Algorithmics

Lecture 2

Dr. Oana Andrei

School of Computing Science
University of Glasgow

oana.andrei@glasgow.ac.uk

Outline of course

Section 0: Quick recap on algorithm analysis

Section 1: Sorting algorithms

Section 2: Strings and text algorithms

Section 3: Graphs and graph algorithms

Section 4: An introduction to NP completeness

Section 5: A (very) brief introduction to computability

Section 2 – Strings and text algorithms

Text compression

- Huffman encoding
- LZW compression/decompression

String comparison

- string distance

String/pattern search

- brute force algorithm
- KMP algorithm
- BM algorithm

Text compression

Problem

- given a string defined over an alphabet (e.g., ASCII or Unicode)
- encode it efficiently into a smaller binary string using only 0s and 1s

When do we want to use it?

- communicating over a low-bandwidth channel, over a limited or poor wireless connection
- storing collections of large documents efficiently on a fixed capacity storage device

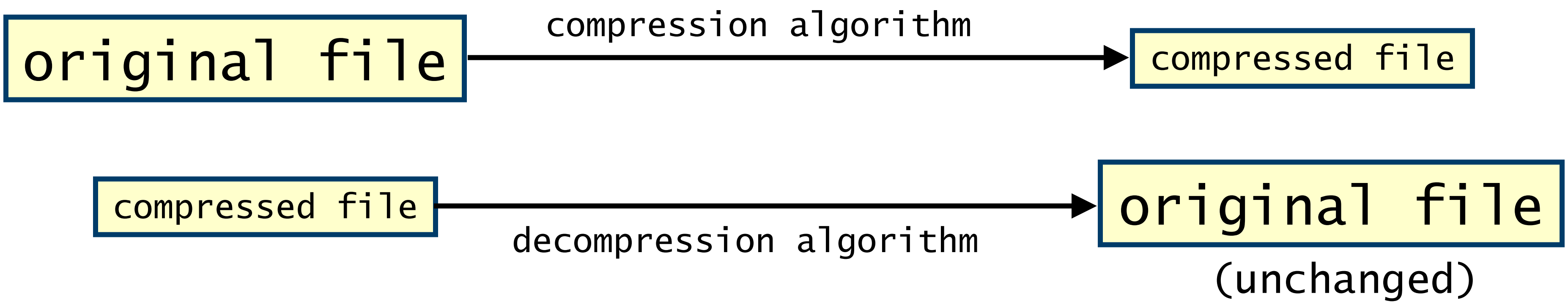
Text compression

A special case of data compression

– saves disk space and transmission time

Text compression must be lossless

– i.e. the original must be recoverable without error



Some other forms of compression can afford to be lossy

– e.g. for pictures, sound, etc. (not considered here)

Text compression

Examples of text compression

- compress, gzip in Unix, ZIP utilities for Windows, …
- two main approaches: statistical and dictionary

Compression ratio: x/y

- x is the size of compressed file and y is the size of original file
- e.g. measured in B, KB, MB, …
- compressing a 10MB file to 2MB would yield a compression ratio of 2/10=0.2

Percentage space saved: (1 – "compression ratio") × 100%

- space saved expressed as a percentage of the original file size
- compressing a 10MB file to 2MB yields a percentage space savings of 80%

Space savings in the range 40% – 60% are typical

- obviously the higher the saving the better the compression

Section 2 – Strings and text algorithms

Text compression

- Huffman encoding
- LZW compression/decompression

String comparison

- string difference

String/pattern search

- brute force algorithm
- KMP algorithm
- BM algorithm

Text compression – Huffman encoding

The classical statistical method

- now mostly superseded in practice by more effective dictionary methods
- fixed (ASCII) code replaced by variable length code for each character
- every character is represented by a unique codeword (bit string)
- frequently occurring characters are represented by shorter codewords

The code has the prefix property

- no codeword is a prefix of another (gives unambiguous decompression)

Based on a Huffman tree (a proper binary tree)

- each character is represented by a leaf node
- codeword for a character is given by the path from the root to the appropriate leaf (`left=0` and `right=1`)
- the prefix property follows from this

Huffman tree construction – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

- the number of times each character appears in the text

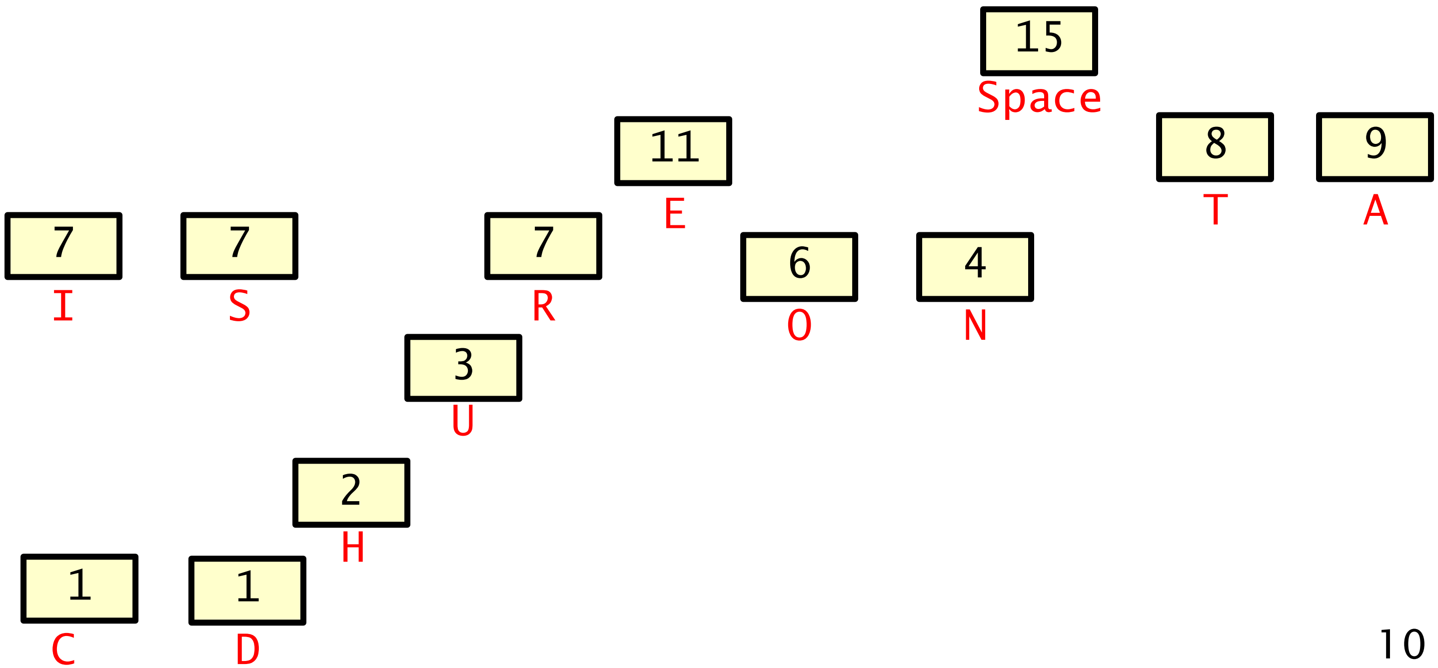
Huffman tree construction – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

