5.5 DATA COMPRESSION

- introduction
- run-length coding
- Huffman compression
- LZW compression

Data compression

Compression reduces the size of a file:
- To save space when storing it.
- To save time when transmitting it.
- Most files have lots of redundancy.

Who needs compression?
- Moore’s law: # transistors on a chip doubles every 18–24 months.
- Parkinson’s law: data expands to fill space available.
- Text, images, sound, video, ...

“Everyday, we create 2.5 quintillion bytes of data—so much that 90% of the data in the world today has been created in the last two years alone. ” — IBM report on big data (2011)

Applications

Generic file compression.
- Files: GZIP, BZIP, 7z.
- Archivers: PKZIP.
- File systems: NTFS, ZFS, HFS+, ReFS, GFS.

Multimedia.
- Images: GIF, JPEG.
- Sound: MP3.
- Video: MPEG, DivX™, HDTV.

Communication.
- ITU-T T4 Group 3 Fax.
- V.42bis modem.
- Skype, Google hangout.

Databases. Google, Facebook, NSA, ....
Lossless compression and expansion

Message. Bitstream \( B \) we want to compress. 

Compress. Generates a "compressed" representation \( C(B) \).

Expand. Reconstructs original bitstream \( B \).

![Basic model for data compression]

Compression ratio. Bits in \( C(B) \) / bits in \( B \).

Ex. 50–75% or better compression ratio for natural language.

Data representation: genomic code

Genome. String over the alphabet \{A, T, C, G\}.


Standard ASCII encoding.

- 8 bits per char.
- \( 8 \times N \) bits.

Two-bit encoding.

- 2 bits per char.
- \( 2 \times N \) bits (25% compression ratio).

<table>
<thead>
<tr>
<th>char</th>
<th>hex</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>'A'</td>
<td>41</td>
<td>01000001</td>
</tr>
<tr>
<td>'T'</td>
<td>54</td>
<td>01010100</td>
</tr>
<tr>
<td>'C'</td>
<td>43</td>
<td>01000011</td>
</tr>
<tr>
<td>'G'</td>
<td>47</td>
<td>01000111</td>
</tr>
</tbody>
</table>

Fixed-length code. \( k \)-bit code supports alphabet of size \( 2^k \).

Amazing but true. Some genomic databases in 1990s used ASCII.

Food for thought

Data compression has been omnipresent since antiquity:

- Number systems.
- Natural languages.
- Mathematical notation.

has played a central role in communications technology,

- Grade 2 Braille.
- Morse code.
- Telephone system.

and is part of modern life.

- JPEG.
- MP3.
- MPEG.

Q. What role will it play in the future?

Reading and writing binary data

Binary standard input. Read bits from standard input.

```java
public class BinaryStdIn {
    boolean readBoolean() {
        read 1 bit of data and return as a boolean value
    }
    char readChar() {
        read 8 bits of data and return as a char value
    }
    char readChar(int r) {
        read \( r \) bits of data and return as a char value
    }
    boolean isEmpty() {
        is the bitstream empty?
    }
    void close() {
        close the bitstream
    }
}
```

Binary standard output. Write bits to standard output

```java
public class BinaryStdOut {
    void write(boolean b) {
        write the specified bit
    }
    void write(char c) {
        write the specified 8-bit char
    }
    void write(char c, int r) {
        write the \( r \) least significant bits of the specified char
    }
    void close() {
        close the bitstream
    }
}
```
Writing binary data

Date representation. Three different ways to represent 12/31/1999.

A character stream (StdOut)

```java
StdOut.print(month + "/" + day + "/" + year);
```

Three ints (BinaryStdOut)

```java
BinaryStdOut.write((int) 0);  
BinaryStdOut.write((int) 1);  
BinaryStdOut.write((int) 2);
```

A 4-bit field, a 5-bit field, and a 12-bit field (BinaryStdOut)

```java
BinaryStdOut.write((short) 0);  
BinaryStdOut.write((short) 1);  
BinaryStdOut.write((short) 2);
```

Universal data compression

ZeoSync. Announced 100:1 lossless compression of random data using Zero Space Tuner™ and BinaryAccelerator™ technology.

Binary dumps

Q. How to examine the contents of a bitstream?

Standard character stream

```java
% more abra.txt
ABRACADABRA!
```

Bitstream represented with hex digits

```java
% java HexDump 4 < abra.txt
41 42 52 41
41 42 44 41
42 52 41 21
```

Universal data compression

Proposition. No algorithm can compress every bitstring.

