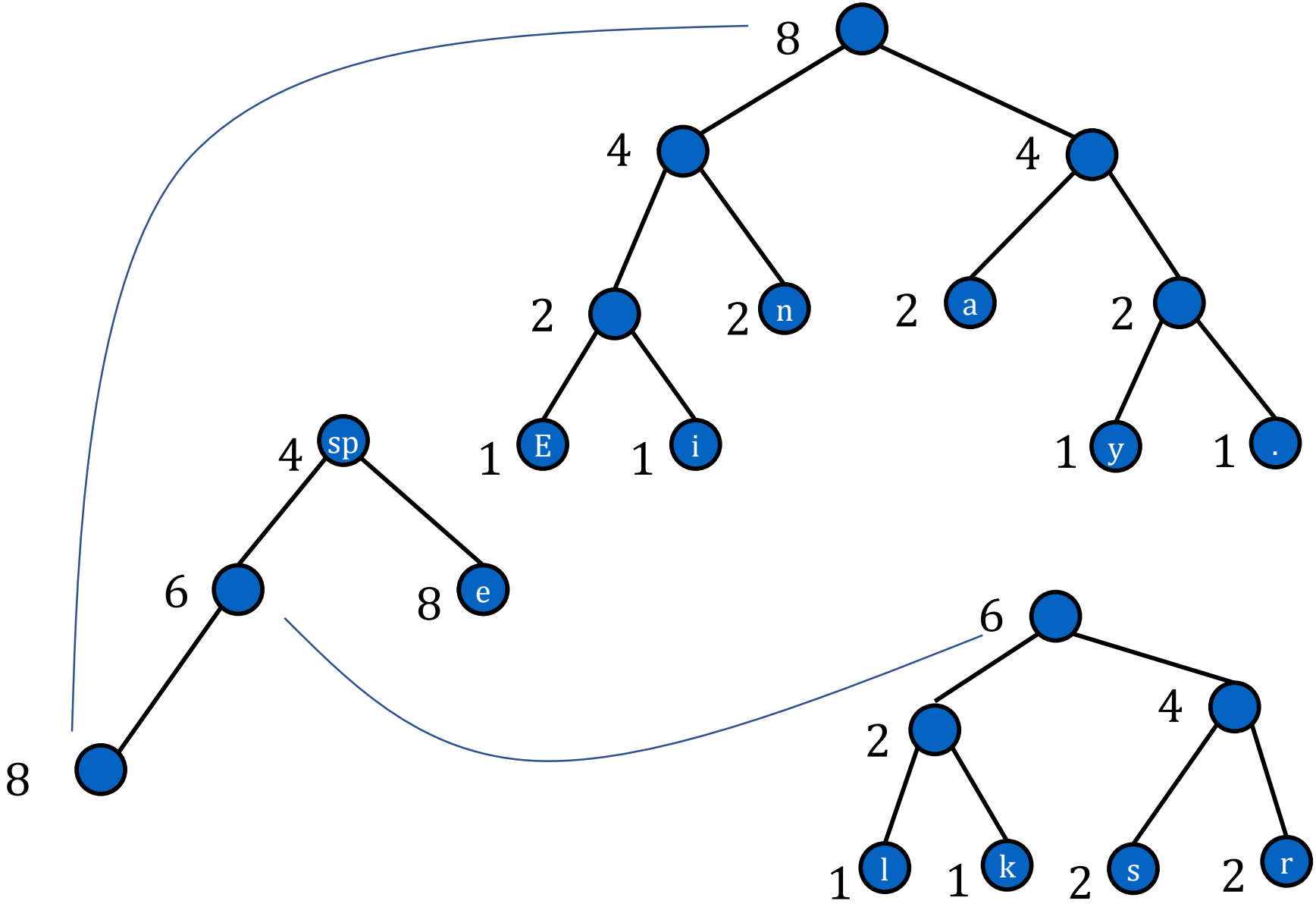

4
2
a
2
1
y
1
.
4
2
s
2
r
2
4
sp
4
4
8
e
4
4
2
2
n
1
E
1
i
2
1
l
1
k

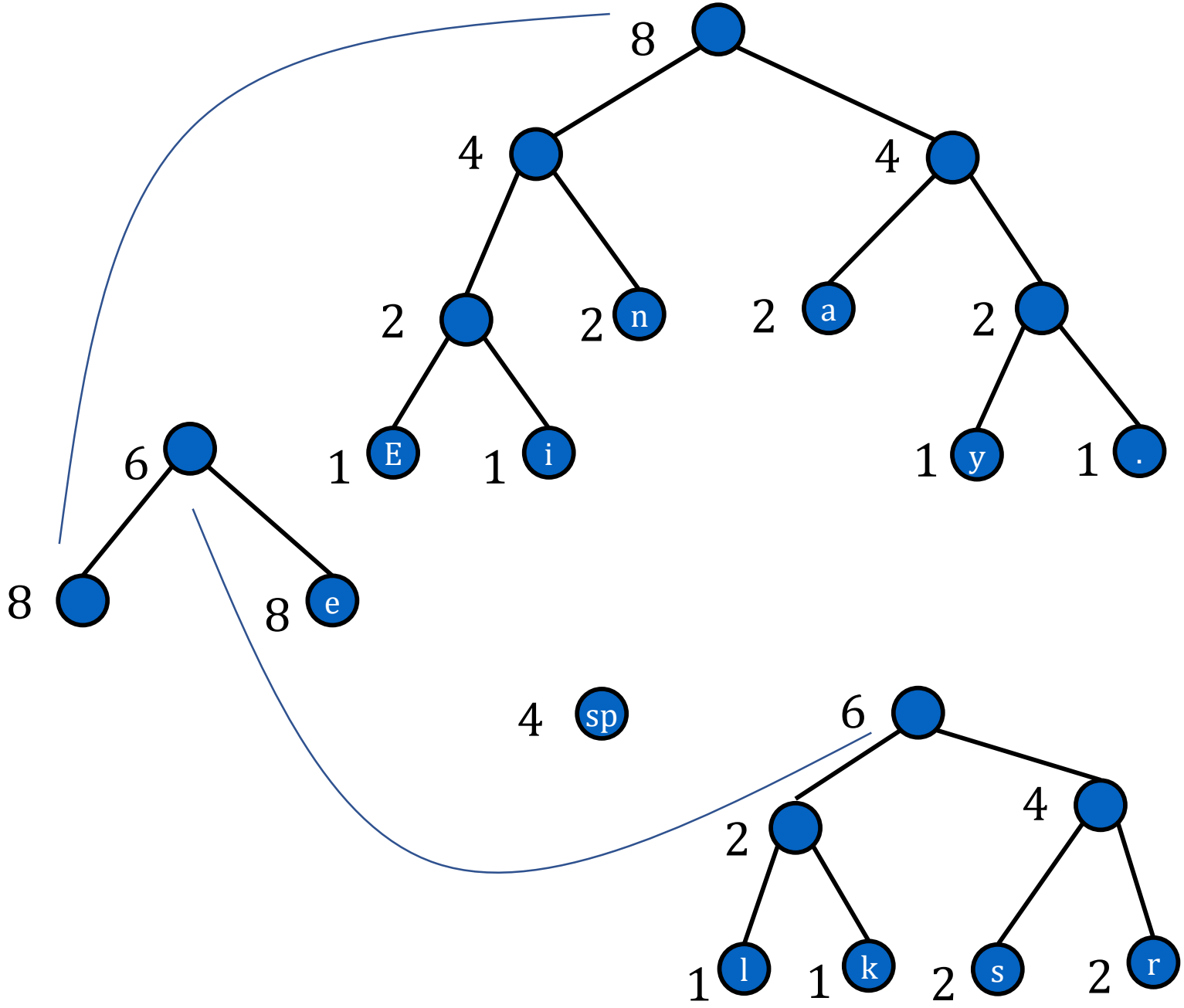

