ECE 264 Spring 2023

Advanced C Programming

Aravind Machiry
Purdue University

slides from yunglu@purdue.edu

Huffman Compression 01

Build Tree and Compress

yunglu@purdue.edu

Fixed-Length vs Variable-Length Code

- ASCII: fixed-length code, every character needs 8 bits
- Some characters (such as ‘s’ and ‘e’) are more often than some others (such as ‘q’ and ‘z’). Variable length can be more efficient:
  - fewer bits for frequently used characters
  - more bits for rarely used characters

  ⇒ fewer bits per character on average
- General design principle: optimize for the frequent cases
- This is ***lossless*** compression. Data can be fully recovered.

yunglu@purdue.edu

Where is data compression used?

- Everywhere
- Image, video, audio (lossy)
- File download
- When network is limited (in data rate), slow, or unstable

yunglu@purdue.edu

Huffman Coding (Compression)

Lossless compression

1. Count the occurrences of the characters (may include symbols and unprintable characters)
2. Sort the characters by their occurrences in the ascending order
3. Take the two least occurrences, make them left and right children of the same parent node, add the occurrences and sort in the ascending order again
4. Continue 3 until only one node is left

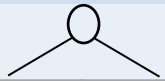
yunglu@purdue.edu

occurrence	4	18	7	22	10	35
letter	#	A	G	c	m	s

⇩ Sort by the occurrences in ascending order

4	7	10	18	22	35
#	G	m	A	c	s

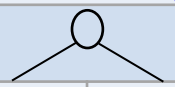
⇩ Make the first two siblings of a binary tree



