Huffman Coding Walkthrough

- Huffman coding is used to compress data.
  - It is used by ZIP files, among many other things.
- The overall process is as follows:
  1. Calculate the frequency of each character in the data.
  2. Build a Huffman tree structure using the frequencies.
  3. Build an encoding table using the Huffman tree.
  4. Encode each character in the data.
- The output will normally contain ≥2 things:
  - Coding table
  - Encoded data

1. Calculate the frequencies

Goal: Make a Huffman code table for compressing the following string.

```
huffman fluffs many mums
```

Next step: Make a frequency table

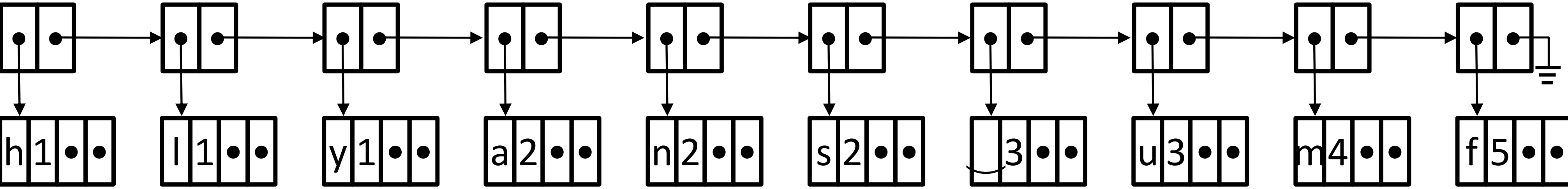

```
huffman fluffs many mums
```

Frequency table

char	frequency
f	5
m	4
u	3
␣	3
s	2
a	2
n	2
y	1
h	1
l	1

Next step: Start creating the Huffman tree.

2. Build the Huffman tree



We start by creating a priority queue where each list node refers to a tree node containing a single character.

Process

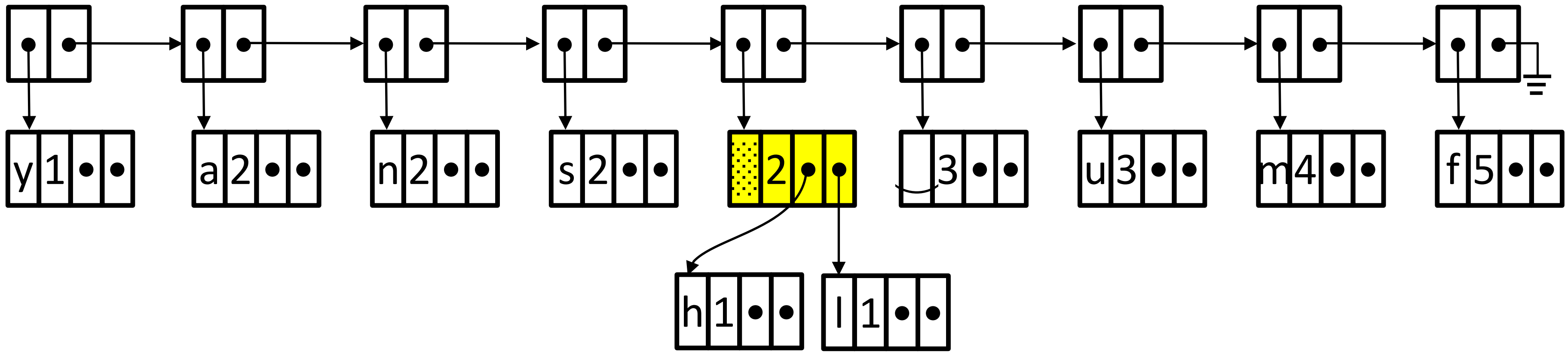
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Process

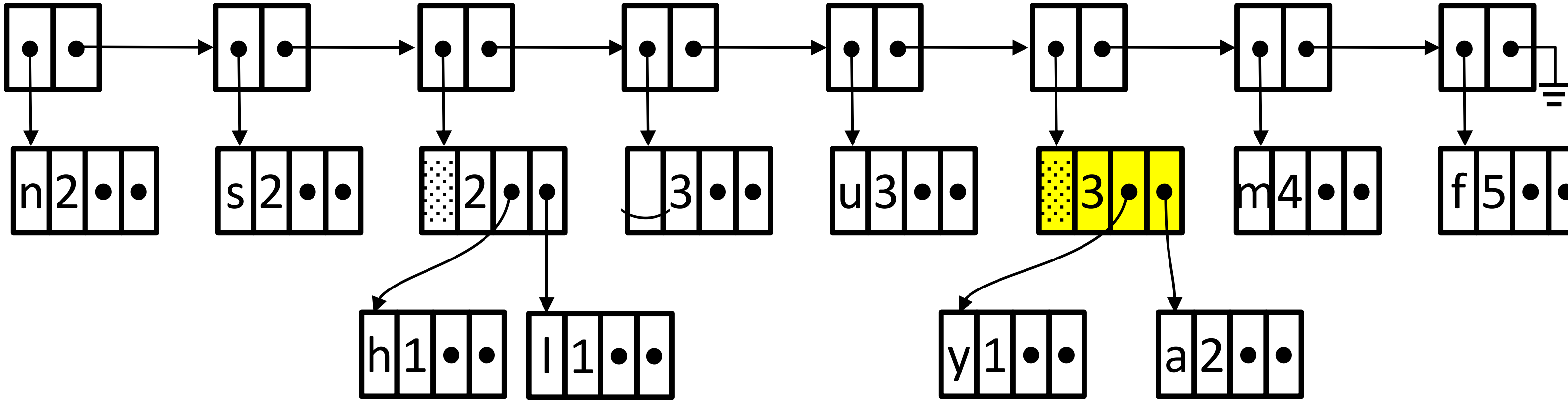
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



