Objectives – Tue 4/5/2022

- Testing
- Coverage
- Huffman coding
  - What it does
  - Building the Huffman tree structure
  - Encoding a file

Whatever we do not finish today, we will do on Thursday

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

3 A's

- Arrange
- Act
- Assert

Also known as the "AAA (Arrange-Act-Assert)" pattern

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

Unit testing

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

Order

- Tests should be able to run in any order
  - Ex: test_read(…) should not depend on test_write(…)
  - It shouldn't matter if you run…

```
mu_run( test_write );
mu_run( test_read );
```

… or …

```
mu_run( test_read );
mu_run( test_write );
```

- You should be able to comment out some tests without affecting others
  - Normally, you should be running all tests together
  - Need enough support code so each test is indepdendent.
- Every test should start with a clean slate

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

No manual inspection required

- The tests should be able to run on their own
  - Running all tests should require no human effort.
- This is the foundation of regression testing
  - Regression testing means running all tests whenever something changes and/or periodically (e.g., nightly).

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

Bugs vs. run-time error handling

- "Bugs" are flaws in <u>your code</u>.
  - Ex: You forgot to check for something.

- "Run-time error handling" means ensuring that the program behaves in a way that is helpful to the user, even when it receives unexpected or malformed inputs
  - Ex: malformed BMP header

© 2020 Alexander J. Quinn   This content is protected and may not be shared, uploaded, or distributed.

Types of test code coverage

- "Line coverage" means every line of the code being tested was executed at least once.
- "Branch coverage" means for every conditional jump (If/While/For/Switch), we took the jump (condition true) and did not take the jump (condition false) at least once.
- "Path coverage" means we tested every possible path through the code (unique combination of branches). *This can be hard.*

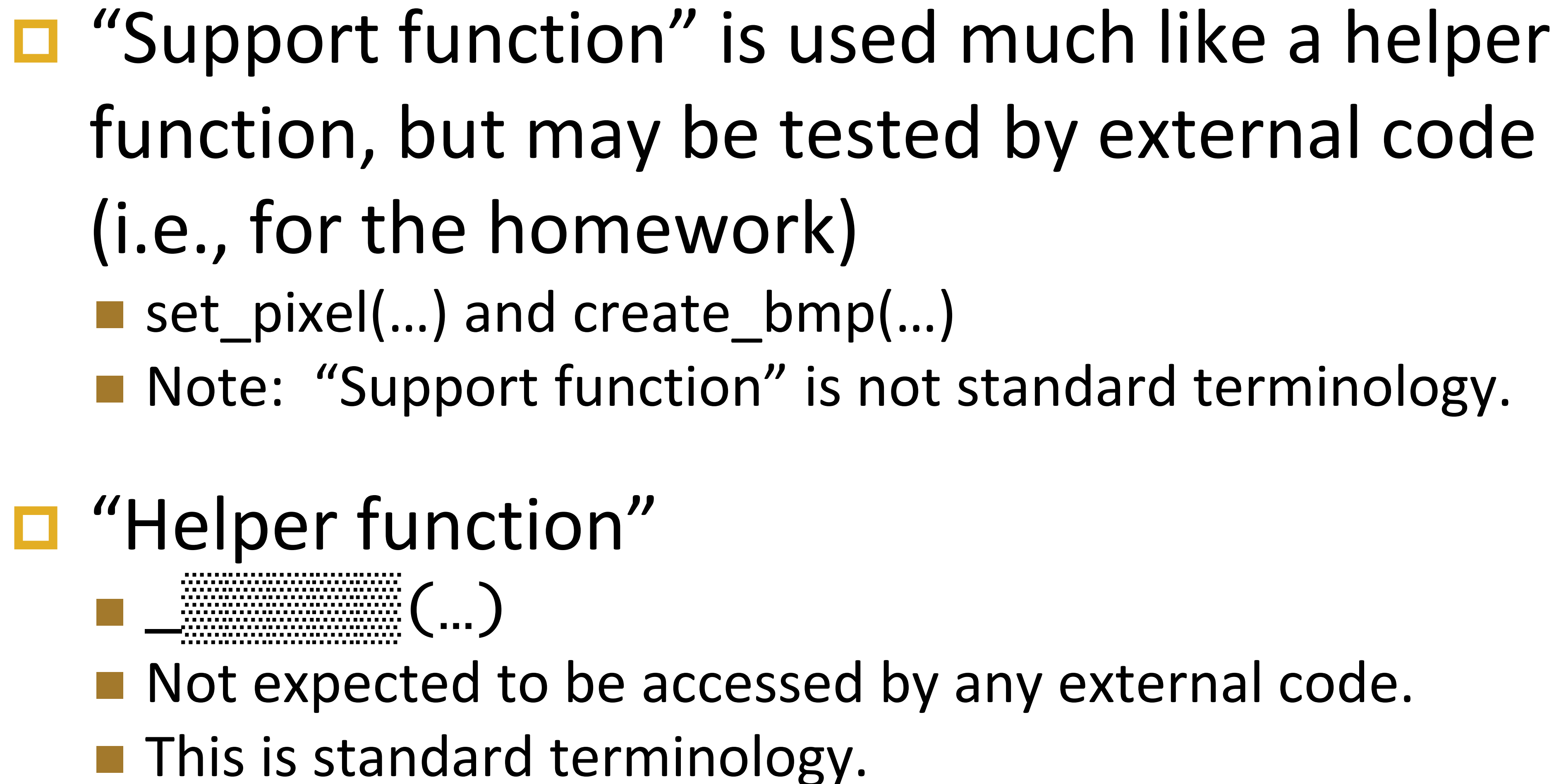
line coverage ⊆ branch coverage ⊆ path coverage