First add leaves of Huffman tree

– characters with their frequencies label the leaf nodes



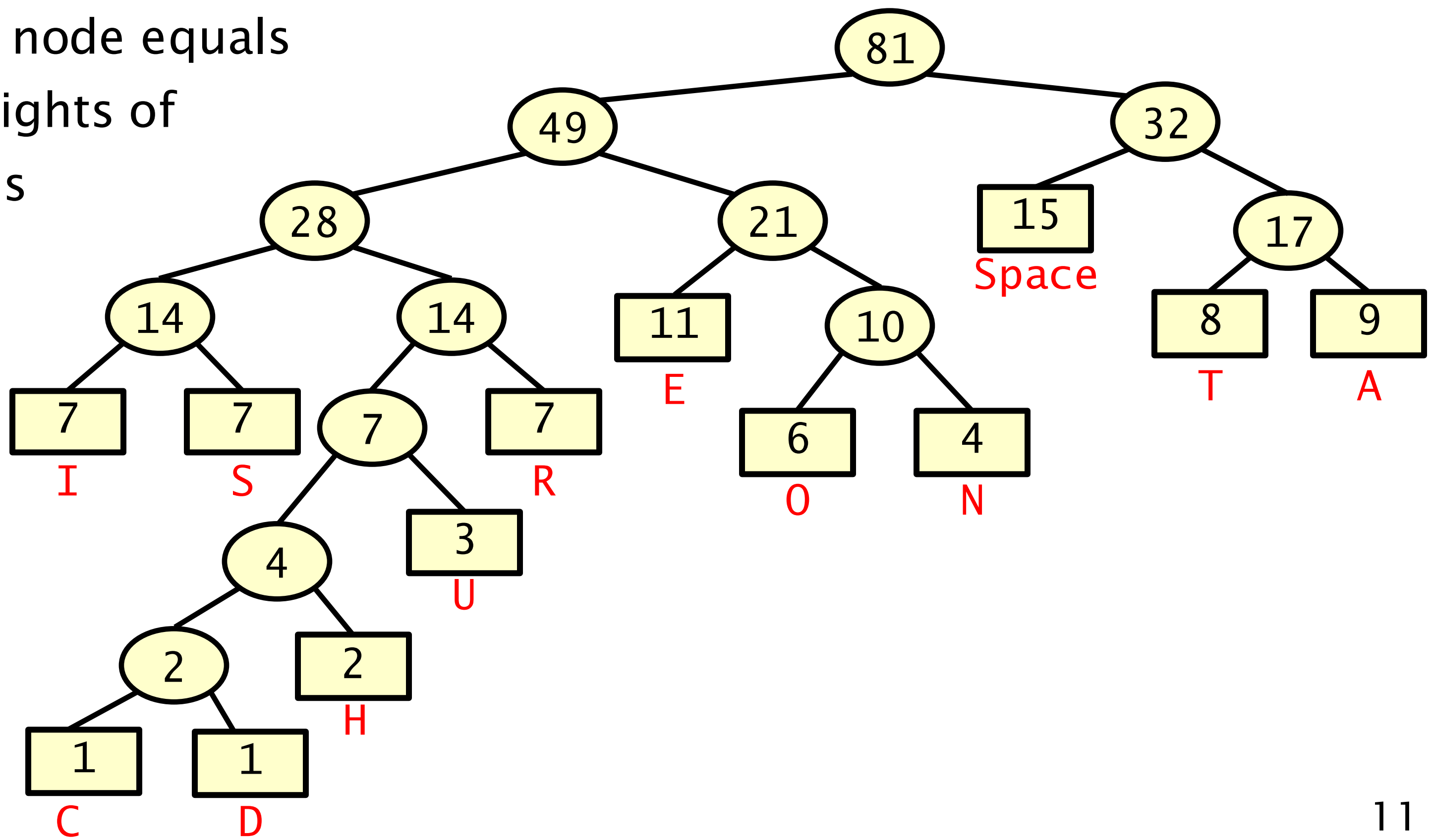
Huffman tree construction – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Next, while there is more than one parentless node

- add new parent to nodes of smallest weight
- weight of new node equals sum of the weights of the child nodes



Huffman tree construction – Pseudocode

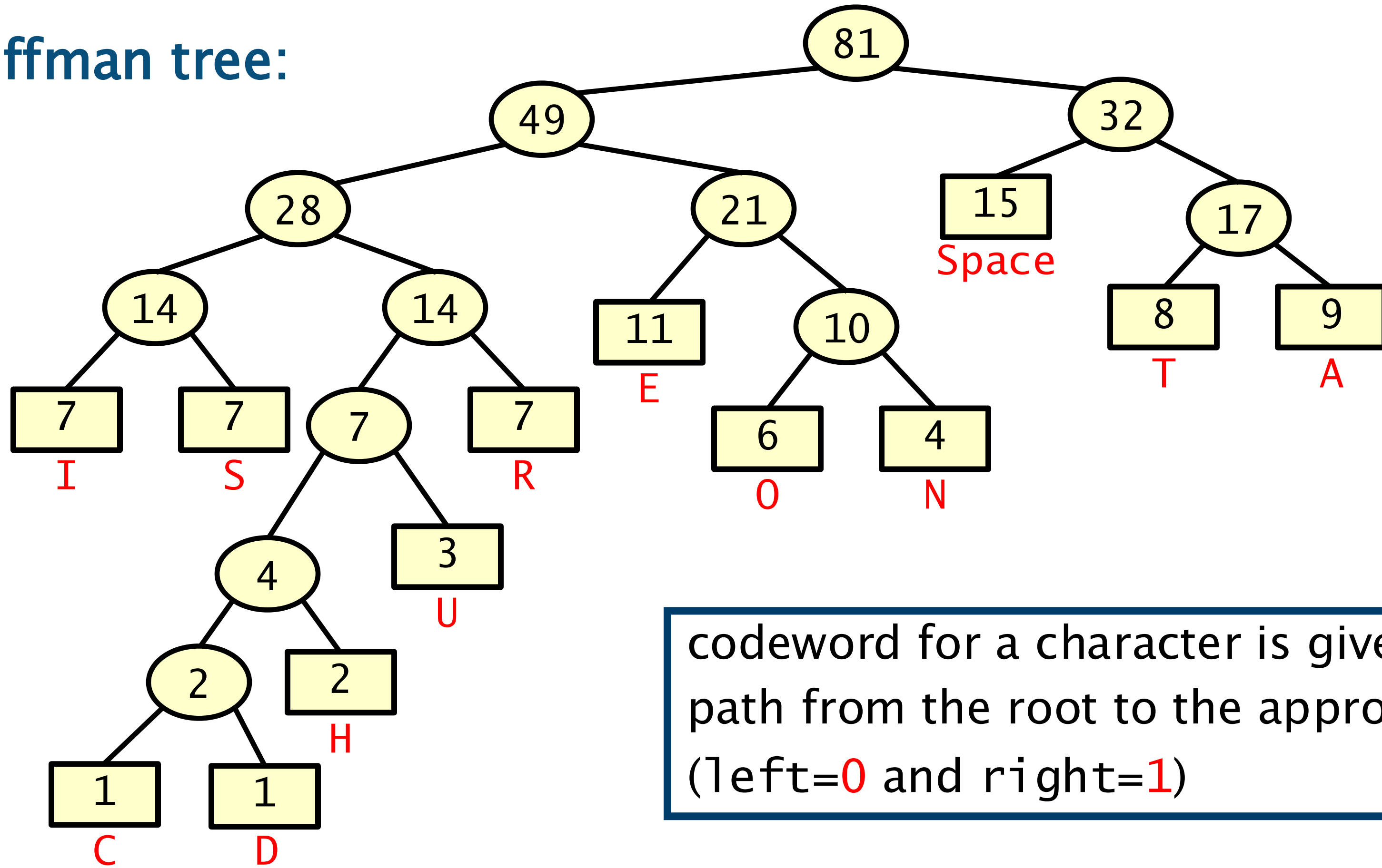
```
// set up the leaf nodes
for (each distinct character c occurring in the text){
  make a new parentless node n; // new node
  int f = frequency count for c;
  n.setWeight(f); // weight equals the frequency
  n.setCharacter(c); // set character value
  // leaf so no children
  n.setLeftChild(null);
  n.setRightChild(null);
}
// construct the branch nodes and links
while (no. of parentless nodes > 1){
  make a new parentless node z; // new node
  x, y = 2 parentless nodes of minimum weight; // its children
  z.setLeftChild(x); // set x to be the left child of new node
  z.setRightChild(y); // set y to be the right child of new node
  int w = x.getWeight()+y.getWeight(); // calculate weight of node
  z.setWeight(w); // set the weight of the new node
}
// the final node z is root of Huffman tree
```