4	7	10	18	22	35
#	G	m	A	c	s

⇩ The occurrence of the parent is the sum

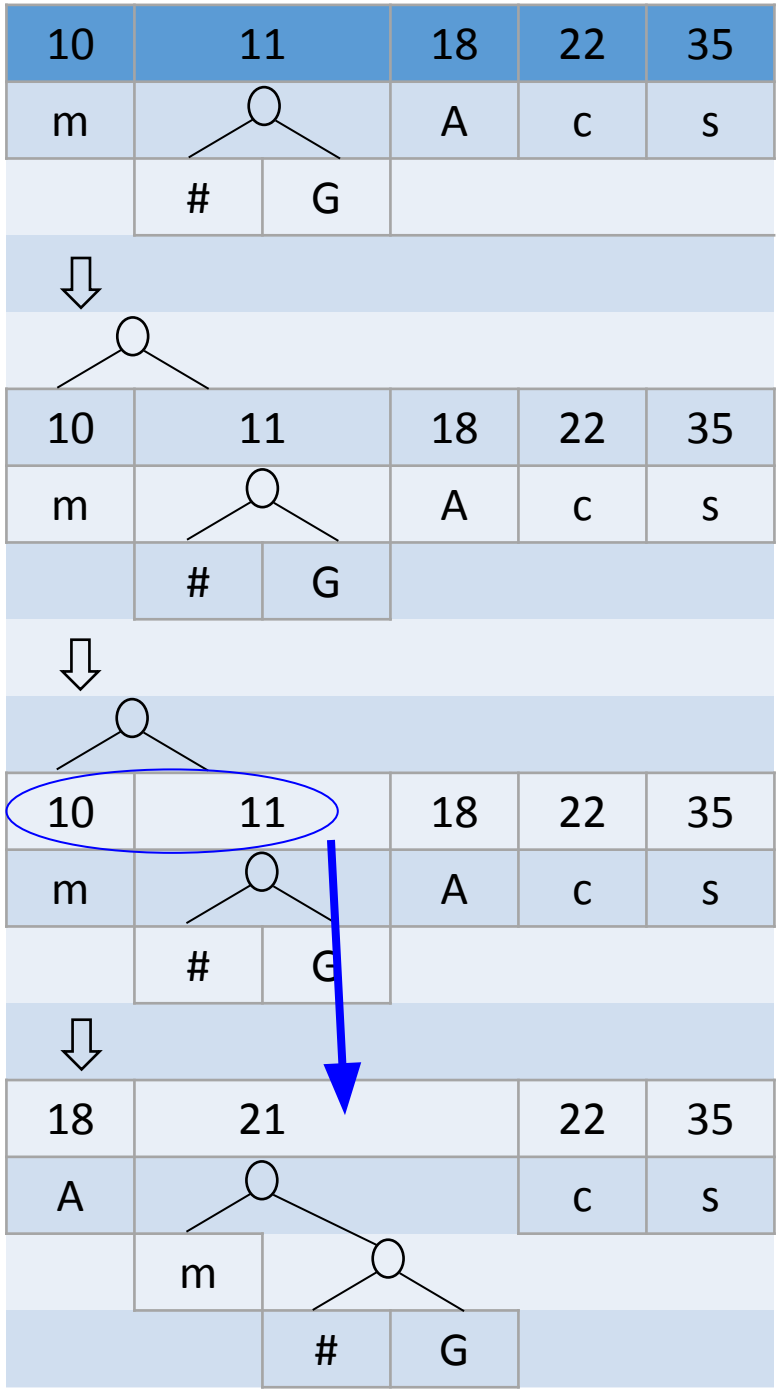
11					

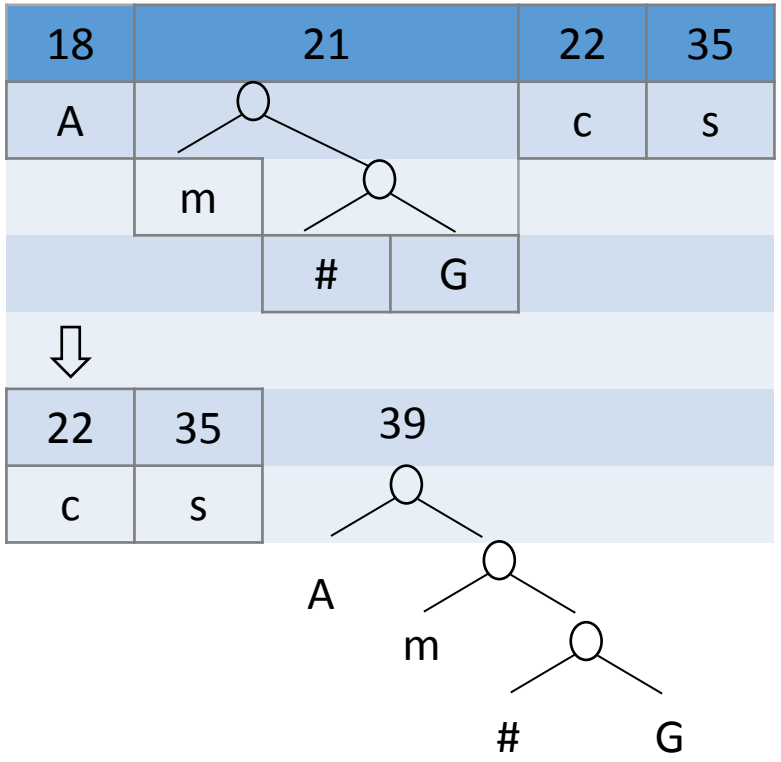


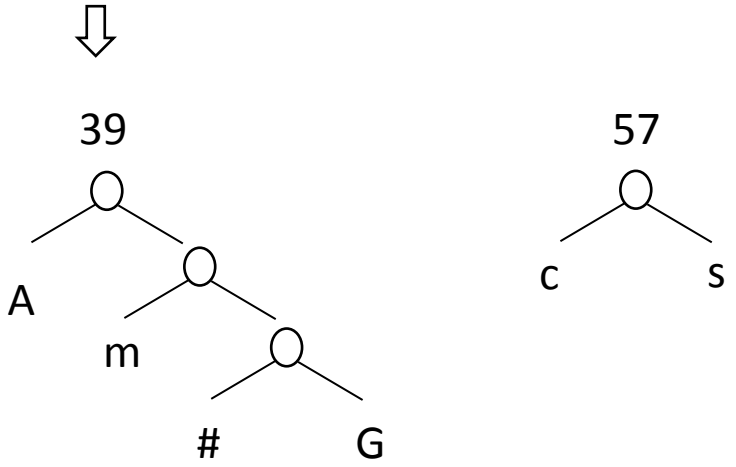
4	7	10	18	22	35
#	G	m	A	c	s

⇩ Insert the parent back in ascending order

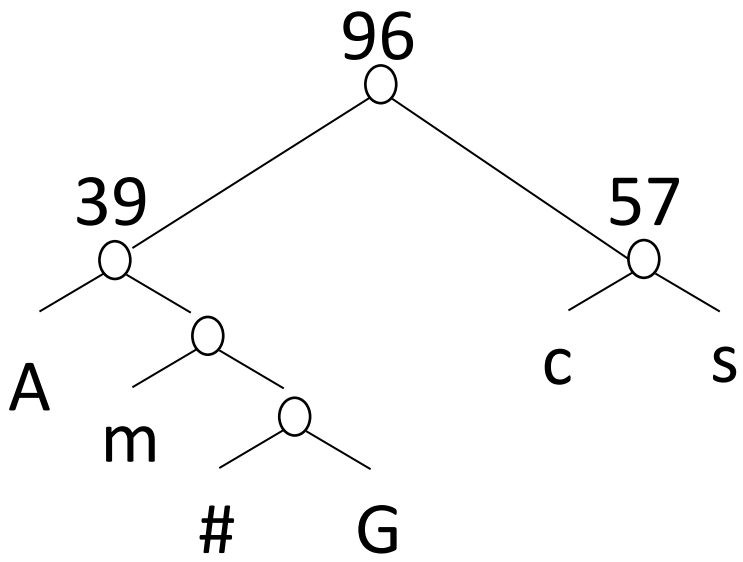
10	11		18	22	35
m			A	c	s
	#	G			

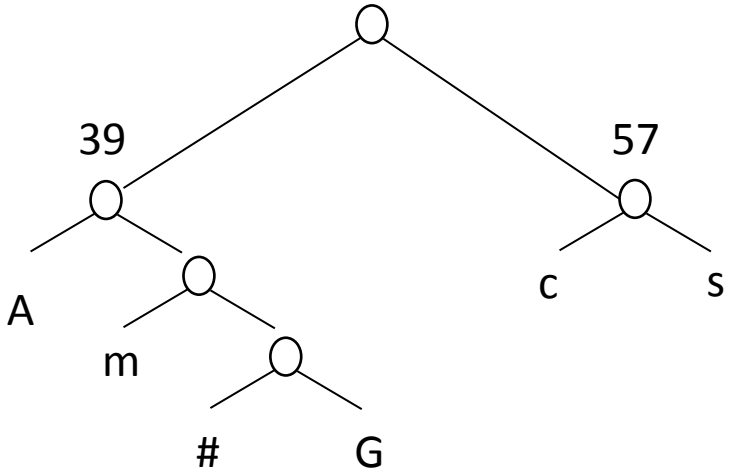

10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
18
21
22
35
A
c
s
m
#
G

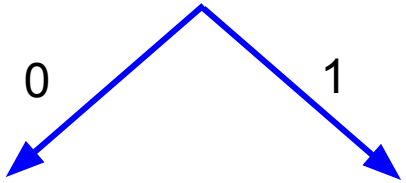




Only the leaf nodes contain characters







character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

yunglu@purdue.edu

character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

- If occurrence (X) < occurrence (Y)

  ⇒ code length (X) ≥ code length (Y)

- code length (X) > code length (Y)

  ⇒ occurrence (X) < occurrence (Y) **WRONG**

yunglu@purdue.edu

character	occurrence	code				length
A	18	0	0			2
m	10	0	1	0		3
#	4	0	1	1	0	4
G	7	0	1	1	1	4
c	22	1	0			2
s	35	1	1			2

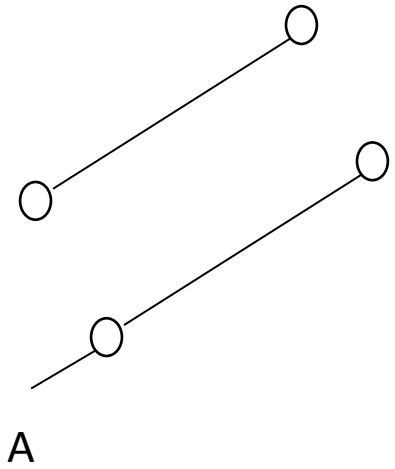
input	A	A	c	s	#	m	G	c	s	A	…
output	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0	0 1 1 1	1 0	1 1	0 0	…