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

```c
////// IMPLEMENTATION CODE //////
void report_weather(bool is_sunny, bool is_raining) {
    if(is_sunny) {
        printf("The sun is shining.\n");
    }
    else {
        printf("The sun is not shining.\n");
    }

    if(is_raining) {
        printf("It is raining.\n");
    }
}

/////////// TEST CODE ///////////
void test_report_weather_1() { // LINE coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(false, true);  // The sun is not shining.  It is raining.
}

void test_report_weather_2() { // BRANCH coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(false, false); // The sun is not shining.
}

void test_report_weather_3() { // PATH coverage
    report_weather(true,  true);  // The sun is shining.  It is raining.
    report_weather(true,  false); // The sun is shining.  It is raining.
    report_weather(false, true);  // The sun is not shining.  It is raining.
    report_weather(false, false); // The sun is not shining.
}
```


© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

“Support functions” vs. “Helper functions”



© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

Thinking of test cases

- Easy cases
  - Answer is obvious (to you). If the test fails, you should have no doubt in your mind about whether the test itself is correct or not.
  - Ex: print_integer(5, 10)
- "Edge cases" (boundaries)
  - Extreme values for inputs (e.g., parameters, input files, etc.).
  - Ex: print_integer(INT_MIN, 10)
- "Corner cases" (turning points)
  - Look for `if(▒){…}`, `while(▒){…}`, `for(▒){…}`, and `▒?▒:▒` in your code
  - Will be captured whenever you have 100% branch coverage (hard)
  - Ex: print_integer(0, 10); print_integer(10, 16); print_integer(9, 16);
- Special cases (look for "except" in spec)
  - Look for words like "… except when…" or "Note: If …" in the specification.
  - Ex: mintf("%")

Note: This is not standard terminology. These are the instructor's invented terms.

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

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

```
huffman fluffs many mums
```

Next step: Make a frequency table

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

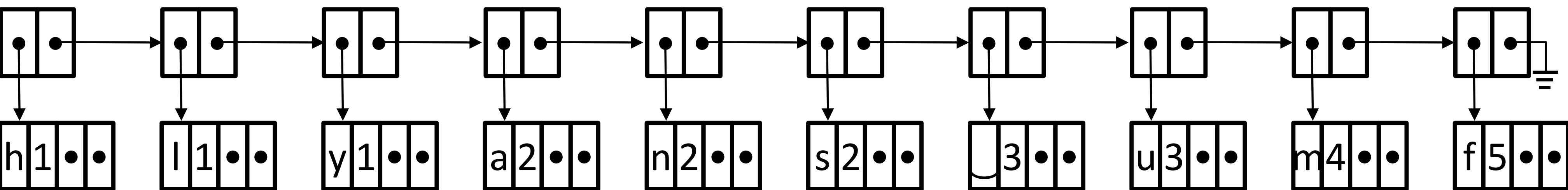
```
huffman fluffs many mums
```

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.

© 2020 Alexander J. Quinn
This content is protected and may not be shared, uploaded, or distributed.



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

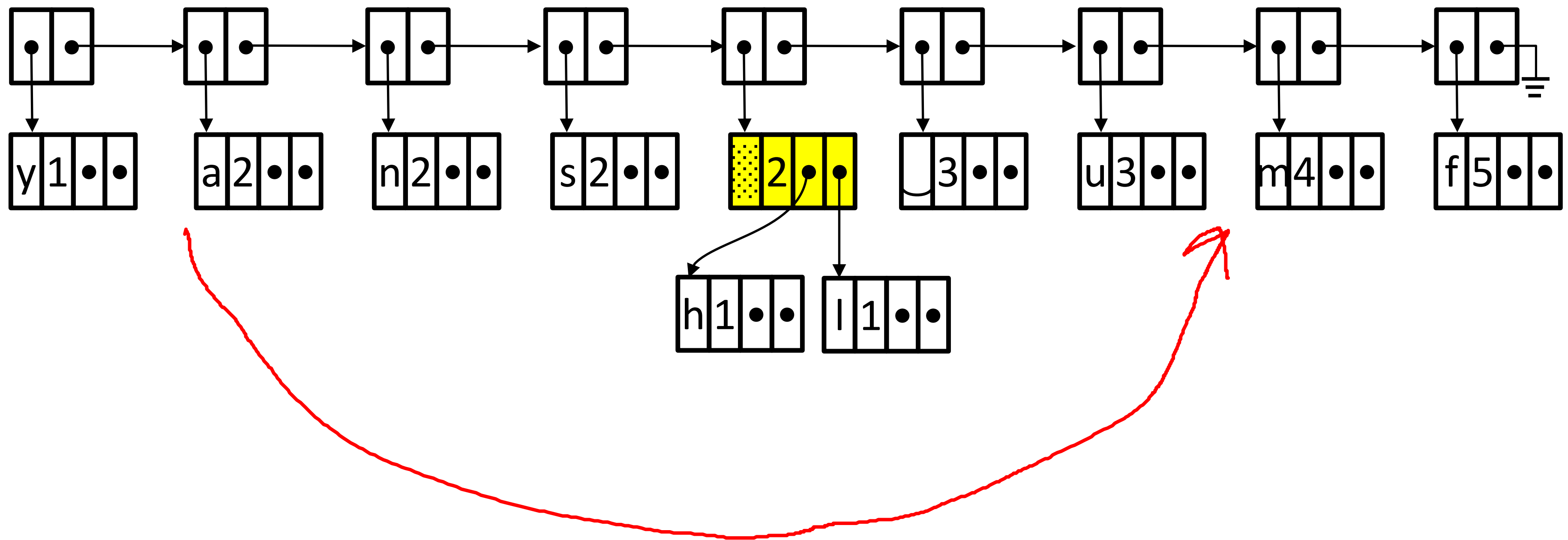
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority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

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



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.

