Data Compression

Data Compression

- Data compression is the process of encoding information using a representation that requires fewer bits than the original representation.
- Although data storage capacity and network speeds are increasing all the time, there are still many occasions when it would be useful to reduce the size of data.

Data Compression

Why reduce the size of data.

- The storage device may be of a fixed size.
- We may be short of space and not able to upgrade.
- We may want to archive data and use as little space as possible to store it.
- Even at fast transmission speeds, it will be faster to send less data bytes. At slow speeds, it may not be practical at all.
- Some file systems limit the size of files, so to store large files may require them to be compressed.

Data Compression

Compression of data is often combined with archiving, e.g. in the WinZip utility, but in fact compression is a separate issue:

- Compression reduces the size of the data to be stored or transmitted
- Archiving creates a single file which contains a number of files/folders within it, making it easier to transfer and manage.

Data Compression

There are two approaches possible:

- Compress the individual files, then place them in the archive (an **archive of compressed files**), typically done by Windows programs such as 7-Zip and Winzip.
- Put the files in an archive, and then compress the result (a **compressed archive**), typically done on Linux/UNIX using tar(1) and a compression program such as gzip(1).

Data Compression

The results can be slightly different:

- Compressing each file separately allows the best compression method for that file to be chosen.
- Compressing the archive as a whole means the overhead of the archive is also compressed.
- Extracting/adding to a compressed archive means the whole archive must be uncompressed first.
- An error in a compressed archive could cause the whole archive to be unreadable.

In reality, there is very little difference.

Data Compression

We can consider data compression to fall into two broad categories:

- Lossless compression
- Lossy compression

Loseless Compression

- On decompressing the data, we get back exactly what we had originally.
- The basis of lossless compression is statistical redundancy in the data:
  - Most data contains repetitive parts. For example, text files often have the same string of characters repeated (“the”).
  - Data is usually encoded in a form that is convenient for the programmer, but may not be the most efficient representation.
    This leads to redundancy.

Lossy Compression

- On decompressing the data, we get back something very similar but of reduced quality because some of the data has been discarded by the compression.

- The success of compression will depend on the type of data, and for this reason there are many different algorithms available.
- Some are general-purpose, and will compress anything, others are designed for particular data types and give better results for that data than a general-purpose algorithm.
- Lossy compression is usually for specific data types, e.g. JPEG for images, MP3 for audio, H.264 for video.

- In general, the better the compression obtained, the more time is required to compress and decompress.
- However, there are algorithms which are slow to compress but fast to decompress, and vice versa.
- No algorithm is perfect for all uses.

Compression Effectiveness

- We can measure the effectiveness of a compression algorithm by using the compression ratio:

  $$CR = (\text{size after compression}) \div (\text{size before compression})$$

- Thus a compression algorithm must give a $CR < 1$ to be any good!
- We usually give a percentage: e.g. compression ratio of 88%, means the compressed file is 88% of the size of the original.
- Most compression programs will show you the compression ratio.

Compression Effectiveness

- Archiving programs can use this to check that files have been reduced by compression, and which algorithms gave the best result.

Note:

- In some cases, compression can result in a file that is larger than the original.
- This usually means an unsuitable algorithm for the data was used.

Alternative Definition of Compression Ratio

- Another way of defining CR is

  CR = 1 – [(compressed size) ÷ (original size)]
- So in the case above we would have CR = 1 – 0.88 = 0.12 or 12% Using this definition, a file that does not reduce at all has a ratio of 0%, while a file that compressed to nothing (unlikely!) would be 100% compressed.
- These are simply different ways of expressing the same thing, so you need to know which one is being used.
- Compression can also be described as “2:1” or similar: this means the original file was twice the size of the compressed result.