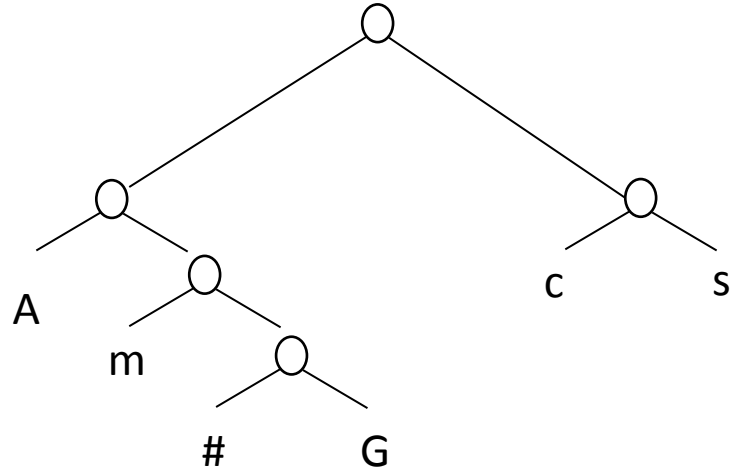
yunglu@purdue.edu

input	0 0 0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0

0

0 0 ⇨ A

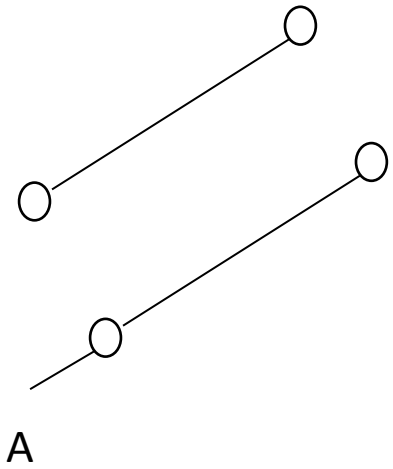




input	0 0	0 0 1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
	A	

0

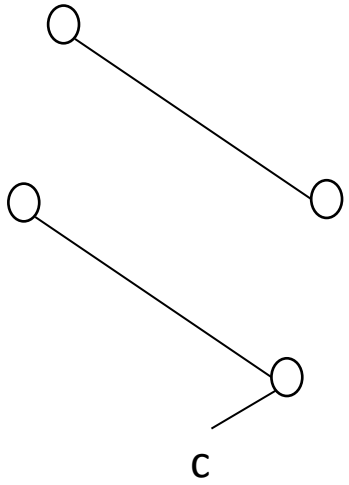
0 0 ⇨ A



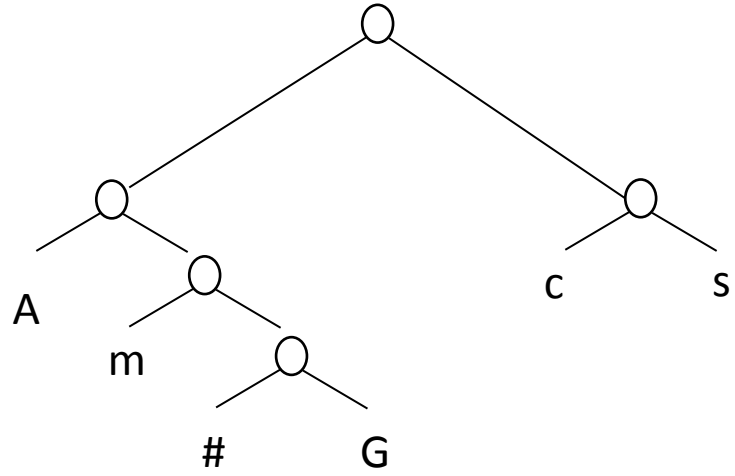
yunglu@purdue.edu

input	0 0	0 0	1 0 1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
	A	A	

1

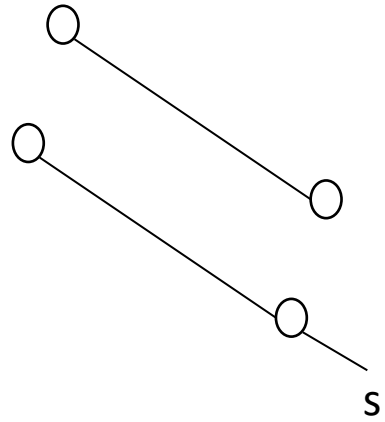


1 0 ⇨ c



input	0 0	0 0	1 0	1 1 0 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0
	A	A	c	

1

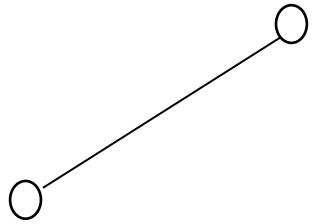


1 1 ⇨ s

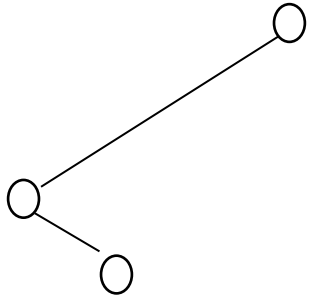
yunglu@purdue.edu

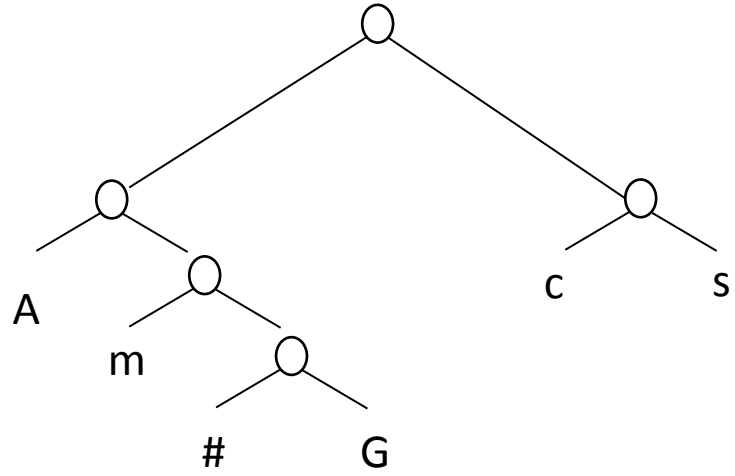
input	0 0	0 0	1 0	1 1	0 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0
	A	A	c	s	