4
2
a
2
1
y
1
.
4
2
s
2
r
4
4
4
sp
4
8
e
4
2
2
n
1
E
1
i
2
1
l
1
k

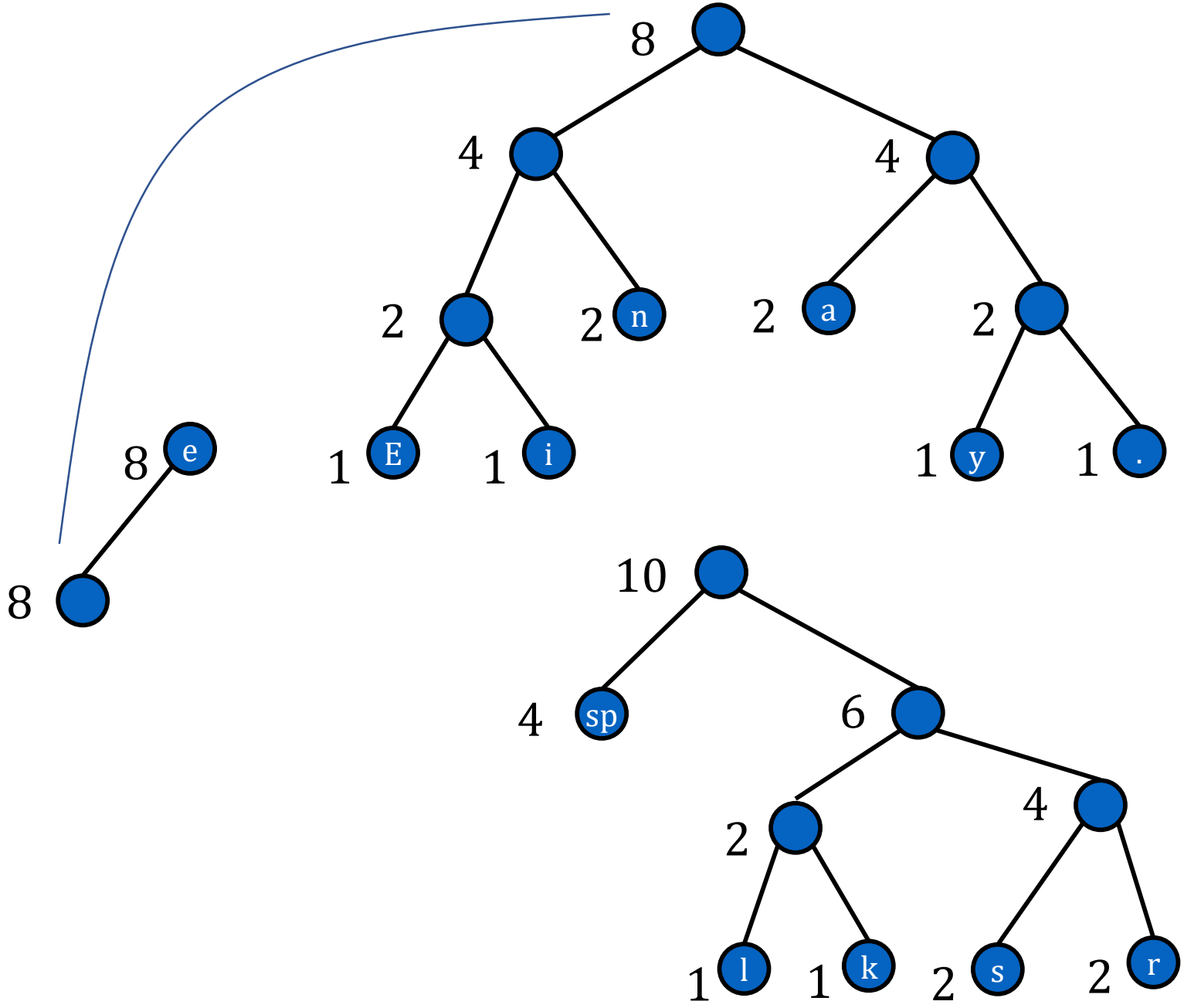

4
2
a
2
1
y
1
.
4
4