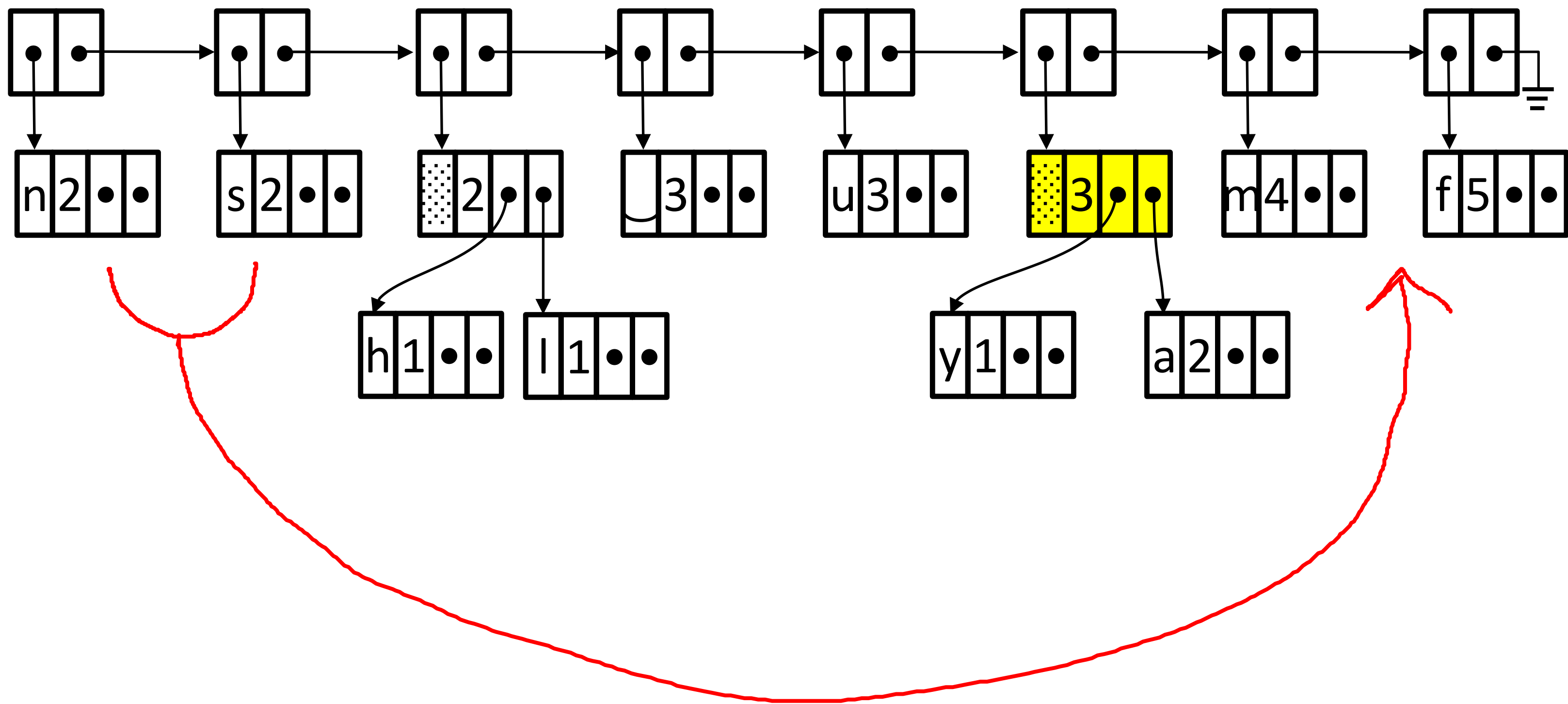
Next step: Join first two nodes

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.

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



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.

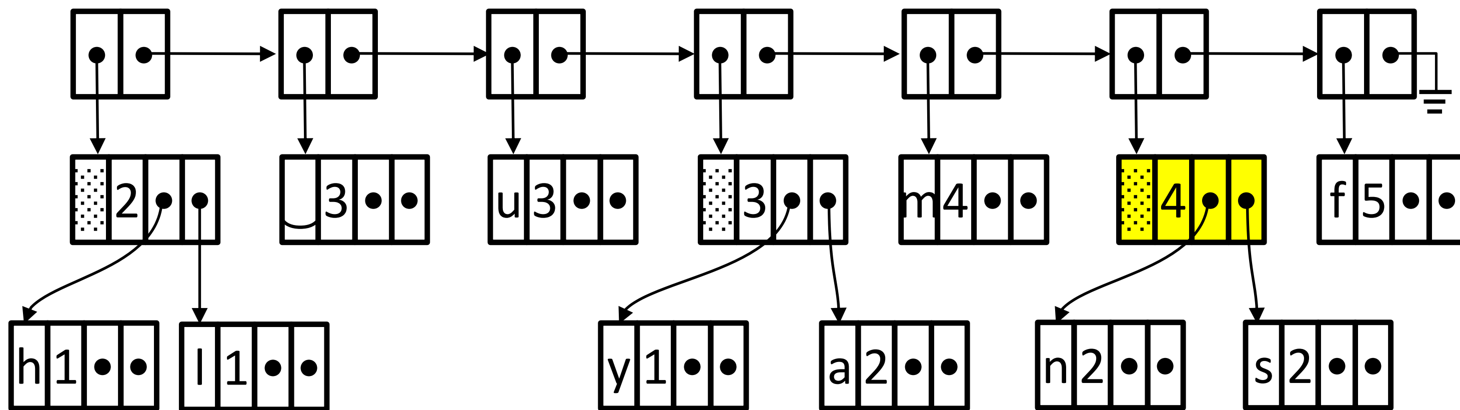
Next step: Join first two nodes

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.