0

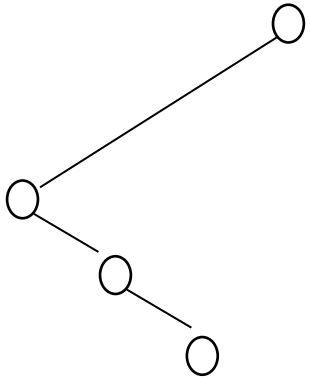


0 1 ⇨

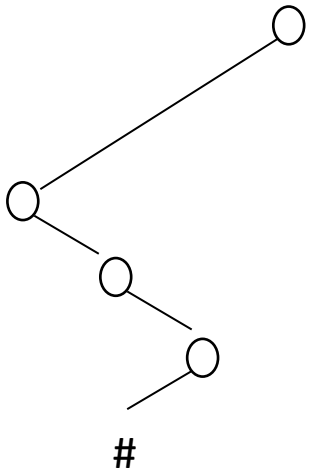




0 1 1



0 1 1 0 ⇨ #



yunglu@purdue.edu

input	0 0	0 0	1 0	1 1	0 1 1 0	0 1 0 0 1 1 1 1 0 1 1 0 0
	A	A	c	s	#	

- 0 goes to the left
- 1 goes to the right
- If reach a leaf node, output the character
- go back to the root

- Characters are stored in only leaf nodes (by construction)

yunglu@purdue.edu

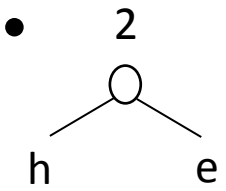
Example

- Hellooo

Example

- Hellooo
- H (1) e (1) l(2) o (3)

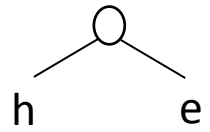
Example

- Hellooo
- H (1) e (1) l(2) o (3)
- 

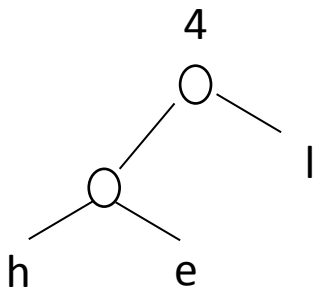
l(2) o (3)

Example

- Hellooo
- H (1) e (1) l(2) o (3)
- 2 l(2) o (3)

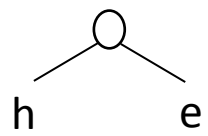


- 4 o (3)

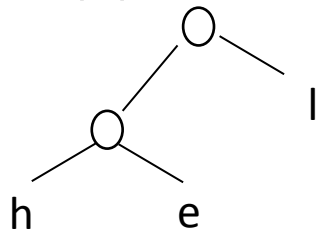


Example

- Hellooo
- H (1) e (1) l(2) o (3)
- 2 l(2) o (3)



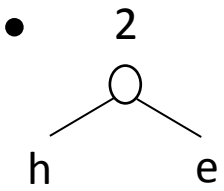
- o (3) 4



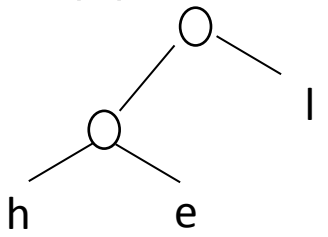
SHOULD BE IN ASCENDING ORDER

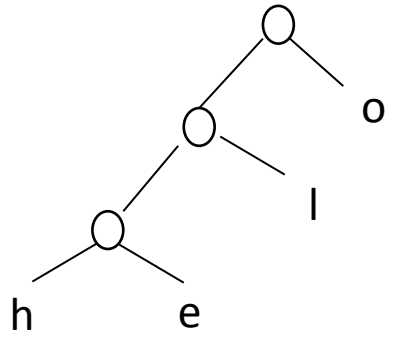
Example

- Hellooo
- H (1) e (1) l(2) o (3)
- 2 l(2) o (3)