4
sp
8
e
4
2
2
n
1
E
1
i
6
2
4
1
l
1
k
2
s
2
r

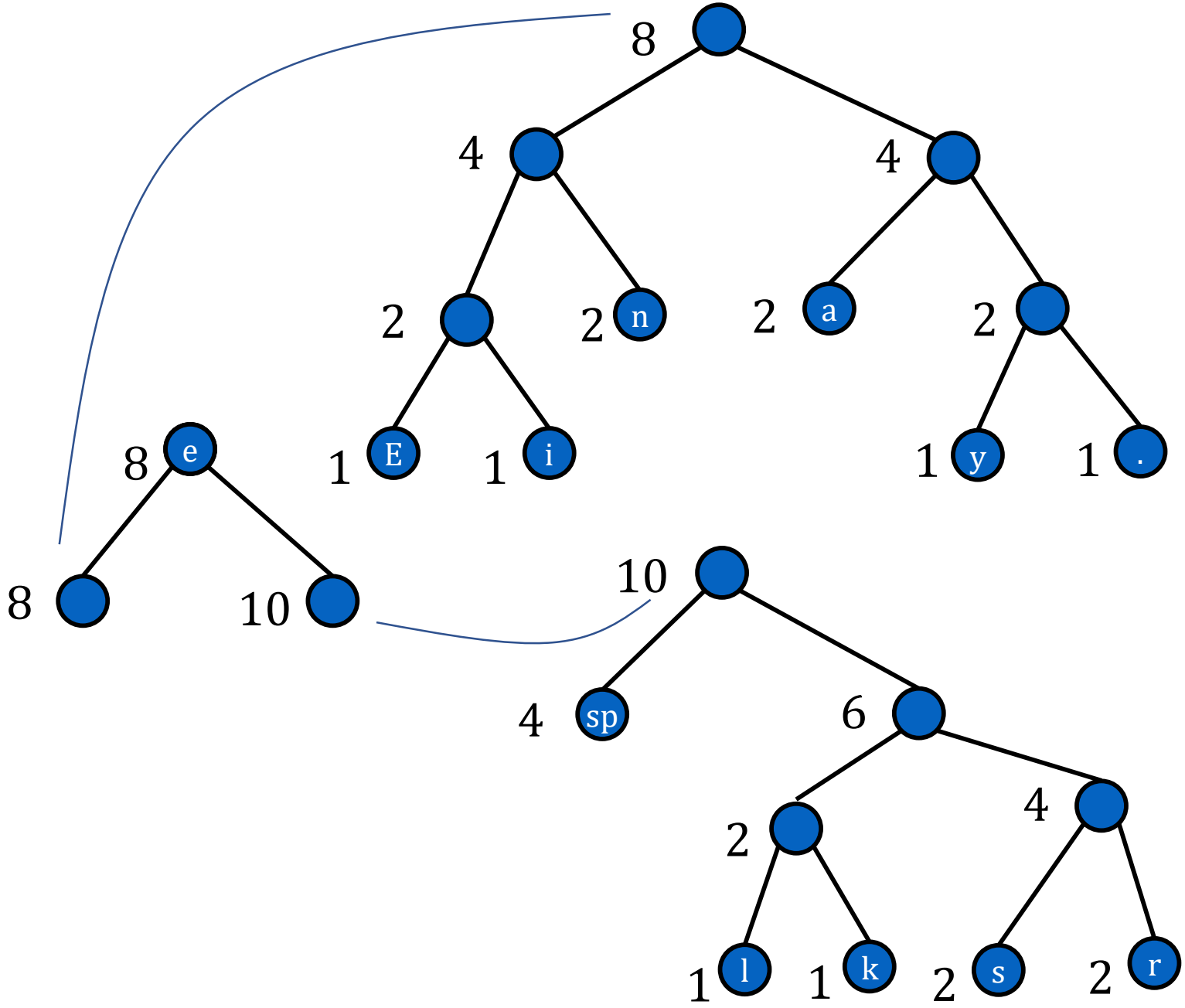

4
2
a
2
1
y
1
.
4
4
4
sp
8
e
6
4
2
2
n
1
E
1
i
6
2