<u>Process</u>

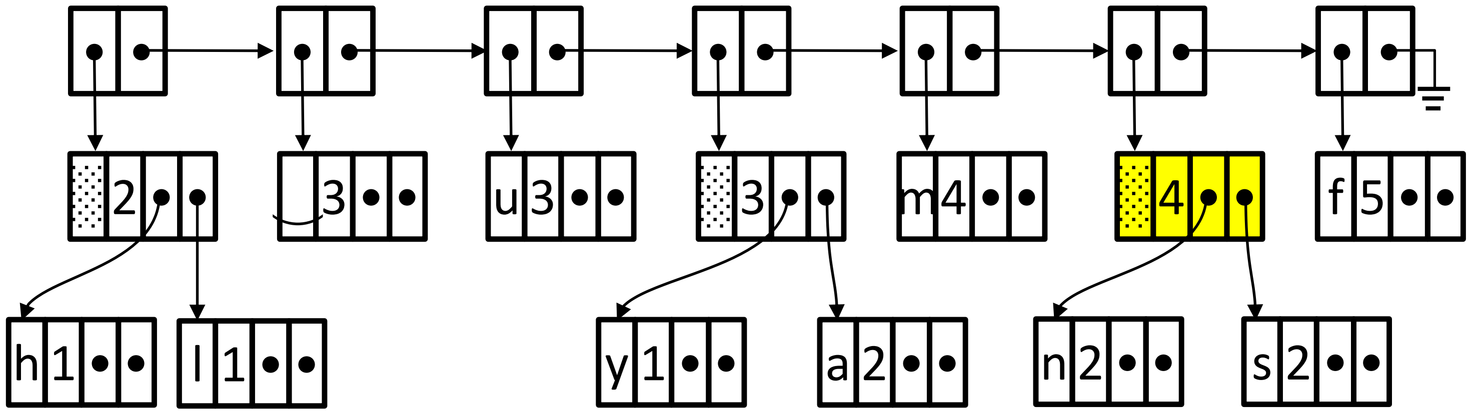
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

<u>Priority queue compare function</u>

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Process

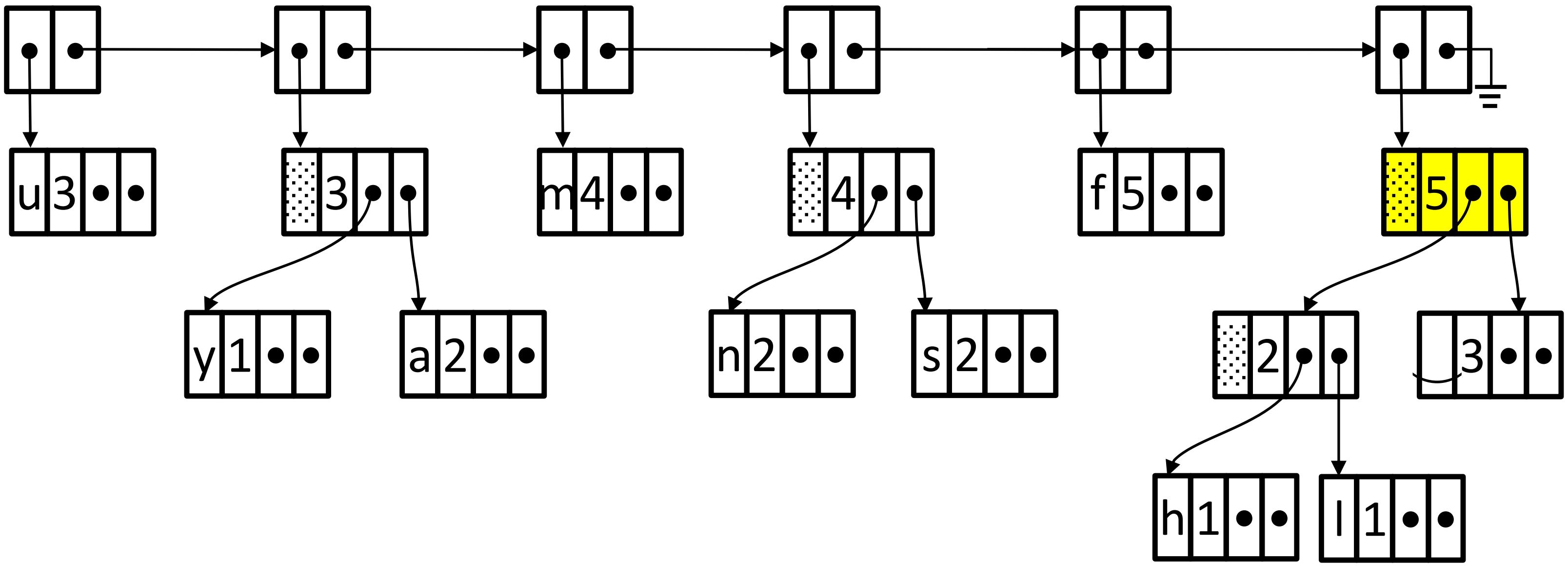
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



