**Shawn Pan - CS 222 Homework 2**

**Code is attached at the end of this document.**

**Problem 1**

We compared gzip (DEFLATE = LZ77 + Huffman), bzip2 (Burrows-Wheeler), xz (LZMA2), and 7zip with the PPM option.

The files we compared included 10MB of random bits, 10MB of 0 bits, text of Alice in Wonderland, a $200 \times 101$ table of numerical data, minimized Javascript code, a PNG image, an mp3 music file, an mp4 video, and parts of a dump of Wikipedia ranging from 1KB to 1GB. The inspiration to use Wikipedia came from this site which has a collection of text compression benchmarks (http://mattmahoney.net/dc/text.html).

Our results show that all the utilities sucessfully compressed text based data by a factor of 3 to 5. However, the media files essentially did not compress at all. As expected, random bits cannot be compressed, and identical zero bits can be compressed almost perfectly. gzip, bzip2, and xz took longer to compress than decompress in general, while ppm was symmetric. Prediction by partial matching (7zip) achieved the best text compression rates. LZ77 (gzip) achieved the fastest times. All the algorithms run in approximately linear time as shown by trend in Wikipedia dumps. Also, larger text files achieved better compression ratios, presumably because more patterns could be exploited.

Table 1: Compressed Sizes (bytes) and Compression Ratios

| File | Original | gzip | ratio | bzip2 | ratio | xz | ratio | 7z ppmd | ratio |
|---|---|---|---|---|---|---|---|---|---|
| random | 10000000 | 10001555 | 1.0002 | 10044419 | 1.0044 | 10000560 | 1.0001 | 10228516 | 1.0229 |
| zero | 10000000 | 9742 | 0.0010 | 49 | 0.0000 | 1584 | 0.0002 | 3791 | 0.0004 |
| alice.txt | 173595 | 61370 | 0.3535 | 49144 | 0.2831 | 54072 | 0.3115 | 44065 | 0.2538 |
| numeric.csv | 505009 | 217219 | 0.4301 | 182355 | 0.3611 | 188288 | 0.3728 | 183292 | 0.3629 |
| jquery.js | 86709 | 29979 | 0.3457 | 27530 | 0.3175 | 28292 | 0.3263 | 24932 | 0.2875 |
| image.png | 736501 | 736514 | 1.0000 | 739754 | 1.0044 | 736600 | 1.0001 | 752219 | 1.0213 |
| music.mp3 | 4426883 | 4385198 | 0.9906 | 4370711 | 0.9873 | 4365156 | 0.9861 | 4386499 | 0.9909 |
| video.mp4 | 905296268 | 904838573 | 0.9995 | 908527162 | 1.0036 | 904758648 | 0.9994 | 925040248 | 1.0218 |
| enwik3 | 1000 | 343 | 0.3430 | 391 | 0.3910 | 400 | 0.4000 | 408 | 0.4080 |
| enwik4 | 10000 | 3728 | 0.3728 | 3733 | 0.3733 | 3656 | 0.3656 | 3393 | 0.3393 |
| enwik5 | 100000 | 36145 | 0.3615 | 31372 | 0.3137 | 33000 | 0.3300 | 28115 | 0.2812 |
| enwik6 | 1000000 | 356264 | 0.3563 | 281323 | 0.2813 | 290692 | 0.2907 | 250071 | 0.2501 |
| enwik7 | 10000000 | 3693807 | 0.3694 | 2916026 | 0.2916 | 2723036 | 0.2723 | 2491710 | 0.2492 |
| enwik8 | 100000000 | 36518329 | 0.3652 | 29008758 | 0.2901 | 26375764 | 0.2638 | 24849802 | 0.2485 |
| enwik9 | 1000000000 | 323742882 | 0.3237 | 253977891 | 0.2540 | 230153052 | 0.2302 | 219169705 | 0.2192 |