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



Next step: Join first two nodes

<u>Process</u>

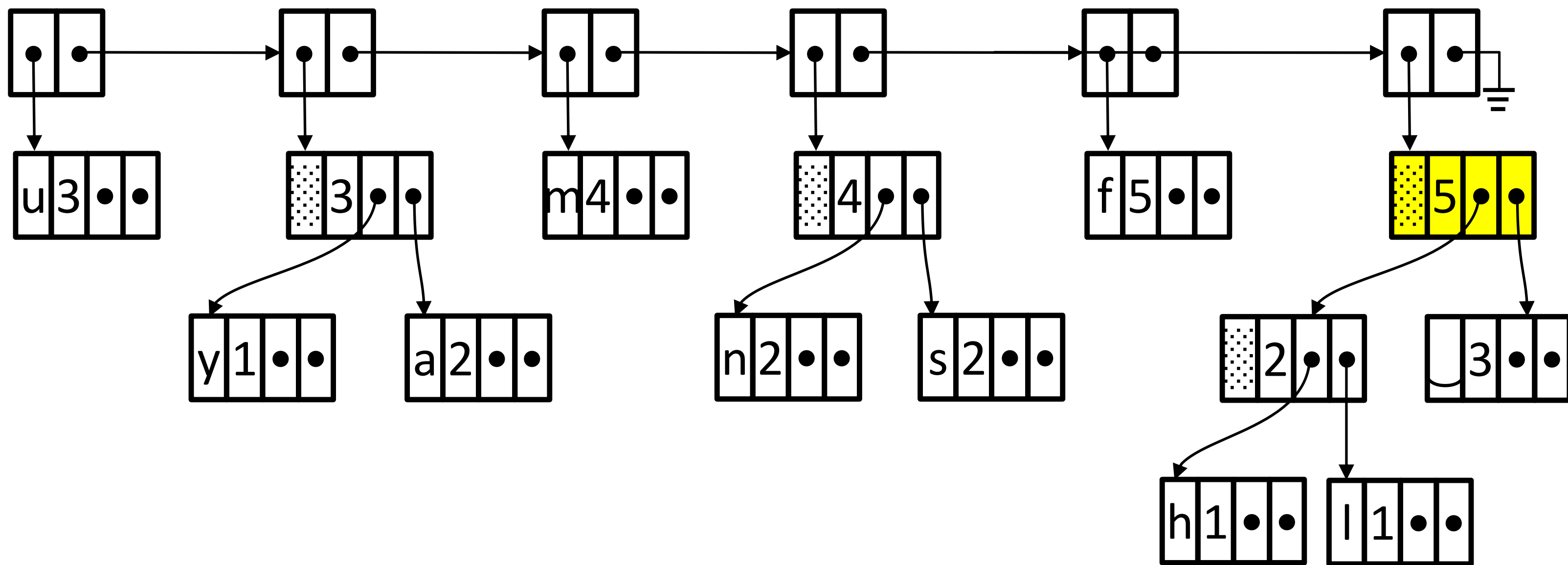
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority queue.

<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.

© 2020 Alexander J. Quinn
This content is protected and may not be shared, uploaded, or distributed.



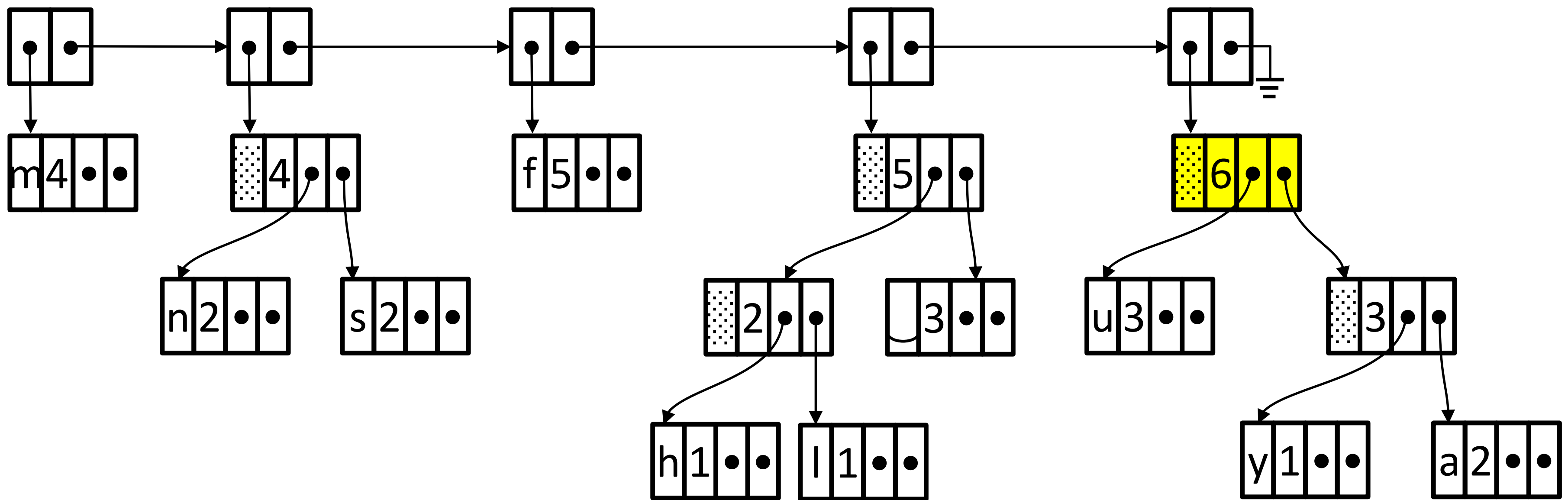
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority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

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



<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.