<u>Process</u>

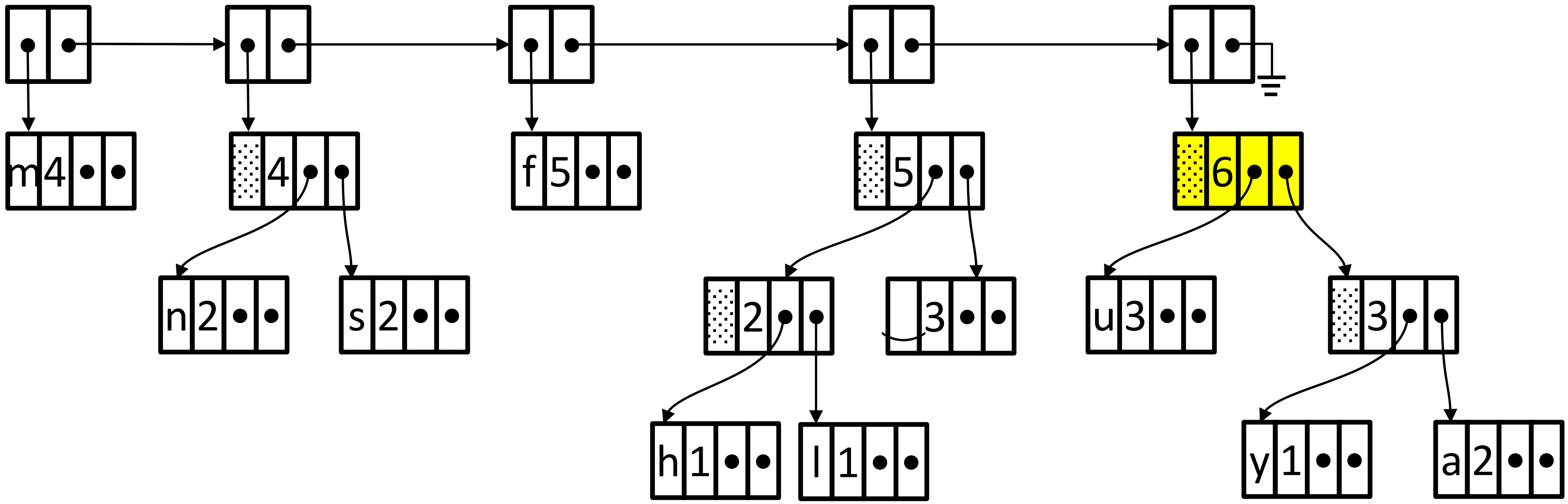
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

<u>Priority queue compare function</u>

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.

