5.5 Data Compression

- basics
- run-length encoding
- 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, ...

“...All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value.” — Carl Sagan

Basic concepts ancient (1950s), best technology recently developed.

Applications

Generic file compression.
- Files: GZIP, BZIP, BOA.
- Archivers: PKZIP.
- File systems: NTFS.

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

Communication.
- ITU-T T4 Group 3 Fax.
- V.42bis modem.

Databases.
- Google.

Lossless compression and expansion

Message. Binary data $B$ we want to compress.
Compress. Generates a "compressed" representation $C(B)$.
Expand. Reconstructs original bitstream $B$.

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

Ex. 50-75% or better compression ratio for natural language.
Food for thought

Data compression has been omnipresent since antiquity:
- Number systems.
- Natural languages.
- Mathematical notation.

has played a central role in communications technology,
- Braille.
- Morse code.
- Telephone system.

and is part of modern life.
- MP3.
- MPEG.

Q. What role will it play in the future?

Reading and writing binary data

Binary standard input and standard output. Libraries to read and write bits from standard input and to standard output.

```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) // similar methods for byte (8 bits); short (16 bits); int (32 bits); long and double (64 bits)
boolean isEmpty() // is the bitstream empty?
void close() // close the bitstream
```

```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 similar methods for byte (8 bits); short (16 bits); int (32 bits); long and double (64 bits)
void close() // close the bitstream
```

Writing binary data

Date representation. Different ways to represent 12/31/1999.

```
A character stream (StdOut)
StdOut.print(month + " / " + day + " / " + year);
```

```
Three ints (BinaryStdOut)
BinaryStdOut.write(month);
BinaryStdOut.write(day);
BinaryStdOut.write(year);
```

```
Two chars and a short (BinaryStdOut)
BinaryStdOut.write((char) month);
BinaryStdOut.write((char) day);
BinaryStdOut.write(year);
```

```
A 4-bit field, a 6-bit field, and a 12-bit field (BinaryStdOut)
BinaryStdOut.write(month, 4);
BinaryStdOut.write(day, 6);
BinaryStdOut.write(year, 12);
```

Four ways to put a date onto standard output
Q. How to examine the contents of a bitstream?

- Standard character stream
  - `% more abra.txt
  - ABRACADABRA!

- Bitstream represented as 0 and 1 characters
  - `% java BinaryDump 16 < abra.txt
  - 0100000100100001
  - 0100001001010010
  - 0100001101000001
  - 0101001001000001
  - 0100000101000010
  - 96 bits

- Bitstream represented with hex digits
  - `% java HexDump 4 < abra.txt
  - 41 42 52 41
  - 43 41 44 41
  - 42 52 41
  - 96 bits

- Bitstream represented as pixels in a Picture
  - `% java PictureDump 16 < abra.txt
  - 66 77 68 65
  - 20 22 20 20
  - 26-by-6 pixel window, magnified
  - 96 bits

Universal data compression

- US Patent 5,533,051 on "Methods for Data Compression", which is capable of compression all files.

- Slashdot reports of the Zero Space Tuner™ and BinaryAccelerator™.

  “ZeoSync has announced a breakthrough in data compression that allows for 100:1 lossless compression of random data. If this is true, our bandwidth problems just got a lot smaller…”

Universal data compression

**Proposition.** No algorithm can compress every bitstring.

**Proof 1.** [by contradiction]
- Suppose you have a universal data compression algorithm U that can compress every bitstream.
- Given bitstring \( B_0 \), compress it to get smaller bitstring \( B_1 \).
- Compress \( B_1 \) to get a smaller bitstring \( B_2 \).
- Continue until reaching bitstring of size 0.
- Implication: all bitstrings can be compressed with 0 bits!

**Proof 2.** [by counting]
- Suppose your algorithm that can compress all 1,000-bit strings.
- 2,000 possible bitstrings with 1000 bits.
- Only \( 1 + 2 + 4 + ... + 2^{998} + 2^{999} \) can be encoded with \( \leq 999 \) bits.
- Similarly, only 1 in \( 2^{999} \) bitstrings can be encoded with \( \leq 500 \) bits!
Perpetual motion machines

Universal data compression is the analog of perpetual motion.

Closed-cycle mill by Robert Fludd, 1618
Gravity engine by Bob Schadevold

Reference: Museum of Unworkable Devices by Donald E. Simanek
http://www.lhup.edu/~dsimanek/museum/unwork.htm

Undecidability

% java RandomBits | java PictureDump 2000 500

100000 bits

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

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();
    }
}

Rdenudcany in Enlgsih Inaguge

Q. How much redundancy is in the English language?

"... 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 demntrasate. In a pubilcation of New Scnieitst you could
randomise all the letetrs, kepeng the first two and last two the
same, and reibadality would hardly be offeeed. My ansaylis did not
come to much beucase the thoery at the time was for shape and
seuence retigciron. Saberi's work sugsegts we may have some
pofrweul palrlael prsooers at work. The resaon for this is suerly
that idnetifying coennt by paarlrel prsooings speeds up
regnicoiton. We only need the first and last two letetrs to spot
chganes in meniag." — Graham Rawlinson

A. Quite a bit.
Genomic code

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

Goal. Encode an N-character genome: \texttt{ATAGATGCATAG...}

Standard ASCII encoding.
• 8 bits per char.
• 8N bits.