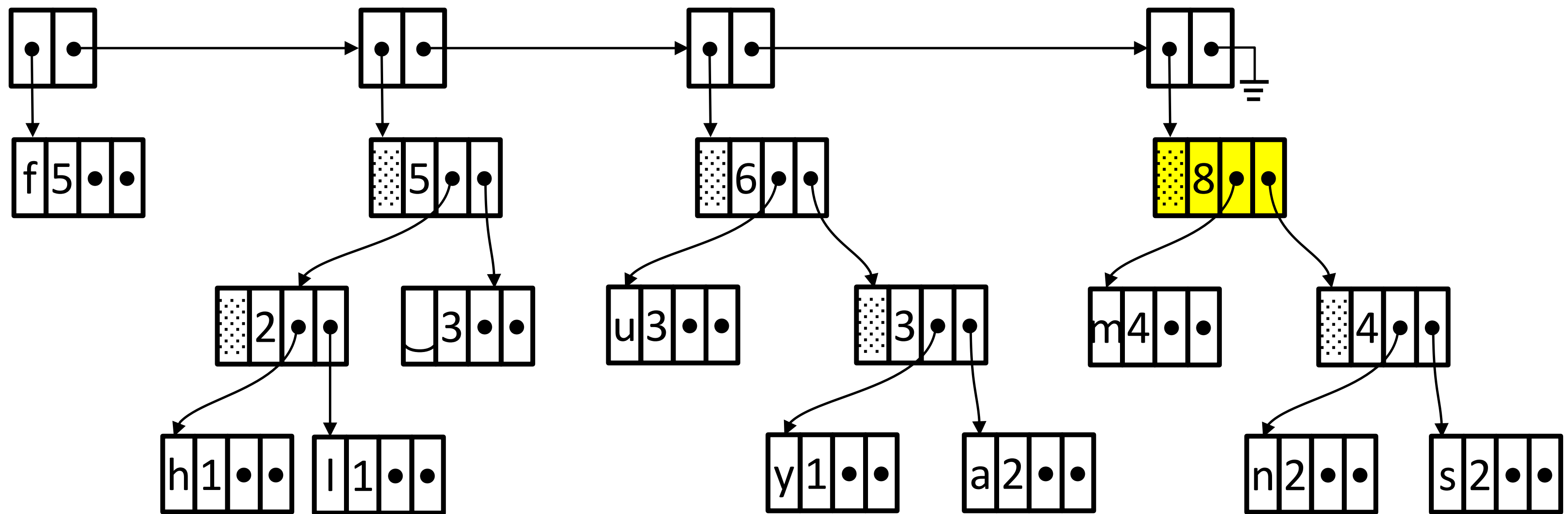
Next step: Join first two nodes

<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.

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



<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.

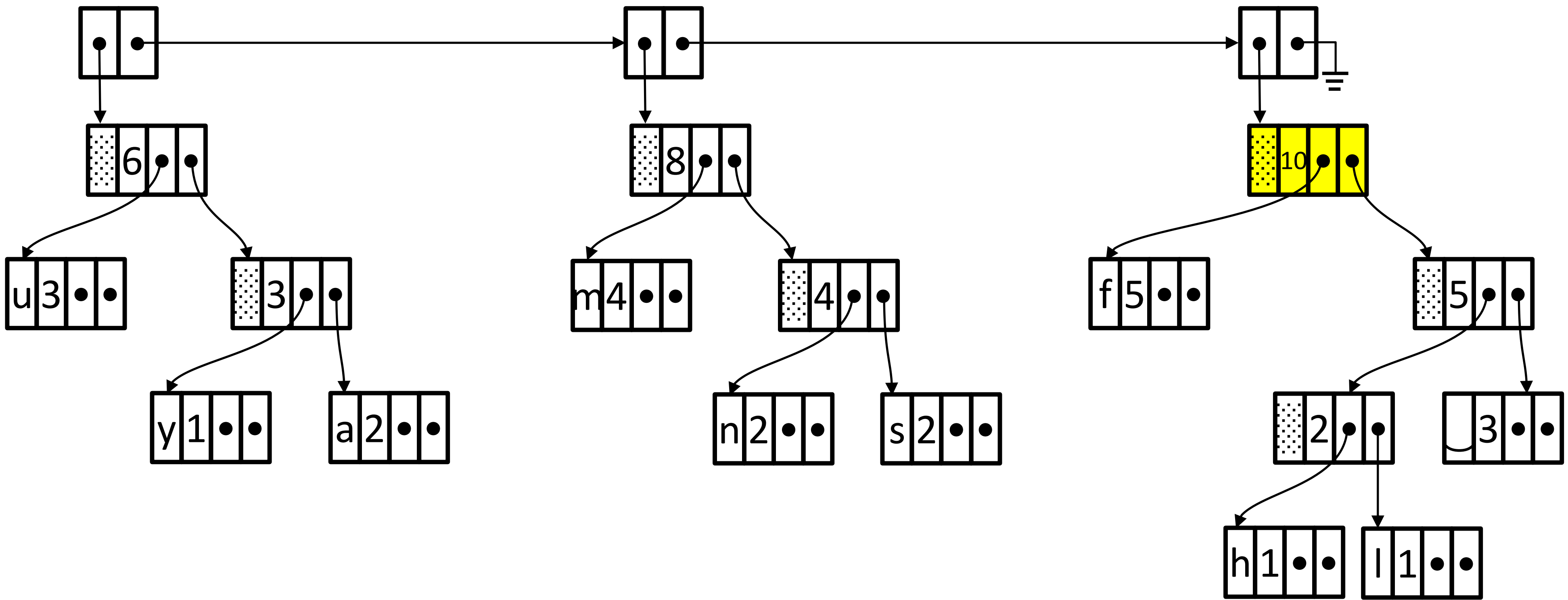
Next step: Join first two nodes

<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.