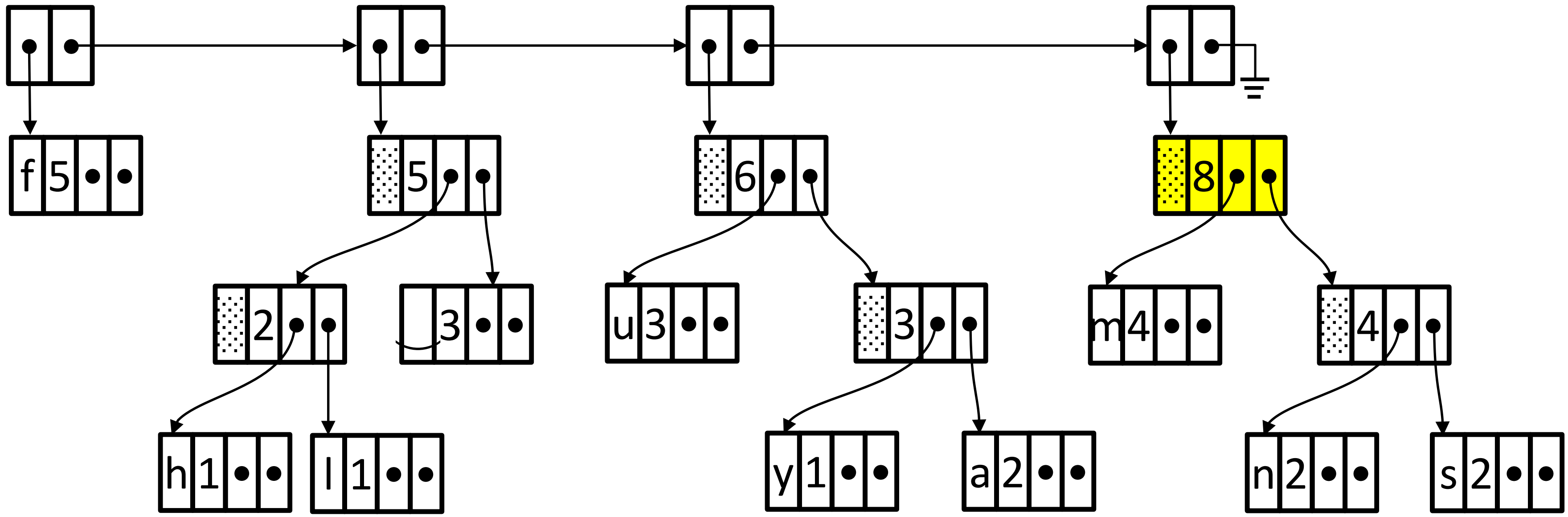


Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Process

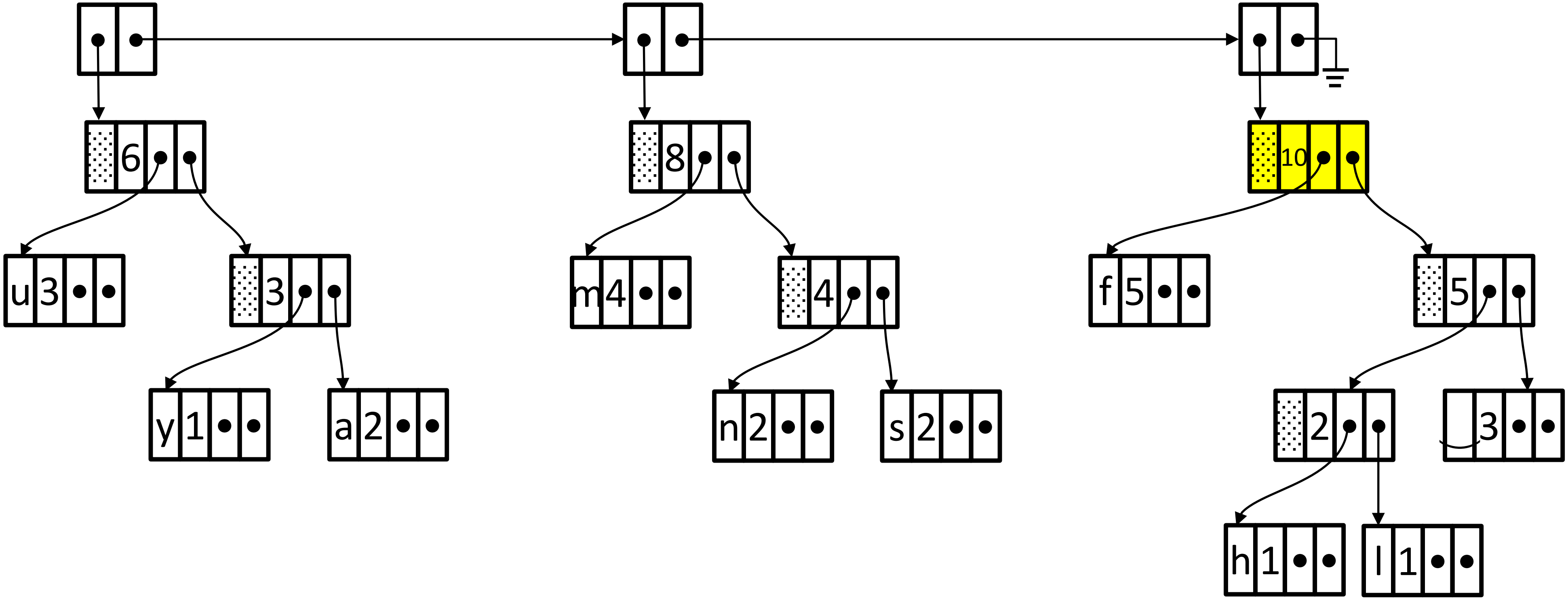
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.