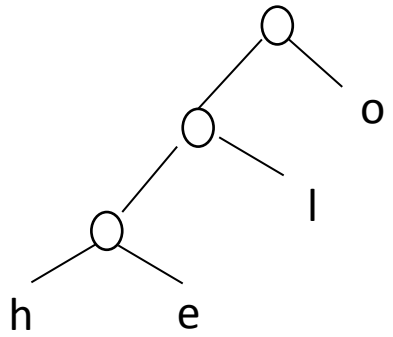
- o (3) 4





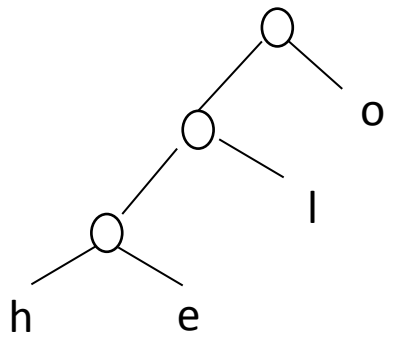
Computing Codes

character	occurrence	code			length
h	1				
e	1				
l	2				
o	3				



Computing Codes

character	occurrence	code			length
h	1	0	0	0	3
e	1	0	0	1	3
l	2	0	1		2
o	3	1			1

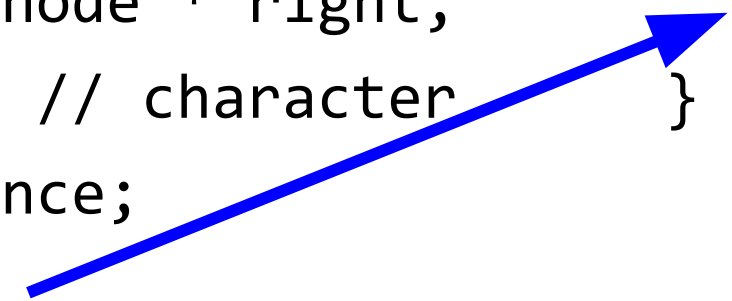


How to build the compression tree

Ch 24 in https://github.com/yunghsianglu/IntermediateCProgramming

```
typedef struct treenode
{
  struct treenode * left;
  struct treenode * right;
  char value; // character
  int occurrence;
} TreeNode;
```

```
typedef struct listnode
{
  struct listnode * next;
  TreeNode * tnptr;
} ListNode;
```

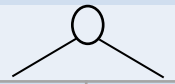


occurrence	4	18	7	22	10	35
letter	#	A	G	c	m	s

⇩ Sort by the occurrences in ascending order

4	7	10	18	22	35
#	G	m	A	c	s

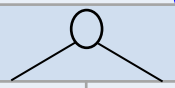
⇩ Make the first two siblings of a binary tree