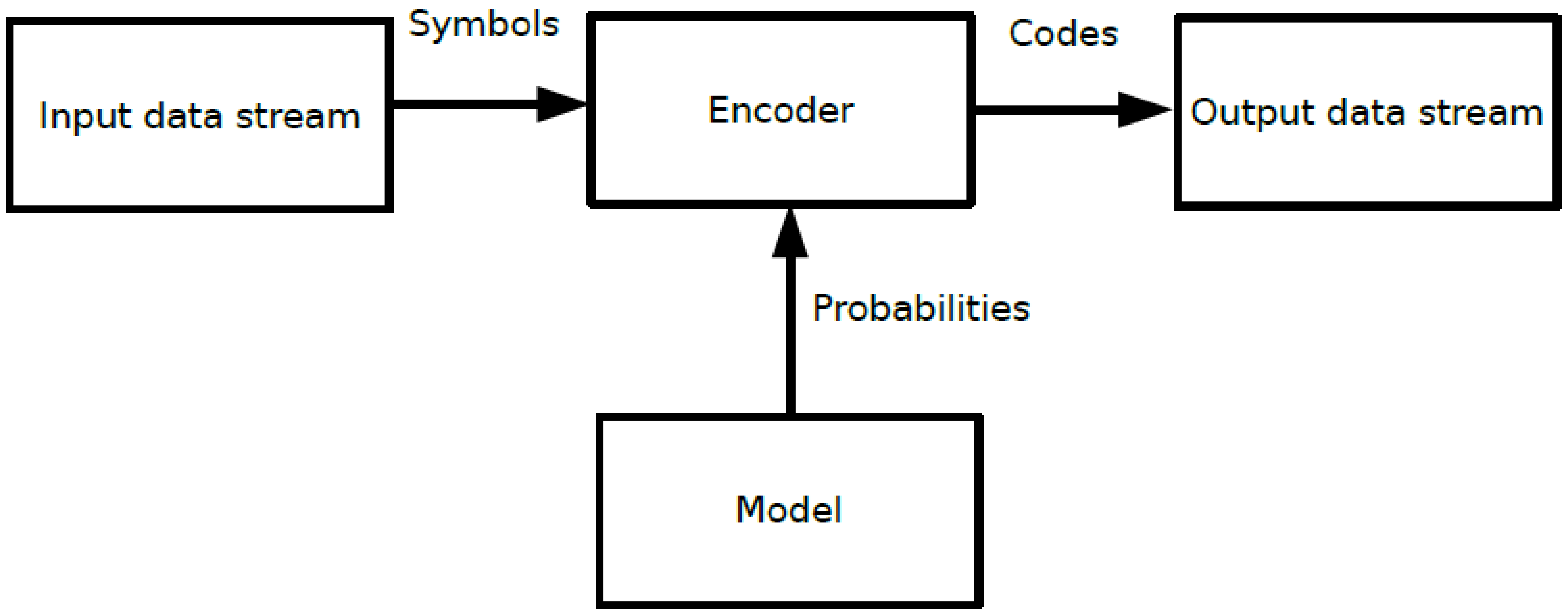
Modelling and Coding

- With lossless compression we are taking a stream of symbols and replacing them with codes.
- We can reverse this process to recover the original data.
- If compression is working on the data, then the stream of codes is smaller than the original stream of symbols.
- We can consider this process as having two parts: the codes, and a model which describes how to choose the codes to output.
- The model should match the type of data we are compressing so that we get the best result.

Modelling and Coding

- An encoder uses the model to decide which code should be used to replace symbols.
- The model describes the probability of each possible symbol (how likely it is to appear).
- If we have the probabilities for a given stream of data, we can choose the best encoding for it.

Modelling and Coding



Modelling and Coding

There are two approaches:

- Profile a type of data in advance, and build the encoder using that information.
- Build the encoder to profile the data as it works and choose the encoding based on the changing data.

“Profiling the data” here means estimating the probability of a symbol occurring in the input stream.

Modelling and Coding

- The first approach was more practical with early compression systems, while the second has the potential for better results in general.
- For us, a symbol will usually correspond to one byte; however, it could be any size “chunk” of data, and all symbols do not have to be the same size in a given data set.
- What we are concerned with is how often they occur in the data.

Modelling and Coding

- The compression ratio gives us a measure of how good the compression is.
- However, it doesn’t tell us how good it could be.
- We need some way to calculate a lower bound on the size of data: how small could we make it without losing anything?
- This would tell us how good our compression method is compared to the best possible method (which might not be achievable in practice).
- To find a lower bound, we need to use theoretical concepts.

Probability

- What do we mean by the probability of a symbol?
- We can look at it in two ways.
  - Frequency-based probability
  - Observer-relative probability

Frequency-based probability

- Suppose we have a series of events.
- Each event is a possible outcome of doing something, or of making a choice, etc.
- Examples are rolling dice (outcome is the number), or picking a character at random from a text file (outcome is the character).
- If there are N events, and event e_i occurs n_i times, then the probability of e_i is defined to be

$$P_i = {}^{n_i}/_{N}$$

- assuming all p_i are independent (i.e. probability of an event does not depend on previous ones).

Frequency-based probability

- In other words, the probability of an event is simply the proportion of the time that the event occurs.
- If we know all the events, we can easily measure this.
- If we do not know them all, then we can estimate the probability of each event based on what we know.
- A probability of 1 means an event is certain to happen; a probability of 0 means it will never happen.
- If you add together all the p_i values for the series of events, they add to 1 – something must happen!

Observer-relative probability

- In this view, probability is a statement of what the observer believes about the likelihood of the event.
- Our assessment of probability will change as we know more about the events.
- This is the basis of Bayesian statistics, and is used for systems like spam filters; however, it is not what we need for compression.

In practice, we often have to estimate frequency-based probability since we don’t know all the events (i.e. the data) in advance.

Information and Entropy

- Information entropy is the average rate at which information is produced by a random source of data.
- When the data source produces a low-probability value (i.e., when a low-probability event occurs), the event carries more "information" than when the source data produces a high-probability value.
- The amount of information conveyed by each event defined in this way becomes a random variable whose expected value is the information entropy.

Information and Entropy

- If we know that a symbol has high probability, then we can use fewer bits to encode it (saving space because it occurs frequently);
- If it has low probability, then we can use more bits for the code (but will have to output it less often).
- This concept is part of information theory developed by Claude Shannon in the 1940s.

Information and Entropy

- The information of an event is defined in terms of its probability:

$$I(e_i) = -\log_2(p_i) = \log_2(1/p_i)$$

- Why minus?
  Since probability p is always less than or equal to 1, and log of a number less than 1 is negative, we would have entropies < 0.
  It is more convenient to multiply by -1 and have positive numbers.

Information and Entropy

- Shannon then defined the entropy of a series of events as follows:

$$H = -\sum_{i=1}^{N} p_i \log_2(p_i)$$

- where p_i is the probability of event e_i as above. Using $\log_2$ gives entropy in bits.
- The symbol Σ means “sum”, i.e. you add together all the different values.
- Σ is the Greek letter capital sigma.

Information and Entropy

- In other words, the entropy is the sum of the information of an event multiplied by the probability of that event, for all events.
- It gives the average information per symbol in the data.
- If there are N symbols, and the entropy is H, then the smallest lossless representation must use $N \times H$ bits.
- This is called the source coding theorem.

Information and Entropy

IMPORTANT:

- This Shannon entropy gives the size of the shortest possible lossless encoding of data, based on the probabilities of the symbols making up the data.
- We can use this to assess how good our encoding actually is, but we can’t do better than this without losing information.

Information and Entropy

- These bits are the same as the electronic/magnetic bits in a computer.
- However, we can use the entropy to decide the minimum amount of storage needed in the computer, or the best compression possible – you can’t compress to less than the number of bits given by the entropy.

Information and Entropy

- We can use these ideas to tell us how efficient a coding system is. As an example, suppose we have a text of 1024 characters where the letter “a” occurs 64 times.
- The probability of “a” is:

  $p(\text{“a”}) = 64/1024 = 1/16$
- The information is:

  $I(\text{“a”}) = -\log_2(1/16) = \log_2(16) = 4$ bits

Information and Entropy

- Suppose we are representing our text using an encoding like ASCII, which has 8 bits for each character.
- We are therefore using twice as many bits to represent "a" as necessary for this text.
- Now suppose there are 8 "q" characters in the text. The information of "q" is:

$$I(\text{“q”}) = -\log_2(8/1024) = -\log_2(1/128) = \log_2(128) = 7 \text{ bits}$$

- If there is only one "z" in the text, its information is 10 bits.