4
1
l
1
k
2
s
2
r

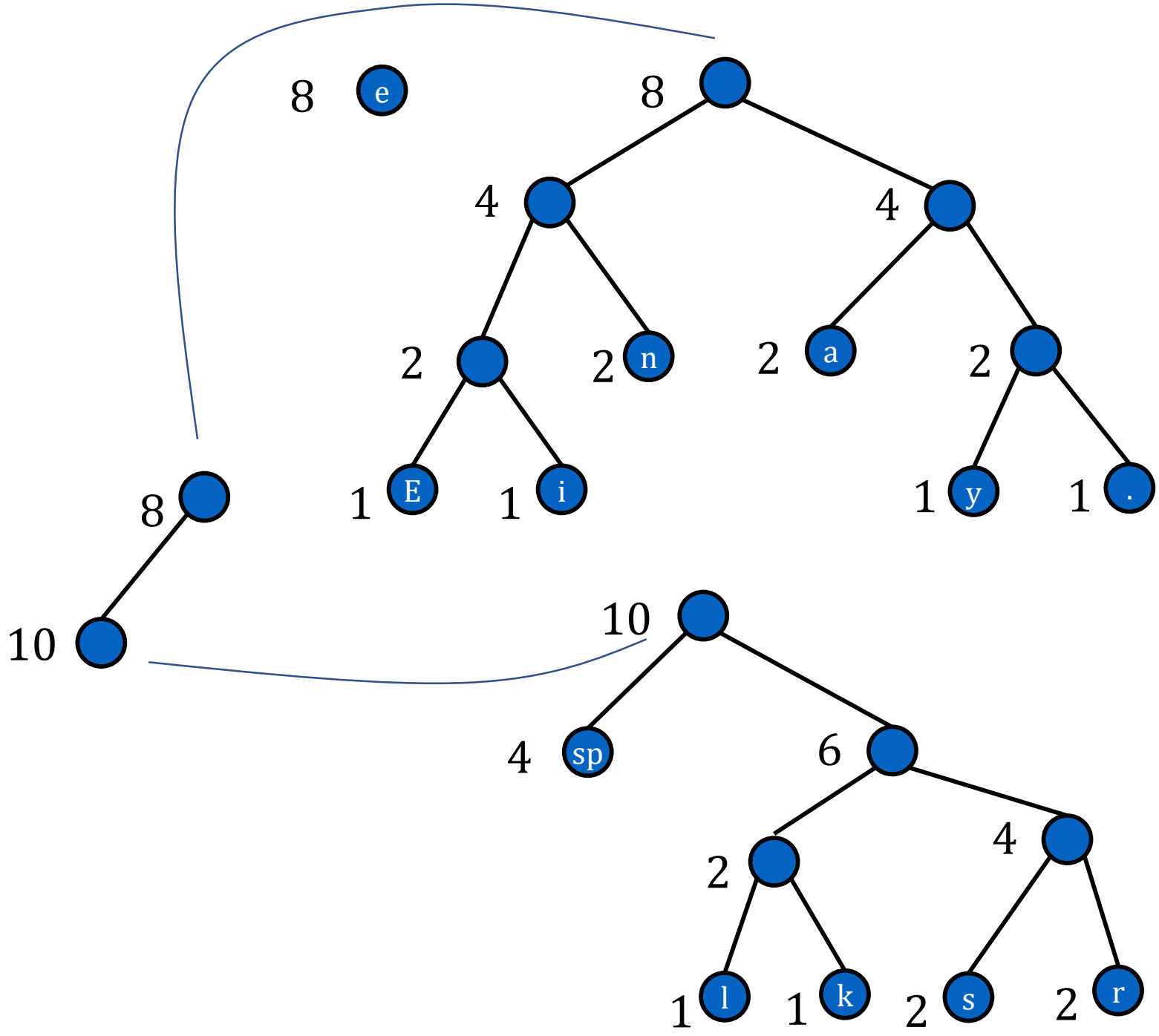

8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
4
sp
6
8
e
6
2
4
1
l
1
k