Table 2: Compression and Decompression Times (CPU seconds)

| File | gzip compress | gzip decompress | bzip2 compress | bzip2 decompress | xz compress | xz decompress | 7z ppmd compress | 7z ppmd decompress |
|---|---|---|---|---|---|---|---|---|
| random | 0.32 | 0.076 | 1.368 | 0.624 | 2.848 | 0.04 | 3.832 | 4.772 |
| zero | 0.064 | 0.064 | 0.092 | 0.052 | 0.508 | 0.052 | 0.132 | 0.172 |
| alice.txt | 0.008 | 0 | 0.012 | 0.004 | 0.052 | 0.004 | 0.02 | 0.02 |
| numeric.csv | 0.036 | 0.004 | 0.04 | 0.02 | 0.172 | 0.02 | 0.064 | 0.072 |
| jquery.js | 0.004 | 0 | 0.008 | 0.004 | 0.024 | 0 | 0.012 | 0.012 |
| image.png | 0.024 | 0.004 | 0.1 | 0.044 | 0.148 | 0.004 | 0.264 | 0.336 |
| music.mp3 | 0.164 | 0.044 | 0.58 | 0.268 | 1.104 | 0.296 | 1.688 | 2.088 |
| video.mp4 | 28.76 | 6.628 | 122.92 | 60.684 | 295.46 | 4.02 | 341.268 | 432.116 |
| enwik3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| enwik4 | 0 | 0 | 0 | 0 | 0.004 | 0 | 0 | 0.004 |
| enwik5 | 0.004 | 0 | 0.008 | 0.004 | 0.028 | 0 | 0.012 | 0.012 |
| enwik6 | 0.044 | 0.008 | 0.084 | 0.036 | 0.328 | 0.02 | 0.096 | 0.116 |
| enwik7 | 0.456 | 0.092 | 0.824 | 0.36 | 5.156 | 0.184 | 1.024 | 1.18 |
| enwik8 | 4.64 | 0.892 | 8.296 | 3.704 | 60.464 | 1.796 | 10.5 | 11.988 |
| enwik9 | 41.464 | 8.78 | 85.788 | 35.452 | 537.176 | 16.376 | 96.168 | 106.332 |

**Problem 2**

For $X = n$, the coin must land on tails $n-1$ times follwed by heads. Each coin flip happens independently with probability $\frac{1}{2}$, so the probability $X = n$ is $2^{-n}$.

$$
\begin{aligned}
H &= -\sum_{n=1}^{\infty} 2^{-n} \log_2 2^{-n} \\
&= \sum_{n=1}^{\infty} n 2^{-n} \\
&= (1)\frac{1}{2} + (2)\frac{1}{4} + (3)\frac{1}{8} + \dots \\
2H &= (1)1 + (2)\frac{1}{2} + (3)\frac{1}{4} + (4)\frac{1}{8} + \dots \\
2H - H &= (1-0)1 + (2-1)\frac{1}{2} + (3-2)\frac{1}{4} + (4-3)\frac{1}{8} + \dots \\
H &= 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \\
&= \frac{1}{1-\frac{1}{2}} \\
&= 2
\end{aligned}
$$

The optimal series of questions to ask are "Is X = 1?" "Is X = 2?" "Is X = 3?" ... With probablity $\frac{1}{2}$ you are correct on your first question. Otherwise, with probability $(\frac{1}{2})(\frac{1}{2})$ you will be correct on your second questions, and so on. The expected number of questions asked is $E = \sum_{n=1}^{\infty} n 2^{-n} = 2$, which is the same infinite series as the entropy calculation above. As expected $H = E$.

**Problem 3**

**Show** $H(Z|X) = H(Y|X)$

$$\begin{aligned}
H(Z|X) &= \sum_x P(X=x) \sum_z P(Z=z|X=x) \log_2 P(Z=z|X=x) \\
&= \sum_x P(X=x) \sum_z P(X+Y=z|X=x) \log_2 P(X+Y=z|X=x) \\
&= \sum_x P(X=x) \sum_{y=z-x} P(Y=z-x|X=x) \log_2 P(Y=z-x|X=x) \\
&= \sum_x P(X=x) \sum_y P(Y=y|X=x) \log_2 P(Y=y|X=x) \\
&= H(Y|X)
\end{aligned}$$

**Show** $H(Z) \geq H(X)$ **when X and Y are independent**

The marginal entropy $H(Z)$ must be at least the conditional entropy $H(Z|Y)$. By symmetry of the case above, $H(Z|Y) = H(X|Y)$. Because of independence, $H(X|Y) = H(X)$.

$$H(Z) \geq H(Z|Y) = H(X|Y) = H(X)$$

**Example** $H(X) > H(Z)$

Consider the case $X = 1, 2, 3, ..., r$ with equal probabilities $\frac{1}{r}$ and $Y = -X$. $Z = X + Y = 0$ everywhere, so $H(Z) = 0$. However, $H(X) = \log_2 r > 0 = H(Z)$.