Information and Entropy

- The interpretation of this is that “z” gives us more information than “a” because it is less likely to occur, and therefore we learn more from it when we see it.
- This idea of information is sometimes surprising in what it says.
- For example: if you know for certain that something is going to happen, then its occurrence gives you no information:

$$p = 1$$

$$\log_2(1/p) = \log_2(1) = 0$$

Information and Entropy

- So if you know you are going to be sent the Encyclopaedia Britannica, even when you receive it you have received no information in the sense of entropy of the event!
- The string “aqz” in ASCII needs 8+8+8 = 24 bits to store it;
- Information theory tells us it could be represented in 4+7+10 = 21 bits for the probabilities above in the 1024 character text.

Information and Entropy

- Finally, suppose that the rest of the text is made up of “b” characters; there must be 951 of them (1024 – 64 – 8 -1).
- The probability of “b” is 951/1024 and its information is 0.1067 bits.

Information and Entropy

- We can calculate the entropy H:

Character	Freq f	=f/1024	-p x $\log_2(p)$
a	64	0.0625	4
q	8	0.0078125	7
z	1	0.0009765625	10
b	951	0.987109375	0.1067
Total	1024	1	0.4135

- So Entropy H = 0.4135451603 bits

H x N = 0.4135 x 1024 = 423.47 bits

Information and Entropy

- This is the minimum size possible for the same information to be represented.
- If we are actually using 8 bits/character, the data requires $1024 \times 8 = 8192$ bits.
- So ideally we would be able to represent common characters like “a” with fewer bits than scarce ones like “q” and “z”.
- We don’t usually do this in practice because it is awkward to have different-sized characters for normal use, but for compression it can be used to tell us how to encode the data by using new codes to replace the symbols.

Information and Entropy

- The model of the data gives us the probabilities of the symbols we are looking at, and therefore we can choose the best codes to represent them.
- Choosing the model determines how well the compression will work: if it accurately describes the data we can approach the best possible compression.

Information and Entropy

- A compression scheme should replace the symbols with a code using a smaller number of bits.
- The question is now how to choose a model and encoding.
- It is important to understand that the information of a symbol depends on the context
- If it is rare then it carries more information.
- So different files have different values for each character, & different types of data have different probabilities for each symbol, e.g. “e” is common in English so has low information in that type of data.

Statistical Models and Adaptive Coding

- We could draw up a static model of the data we expect to see and assign codes based on that.
- The drawback is that changing data would not be accurately modelled.
- Modern compression uses adaptive coding, which examines the data as compression proceeds to build a statistical model of the data being compressed.
- The statistical model tracks the probability of each symbol based on the symbols that have previously appeared in the input stream.

Statistical Models and Adaptive Coding

- The idea is, to build a table of data values and codes as we go.
- The table is then sent with the compressed data. The model is described by how many previous symbols are considered:
  - Order 0: no previous symbols
  - Order 1: one previous symbol
  - etc.

Statistical Models and Adaptive Coding

- The order can affect the table significantly.
- For example, in an order 0 model the letter “u” may be fairly low probability.
- However, in an order 1 model “u” would be seen as highly probable if the previous symbol was “q” (for English text).
- The table tells what code to use for the symbol based on the previous symbols, so the higher the order the larger the table will have to be.

Statistical Models and Adaptive Coding

- With an order 0 model considering one byte at a time, there would need to be a table with 256 entries:

Symbol	Code
U	…

Statistical Models and Adaptive Coding

- With an order 1 model, we need to have entries for every possible pair of symbols: instead of just “u” we must have

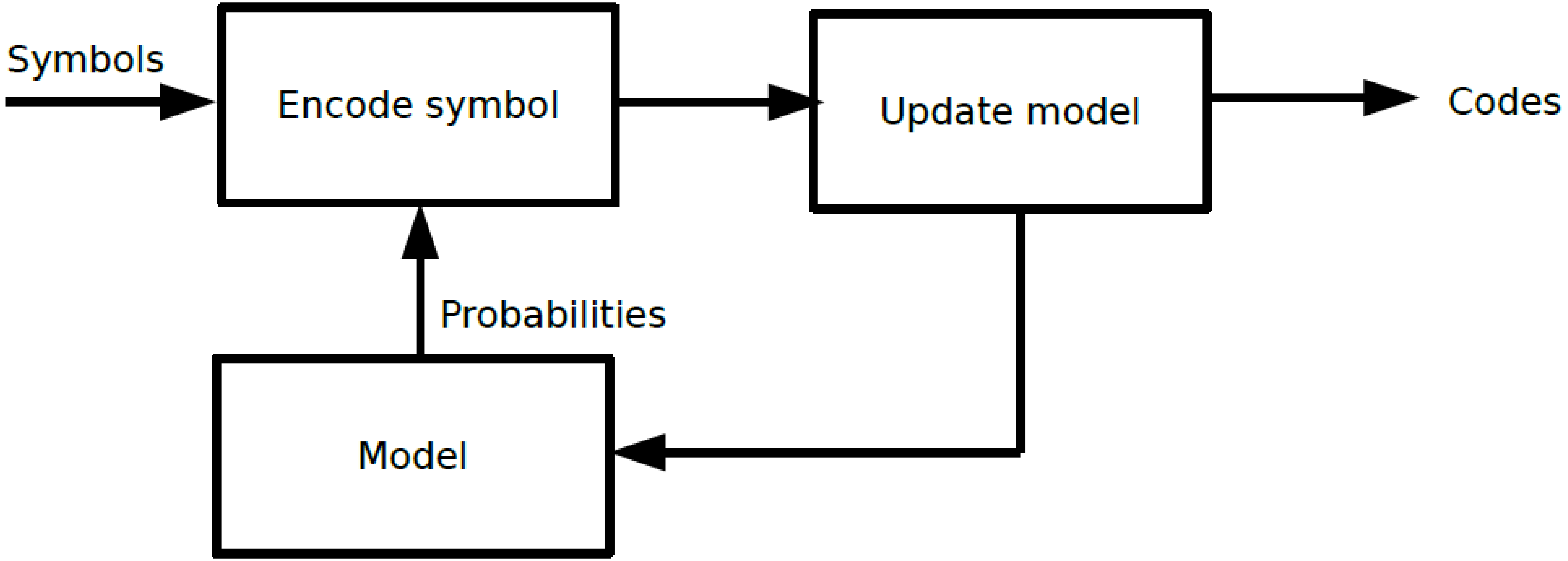
Symbol	Code
au	…
bu	…
…	…
…	…
yu	…
zu	…

There will be $256 \times 256 = 65{,}536$ entries.