2
s
2
r

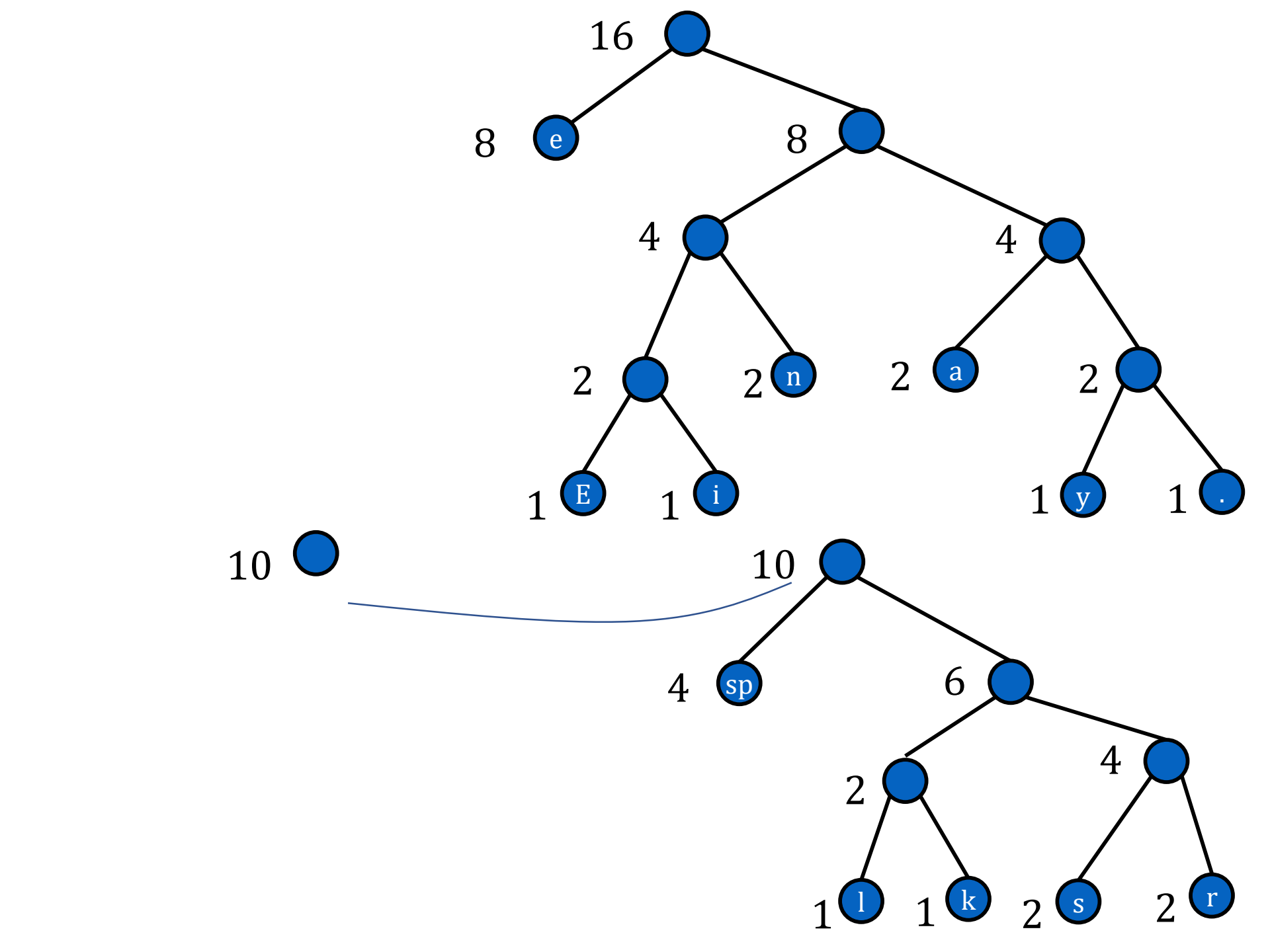

8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
4
sp
6
8
e
8
6
2
4
1
l
1
k
2
s
2
r