**Conditions for** $H(Z) = H(X) + H(Y)$

$H(Z) = H(X) + H(Y)$ when $X$ and $Y$ are independent and all sums $z_{ij} = x_i + y_j$ are distinct. Intuitively, for $Z$ to have the same information content as $X$ and $Y$ independently combined, we must be able to recover x and y given a z.

We will show that these 2 conditions are sufficient. The key is that $P(z_{ij}) = P(x_i)P(y_j)$ because of independence and distinctness.

$$\begin{aligned}
H(Z) &= -\sum_z P(z) \log_2 P(z) \\
&= -\sum_x \sum_y P(x)P(y) \log_2 P(x)P(y) \\
&= -\sum_x \sum_y P(x)P(y) \left(\log_2 P(x) + \log_2 P(y)\right) \\
&= -\sum_x \sum_y P(x)P(y) \log_2 P(x) - \sum_x \sum_y P(x)P(y) \log_2 P(y) \\
&= -\sum_y P(y) \sum_x P(x) \log_2 P(x) - \sum_x P(x) \sum_y P(y) \log_2 P(y) \\
&= -\sum_x P(x) \log_2 P(x) - \sum_y P(y) \log_2 P(y) \\
&= H(X) + H(Y)
\end{aligned}$$

Note that both these conditions are necessary as shown by these two counterexamples:

1. Consider independent $X = 0, 1$ and $Y = 0, 1$ with equal probability for each option. Note that two of the pairs sum to 1, and $Z = 0, 1, 2$ with probabilties $0.25, 0.5, 0.25$. $H(X) = H(Y) = 1$ and $H(Z) = 1.5 \neq H(X) + H(Y)$.

2. Consider $X = 1, 2$ with equal probability and $Y = 2X = 2, 4$. $Z = 3, 4, 5, 6$ with probabilities $0.5, 0, 0, 0.5$. $H(X) = H(Y) = H(Z) = 1$ and $H(Z) \neq H(X) + H(Y)$.

**Problem 4**

Consider any pair of 2 faces of the die. Let $p + \epsilon$ and $p - \epsilon$ be the probabilites of landing on the those 2 faces. Note that the sum is a constant $2p$ independent of $\epsilon$, and we want to figure out how to distribute the probability by varying $\epsilon$ to maximize entropy.

$$
\begin{aligned}
H &= -(p+\epsilon)\log_2(p+\epsilon) - (p-\epsilon)\log_2(p-\epsilon) \\
\frac{dH}{d\epsilon} &= -\log_2(p+\epsilon) - (p+\epsilon)\frac{1}{\ln 2}\frac{1}{p+\epsilon} + \log_2(p-\epsilon) + (p-\epsilon)\frac{1}{\ln 2}\frac{1}{p-\epsilon} \\
&= \log_2\left(\frac{p-\epsilon}{p+\epsilon}\right) \\
&= 0 \\
\frac{p-\epsilon}{p+\epsilon} &= 1 \\
\epsilon &= 0 \\
\frac{d^2H}{d\epsilon^2} &= \left(\frac{1}{\ln 2}\right)\left(\frac{p+\epsilon}{p-\epsilon}\right)\left(\frac{-(p+\epsilon)-(p-\epsilon)}{(p+\epsilon)^2}\right) \\
&= \frac{-2p}{\ln 2(p+\epsilon)(p-\epsilon)} \\
&< 0
\end{aligned}
$$

The derivative is 0 when $\epsilon = 0$, and that point is a maximum because the second derivative is negative. Therefore, entropy is maximized by splitting probabilities equally between 2 faces. By considering all pairs of faces, we can generalize the result to say the entropy of a die is maximized when all faces have equal probability.