Two-bit encoding encoding.
• 2 bits per char.
• 2N bits.

Amazing but true. Initial genomic databases in 1990s did not use such a code!

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

<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>C</td>
<td>43</td>
<td>01000011</td>
</tr>
<tr>
<td>T</td>
<td>54</td>
<td>01010100</td>
</tr>
<tr>
<td>G</td>
<td>47</td>
<td>01000111</td>
</tr>
</tbody>
</table>

Genomic code: test client and sample execution

```java
public class Genome {
    public static void compress() {
        Alphabet DNA = new Alphabet("ACTG");
        String s = BinaryStdIn.readString();
        int N = s.length();
        BinaryStdOut.write(N);
        for (int i = 0; i < N; i++) {
            int d = DNA.toIndex(s.charAt(i));
            BinaryStdOut.write(d, 2);
        }
        BinaryStdOut.close();
    }

    public static void expand() {
        Alphabet DNA = new Alphabet("ACTG");
        int N = BinaryStdIn.readInt();
        for (int i = 0; i < N; i++) {
            char c = BinaryStdIn.readChar(2);
            BinaryStdOut.write(DNA.toChar(c));
        }
        BinaryStdOut.close();
    }
}
```

Compressing and expanding genomic sequences with 2-bit encoding

An actual virus (50000 bits)

Tiny test case (264 bits)

```
% java Genome - < genomeVirus.txt | java PictureDump 512 25
12536 bits
```

```
% more genomeTiny.txt
ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
```

```
% java BitsDump 64 < genomeTiny.txt
01000001010101000100000101000111010000010101010001000111010000110100001101000001
0101010001000001010001110100001101010100010000010100011101000001
```

```
% java Genome - < genomeTiny.txt | java BinaryDump 64
```

```
00000000000000000000000000100001001000110010110100100011011101001000110110001100101110110001101000000
```

```
% java Genome - < genomeTiny.txt | java HexDump 8
00 00 00 21 23 2d 23 74
```

```
% java Genome - < genomeTiny.txt | java Genome +
ATAGATGCATAGCGCATAGCTAGATGTGCTAGC
cannot see bitstream on standard output
```

compress-expand cycle produces original input

Genomic encoding
• genomic encoding
• run-length encoding
• Huffman compression
• LZW compression
Run-length encoding

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

```
000000000000000111111110000001111111111
```

Representation. Use 4-bit counts to represent alternating runs of 0s and 1s:

```
1111 0111 1011 1111
```

Q. How many bits to store the counts?
A. We'll use 8.

Q. What to do when run length exceeds max count?
A. If longer than 255, intersperse runs of length 0.

Applications. JPEG, ITU-T T4 Group 3 Fax, ...

---

An application: compress a bitmap

Typical black-and-white-scanned image.
- 300 pixels/inch.
- 8.5-by-11 inches.
- 300 \times 8.5 \times 300 \times 11 = 8,415 million bits.

Observation. Bits are mostly white.

Typical amount of text on a page.
40 lines \times 75 \text{ chars per line} = 3,000 \text{ chars.}
### Variable-length codes

Use different number of bits to encode different chars.

**Ex.** Morse code: ⋅ ⋅ ⋅ − − − − ⋅ ⋅ ⋅

**Issue.** Ambiguity.
- SOS?
- IAMIE?
- EEDNII?
- V7?

In practice. Use a medium gap to separate codewords.

### Prefix-free codes: trie representation

**Q.** How to represent the prefix-free code?

**A.** A binary trie!
- Chars in leaves.
- Codeword is path from root to leaf.

**Ex.** Morse code: ⋅ ⋅ ⋅ − − − − ⋅ ⋅ ⋅

**Letter** | **Number**
--- | ---
A | ⋅ ⋅ ⋅
B | ⋅ ⋅ ⋅ −
C | ⋅ ⋅ ⋅ − −
D | ⋅ ⋅ ⋅ − − −
E | ⋅ ⋅ ⋅ − − − −
F | ⋅ ⋅ − − − − −
G | ⋅ − − − − − − −
H | ⋅ − − − − − − − −
I | ⋅ − − − − − − − − −
J | ⋅ − − − − − − − − − −
K | ⋅ − − − − − − − − − − −
L | − − − − − − − − − − − − −
M | − − − − − − − − − − − − − −
N | − − − − − − − − − − − − − − −
O | − − − − − − − − − − − − − − − −
P | − − − − − − − − − − − − − − − − −
Q | − − − − − − − − − − − − − − − − − −
R | − − − − − − − − − − − − − − − − − − −
S | − − − − − − − − − − − − − − − − − − − −
T | − − − − − − − − − − − − − − − − − − − − −
U | − − − − − − − − − − − − − − − − − − − − − −
V | − − − − − − − − − − − − − − − − − − − − − − −
W | − − − − − − − − − − − − − − − − − − − − − − − −
X | − − − − − − − − − − − − − − − − − − − − − − − − −
Y | − − − − − − − − − − − − − − − − − − − − − − − − − −
Z | − − − − − − − − − − − − − − − − − − − − − − − − − − −