4	7	10	18	22	35
#	G	m	A	c	s

⇩ The occurrence of the parent is the sum

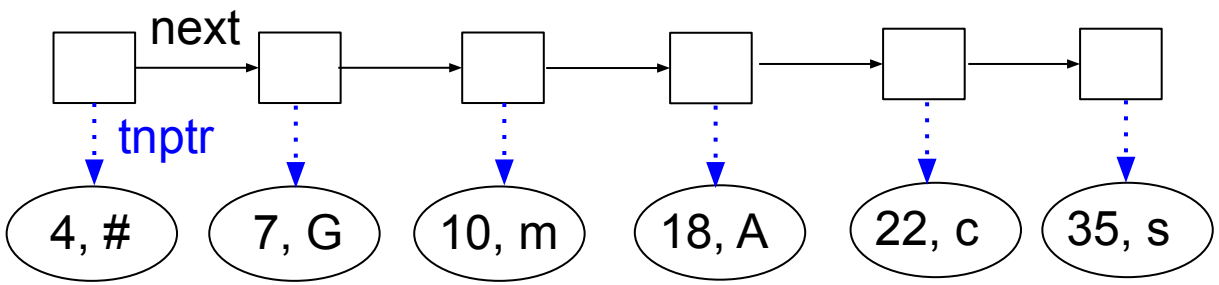
11					

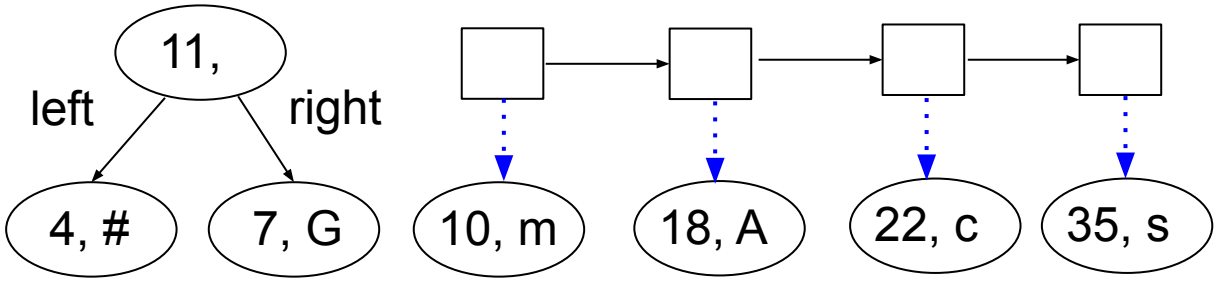


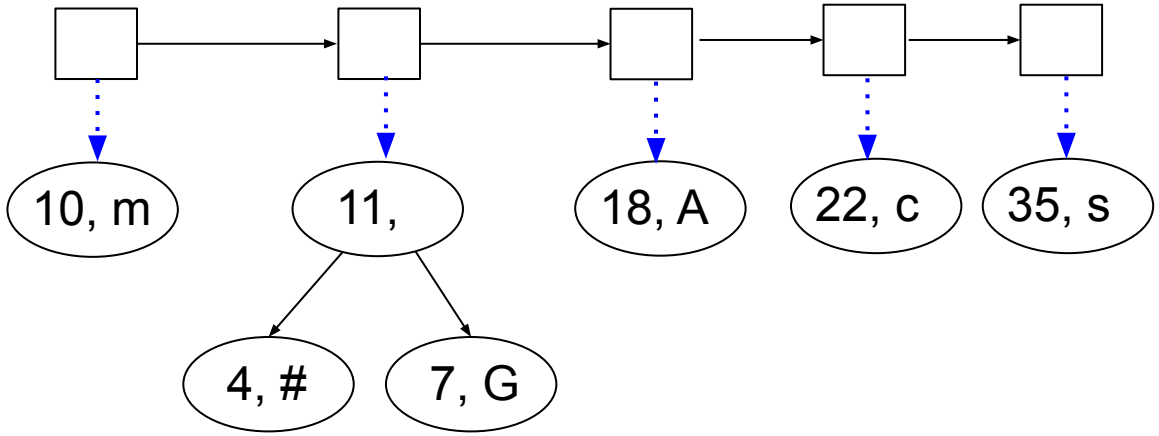
4	7	10	18	22	35
#	G	m	A	c	s

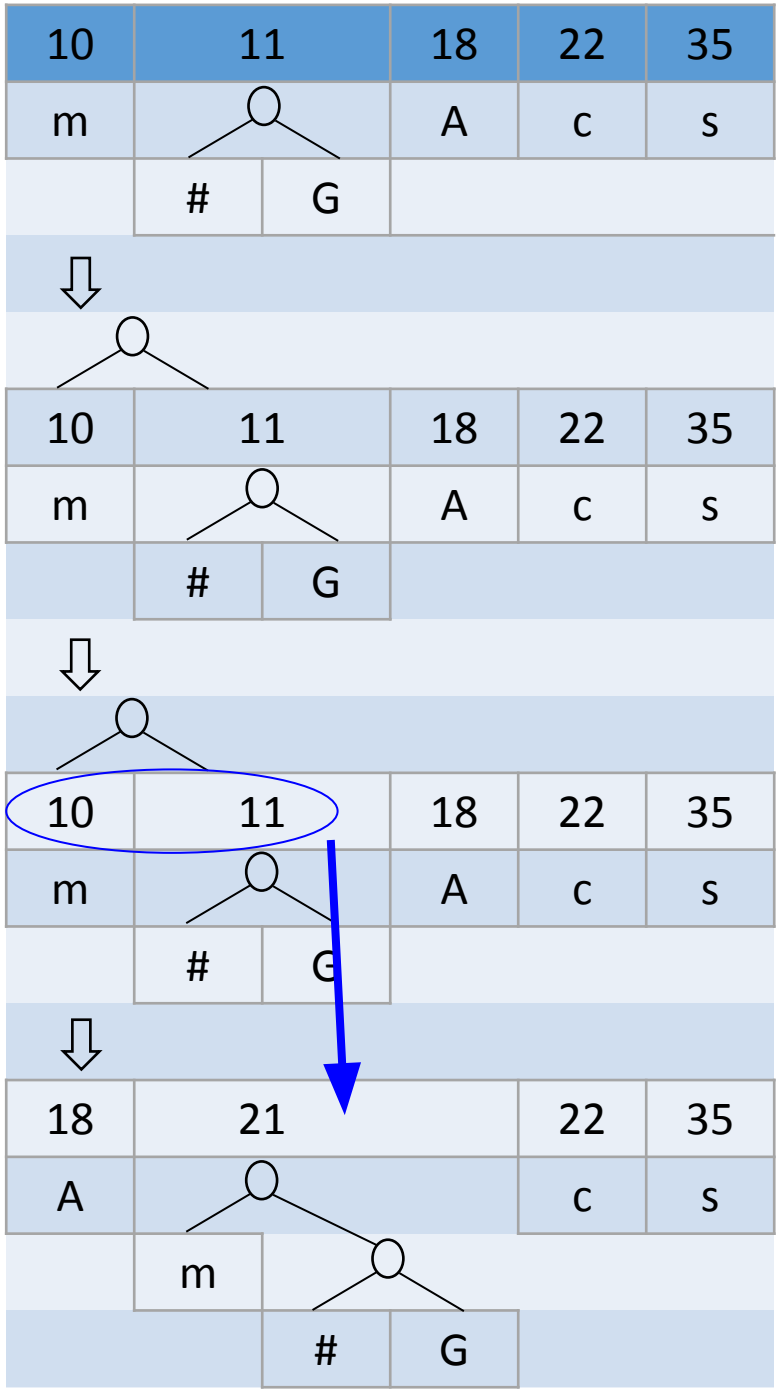
⇩ Insert the parent back in ascending order