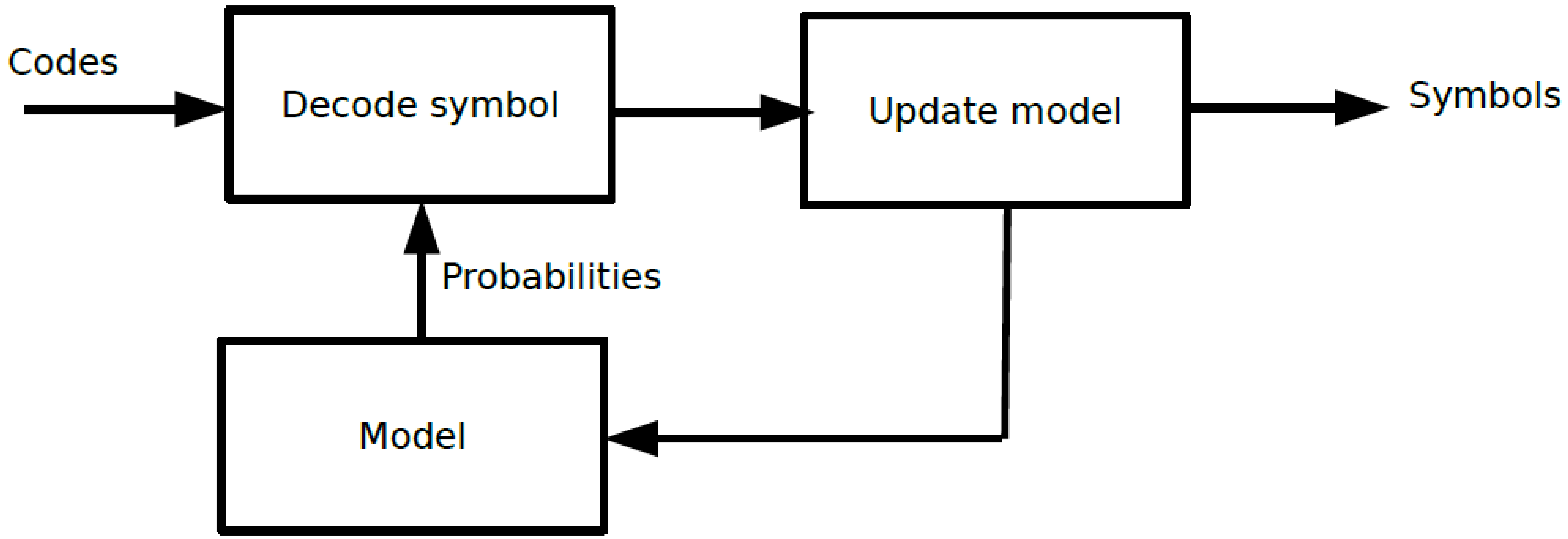
Statistical Models and Adaptive Coding

- Clearly this limits the effectiveness of the compression on small amounts of data: the table may be larger than the data.
- We can update the table as we go: when a symbol becomes more common it should be represented with a smaller code.
- The model for adaptive compression/decompression is on the following slides.

Model for Adaptive Compression

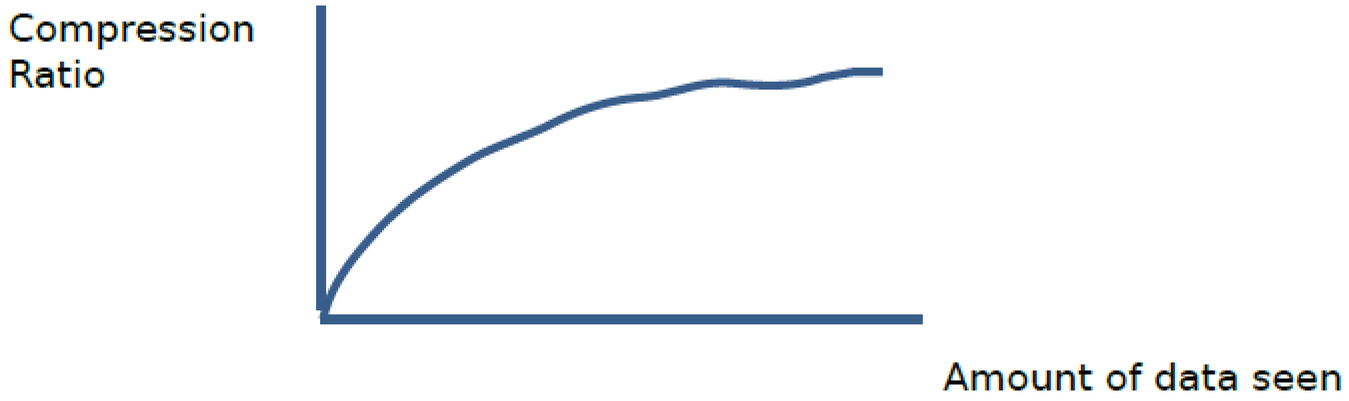


Model for Adaptive Decompression



Statistical Models and Adaptive Coding

- An adaptive model initially knows nothing about the data, but “learns” as it goes along.
- So the compression ratio will improve as it goes, eventually levelling off.
- After this, more data won’t improve the compression.