Huffman code – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Huffman tree:



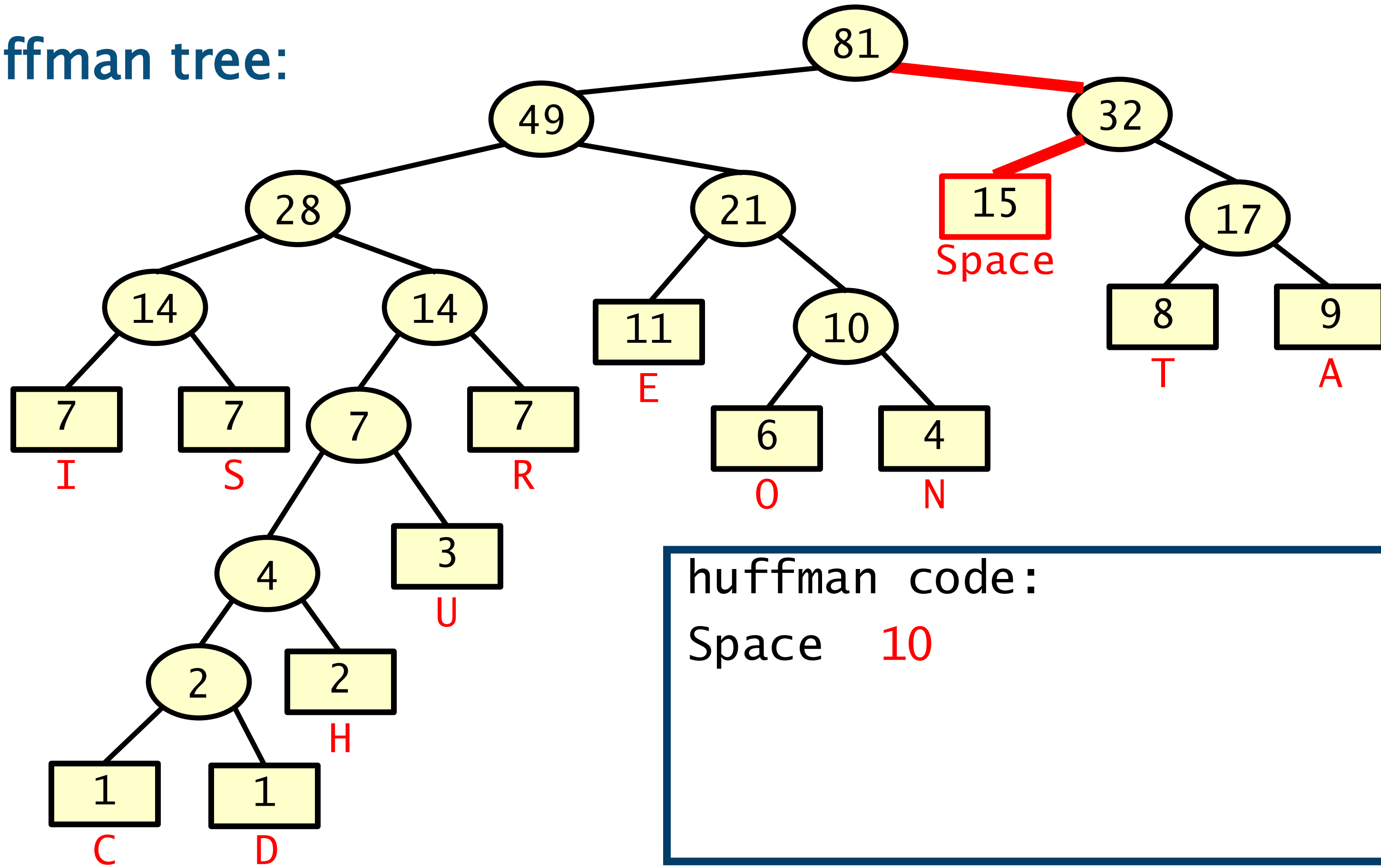
codeword for a character is given by the path from the root to the appropriate leaf (`left`=0 and `right`=1)

Huffman code – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Huffman tree:



right=1
left=0

huffman code:

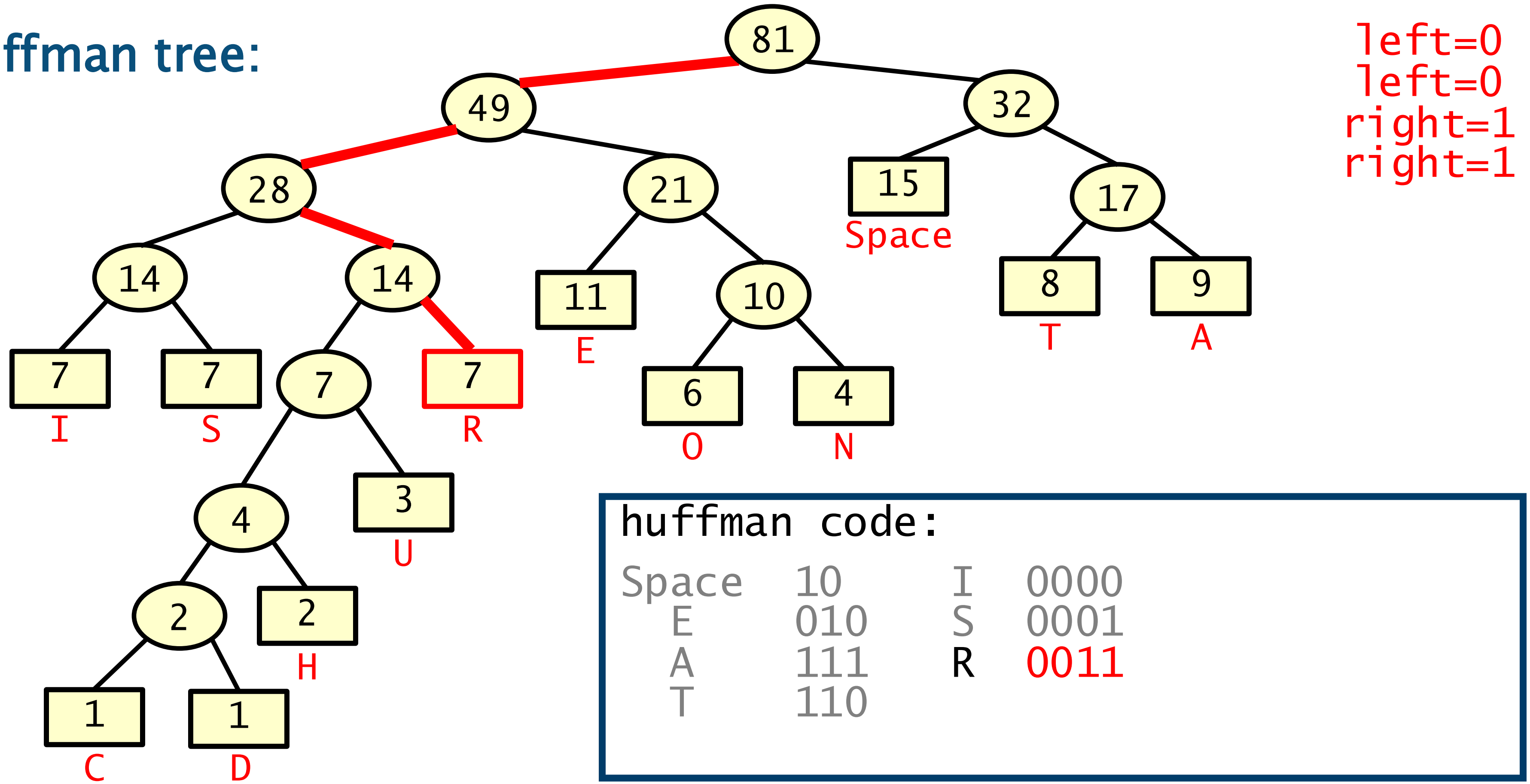
Space 10

Huffman code – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Huffman tree:




```
huffman code:
```