Pf 1. [by contradiction]

- Suppose your algorithm that can compress all 1,000-bit strings.
- \(2^{1000} \) possible bitstrings with 1,000 bits.
- Only \(1 + 2 + 4 + \ldots + 2^{998} + 2^{999} \) can be encoded with \( \leq 999 \) bits.
- Similarly, only 1 in \(2^{999}\) bitstrings can be encoded with \( \leq 500 \) bits!
Undecidability

A difficult file to compress: one million (pseudo-) random bits

```java
public class RandomBits {
    public static void main(String[] args) {
        int x = 11111;
        for (int i = 0; i < 1000000; i++) {
            x = x * 314159 + 218281;
            BinaryStdOut.write(x > 0);
        }
        BinaryStdOut.close();
    }
}
```

Rudundancy in English Language

Q. How much redundancy in the English language?
A. Quite a bit.

“... randomising letters in the middle of words [has] little or no effect on the ability of skilled readers to understand the text. This is easy to demntrasote. In a pubiltacion of New Scnieitst you could ramdinoe all the letetr, keipeng the first two and last two the same, and reibadality would hardly be afeeted. My ansaylis did not come to much beucase the thoery at the time was for shape and senqeuce retigiconon. Saberi’s work sugsegts we may have some pofrweul palrlael prsooscers at work. The resaon for this is suerly that idnetifyng coentnt by paarllel prseocsing speeds up regnicoiton. We only need the first and last two letetrs to spot chganes in meniang.” — Graham Rawlinson

The goal of data cmperisoson is to inetdify rdenudcaney and exploit it.

Run-length encoding

Simple type of redundancy in a bitstream. Long runs of repeated bits.

```
00000000000111111000000111111111
```

40 bits

Representation. 4-bit counts to represent alternating runs of 0s and 1s:
15 0s, then 7 1s, then 7 0s, then 11 1s.

```
1111011101110111
```

16 bits (instead of 40)

Q. How many bits to store the counts?
A. We typically use 8 (but 4 in the example above for brevity).

Q. What to do when run length exceeds max count?
A. Intersperse runs of length 0.

Applications. JPEG, ITU-T T4 Group 3 Fax, ...
Run-length encoding: Java implementation

```java
public class RunLength {
    private final static int R = 256;
    private final static int lgR = 8;

    public static void compress() {
        /* see textbook */
    }

    public static void expand() {
        boolean bit = false;
        while (!BinaryStdIn.isEmpty()) {
            int run = BinaryStdIn.readInt(lgR);
            for (int i = 0; i < run; i++)
                BinaryStdOut.write(bit);
            bit = !bit;
        }
        BinaryStdOut.close();
    }
}
```

Data compression: quiz 1

What is the best compression ratio achievable from run-length coding when using 8-bit counts?

A. 1 / 256
B. 1 / 16
C. 8 / 255
D. 24 / 510 = 4 / 85
E. I don't know.

Variable-length codes

Use different number of bits to encode different chars.

Ex. Morse code: • • • – – – • • •

Issue. Ambiguity.

SOS?
V7?
IAMIE?
EEJNI?

In practice. Use a medium gap to separate codewords.

Codeword for S is a prefix of codeword for V.
Variable-length codes

Q. How do we avoid ambiguity?
A. Ensure that no codeword is a prefix of another.

Ex 1. Fixed-length code.
Ex 2. Append special stop character to each codeword.
Ex 3. General prefix-free code.

Prefix-free codes: trie representation

Q. How to represent the prefix-free code?
A. A binary trie!
   - Characters in leaves.
   - Codeword is path from root to leaf.

Prefix-free codes: expansion

Expansion.
- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, write character; return to root node; repeat.

Prefix-free codes: compression

Compression.
- Method 1: start at leaf; follow path up to the root; print bits in reverse.
- Method 2: create ST of key-value pairs.
Data compression: quiz 2

Consider the following trie representation of a prefix-free code. Expand the compressed bit string 100101000111011?

