Here is an explicit toy example to see how Problem 1.3 works. Suppose our symbol alphabet consists of the 5 capital English letters A,B,C,D,E. These have ASCII values 65,66,67,68,69, respectively.

Let us illustrate the encoding process to create a bit stream. Suppose the input symbol stream is: B,A,B,D,E,...

Initially the counters are set as Counts: A (0), B(0), C(0), D(0), E(0).

I.e. g(x) = 0 for all x ∈ {A, B, C, D, E}.

Let’s build the initial tree. All counts are zero, so we need to find the 2 symbols with the smallest f value, namely A (i.e. f(A) = 65) and B (i.e. f(B) = 66) and merge them to form a new node AB with count 0. The left child is A and the right child is B since g(A) = g(B) = 0 and f(A) < f(B).

Now let’s repeat the process recursively. Again all node counts are zero, so we merge the 2 nodes with the smallest f value. These 2 nodes are AB (with f(AB) = min(65, 66) = 65) and C (with f(C) = 67) and we form a new node ABC with count 0. The left child is AB and the right child is C since g(AB) = g(B) = 0 and f(AB) < f(C).

The next merger will be ABC (with f(ABC) = min(65, 66, 67) = 65) and D (with f(D) = 68) to form the new node ABCD. The left child is ABC and the right child is D since g(ABC) = g(D) = 0 and f(ABC) < f(D).

The final merger will be ABCD with E to form ABCDE, i.e. the root node. The left child is ABCD and the right child is E since g(ABCD) = g(E) = 0 and f(ABCD) < f(E). Now the initial tree is complete.

Now the encoder takes the first input symbol, B, and encodes it with the initial tree to get the binary string “001” and then updates the counters to be: Counts: A (0), B(1), C(0), D(0), E(0).

Now it’s time to build the next Huffman tree in order to encode the 2nd input symbol. First we merge A and C since g(A) = g(C) = 0 and f(A), f(C) < f(D), f(E). This forms a new node AC with count 0. The left child is A and the right child is C since g(A) = g(C) = 0 and f(A) < f(C).

Next we merge node AC with node D to form new node ACD. The left child is AC and the right child is D since g(AC) = g(D) = 0 and f(AC) < f(D). Then we merge node ACD with node E to form node ACDE. The left child is ACD and the right child is E since g(ACD) = g(E) = 0 and f(ACD) < f(E). Finally we merge node ACDE with node B (note g(B) = 1 so it’s merged last) to form the root node ABCDE. The left child is ACDE and the right child is B since g(ACDE) < g(B).

So the 2nd tree looks like this:
Now, the encoder takes the 2nd input symbol, A, and encodes it with the 2nd tree to get the binary string “0000” and then updates the counters to be: \textbf{Counts:} A (1), B(1), C(0), D(0), E(0).

Let’s build the 3rd tree now. First merge nodes C and D to form new node CD with C on the left and D on the right. Then merge nodes CD and E to form new node CDE, with CD on the left and E on the right. Then merge nodes A and CDE to form new node ACDE. This time, CDE will be on the left and A on the right, since \( g(CDE) < g(A) \). Finally merge ACDE and B to form the root. Node ACDE will be on the left and B on the right, since \( g(ACDE) = g(B) = 1 \) and \( f(ACDE) = f(A) = 65 < 66 = f(B) \).

So the 3rd tree looks like this:

Now, the encoder takes the 3rd input symbol, B, and encodes it with the 3rd tree to get the binary string “1” and then updates the counters to be: \textbf{Counts:} A (1), B(2), C(0), D(0), E(0).

Let’s build the 4th tree. First merge nodes C and D to form new node CD with C on the left and D on the right. Then merge nodes CD and E to form new node CDE, with CD on the left and E on the right. Then merge nodes A and CDE to form new node ACDE. This time, CDE will be on the left and A on the right, since \( g(CDE) < g(A) \). Finally merge ACDE and B to form the root. Node ACDE will be on the left and B on the right, since \( g(ACDE) = 1 < 2 = g(B) \).

So the 4th tree looks identical to the 3rd tree:

Now, the encoder takes the 4th input symbol, D, and encodes it with the 4th tree to get the binary string “0001” and then updates the counters to be: \textbf{Counts:} A (1), B(2), C(0), D(1), E(0).

Let’s build the 5th tree. First merge nodes C and E to form new node CE with C on the left and E on the right. Then merge nodes CE and A to form new node ACE, with CE on the left and A on the right.
Then merge nodes ACE and D to form new node ACDE. This time, ACE will be on the left and D on the right, since $g(ACE) = g(D) = 1$ and $f(ACE) = f(A) = 65 < 68 = f(D)$. Finally merge ACDE and B to form the root. Node ACDE will be on the left and B on the right, since $g(ACDE) = g(B) = 2$ and $f(ACDE) = f(A) < f(B)$.

So the 5th tree looks like this:

```
   0
  / \
 1   B
/   /
0   1
  \
  D
/   /
0   1
  \
A   E
```

Now, the encoder takes the 5th input symbol, E, and encodes it with the 5th tree to get the binary string “0001” and then updates the counters to be: **Counts: A (1), B(2), C(0), D(1), E(1)**.

Continue this process.