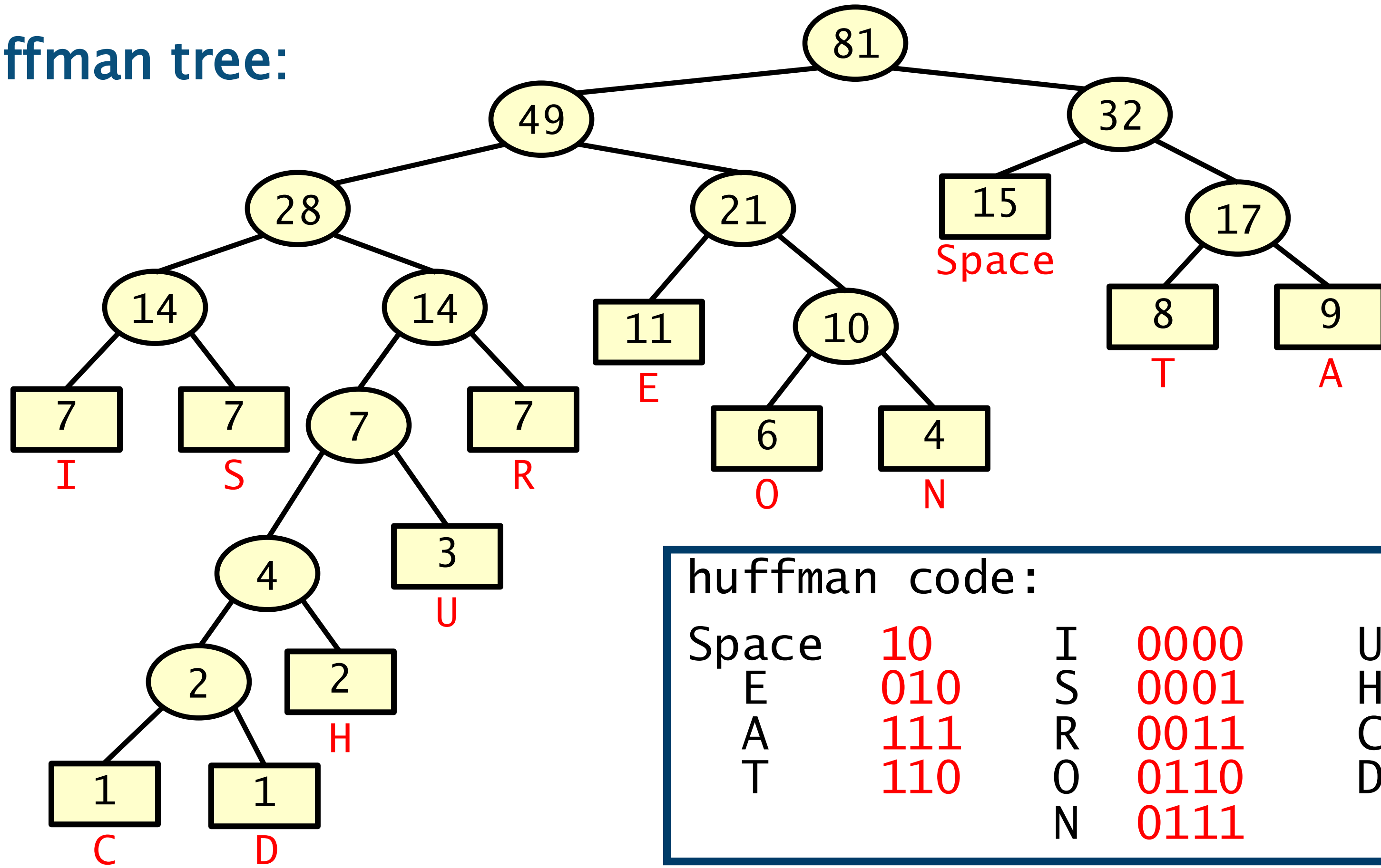
Space	10	I	0000
E	010	S	0001
A	111	R	0011
T	110		

Huffman code – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Huffman tree:



huffman code:

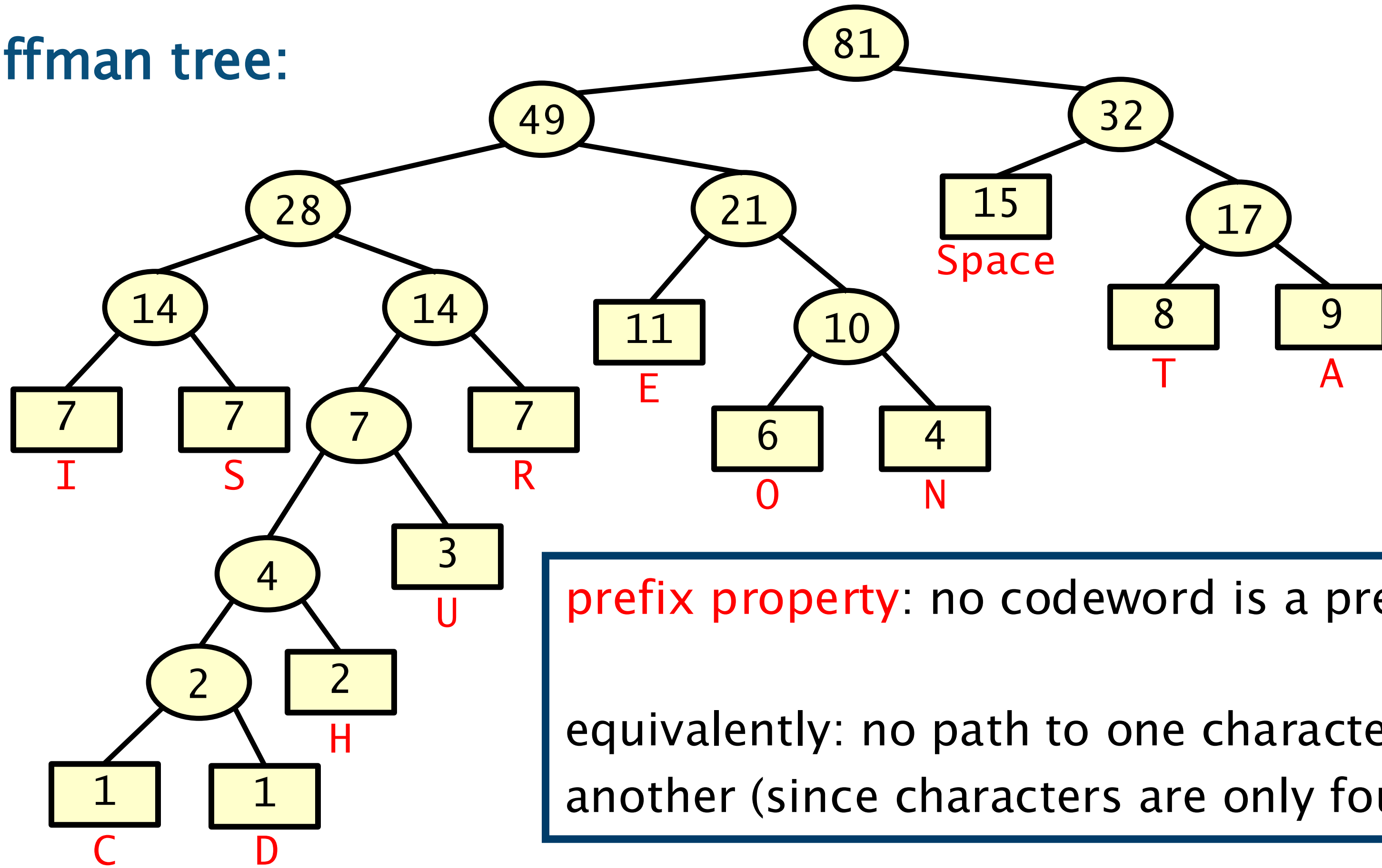
Space	10	I	0000	U	00101
E	010	S	0001	H	001001
A	111	R	0011	C	0010000
T	110	O	0110	D	0010001
		N	0111		