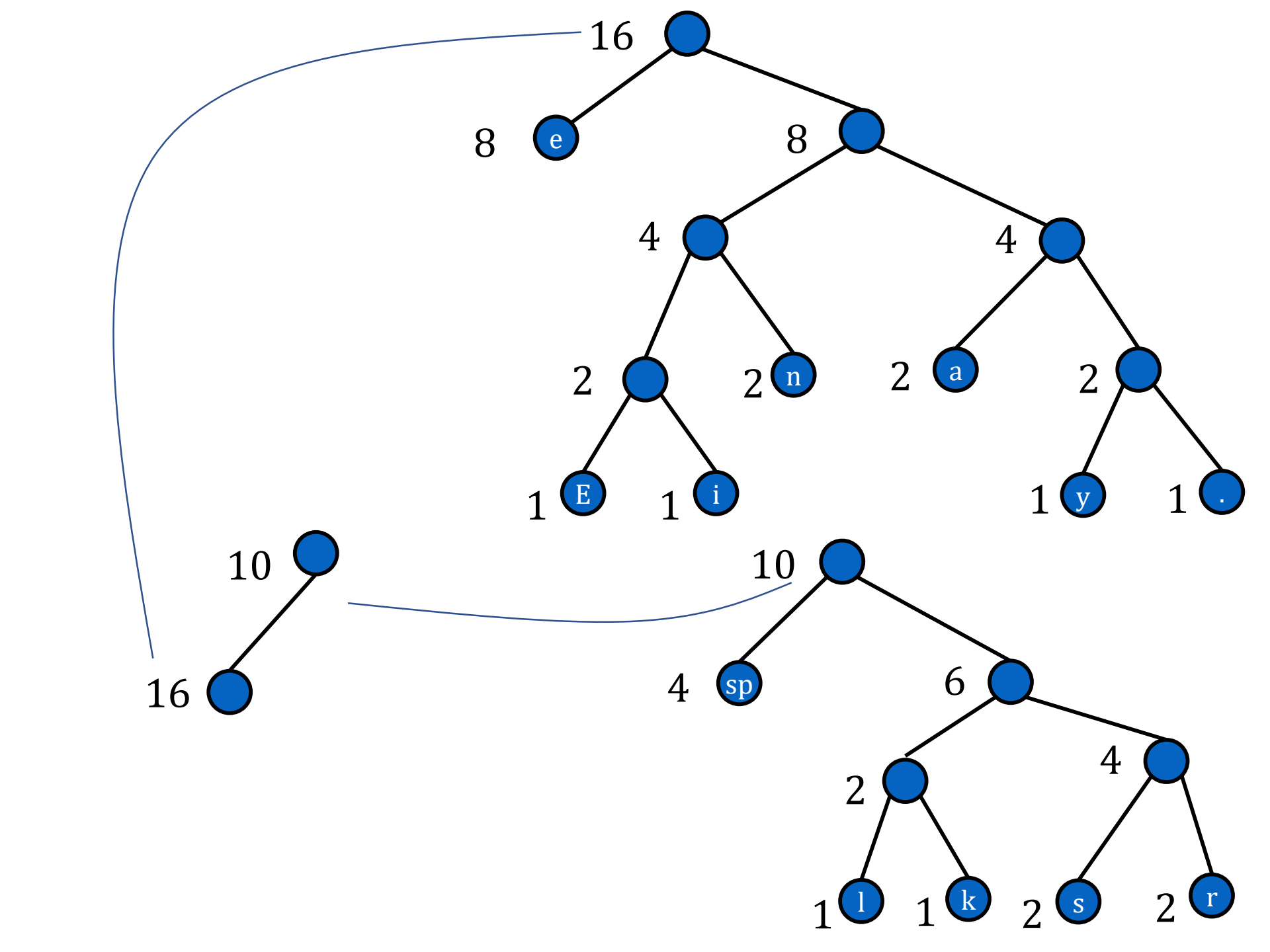

8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
6
8
8
e
4
sp
6
2
4
1
l
1
k
2
s
2
r