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



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.

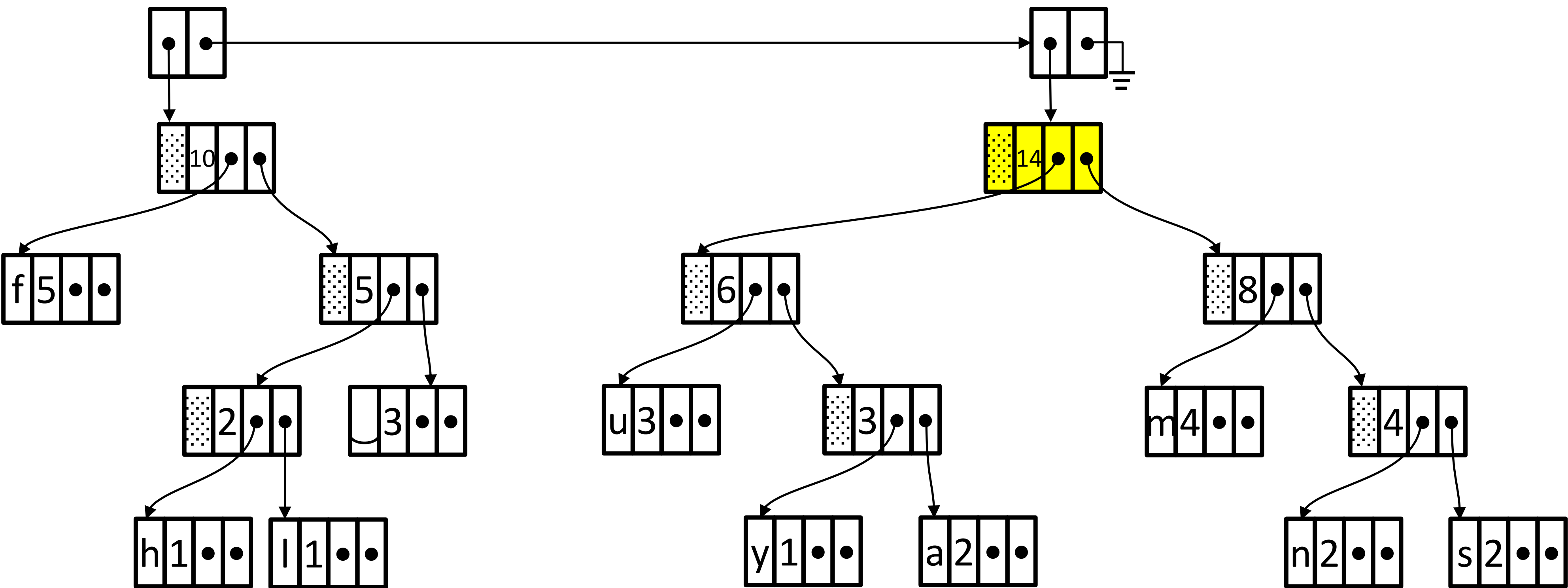
Next step: Join first two nodes

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.

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



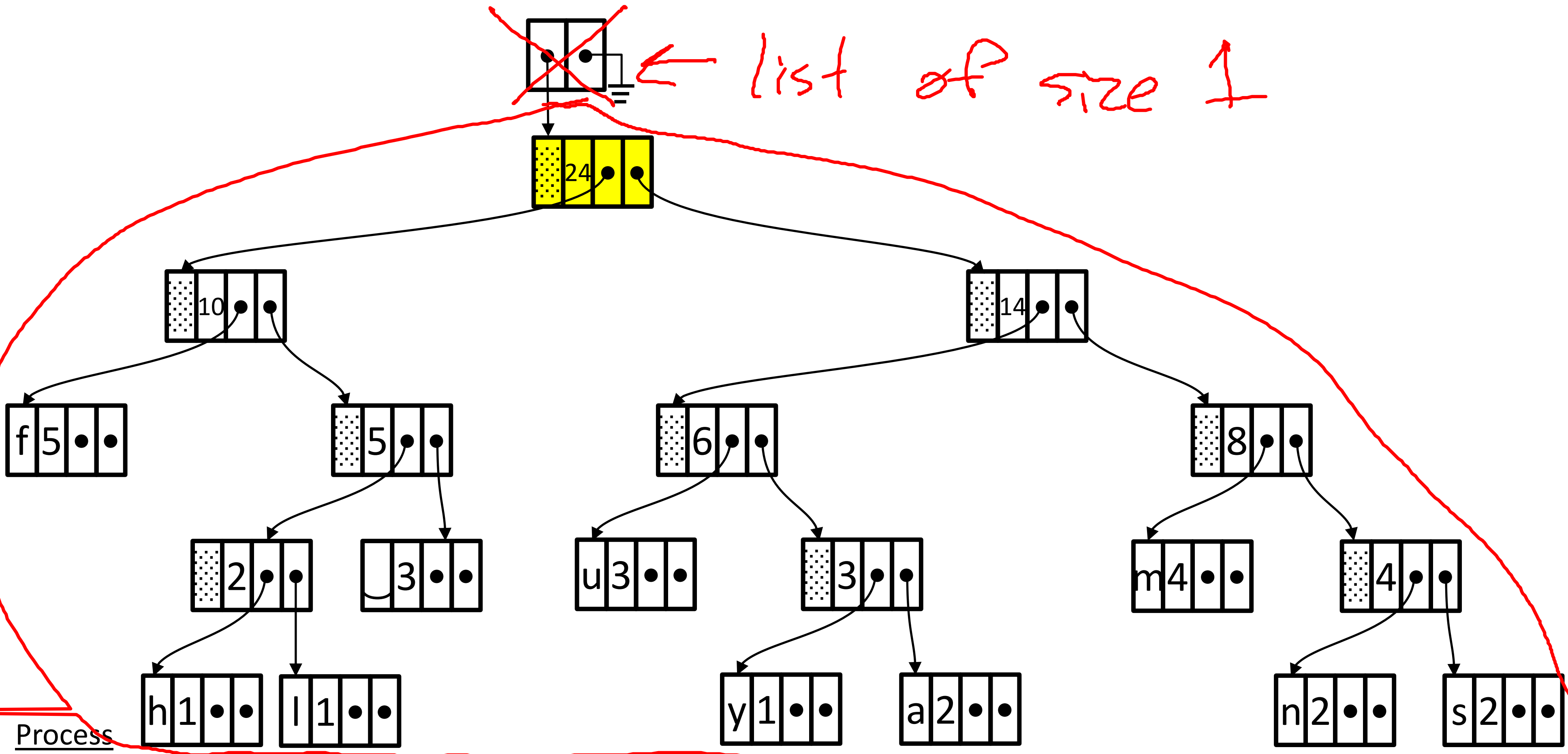
Process
1. Take first two nodes from priority queue.
2. Combine them into a cluster. (Will require creating a new tree node.) The cluster will have the sum of the frequencies of its children.
3. Insert the cluster into priority queue.
4. Repeat (from step 1) until there is only one node in the priority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

