This lecture explains how to use bits in Huffman compression.

Let’s review what we have done so far. The Huffman compression method gives fewer bits to the characters that appear frequently, and more bits to the characters that appear rarely. On average, fewer bits are needed per character.

A compressed file has three parts: the first part describes the code tree using post-order traversal. The second part is a 4-byte integer as the length of the original file measured in bytes. The third part is the codes of the original data.

Consider this example we showed earlier. The post order traversal generates this description:

one, ei, one, M, one, #, one, gee, zero, zero, zero, one, cee, one, ass, zero, zero, zero.

The data may be expressed as

Zero, zero for ei.

Zero, zero for ei.

one, zero for cee.

one, one for ass.

etc. .

How can we express the information using bits?

First, let’s write down the Ass key code for the characters. Ei is 65 in decimal and 41 in hexadecimal. M is six dee in hexadecimal. # is two three in hexadecimal. The ones and zeros for control are marked.

We need to use only one bit for each one or zero for control.

The very first bit is a control bit and it is one.

This control bit is followed by character ei. Character ei is four one in hexadecimal: zero, one, zero, zero,

Zero, zero, zero, one.

The control bit one already uses one bit. Thus only seven bits are available in this byte. the next seven bits come from the first seven bits of character ei.

The last bit of ei has to be moved to the next byte.

The first byte is one, zero, one, zero, zero, zero, zero, zero.

The second byte starts with one. This is the last bit from the character ei.

The next bit is a control bit one.

Since the first bit is already occupied by the last bit of ei, this control bit has to be the second bit in the byte.

The next six bits come from character ‘M’. .

‘M’ is six dee in hexadecimal. The first four bits are zero, one, one zero. The next four bits are one, one, zero, one.

Two bits in this byte are already used. Thus, only six bits from ‘M’ can fit in this byte.

The last two bits of ‘M’ have to move to the next byte.

The next is a control bit of one. This is the third bit of the byte because the first two bits are already used for the last two bits of ‘M’. .

The first three bits of this byte are already used; thus, only five bits from # can be stored in this byte.

The last three bits of # have to go to the next byte.

The next is a control bit of one. This is the fourth bit of the byte because of the first three bits are already used.

How do we implement the process in a Cee program? We are going to use the bit wise operations explained earlier.

First, we can create an unsigned char storing one, followed by seven zeros. This can be done by create a hexadecimal number eight zero.

This will be the first control bit.

Next, we can take ei and shift right by one bit. Only seven bits are left.

The first byte uses bit wise or of the two values earlier.

Next, we want to get the last bit of ei. This is done by using bit wise and with hexadecimal zero one. Only the last bit is kept and all the other bits are set to zero because of bit wise and.

We need to shift this bit left by seven bits so that it becomes the first bit of the next byte.

The next is a control bit one. It is the second bit of this byte. Thus, we can take hexadecimal eight zero and shift it right byte one bit. It is the same as hexadecimal four zero.

The next six bits come from the first six bits of ‘M’. .

To get the first six bits of ‘M’, we take the character and shift it right by two bits. This will discard the right most two bits.

This byte is composed of three things: The first bit comes from the last bit of ei. The second bit is a control bit of one. The other six bits come from the first six bits of ‘M’. .

To put all of them together, we use bit wise OR. .

Please notice that when we shift right the first time, we shift right only one bit.

When we shift right the second time, we shift right two bits. Later, we need to shift right three bits.

For shift left, the first time we need to shift left 7 bits. Later, we need to shift left only six bits.

We can continue is the process as shown on this slide.

We will use hexadecimal zero three to get the last two bits of ‘M’. .

This needs to be shifted left six bits.

The next is a control bit of one. This needs to be shifted right by two bits because the first two bits are already occupied by the last two bits of ‘M’. .

What is the general solution?

We can use hexadecimal eight zero to get the control bit of one. This bit needs to be shifted to the correct place.

We need to keep track where the bit is shifted to. At the very beginning, hexadecimal eight zero does not need to be shifted. Then, it needs to be shifted right by one bit. Then, it needs to be shifted right by two bits. Then, it needs to be shifted right by three bits.

We also need to shift the bits for the characters. The first character needs to be shifted right by one bit so that only seven bits are used for the first byte. Then, the last bit needs to be shifted left seven bits and becomes the first bit in the next byte.

The second character needs to be shifted right two bits first to keep the first six bits. The last two bits need to be shifted left by six bits for the next byte.

We introduce a concept called mask here. What is a mask? A mask can block some information and allow some other information to pass through. Think of these two masks drawn here. They will block the wearers’ cheeks and foreheads. Meanwhile, the masks allow the eyes to see.

The example here is a mask of hexadecimal zero one. If we do bit wise and with this mask, then all bits, except the last bit, will be set to zero. This mask blocks the first seven bits and allows the last bit to go through.

It is likely that the post-order description of the code tree has some unused bits in the last byte. If this occurs, the unused bits will be zeros.

The second part of the header is the four byte integer storing the length of the file.

The third part of the file is the data expressed by the code. Each of the first four characters needs only two bits. Thus, the first four characters can be compressed into using only one byte.

The next two characters, # and M., need only seven bits.

The first bit of gee is moved to the last bit of the second byte.

This process continues until all characters are represented using the codes.

It is possible that some bits in the last byte are not used. If this occurs, the unused bits are zeros.