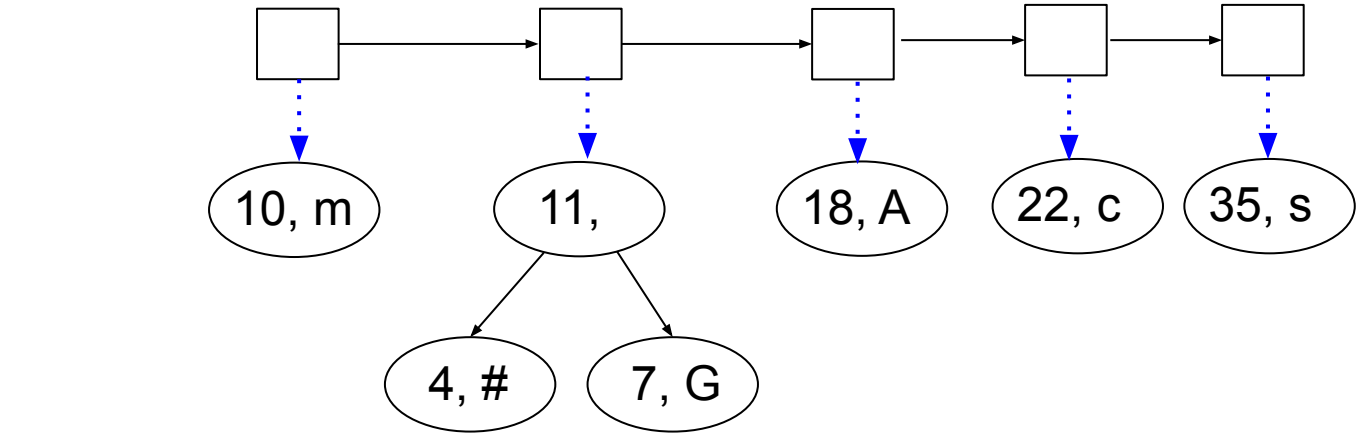
10	11		18	22	35
m			A	c	s
	#	G			

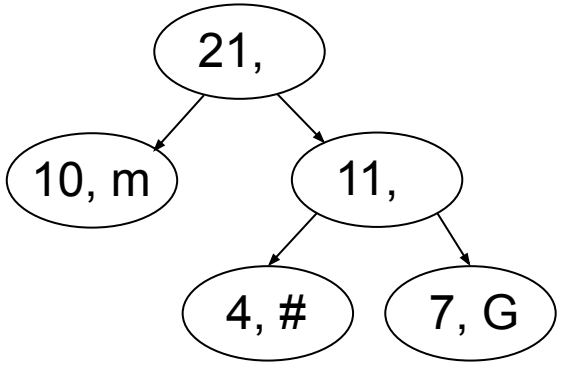


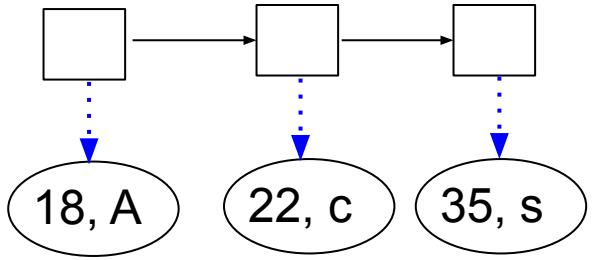


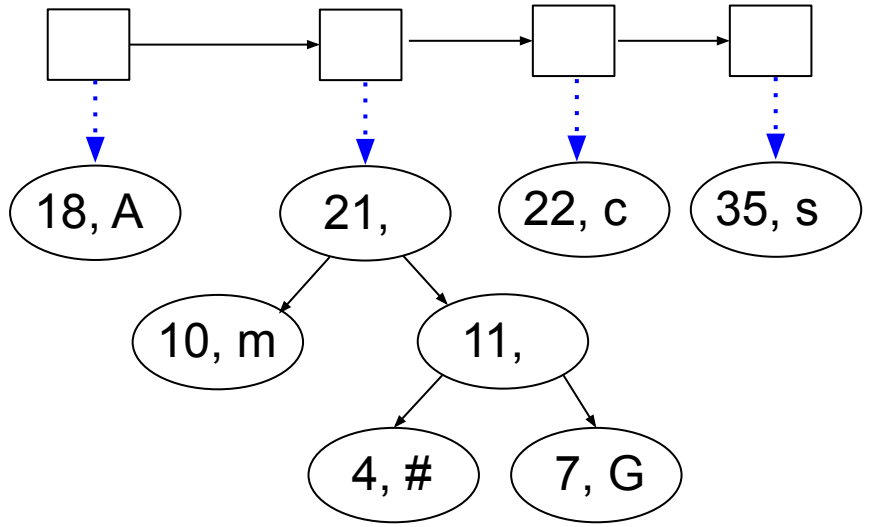



10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
10
11
18
22
35
m
A
c
s
#
G
18
21
22
35
A
c
s
m
#
G


10, m
11,
18, A
22, c
35, s
4, #
7, G


21,
10, m
11,
4, #
7, G


18, A
22, c
35, s


18, A
21,
22, c
35, s
10, m
11,
4, #
7, G