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



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.

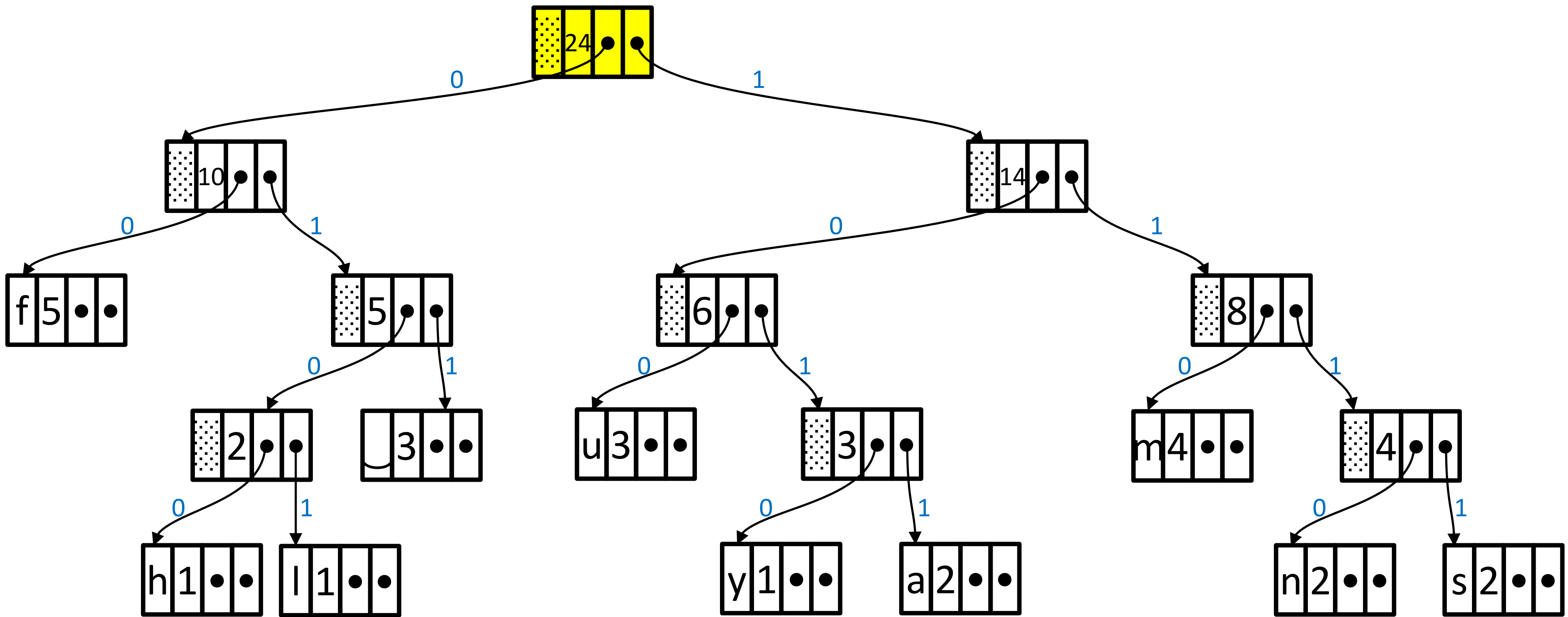
Priority queue compare function

- 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.