Huffman Coding

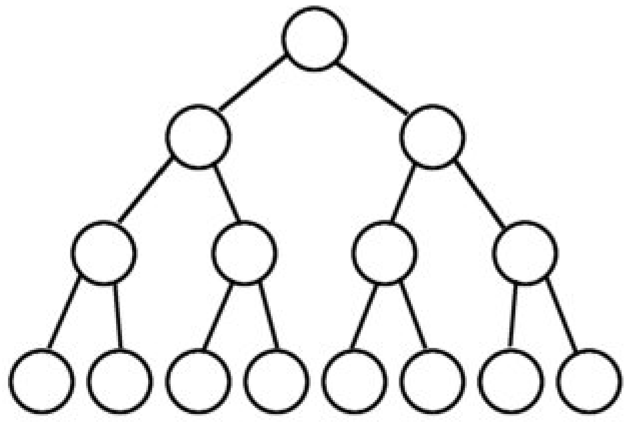
- Huffman coding was developed in 1952 and is widely used in lossless compression, although better algorithms have been developed since.
- For example, it is part of the JPEG standard and is also used in the DEFLATE algorithm – in both cases combined with other compression.
- It is useful to reduce the size of tables and other data which are part of the initial compression.

Huffman Coding

- It is a form of minimum redundancy coding – i.e. we try to encode data using codes which have the minimum amount of redundancy in them (no unnecessary bits used).
- This requires the usual fixed-size symbols (typically bytes) to be replaced with variable-size codes.
- The process of Huffman coding is quite simple.
- It works by building a binary tree which links all the symbols.
- Codes are worked out from the positions of symbols in the tree.

Huffman Coding

- A Binary Tree is a data structure in which a record (known as a node) is linked to two successor records, usually referred to as the left branch when greater and the right when less than the previous record.
- Every node can have no more than 2 children (left & right) but can have none or 1.
- This gives rise to an upside down tree like structure
- The starting (top) node is known as the root.



Huffman Coding

- Lay out all symbols, and assign to each its probability/weight, these are the leaves of the tree.
- Find the 2 nodes with the lowest weight, & create a parent for them. Its weight is the sum of the 2 children's weights.
- Assign the path to one child node the bit value 0, and the other 1.
- The children can be ignored from now on.
- Continue this process, creating parents and linking them together, until only one node (the root) is left.

Huffman Coding

- To find the codes for the symbols, start at the bottom and work up the tree noting the bit values.
- This gives the codes (although the bits are in reverse order, so you need to rearrange them).

Huffman Coding

- This algorithm has some important features:
  - It has the “unique prefix” property.
    No code is a prefix to another, so there is no ambiguity when decoding.
  - If you know the symbols in advance, it is the best you can do with a whole number of bits per code (proved by Huffman).

Huffman Coding

- Consider the following five symbols and their frequencies in a message with 39 symbols:

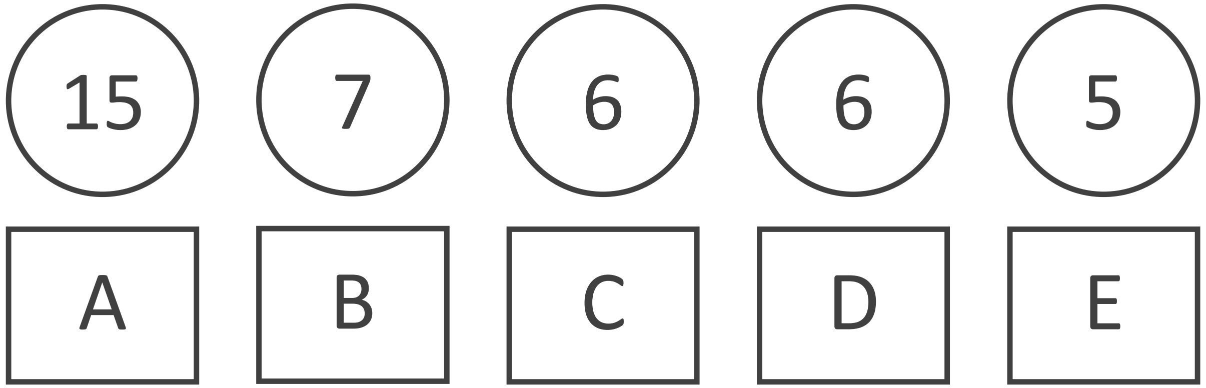
A	B	C	D	E
15	7	6	6	5

- Using ASCII with 8 bits per symbol, we require $(15 + 7 + 6 + 6 + 5) \times 8 = 312$ bits.
- What happens if we use Huffman coding to create a better coding? Using the entropy definition, what is the best possible?

Huffman Coding

Step 1:

- Start by laying out the symbols with their frequencies:

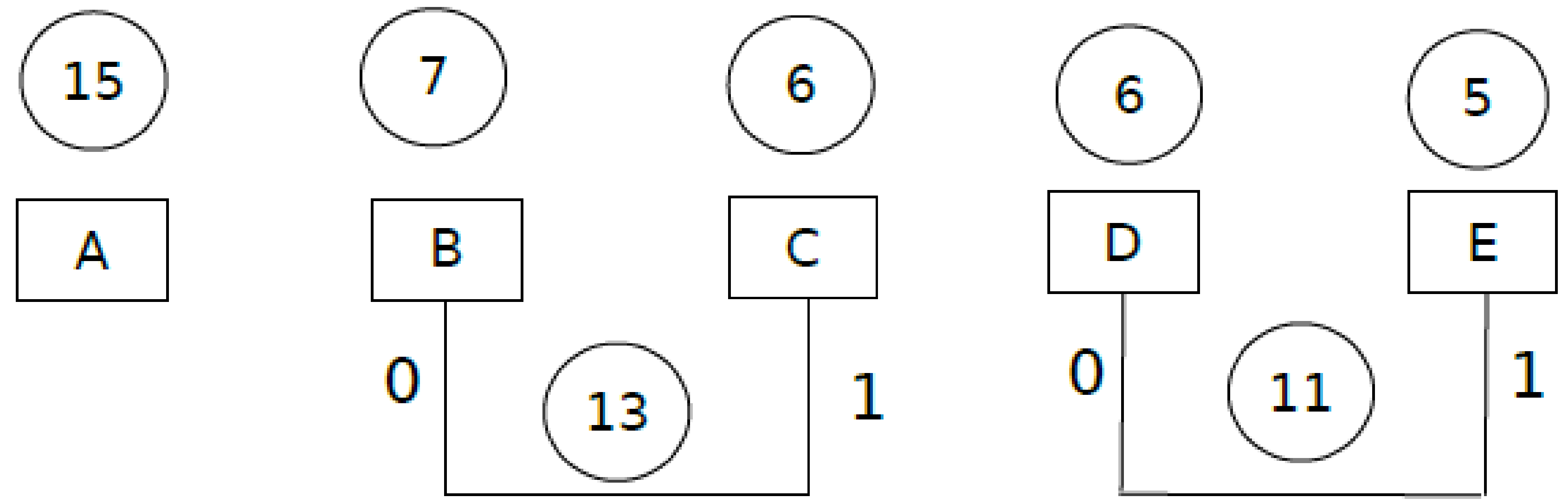


- Select the two with lowest frequency and combine, labelling the branches 1 and 0 (it does not matter which branch is labelled with which bit).

Huffman Coding

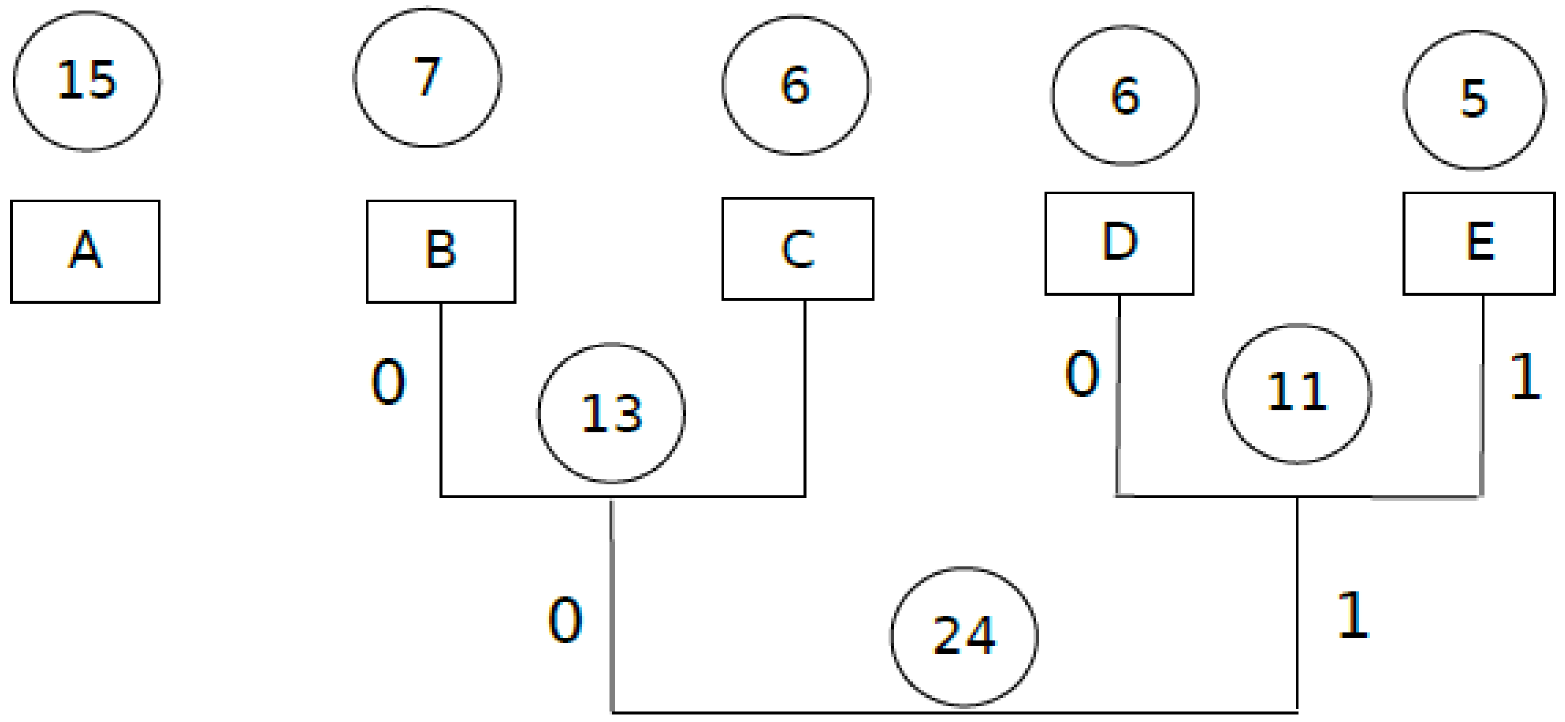
Step 2:

- Repeat this procedure until we have only one node left



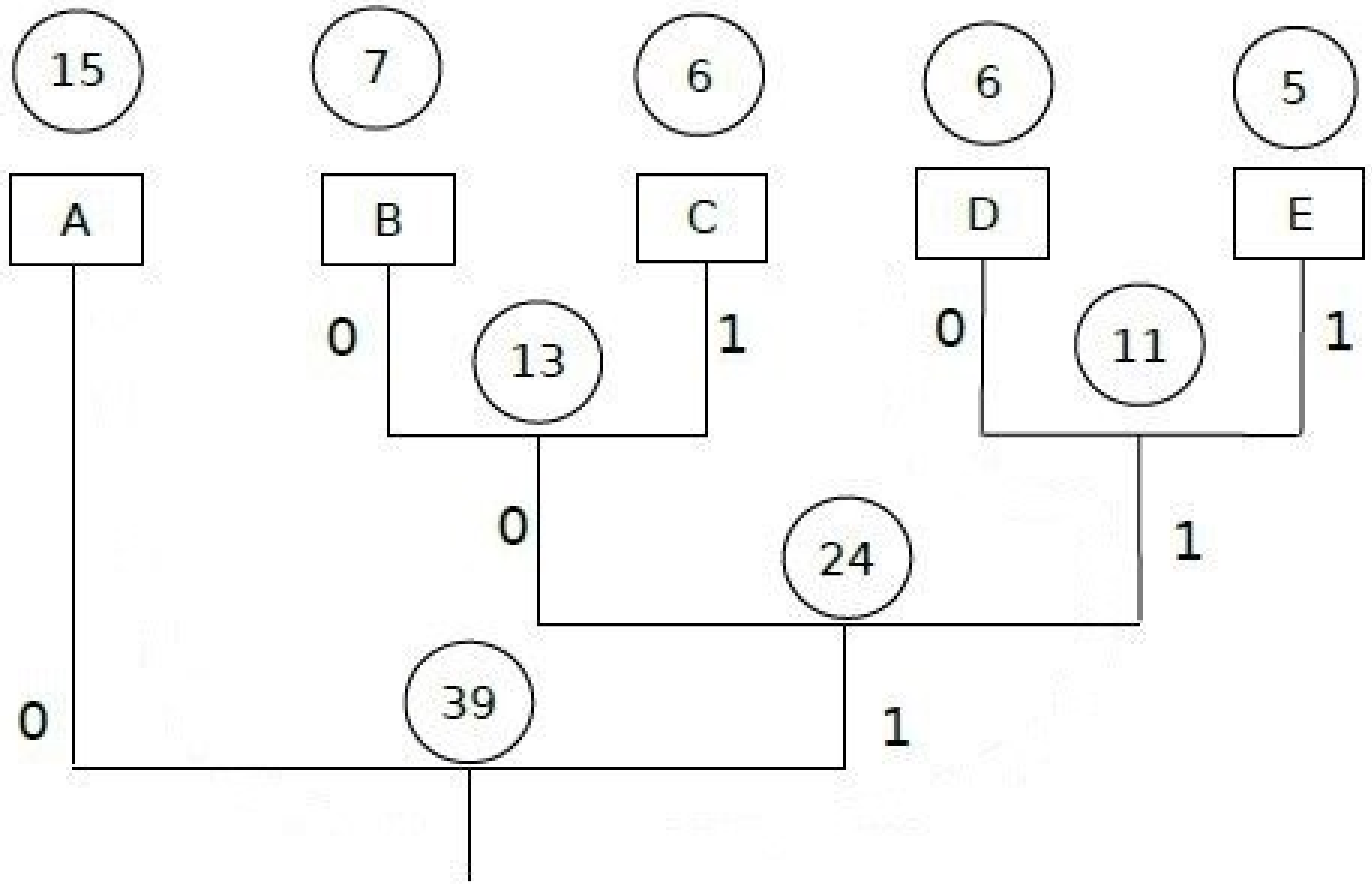
Huffman Coding

Step 2:



Huffman Coding

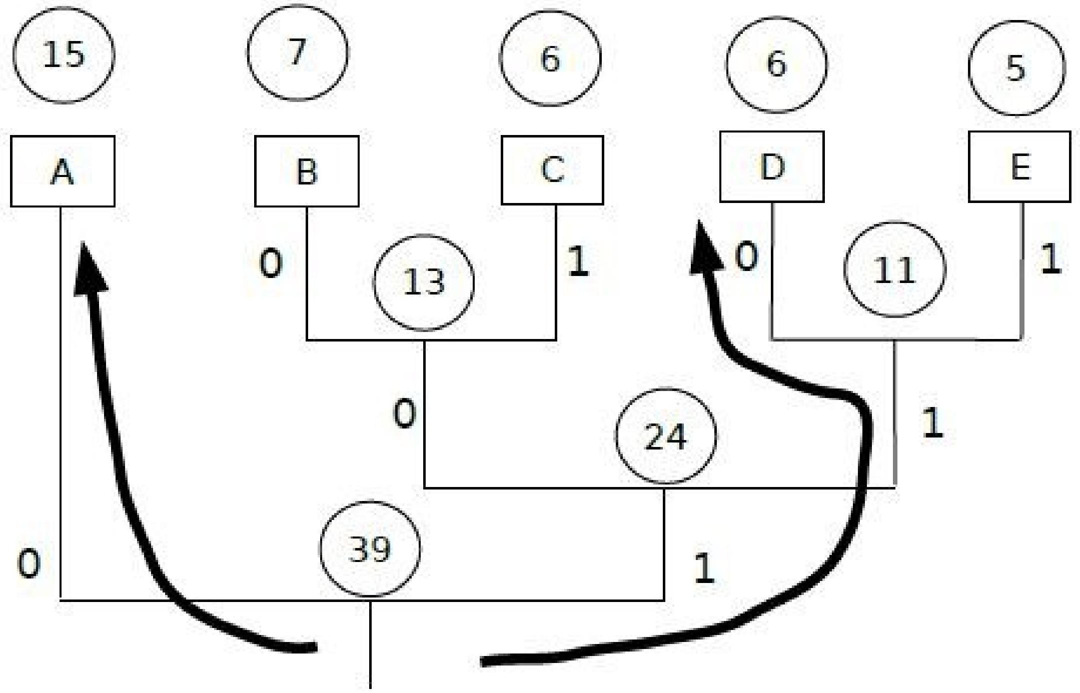
Step 3:



Huffman Coding

Step 4:

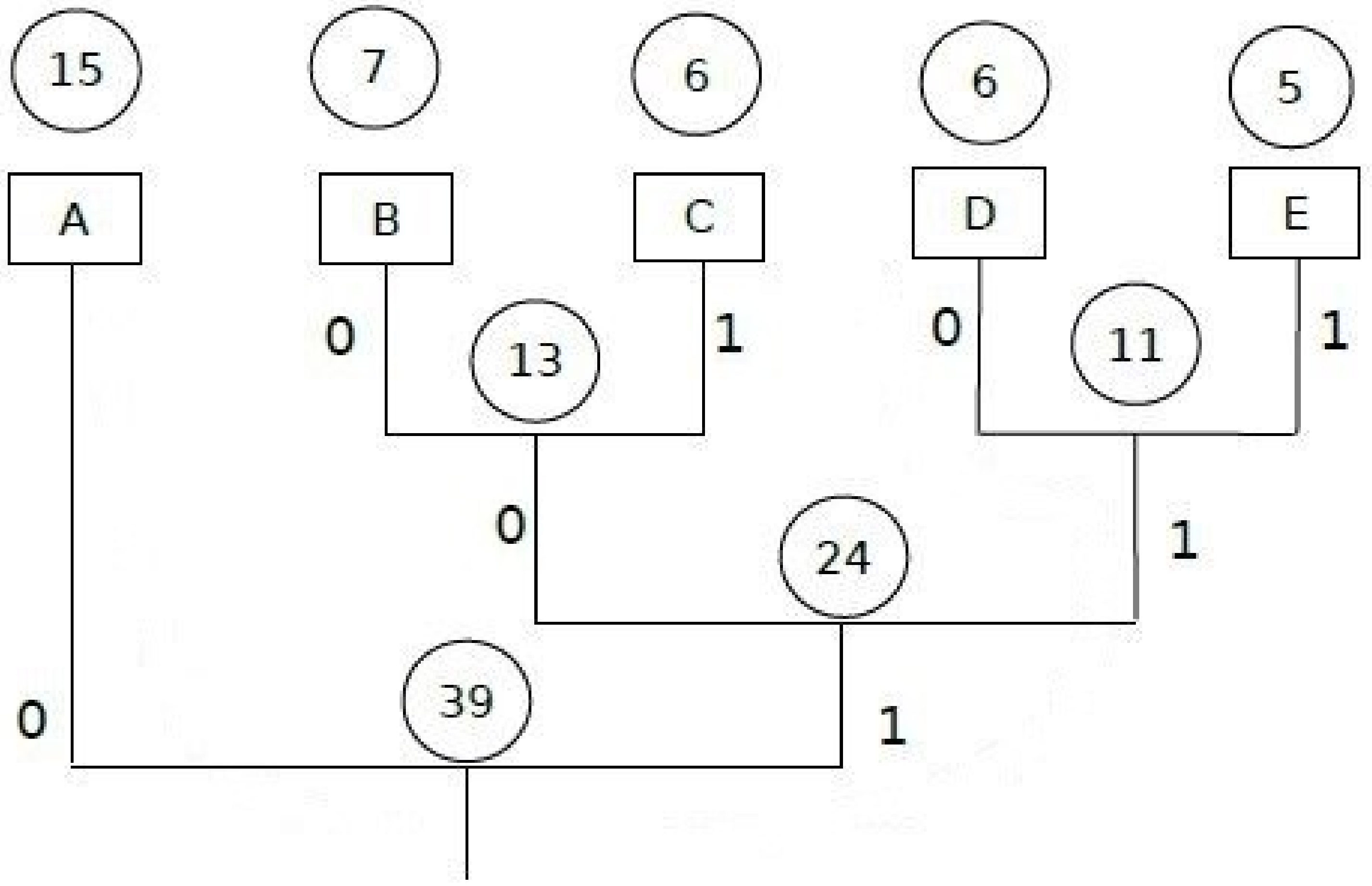
- Now we can find the code for each symbol simply by traversing the tree from the root to the symbol (or, equivalently, going from symbol to root, which would give the symbol's bits in reverse order). To find the codes for "D" and "A"



Huffman Coding

- Thus “D” is given the three-bit symbol 110, and “A” is given code 0.
- Notice what has happened: because “A” is more frequent than “D”, it gets a smaller code.
- The full table for mapping symbols to codes is on the following Slide

Huffman Coding



Symbol	Code
A	0
B	100
C	101
D	110
E	111

Huffman Coding

- How many bits can we now use to represent the message? In other words, how much does this compress it?

Symbol	Code	Frequency	Huffman encoded Bits	ASCII Bits
A	0	15	15 × 1 = 15	15 × 8 = 120
B	100	7	7 × 3 = 21	7 × 8 = 56
C	101	6	6 × 3 = 18	6 × 8 = 48
D	110	6	6 × 3 = 18	6 × 8 = 48
E	111	5	5 × 3 = 15	5 × 8 = 40
		Total Bits	**87**	**312**

- Huffman encoding = 87 bits V 312 needed for ASCII.

Huffman Coding

- Compare the entropy of the message to the number of bits resulting from Huffman

Symbol	Frequency	Probability P=f/39	Information -log2(prob.)	ASCII Bits
A	15	0.384	1.38	0.530
B	7	0.179	2.48	0.445
C	6	0.154	2.70	0.415
D	6	0.154	2.70	0.415
E	5	0.128	2.96	0.380
	39		**Entropy**	**2.185**

Huffman Coding

- So entropy $H = 2.185$ bits, & total information in $N=39$ symbols is:

  $$H\times N = 2.185 \times 39 = 85.25 \text{ bits}$$

- So the Huffman encoding (87 bits) approaches very close to the maximum we could get in theory (85.25 bits).
- In fact, the only way we could do better than this would be to have fractional bits, which Huffman coding does not allow.

Features of Huffman Coding

- First determines the probabilities of the different symbols that constitute the message.
- Next creates a variable length code that assigns the most frequently used symbols to the shortest code words.
- The message is then stored using the devised code along with the information required to decode the message (the table mapping codes to symbols).
- The number of bits used to represent a symbol is always an integral number (integer).

Features of Huffman Coding

- The number of bits that the devised code uses to represent the symbols is pretty close to the theoretical minimum.
- Codes are variable length.
- The codes have the “unique prefix” property: we can easily identify codes when decoding.
- Our examples used one byte as the symbol, but the same ideas work if we used more than one byte at a time. The best approach will depend on the data we are compressing.

elaine.tynan@lit.ie