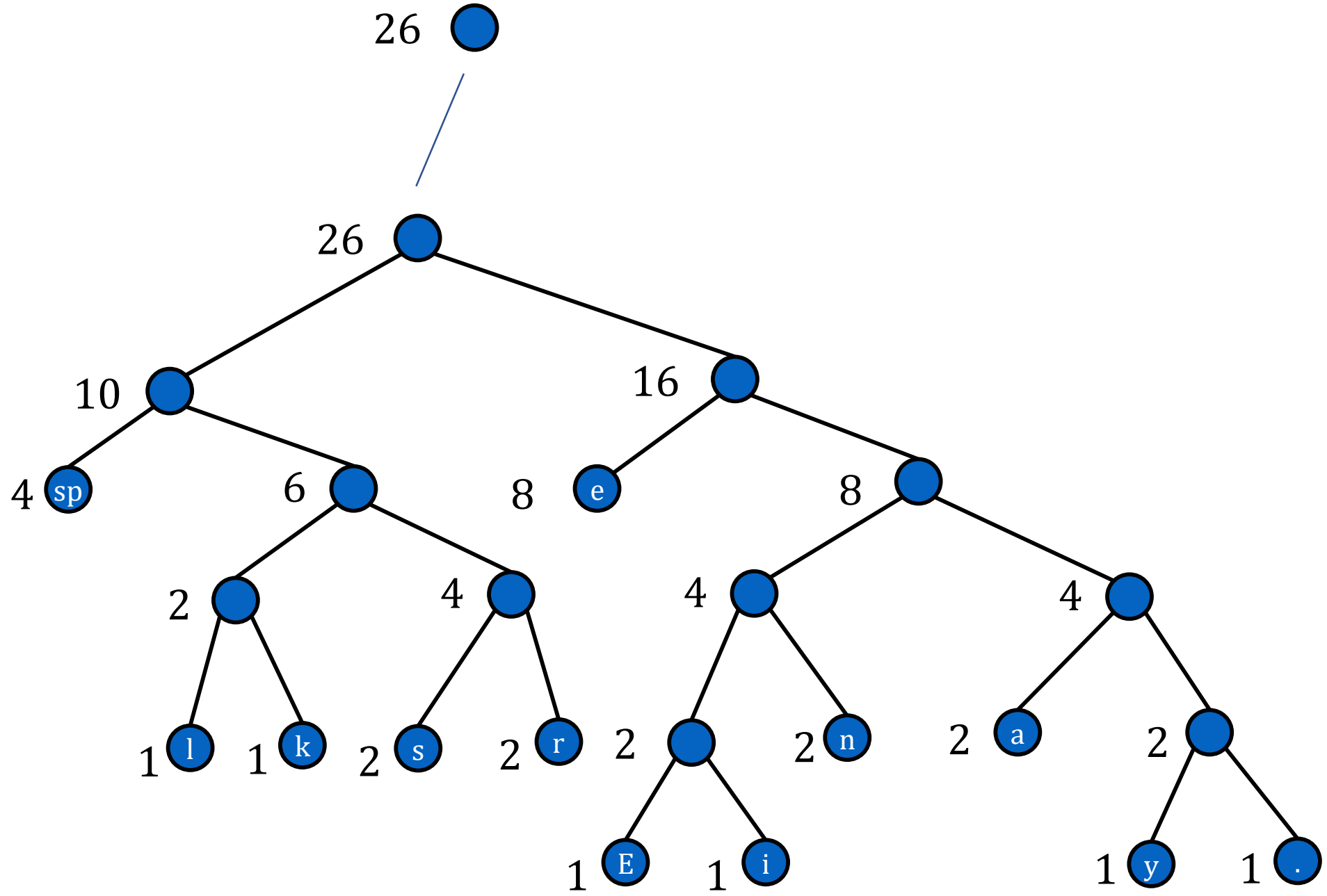

8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
8
e
8
10
4
sp
6
2
4
1
l
1
k
2
s
2
r


8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
8
e
8
10
10
4
sp
6
2
4
1
l
1
k
2
s
2
r