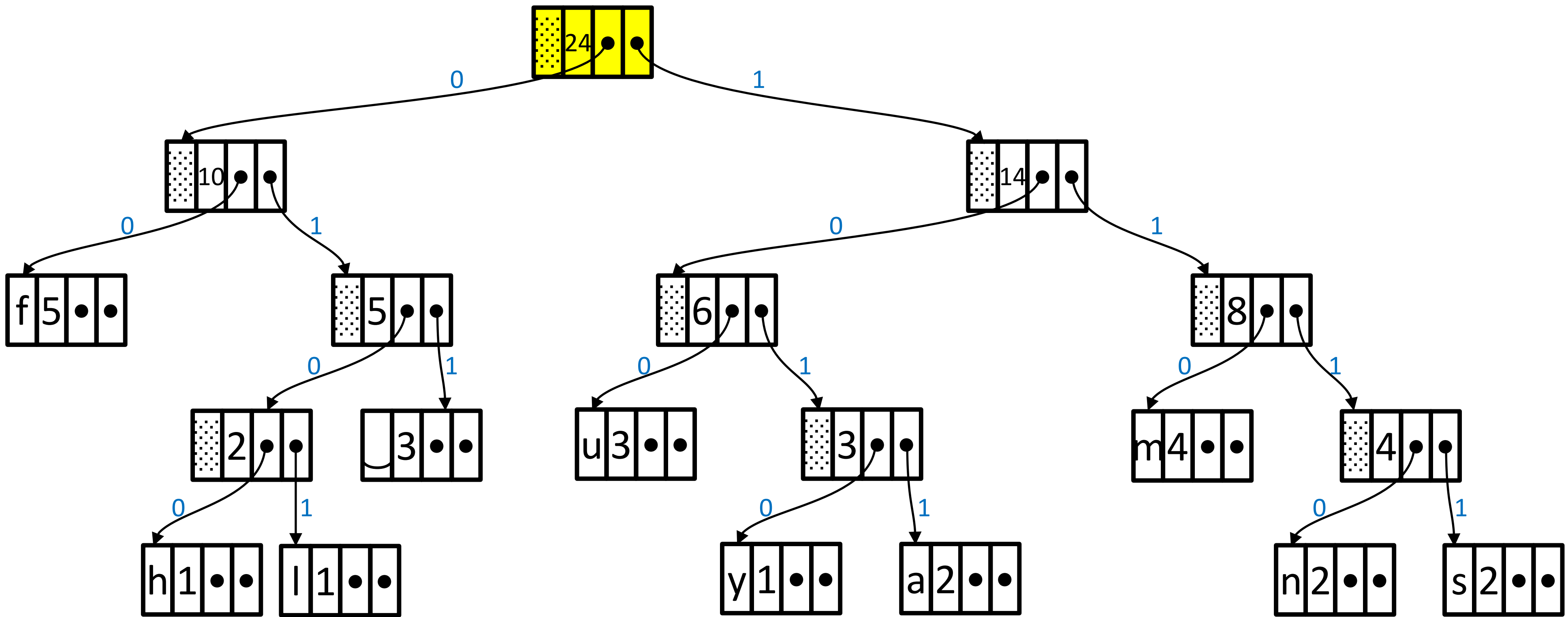
This summary is not a substitute for reading the homework description. In case of any discrepancy, it takes precedence.

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



Next step: Create the code table

 © 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

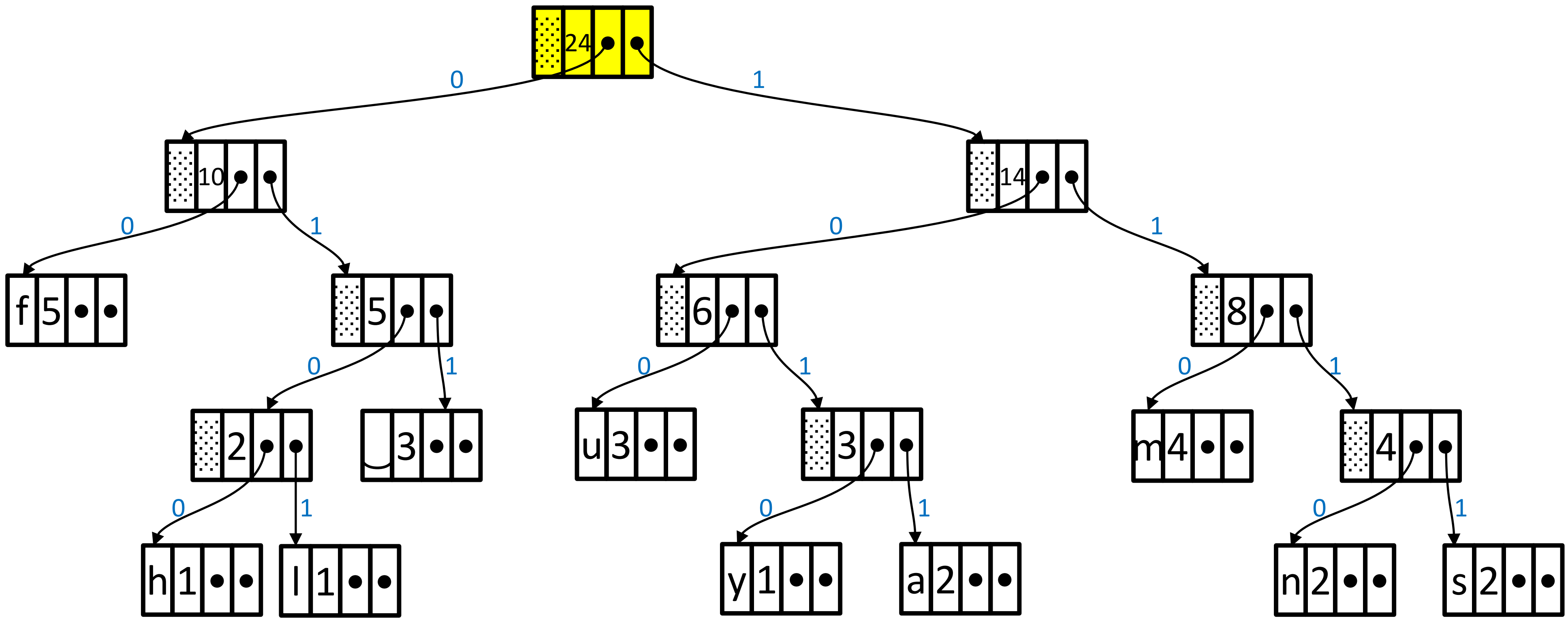


Code table

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.

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.



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.

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

huffman fluffs many mums

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

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

© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.

```
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
```


© 2020 Alexander J. Quinn This content is protected and may not be shared, uploaded, or distributed.