Huffman code – Example

Character frequencies:

Space	E	A	T	I	S	R	O	N	U	H	C	D
15	11	9	8	7	7	7	6	4	3	2	1	1

Huffman tree:



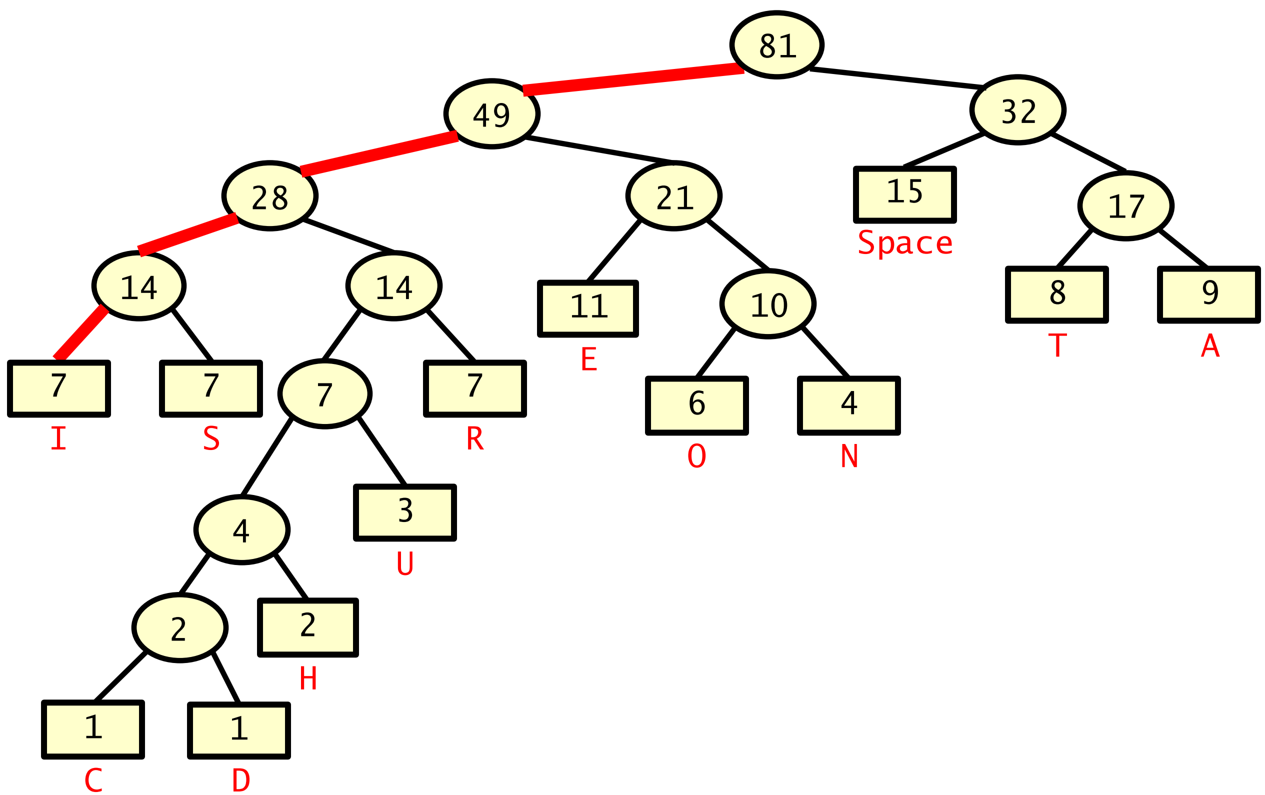
prefix property: no codeword is a prefix of another

equivalently: no path to one character is a prefix of another (since characters are only found at leaves)

Huffman encoding – Optimality

Weighted path length (WPL) of a tree T

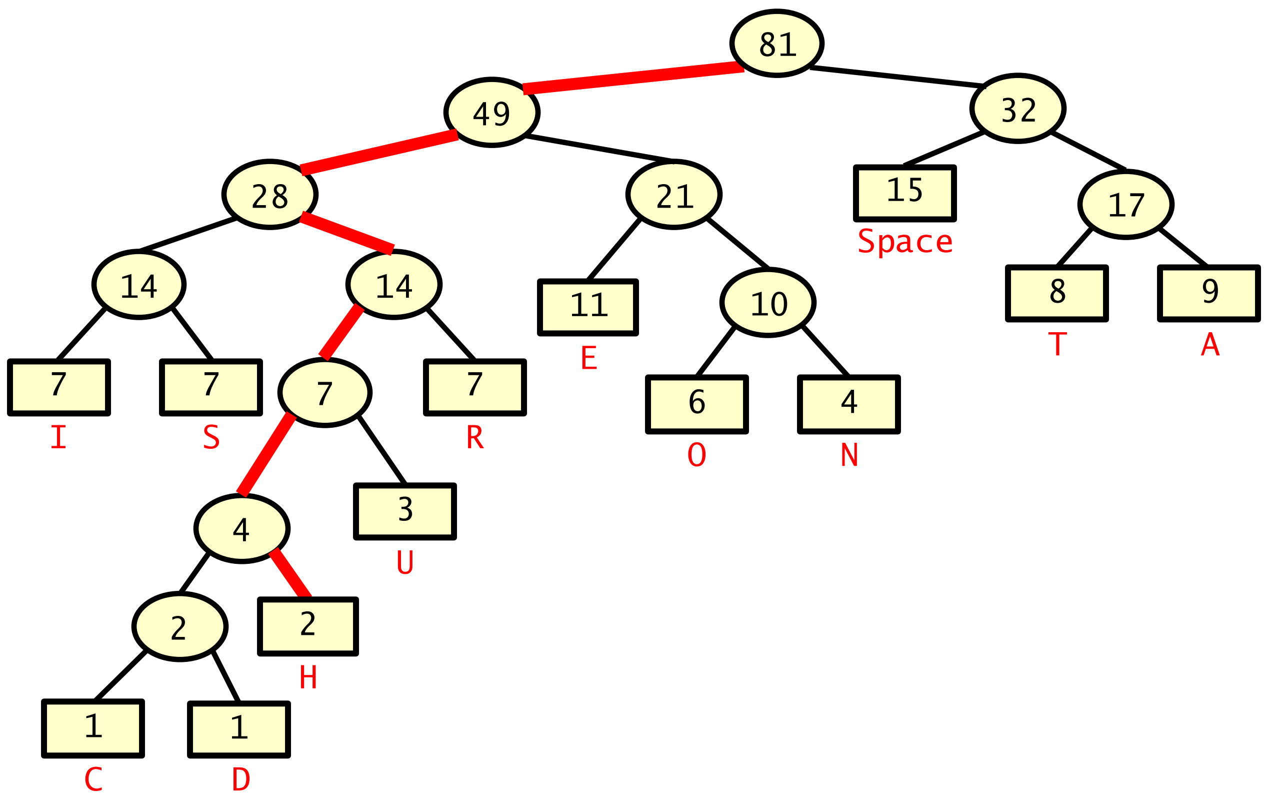
- Σ (weight)×(distance from root) where sum is over all leaf nodes
- for the example tree: WPL equals: 7×4 +