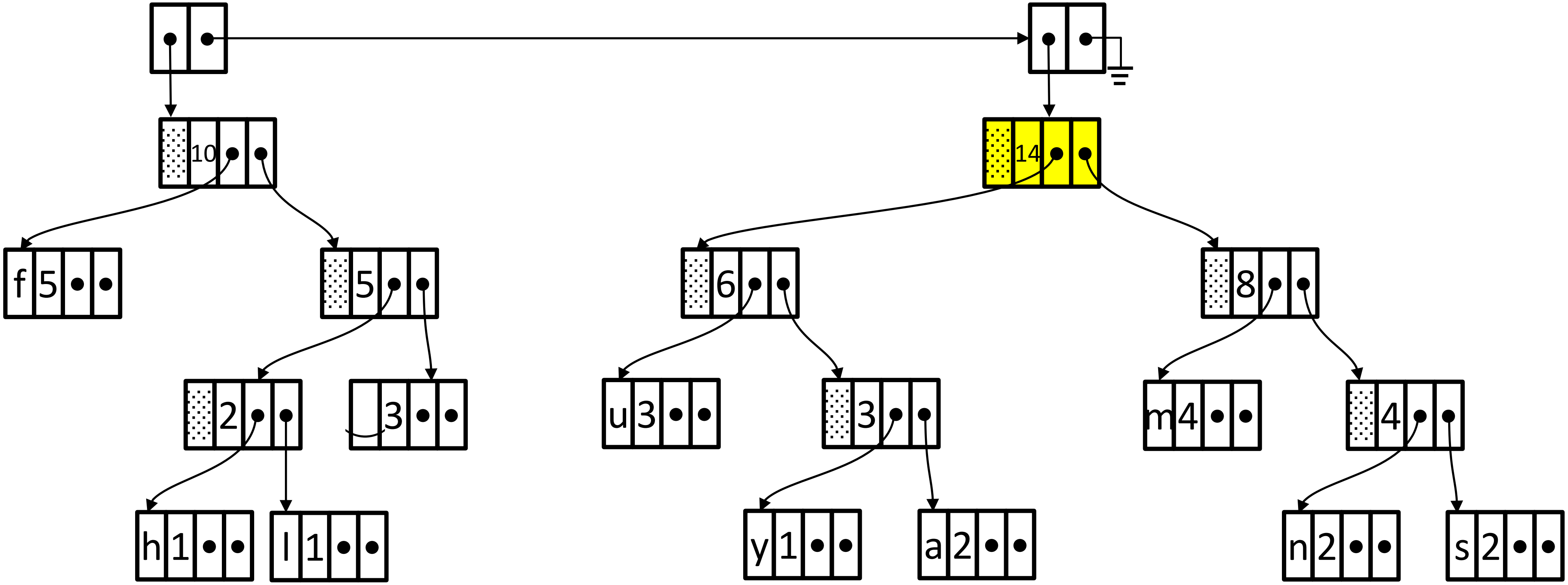


Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function
- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Process

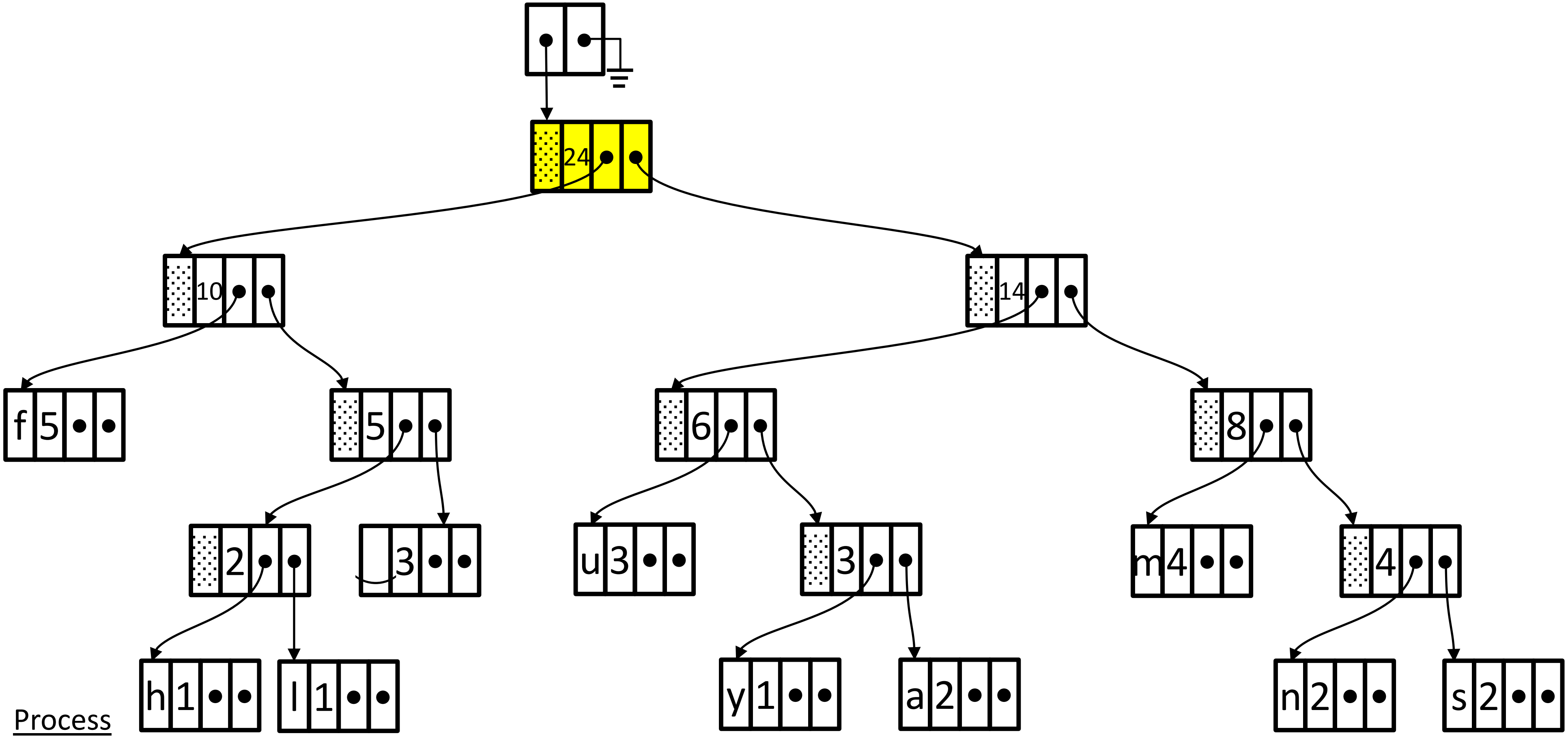
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Next step: Join first two nodes