A. PEED
B. PESDEY
C. SPED
D. SPEDEY
E. I don’t know.

Huffman trie node data type

```java
private static class Node implements Comparable<Node> {
    private final char ch; // used only for leaf nodes
    private final int freq; // used only by compress()
    private final Node left, right;

    public Node(char ch, int freq, Node left, Node right) {
        this.ch = ch;
        this.freq = freq;
        this.left = left;
        this.right = right;
    }

    public boolean isLeaf() {
        return left == null && right == null;
    }

    public int compareTo(Node that) {
        return this.freq - that.freq;
    }
}
```

Prefix-free codes: expansion

```java
public void expand() {
    Node root = readTrie();
    int N = BinaryStdIn.readInt();

    for (int i = 0; i < N; i++) {
        Node x = root;
        while (!x.isLeaf()) {
            if (!BinaryStdIn.readBoolean())
                x = x.left;
            else
                x = x.right;
        }
        BinaryStdOut.write(x.ch, 8);
    }
    BinaryStdOut.close();
}
```

Running time. Linear in input size $N$.
Prefix-free codes: how to transmit

Q. How to write the trie?
A. Write preorder traversal of trie; mark leaf and internal nodes with a bit.

```
private static void writeTrie(Node x) {
    if (x.isLeaf())
        { 
            if (x.isLeaf()))
            BinaryStdOut.write(true);
            BinaryStdOut.write(x.ch, 8);
            return;
        } 
        BinaryStdOut.write(false);
        writeTrie(x.left);
        writeTrie(x.right);
}
```

Using preorder traversal to encode a trie as a bitstream

```
public static Node readTrie() {
    if (BinaryStdIn.readBoolean())
        { 
            char c = BinaryStdIn.readChar(8);
            return new Node(c, 0, null, null);
        } 
        Node x = readTrie();
        Node y = readTrie();
        return new Node("\0", 0, x, y);
}
```

Note. If message is long, overhead of transmitting trie is small.

Huffman codes

Q. How to find best prefix-free code?

Huffman algorithm:
- Count frequency freq[i] for each char i in input.
- Start with one node corresponding to each char i (with weight freq[i]).
- Repeat until single trie formed:
  - select two tries with min weight freq[i] and freq[j]
  - merge into single trie with weight freq[i] + freq[j]

Applications:

![Images of file formats: JPEG, PDF, mp3, DJVU, and GZIP]

Constructing a Huffman encoding trie: Java implementation

```
private static Node buildTrie(int[] freq) {
    MinPQ<Node> pq = new MinPQ<Node>();
    for (char i = 0; i < R; i++)
        if (freq[i] > 0)
            pq.insert(new Node(i, freq[i], null, null));

    while (pq.size() > 1) {
        Node x = pq.delMin();
        Node y = pq.delMin();
        Node parent = new Node("\0", x.freq + y.freq, x, y);
        pq.insert(parent);
    }
    return pq.delMin();
}
```

initialize PQ with singleton tries
merge two smallest tries
not used for internal nodes
total frequency two subtrees
Huffman compression summary

**Proposition.** Huffman’s algorithm produces an optimal prefix-free code.
**Pf.** See textbook.

Two-pass implementation (for compression).
- Pass 1: tabulate character frequencies; build trie.
- Pass 2: encode file by traversing trie (or symbol table).

Running time (for compression). Using a binary heap \(\Rightarrow N + R \log R\).
Running time (for expansion). Using a binary trie \(\Rightarrow N\).

Q. Can we do better? [stay tuned]

---

5.5 DATA COMPRESSION

- introduction
- run-length coding
- Huffman compression
- LZW compression

Jacob Ziv
Abraham Lempel

---

Statistical methods

**Static model.** Same model for all texts.
- Fast.
- Not optimal: different texts have different statistical properties.
- Ex: ASCII, Morse code.

**Dynamic model.** Generate model based on text.
- Preliminary pass needed to generate model.
- Must transmit the model.
- Ex: Huffman code.

**Adaptive model.** Progressively learn and update model as you read text.
- More accurate modeling produces better compression.
- Decoding must start from beginning.
- Ex: LZW.

---

LZW compression demo

input: A B R A C A D A B R A B R A A
matches: A B R A C A D A B R A B R A A
value: 41 42 52 41 43 41 44 81 83 82 88 41 80

LZW compression for A B R A C A D A B R A B R A A
codeword table

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>81</td>
</tr>
<tr>
<td>BR</td>
<td>82</td>
</tr>
<tr>
<td>RA</td>
<td>83</td>
</tr>
<tr>
<td>AC</td>
<td>84</td>
</tr>
<tr>
<td>CA</td>
<td>85</td>
</tr>
<tr>
<td>AD</td>
<td>86</td>
</tr>
<tr>
<td>DA</td>
<td>87</td>
</tr>
<tr>
<td>ABR</td>
<td>88</td>
</tr>
<tr>
<td>RAB</td>
<td>89</td>
</tr>
<tr>
<td>BRA</td>
<td>8A</td>
</tr>
<tr>
<td>ABRA</td>
<td>8B</td>
</tr>
</tbody>
</table>
Lempel-Ziv-Welch compression