Huffman encoding – Optimality

Weighted path length (WPL) of a tree T

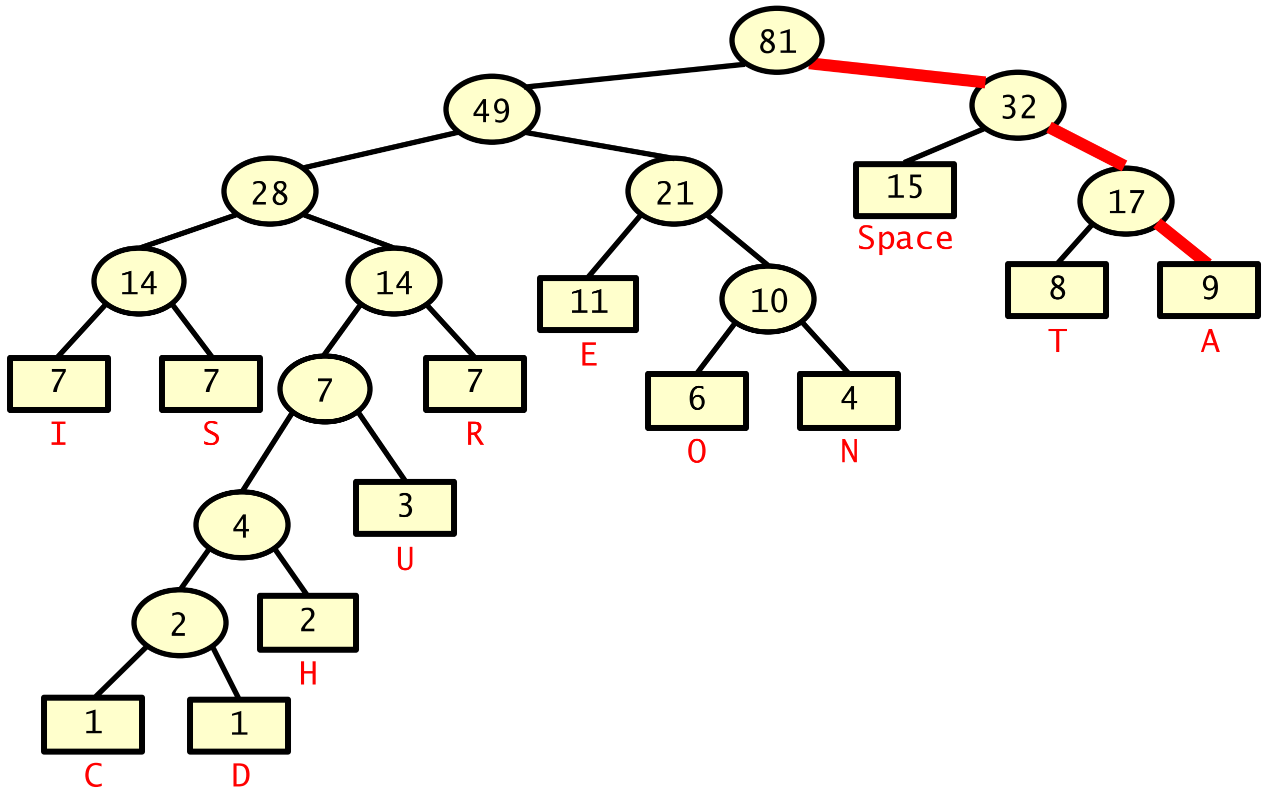
- Σ (weight)$\times$(distance from root) where sum is over all leaf nodes
- for the example tree: WPL equals: $7\times4 + 7\times4 + 1\times7 + 1\times7 + 2\times6 +$



Huffman encoding – Optimality

Weighted path length (WPL) of a tree T

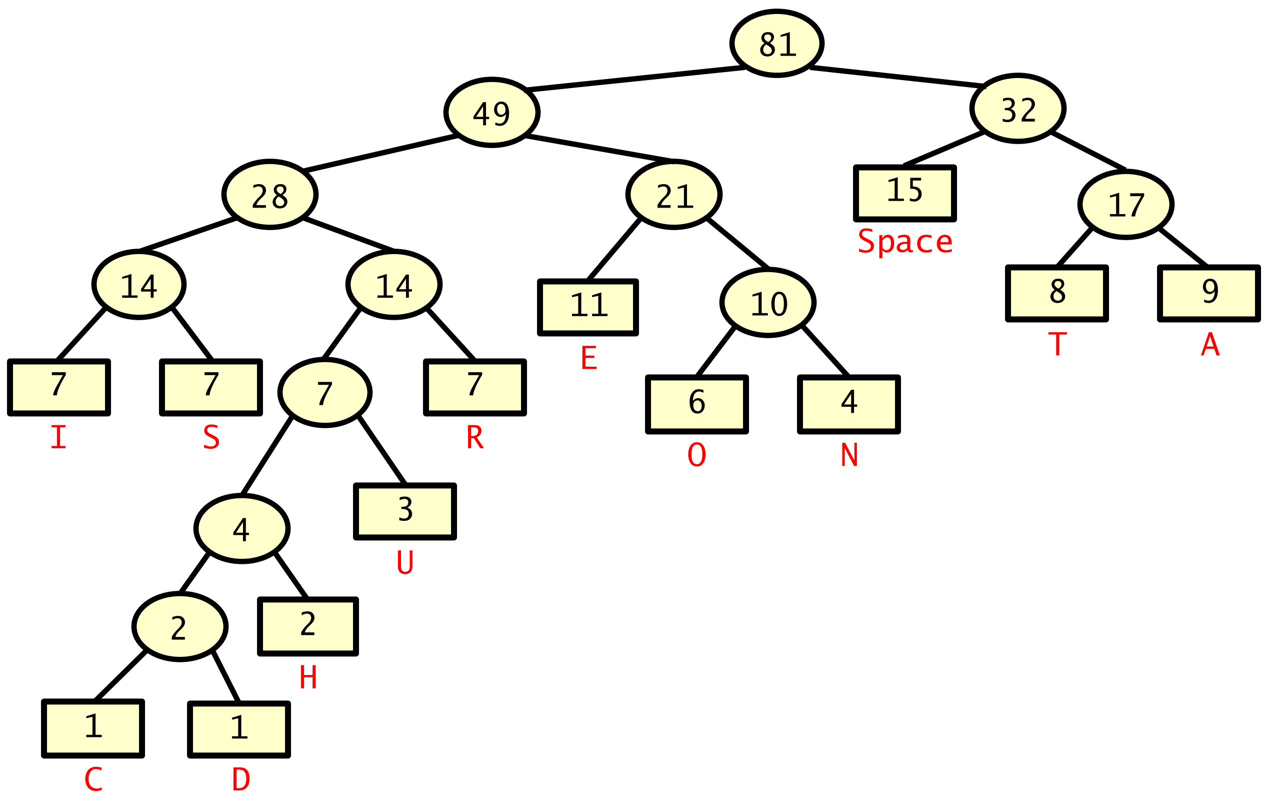
- Σ (weight)$\times$(distance from root) where sum is over all leaf nodes
- for the example tree: WPL equals: $7\times4 + 7\times4 + 1\times7 + 1\times7 + 2\times6 + 3\times5 + 7\times4 + 11\times3 + 6\times4 + 4\times4 + 15\times2 + 8\times3 + 9\times3$