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Process

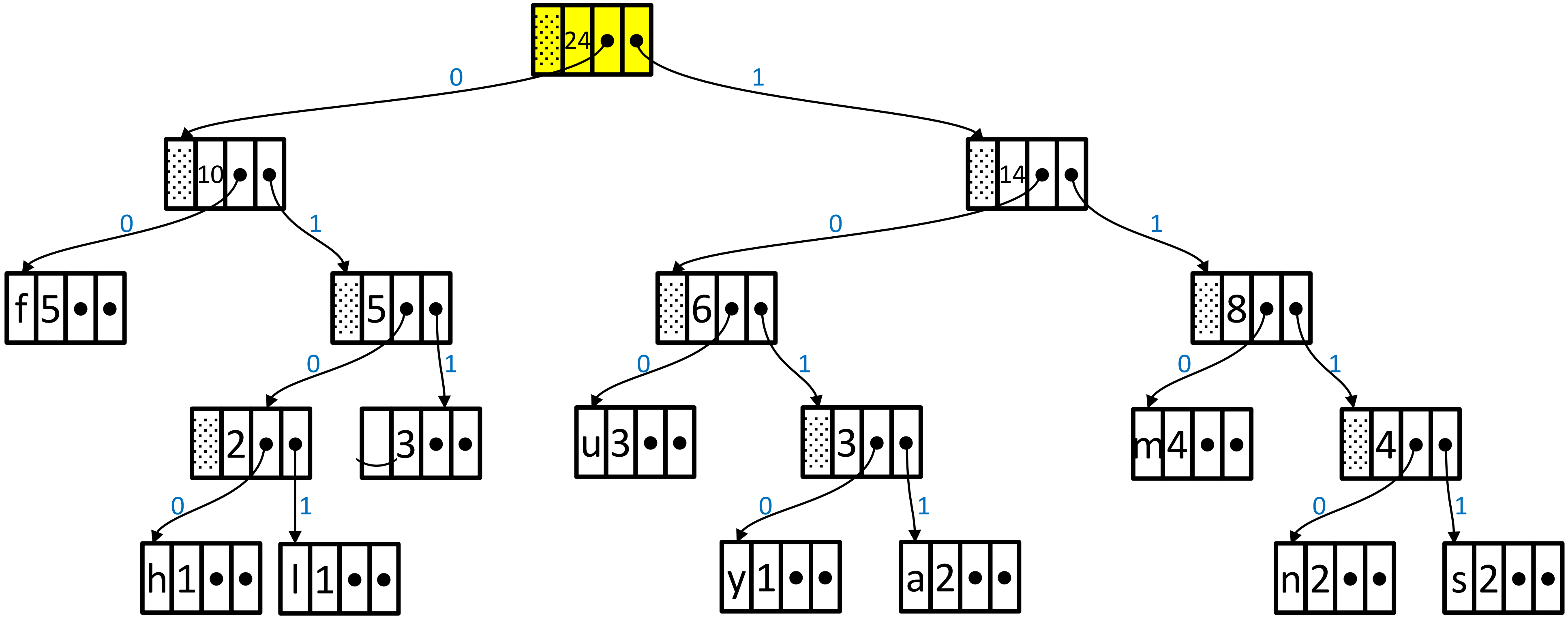
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

Priority queue compare function

- Order by the frequency.
- If frequency is same, then nodes with just a single character come before clusters.
- If frequency is same and both are single-character nodes (i.e., not clusters order by ASCII value of character.