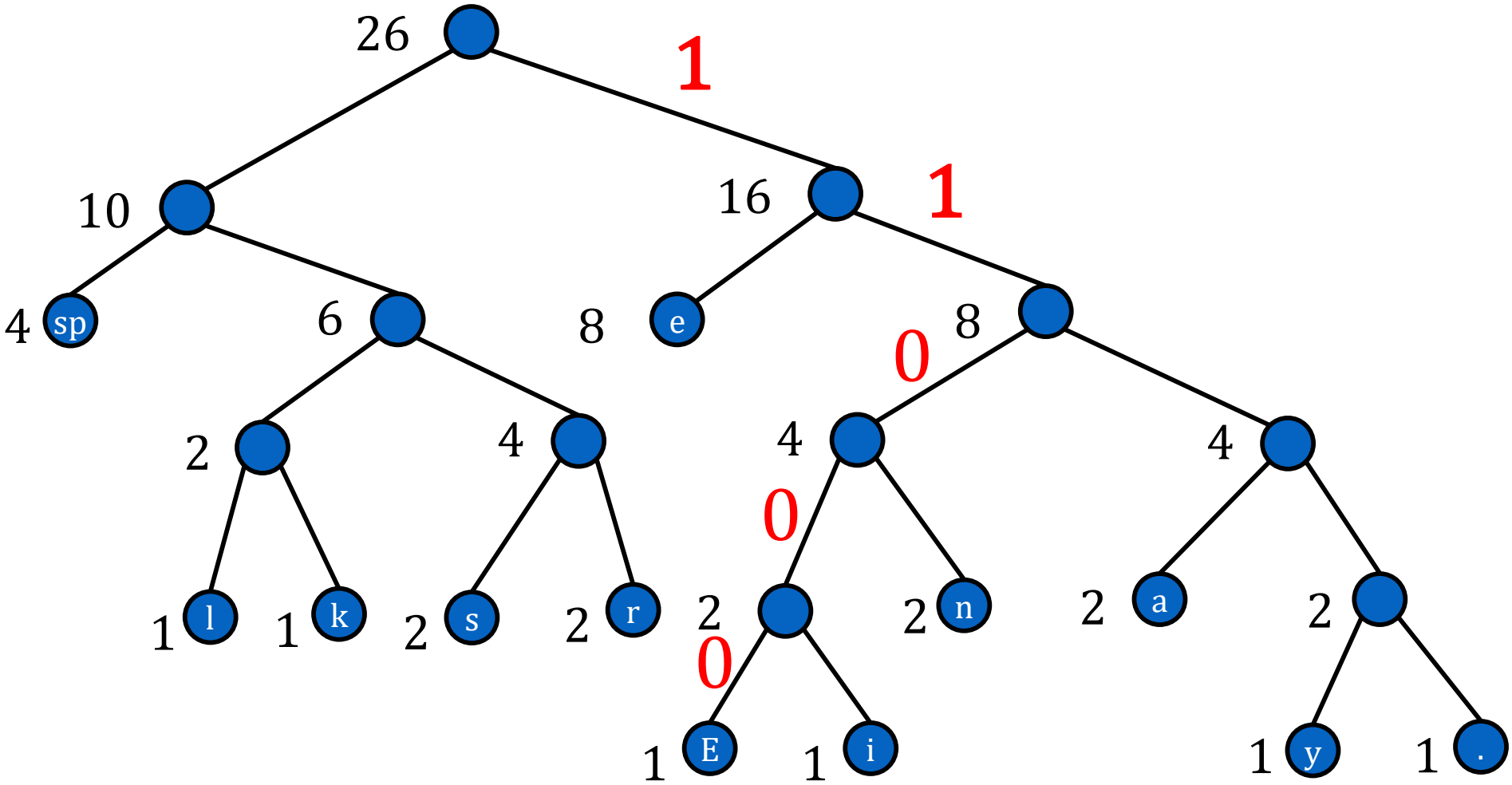

8
e
8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
8
10
10
4
sp
6
2
4
1
l
1
k
2
s
2
r


16
8
e
8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
10
10
4
sp
6
2
4
1
l
1
k
2
s
2
r


16
8
e
8
4
4
2
2
n
2
a
2
1
E
1
i
1
y
1
.
10
16
10
4
sp
6
2
4
1
l
1
k
2
s
2
r


26
26
10
16
4
sp
6
8
e
8
2
4
4
4
1
l
1
k
2
s
2
r
2
2
n
2
a
2
1
E
1
i
1
y
1
.

char	E	e	r	i	space	y
code	11000	10	0111	11001	00	11110
char	**s**	**n**	**a**	**l**	**k**	**.**
code	0110	1101	1110	0100	0101	11111



Input Sample: text.txt

```
Eerie eyes seen near lake.
```

char	E	e	r	i	space	y
code	11000	10	0111	11001	00	11110
char	s	n	a	l	k	.
code	0110	1101	1110	0100	0101	11111

Output Sample: code.txt

```
12                  Number of different
                    characters
E 1 11000           char1 freq1 code1
e 8 10              Char2 freq2 code2
r 2 0111            …
i 1 11001           Encrypted text
space 4 00
y 1 11110
s 2 0110
n 2 1101
a 2 1110
l 1 0100
k 1 0101
. 1 11111
84
11000100111110011000101111010011000011010101
10100110110111001110010011100101011011111
```

Code book

Note

- You must implement the project with a priority queue
- Tree node should be declared as follows

```
typedef struct node *nodePointer;
typedef struct node {
   char character;
   int frequency;
   nodePointer leftChild;
   nodePointer rightChild;
};
int MAX_QUEUE_SIZE;
nodePointer priorityQueue[MAX_QUEUE_SIZE];
```

- You have to create a node with **malloc** function

Note

- Your code must be able to:
  - read the text from a file
  - output the code book and encrypted text to a file

- Deadline:
  11/22 Thu (有問題可以再調整)

- E-course

- **C Source code**