Huffman encoding – Optimality

Weighted path length (WPL) of a tree T

- Σ (weight)$\times$(distance from root) where sum is over all leaf nodes
- for the example tree: WPL equals: $7\times4 + 7\times4 + 1\times7 + 1\times7 + 2\times6 + 3\times5 + 7\times4 + 11\times3 + 6\times4 + 4\times4 + 15\times2 + 8\times3 + 9\times3 = 279$



Huffman encoding – Optimality

Weighted path length (WPL) of a tree T

- `Σ (weight)×(distance from root)` where sum is over all leaf nodes
- for the example tree: WPL equals: $7\times4 + 7\times4 + 1\times7 + 1\times7 + 2\times6 + 3\times5 + 7\times4 + 11\times3 + 6\times4 + 4\times4 + 15\times2 + 8\times3 + 9\times3 = 279$

Huffman tree has minimum WPL over all binary trees with the given leaf weights

- Huffman tree need not be unique (e.g., nodes>2 with min weight)
- however, all Huffman trees for a given set of frequencies have same WPL
- so what?
- weighted path length (WPL) is the number of bits in compressed file
  - bits = sum over chars (frequency of char × code length of char)
- so a Huffman tree minimises this number
- hence Huffman coding is optimal, for all possible codes built in this way

Huffman encoding – Algorithmic requirements

Building the Huffman tree

- if the text length equals n and there are m distinct chars in text
- $O(n)$ time to find the frequencies
- $O(m\log m)$ time to construct the code, for example using a (min-) heap to store the parentless nodes and their weights
  - initially build a heap where nodes correspond to the m characters labelled by their frequencies, therefore takes $O(m)$ time to build the heap
  - one iteration takes $O(\log m)$ time:
    - find and remove ($O(\log m)$) two minimum weights
    - then insert ($O(\log m)$) new weight (sum of minimum weights found)
  - and there are $m-1$ iterations before the heap is empty
    - each iteration decreases the size of the heap by 1
- so $O(n + m\log m)$ overall
- in fact, m is essentially a constant, so it is really $O(n)$

Huffman encoding – Algorithmic requirements

Compression & decompression are both $O(n)$ time

- assuming m is constant

Compression uses a code table (an array of codes, indexed by char)

- $O(m\log m)$ to build the table as m characters so m paths of length $\leq \log m$
- $O(n)$ to compress: n characters in the text so n lookups in the array $O(1)$
- so $O(m\log m) + O(n)$ overall

Decompression uses the tree directly (repeatedly trace paths in tree)

- $O(n\log m)$ as n characters so n paths of length $O(\log m)$

Huffman encoding – Algorithmic requirements

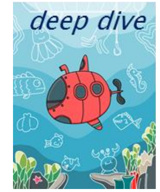
Problem: some representation of the Huffman tree must be stored with the compressed file

- (because the there is no unique Huffman tree)
- otherwise decompression would be impossible

Alternatives

- use a fixed set of frequencies based on typical values for text
  - but this will usually reduce the compression ratio
- use adaptive Huffman coding: the (same) tree is built and adapted by the compressor and by the decompressor as characters are encoded/decoded
  - this slows down compression and decompression (but not by much if done in a clever way)
  - for further reference read about the FGK algorithm and the Vitter algorithm

Huffman encoding – application

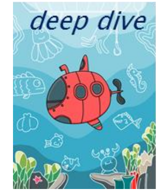


A video streaming service, like Netflix or YouTube, faces the challenge of delivering high-quality video content to millions of users worldwide.

Some requirements:

- Bandwidth efficiency: How to maximise the use of available bandwidth to stream high-quality video without constant buffering or data caps overruns?
- Adaptive streaming: How to dynamically adjust the video quality based on the user's internet speed without requiring manual adjustment?
- Cost reduction: How to minimize the costs associated with data transmission while maintaining or improving service quality?

Huffman encoding – application



A video streaming service, like Netflix or YouTube, faces the challenge of delivering high-quality video content to millions of users worldwide.

Possible solution using Huffman encoding:

- Compress video files before transmission; by analysing the frequency of pixel values or encoding symbols in the video data, Huffman trees can be used to create optimal prefix codes, reducing the overall size of the files
- Implement adaptive streaming algorithms to compress multiple quality versions of the same video; the server can dynamically select & transmit the most appropriate version based on the user's current bandwidth.
- Compressing video files using Huffman encoding significantly reduces the amount of data transmitted over the network, leading to lower operational costs

Section 2 – Strings and text algorithms

Text compression

- Huffman encoding
- LZW compression/decompression

String comparison

- string difference

String/pattern search

- brute force algorithm
- KMP algorithm
- BM algorithm

LZW compression

A popular dictionary-based method

- the basis of `compress` and `gzip` in Unix, also used in gif and tiff formats
- due to Abraham **L**empel, Jacob **Z**iv and Terry **W**elch
- algorithm was under patented to Unisys (but patent now expired)

The dictionary is a collection of strings

- each with a codeword that represents it
- the codeword is a bit pattern
- but it can be interpreted as a non-negative integer

Whenever a codeword is outputted during compression, what is written to the compressed file is the **bit pattern**

- using a number of bits determined by the current codeword length
- at any point all codewords are the same length (no ambiguity)

LZW compression

The dictionary is built dynamically during compression

- and also during decompression

Initially dictionary contains all possible strings of length 1

Throughout the execution the dictionary is closed under prefixes

- i.e., if the string s is represented in the dictionary, so is every prefix of s

It follows that a trie is an ideal representation of the dictionary

- every node in the trie represents a 'word' in the dictionary
- a trie is effective and efficient for other reasons too

LZW compression

Key question: how many bits are in a codeword?

- in the most used version of the algorithm, this value changes as the compression (or decompression) algorithm proceeds

At any given time during compression (or decompression)

- there is a current codeword length k
- so there are exactly 2^k distinct codewords available
  - i.e., all possible bit-strings of length k
- this limits the size of the dictionary
- however the codeword length can be incremented when necessary
  - thereby doubling the number of available codewords
- initial value of k should be large enough to encode all strings of length 1

A word on the example inputs for LZW

Example inputs over 2-4 characters for example:

- DNA sequence using the 4 nucleotides A (adenine), T (thymine), G (guanine), and C (cytosine)
- RNA: A, C, G, and U (uracil)

Compression algorithms like LZW are typically more effective on large datasets where the repetitive patterns are more likely to occur

- hence DNA sequence sample examples with repetitive patterns
- reduce the risk of compression overhead
  - producing a larger output than the input

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

`Initial dictionary: A:00   C:01   G:10   T:11`
`Initial codeword length: 2 (since` $2^2=4$ `codewords)`

Setup

- initial dictionary contains all strings of length 1
- codewords are a bit pattern but can be interpreted as integers

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

`Initial dictionary: A:0   C:1   G:2   T:3`
Initial codeword length: 2 (since $2^2=4$ codewords)

Setup

- initial dictionary contains all strings of length 1
- codewords are a bit pattern but can be interpreted as integers
- will use integers in the dictionary (to simplify presentation)

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10		

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10		

dictionary full, no available codewords so increase codeword length

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10		

dictionary full, no available codewords so increase codeword length

LZW compression – Example

Text = G **A** C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	**G**	10	**GA**	4

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000		

LZW compression – Example

Text = G A **C** G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	**A**	000	**AC**	5