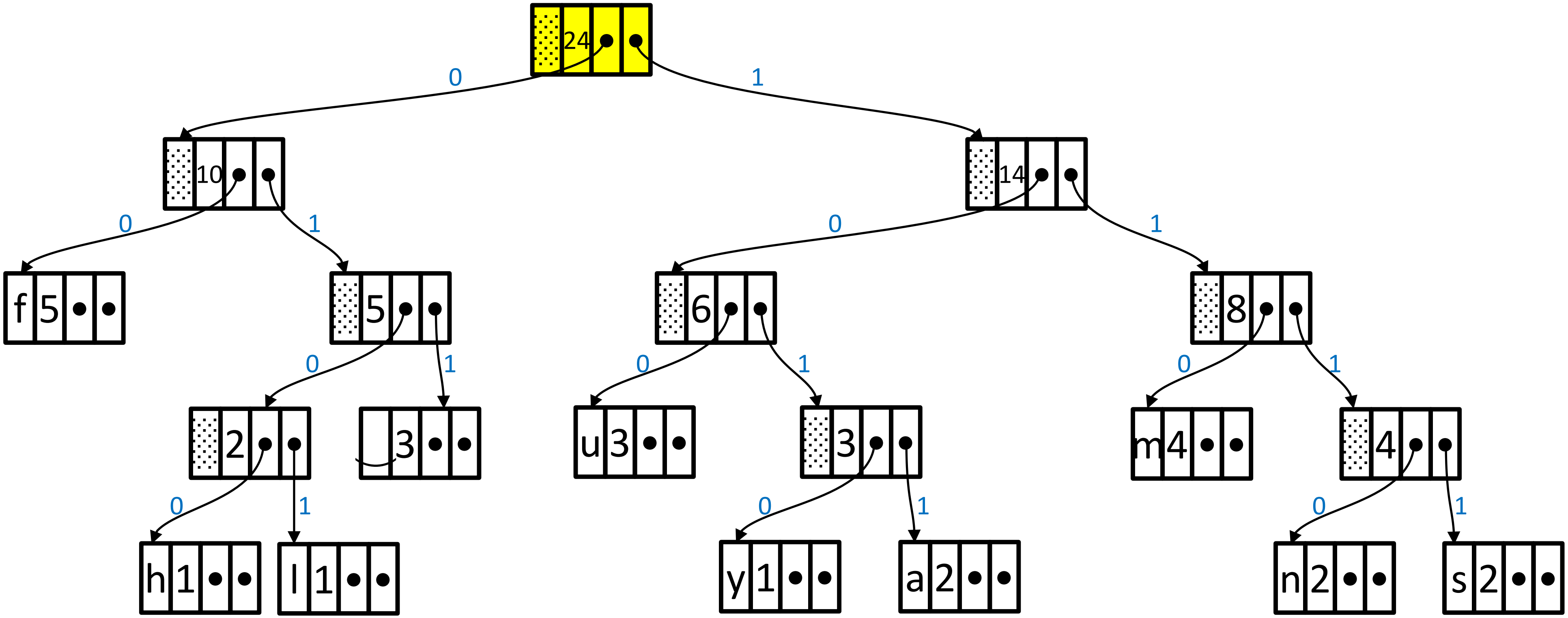
Next step: Remove head of priority queue, leaving only the tree.

This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.



