Lecture 15: Compression

http://www.cs.virginia.edu/cs216

Fighting Finalizers

- `finalize()` – method in `java.lang.Object`
  - Class can override
  - It is called when GC determines the object is garbage (before collecting it)

```java
x = NULL;
gc;
```

```java
a.finalize();
b.finalize();
```

Problem due to Paul Tyma, Google

Finality?

```java
class A {
    B p;
}
class B {
    A r;
    void finalize() {
        r.p = this;
    }
}
```

```java
a = new A();
a.p = new B();
a.p.r = a;
x = a;
a.p = NULL;
gc
```

```java
b.finalize();
a.p.toString();
```

Problem due to Paul Tyma, Google

Encoding

- Huffman Encoding
  - We proved there is no better encoding that is:
    - Prefix encoding (can divide coded message into symbols without looking ahead)
    - One-to-one mapping (Symbol → Bits)
    - Fixed mapping
  - Can we do better without these constraints?

Lempel-Ziv-Wench (LZW)

- Terry Wench refined the L-Z scheme
- Fixed-length (typically 12-bits) codewords
- Dictionary maps each codeword to text
- Greedy scheme for building dictionary
LZW Encoding Algorithm

def LZWEncode(s):
    w = ""
    res = ""
    dictionary.initialize()  # code for each alphabet symbol
    foreach k in s:
        if dictionary.contains(w + k):
            w = w + k;
        else:
            # need to do something if dictionary is full
            dictionary.append(w + k)
            res = res + dictionary.find(w)
            w = k;
        w
    return res

Compression Bake-off

Declaration of Independence

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>8586</td>
</tr>
<tr>
<td>Huffman (PS4)</td>
<td>5123 (60%)</td>
</tr>
<tr>
<td>Compress (LZW)</td>
<td>4441 (52%)</td>
</tr>
<tr>
<td>Gzip (not LZW)</td>
<td>3752 (44%)</td>
</tr>
</tbody>
</table>

Random Characters

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>10000</td>
</tr>
<tr>
<td>Huffman (PS4)</td>
<td>9517</td>
</tr>
<tr>
<td>Compress (LZW)</td>
<td>10000 (&quot;file unchanged&quot;)</td>
</tr>
<tr>
<td>Gzip (not LZW)</td>
<td>8800</td>
</tr>
</tbody>
</table>

This is quite surprising!

GIF

- Graphics Interchange Format developed by Compuserve (1987)
- Algorithm:
  - Divide image into 8x8 blocks
  - Find optimal Huffman encoding for those blocks
  - Encode result using LZW

How is GIF different from JPEG?

Lossy/Lossless Compression

- Lossless Compression:
  - uncompress (compress (S)) = S
- Lossy Compression:
  - uncompress (compress (S)) similar to S
- For images, sound, video: lossy compression is usually okay
- For computer programs, declarations of independence, email: only lossless compression will do!

What’s wrong with GIF?

- 1978: LZ patented by Sperry
- 1984: (June) Welch’s article on LZW
- 1984: (July) Unix compress implemented using LZW
- 1987: Compuserve develops GIF (Graphics Interchange Format) image format, used LZW but didn’t know it was patented
- GIF becomes popular
- 1994: Unisys/Compuserve decide that developers who implement LZW (including in GIF) will have to pay a licensing fee
- 2003: LZ patent expired

PNG (“PNG’s Not GIF”)

Representing Numbers
Binary Representation

\[ b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \ldots + b_1 \cdot 2^1 + b_0 \cdot 2^0 \]

What should \( n \) be?

What is \( n \)?

- Java:
  - byte, char = 8 bits
  - short = 16 bits
  - int = 32 bits
  - long = 64 bits
- C: implementation-defined
  - int: can hold between 0 and UINT_MAX
    - UINT_MAX must be at least 65535
  - int >= 16, typical current machines \( n = 32 \)
- Python?
  - \( n \) is not fixed (numbers work)

Charge

- Is Java a “high-level language“?
  - Only if you never use numbers bigger than 231. If you have to worry about how numbers are represented, you are doing low-level programming
- PS4 Due Wednesday