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001		

LZW compression – Example

Text = G A C **G** A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	**C**	001	**CG**	6

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100		

LZW compression – Example

Text = G A C G A **T** A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	**GA**	100	**GAT**	7

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 3

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011		

no available code words so increase code word length

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

```
dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 4
```

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011		

no available code words so increase code word length

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	**T**	011	**TA**	8

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101		

LZW compression – Example

Text = G A C G A T A C **G** A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	**ACG**	9

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9				

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

```
dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4
```

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111		

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9 GATA:10
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	**GAT**	0111	**GATA**	10

add: string + char at next position

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9 GATA:10
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12				

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9 GATA:10
codeword length: 4

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001		

LZW compression – Example

```
Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char
```

```
dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9 GATA:10
codeword length: 4
```

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

Compressed file: 10 000 001 100 011 0101 0111 1001
file size = 26 bits

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

new file: concatenate all the codewords

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

Compressed file: 10 000 001 100 011 0101 0111 1001
file size = 26 bits

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

Compressed file: 10 000 001 100 011 0101 0111 1001
file size = 26 bits

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

Compressed file: 10 000 001 100 011 0101 0111 1001
file size = 26 bits

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

LZW compression – Example

Text = G A C G A T A C G A T A C G
File size = 14 bytes, or 28 bits if 2 bits/char

Compressed file: 10000001100011010101111001
file size = 26 bits

step	position in string	longest string in dictionary	b	add to dictionary	code
1	1	G	10	GA	4
2	2	A	000	AC	5
3	3	C	001	CG	6
4	4	GA	100	GAT	7
5	6	T	011	TA	8
6	7	AC	0101	ACG	9
7	9	GAT	0111	GATA	10
8	12	ACG	1001	-	-

LZW compression – Pseudo code

```
set current text position i to 0;
initialise codeword length k (say to 8);
initialise the dictionary d;

while (the text t is not exhausted) {

  identify the longest string s, starting at position i of text t
  that is represented in the dictionary d;
  // there is such string, as all strings of length 1 are in d

  output codeword for the string s; // using k bits

  // move to the next position in the text
  i += s.length(); // move forward by the length of string just encoded
  c = character at position i in t; // character in next position

  add string s+c to dictionary d, paired with next available codeword;
 // this involves adding a new leaf node if d is represented by a trie
 // may have to increment the codeword length k to make this possible
}
```

LZW decompression

Decompression algorithm builds same dictionary as compression algorithm

- but one step out of phase

LZW decompression – Pseudo code

```
initialise codeword length k;
initialise the dictionary;

read the first codeword x from the compressed file f; // i.e. read k bits
String s = d.lookUp(x); // look up codeword in dictionary
output s; // output decompressed string

while (f is not exhausted){

  String oldS = s.clone(); // copy last string decompressed

  if (d is full) k++; // dictionary full so increase the code word length

  get next codeword x from f;  // i.e. read k bits
  s = d.lookUp(x); // look up codeword in dictionary
  output s; // output decompressed string

  String newS = oldS + s.charAt(0); // string to add to dictionary
  add string newS to dictionary d paired with next available codeword;
}
```

LZW decompression – Example

```
Compressed file: 10000001100011010101111001
file size = 26 bits
```

```
dictionary: A:0 C:1 G:2 T:3
codeword length: 2
```

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 2

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-

do not add to dictionary first time around

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-

dictionary full so increased size

LZW decompression – Example

Compressed file: 10000000110001101010111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	**G**	000	**A**	**GA**	4

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	**A**	001	**C**	**AC**	5

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	**C**	100	**G**A	**CG**	6

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 3

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	**GA**	011	**T**	**GAT**	7

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7

dictionary full so increased size ($2^3=8$)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC				

LZW decompression – Example

```
Compressed file: 10000001100011010101111001
file size = 26 bits
```

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC	0111	GAT		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	**AC**	0111	**G**AT	**ACG**	9

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC	0111	GAT	ACG	9
7	23	GAT				

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC	0111	GAT	ACG	9
7	23	GAT	1001	ACG		

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

dictionary: A:0 C:1 G:2 T:3 GA:4 AC:5 CG:6 GAT:7 TA:8 ACG:9 GATA:10
codeword length: 4

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC	0111	GAT	ACG	9
7	23	**GAT**	1001	ACG	**GATA**	10

new string = old string + string.charAt(0)

LZW decompression – Example

Compressed file: 10000001100011010101111001
file size = 26 bits

Uncompressed Text = G A C G A T A C G A T A C G

step	position in file	old string	code from dictionary	string	add to dictionary	code
0	1	-	10	G	-	-
1	3	G	000	A	GA	4
2	6	A	001	C	AC	5
3	9	C	100	GA	CG	6
4	12	GA	011	T	GAT	7
5	15	T	0101	AC	TA	8
6	19	AC	0111	GAT	ACG	9
7	23	GAT	1001	ACG	GATA	10

LZW decompression – Special case

It is possible to encounter a codeword that is not (yet) in the dictionary

- because decompression is 'out of phase' with compression
- but in that case it is possible to deduce what string it must represent
- consider: A A B A B A B A A and **work through compression and decompression for this text yourselves !**

The solution:

```
if (lookUp fails)  s = oldS + oldS.charAt(0);
```

Example of this special case is available on Moodle next to the slides of this lecture

LZW decompression

Appropriate data structure for decompression is a simple table

Complexity of compression and decompression both O(n)

- for a text of length n (if suitably implemented)
- algorithms essentially involves just one pass through the text
- do need to search for longest strings in the dictionary
  - can do this efficiently with a `Trie` representing the dictionary

LZW compression – Variants

Constant codeword length: fix the codeword length for all time

– the dictionary has fixed capacity: when full, just stop adding to it

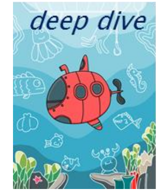
Dynamic codeword length (the version described here)

– start with shortest reasonable codeword length, say, 8 for normal text
– whenever dictionary becomes full
  - add 1 to current codeword length (doubles the number of codewords)
  - does not affect the sequence of codewords already output
– may specify a maximum codeword length, as increasing the size indefinitely may become counter-productive

LRU version: when dictionary full and codeword length maximal

– current string replaces Least Recently Used string in dictionary

LZW (de)compression – application



A digital library stores a vast amount of textual data, including books, academic papers, articles, and historical documents. The library faces challenges in efficiently storing, managing, and retrieving these documents while ensuring they remain accessible and intact for future generations.

Some requirements:

- **Storage optimisation**: How to minimize the storage space required for the vast and ever-growing collection of digital documents?
- **Fast retrieval**: How to ensure that users can quickly search for and access documents without long wait times, despite the library's large size?
- **Bandwidth efficiency**: How to enable efficient downloading or viewing of documents over the internet, especially important for users with limited bandwidth or data plans?

LZW (de)compression – application

Possible solution using LZW:

- Storage space reduction:
  - apply LZW compression to all documents in the digital library
  - LZW is highly effective for compressing text documents,
- Improved retrieval speeds:
  - storing documents in compressed form can also speed up retrieval times
  - smaller files require less disk I/O time to read from storage, and when combined with efficient decompression algorithms, the documents can be quickly accessed and decompressed on-the-fly for user access
- Efficient document transmission:
  - when users request to download/view documents, transmitting the compressed files uses less bandwidth hence faster download times
  - the LZW decompression can either occur server-side for web viewing or client-side after download, ensuring that the document is presented in its original form without loss of information

Next lecture

Text compression

- Huffman encoding
- LZW compression/decompression

String comparison

- string distance

String/pattern search

- brute force algorithm
- KMP algorithm
- BM algorithm