Next step: Create the code table

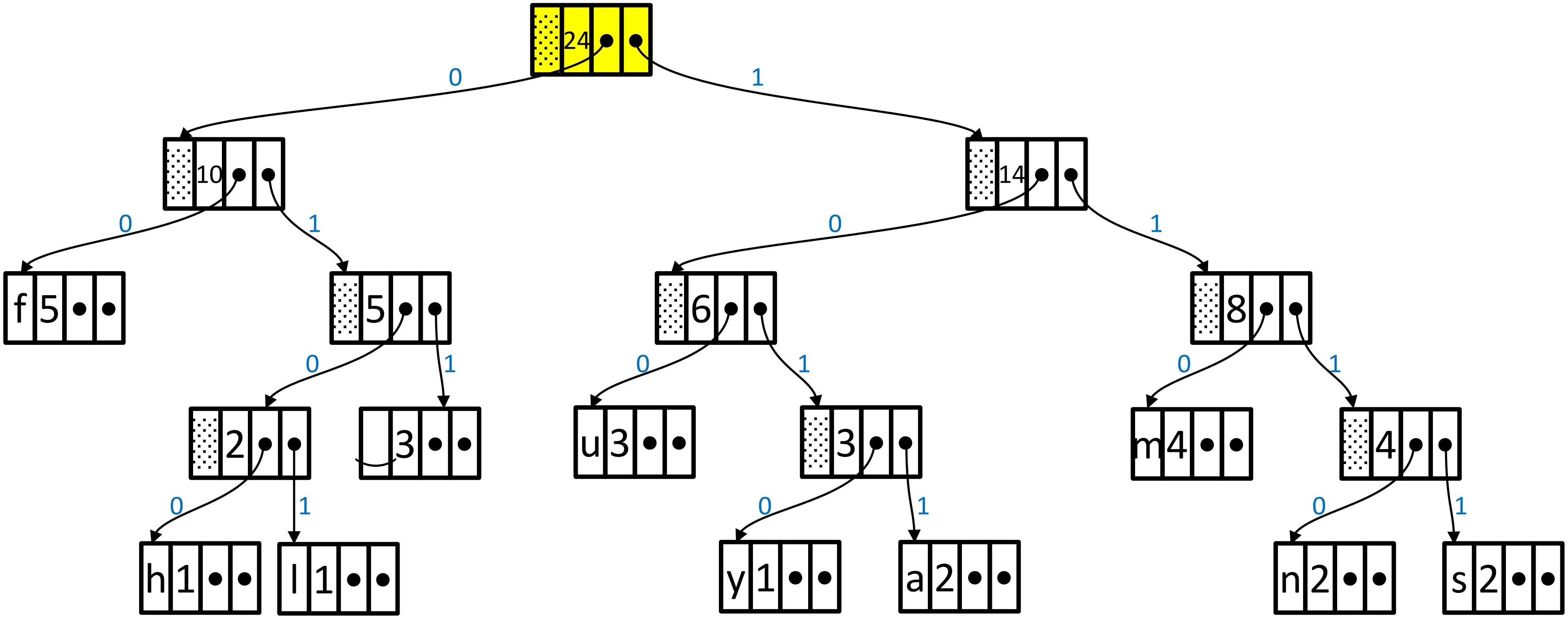
3. Build the encoding table.



Code table

char	code	# of bits	frequency
f	00	2	5
m	110	3	4
␣	011	3	3
u	100	3	3
s	1111	4	2
a	1011	4	2
n	1110	4	2
y	1010	4	1
h	0100	4	1
l	0101	4	1

Notice that no code is a prefix of another.



Code table

char	code	# of bits	frequency
f	00	2	5
m	110	3	4
‿	011	3	3
u	100	3	3
s	1111	4	2
a	1011	4	2
n	1110	4	2
y	1010	4	1
h	0100	4	1
l	0101	4	1

More frequently occurring characters get shorter codes.

4. Encode each character in the data.

huffman fluffs many mums

h	0100
u	100
f	00
f	00
m	110
a	1011
n	1110
⌣	011
f	00
l	0101
u	100
f	00
f	00
s	1111
⌣	011
m	110
a	1011
n	1110
y	1010
⌣	011
m	110
u	100
m	110
s	1111

Encoded string

```
0100 100 00 00 110
h    u   f  f  m

1011 1110 011 00 0101
a    n    ⌣   f  l

100 00 00 1111 011 110
u   f  f  s    ⌣   m

1011 1110 1010 011 110
a    n    y    ⌣   m
```

Code table

char	code	# of bits	frequency
f	00	2	5
m	110	3	4
⌣	011	3	3
u	100	3	3
s	1111	4	2
a	1011	4	2
n	1110	4	2
y	1010	4	1
h	0100	4	1
l	0101	4	1

huffman fluffs many mums

h	0100
u	100
f	00
f	00
m	110
a	1011
n	1110
␣	011
f	00
l	0101
u	100
f	00
f	00
s	1111
␣	011
m	110
a	1011
n	1110
y	1010
␣	011
m	110
u	100
m	110
s	1111

Code table

char	code	# of bits	frequency
f	00	2	5
m	110	3	4
␣	011	3	3
u	100	3	3
s	1111	4	2
a	1011	4	2
n	1110	4	2
y	1010	4	1
h	0100	4	1
l	0101	4	1

Encoded string

10 bytes

01001000 00011010
h u f f m a

11111001 10001011
n ␣ f l u

00000011 11011110
f f s ␣ m

10111110 10100111
a n y ␣ m

10100110 11110000
u m s

Input: 24 bytes

- ```
  01101000 01110101 01100110 01100110 01101101
  h        u        f        f        m
  01100001 01101110 00100000 01100110 01101100
  a        n        ‿        f        l
  01110101 01100110 01100110 01110011 00100000
  u        f        f        s        ‿
  01101101 01100001 01101110 01111001 00100000
  m        a        n        y        ‿
  01101101 01110101 01101101 01110011
  m        u        m        s
  ```

Output: 10 bytes

- ```
  01001000 00011010 11111001 10001011 00000011
  11011110 10111110 10100111 10100110 11110000
  ```

Compression ratio: 10 / 24 = 42%