LZW compression.
- Create ST associating $W$-bit codewords with string keys.
- Initialize ST with codewords for single-character keys.
- Find longest string $s$ in ST that is a prefix of unscanned part of input.
- Write the $W$-bit codeword associated with $s$.
- Add $s + c$ to ST, where $c$ is next character in the input.

Q. How to represent LZW compression code table?
A. A trie to support longest prefix match.

LZW expansion

LZW expansion.
- Create ST associating string values with $W$-bit keys.
- Initialize ST to contain single-character values.
- Read a $W$-bit key.
- Find associated string value in ST and write it out.
- Update ST.

Q. How to represent LZW expansion code table?
A. An array of length $2^W$.

Data compression: quiz 3

What is the LZW compression of A B A B A B A?

A. 41 42 41 42 41 42 80
B. 41 42 41 81 81
C. 41 42 81 81 41
D. 41 42 81 83 80
E. I don't know.
LZW tricky case: compression

```
input  A  B  A  B  A  B  A
matches A  B  A  B  A  B  A
value  41 42  81  83  80
```

LZW compression for ABABABA

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AB</td>
</tr>
<tr>
<td>A</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

value

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AB</td>
</tr>
<tr>
<td>A</td>
<td>41</td>
</tr>
<tr>
<td>B</td>
<td>42</td>
</tr>
<tr>
<td>C</td>
<td>43</td>
</tr>
<tr>
<td>D</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

output

```
A  B  A  B  A  B  A
```

LZW tricky case: expansion

```
value  41  42  81  83  80
```

LZW expansion for 41 42 81 83 80

```
<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

LZW in the real world

Lempel-Ziv and friends.
- LZ77.
- LZ78.
- LZW.
- Deflate / zlib = LZ77 variant + Huffman.

Unix compress, GIF, TIFF, V.42bis modem: LZW.
- zip, 7zip, gzip, jar, png, pdf: deflate / zlib.
- iPhone, Wii, Apache HTTP server: deflate / zlib.

Apache HTTP Server Project
## Lossless data compression benchmarks

<table>
<thead>
<tr>
<th>Year</th>
<th>Scheme</th>
<th>Bits / Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>1967</td>
<td>ASCII</td>
<td>7.00</td>
</tr>
<tr>
<td>1950</td>
<td>Huffman</td>
<td>4.70</td>
</tr>
<tr>
<td>1977</td>
<td>LZ77</td>
<td>3.94</td>
</tr>
<tr>
<td>1984</td>
<td>LZW</td>
<td>3.32</td>
</tr>
<tr>
<td>1987</td>
<td>move-to-front</td>
<td>3.24</td>
</tr>
<tr>
<td>1987</td>
<td>LZ8</td>
<td>3.18</td>
</tr>
<tr>
<td>1987</td>
<td>gzip</td>
<td>2.71</td>
</tr>
<tr>
<td>1988</td>
<td>PPM</td>
<td>2.48</td>
</tr>
<tr>
<td>1994</td>
<td>SAKDC</td>
<td>2.47</td>
</tr>
<tr>
<td>1994</td>
<td>PPM</td>
<td>2.34</td>
</tr>
<tr>
<td>1995</td>
<td>Burrows–Wheeler</td>
<td>2.29</td>
</tr>
<tr>
<td>1997</td>
<td>BOA</td>
<td>1.99</td>
</tr>
<tr>
<td>1999</td>
<td>RK</td>
<td>1.89</td>
</tr>
</tbody>
</table>

Data compression using Calgary corpus

## Data compression summary

**Lossless compression.**
- Represent fixed-length symbols with variable-length codes. [Huffman]
- Represent variable-length symbols with fixed-length codes. [LZW]

**Lossy compression.** [not covered in this course]
- JPEG, MPEG, MP3, ...
- FFT/DCT, wavelets, fractals, ...

**Theoretical limits on compression.** Shannon entropy: 
\[ H(X) = - \sum_{i} p(x_i) \log p(x_i) \]

**Practical compression.** Exploit extra knowledge whenever possible.