**Codeword**

- **Ex 2.** Append special stop char to each codeword.
- **Ex 3.** General prefix-free code.

### Prefix-free codes: compression and expansion

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

**Expansion.**
- Start at root.
- Go left if bit is 0; go right if 1.
- If leaf node, print char and return to root.
Huffman trie node data type

```java
private static class Node implements Comparable<Node> {
    private char ch;  // Unused for internal nodes.
    private int freq; // Unused for expand.
    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);
    }
    BinaryStdOut.close();
}
```

Running time. Linear in input size (constant amount of work per bit read).

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.

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

Q. How to read in the trie?
A. Reconstruct from preorder traversal of trie.

```java
private static Node readTrie() {
    if (BinaryStdIn.readBoolean()) {
        char c = BinaryStdIn.readChar();
        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?

**A.** Huffman algorithm.

**Huffman algorithm (to compute optimal prefix-free code):**

- 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.** JPEG, MP3, MPEG, PKZIP, GZIP, ...

---

**Constructing a Huffman encoding trie: Java implementation**

```java
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();
}
```

---

**Huffman encoding summary**

**Proposition.** [Huffman 1950s] Huffman algorithm produces an optimal prefix-free code.

**Pf.** See textbook.

**Implementation.**

- **Pass 1:** tabulate char frequencies and build trie.
- **Pass 2:** encode file by traversing trie or lookup table.

**Running time.** Using a binary heap $\Rightarrow O(N + R \log R)$.

**Q.** Can we do better? [stay tuned]
LZW compression example

LZW compression for ABRACADABRABRA

LZW compression example

LZW compression for ABRACADABRABRA

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.

Lempel-Ziv-Welch compression example

LZW compression for ABRACADABRABRA

LZW compression

- Create ST associating W-bit codewords with string keys.
- Initialize ST with codewords for single-char 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 char in the input.
**LZW expansion**

1. Create ST associating string values with W-bit keys.
2. Initialize ST to contain with single-char values.
3. Read a W-bit key.
4. Find associated string value in ST and write it out.
5. Update ST.

**Remark.** Every prefix of a key in encoding table is also in encoding table.

**LZW expansion: tricky situation**

**Q.** What to do when next codeword is not yet in ST when needed?

```java
public static void compress()
{
    String input = BinaryStdIn.readString();
    TST<Integer> st = new TST<Integer>();
    for (int i = 0; i < R; i++)
        st.put(“” + (char) i, i);
    int code = R + 1;
    while (input.length() > 0)
    {
        String s = st.longestPrefixOf(input);
        BinaryStdOut.write(st.get(s), W);
        int t = s.length();
        if (t < input.length() && code < L)
        {
            st.put(input.substring(0, t+1), code++);
        }
        input = input.substring(t);
    }
    BinaryStdOut.write(R, W);
    BinaryStdOut.close();
}
```
LZW implementation details

How big to make ST?
• How long is message?
• Whole message similar model?
  • [many variations have been developed]

What to do when ST fills up?
• Throw away and start over. [GIF]
• Throw away when not effective. [Unix compress]
  • [many other variations]

Why not put longer substrings in ST?
• [many variations have been developed]

LZW in the real world

Lempel-Ziv and friends.
• LZ77. [LZ77 not patented — widely used in open source]
• LZ78. [LZW patent #4,558,302 expired in US on June 20, 2003]
• LZW. [some versions copyrighted]
  • Deflate = LZ77 variant + Huffman.

PNG: LZ77.
Winzip, gzip, jar: deflate.
Unix compress: LZW.
Pkzip: LZW + Shannon-Fano.
GIF, TIFF, V.42bis modem: LZW.
Google: zlib which is based on deflate.
  • never expands a file

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>LZMW</td>
<td>3.32</td>
</tr>
<tr>
<td>1987</td>
<td>LZH</td>
<td>3.30</td>
</tr>
<tr>
<td>1987</td>
<td>move-to-front</td>
<td>3.24</td>
</tr>
<tr>
<td>1987</td>
<td>LZB</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>SAHDC</td>
<td>2.47</td>
</tr>
<tr>
<td>1994</td>
<td>PM</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 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, wavelets, fractals, ...

Theoretical limits on compression. Shannon entropy.

Practical compression. Use extra knowledge whenever possible.