**Problem 5**

First we append an end-of-file character to the input and generate all the cyclic shifts.

```
wabbawabbawoo|
abbawabbawoo|w
bbawabbawoo|wa
bawabbawoo|wab
awabbawoo|wabb
wabbawoo|wabba
abbawoo|wabbaw
bbawoo|wabbawa
bawoo|wabbawab
awoo|wabbawabb
woo|wabbawabba
oo|wabbawabbaw
o|wabbawabbawo
|wabbawabbawoo
```

Then we sort the shifts alphabetically.

```
abbawabbawoo|w
abbawoo|wabbaw
awabbawoo|wabb
awoo|wabbawabb
bawabbawoo|wab
bawoo|wabbawab
bbawabbawoo|wa
bbawoo|wabbawa
oo|wabbawabbaw
o|wabbawabbawo
wabbawabbawoo|
wabbawoo|wabba
woo|wabbawabba
|wabbawabbawoo
```

We output the last column wwbbbbaawo|aao.

So far, we have simply transformed the data. The first column has been sorted and is a good predictor for the last column which is cyclicly adjacent. Therefore, the last column should have clusters of characters and can be compressed by move-to-front encoding. We start with a dictionary of all the characters and move them to the front each time a character is used.

```
['a', 'b', 'o', 'w', '|'] [3]
['w', 'a', 'b', 'o', '|'] [3, 0]
['w', 'a', 'b', 'o', '|'] [3, 0, 2]
['b', 'w', 'a', 'o', '|'] [3, 0, 2, 0]
['b', 'w', 'a', 'o', '|'] [3, 0, 2, 0, 0]
['b', 'w', 'a', 'o', '|'] [3, 0, 2, 0, 0, 0]
['b', 'w', 'a', 'o', '|'] [3, 0, 2, 0, 0, 0, 2]
['a', 'b', 'w', 'o', '|'] [3, 0, 2, 0, 0, 0, 2, 0]
```

```
['a', 'b', 'w', 'o', '|'] [3, 0, 2, 0, 0, 0, 2, 0, 2]
['w', 'a', 'b', 'o', '|'] [3, 0, 2, 0, 0, 0, 2, 0, 2, 3]
['o', 'w', 'a', 'b', '|'] [3, 0, 2, 0, 0, 0, 2, 0, 2, 3, 4]
['|', 'o', 'w', 'a', 'b'] [3, 0, 2, 0, 0, 0, 2, 0, 2, 3, 4, 3]
['a', '|', 'o', 'w', 'b'] [3, 0, 2, 0, 0, 0, 2, 0, 2, 3, 4, 3, 0]
['a', '|', 'o', 'w', 'b'] [3, 0, 2, 0, 0, 0, 2, 0, 2, 3, 4, 3, 0, 2]
```

The output of the move-to-front encoding [3, 0, 2, 0, 0, 0, 2, 0, 2, 3, 4, 3, 0, 2] can be efficiently encoded with Huffman or arithmetic coding.

To decoder receives the encoded last column. The move-to-front procedure is applied in reverse to recover the last column wwbbbbaawo|aao. We determine the index of correct row (10) by finding the end-of-file character | in the last column. We sort the last column to recover the first column aaaabbbboowww|. Note that the last and first columns together form all the pairs in the original data.

$$
\begin{aligned}
w_1a_1\\
w_2a_2\\
b_1a_3\\
b_2a_4\\
b_3b_1\\
b_4b_2\\
a_1b_3\\
a_2b_4\\
w_3o_1\\
o_1o_2\\
|_1w_1\\
a_3w_2\\
a_4w_3\\
o_2|_1
\end{aligned}
$$

If we index the identical characters in order, we can follow these pairs along a chain to recover any row.

$$|_1w_1 \to w_1a_1 \to a_1b_3 \to b_3b_1 \to b_1a_3 \to a_3w_2 \to w_2a_2 \to a_2b_4 \to b_4b_2 \to b_2a_4 \to a_4w_3 \to w_3o_1 \to o_1o_2$$

We recover the original phrase wabbawabbawoo.

**Problem 6**

```
Encode
(0.0, 0.2, 1)
(0.0, 0.04000000000000001, 2)
(0.020000000000000004, 0.04000000000000001, 3)
(0.024000000000000004, 0.030000000000000006, 4)
(0.027000000000000003, 0.030000000000000006, 5)
(0.027000000000000003, 0.027600000000000003, 6)
```

The interval from $[0.027, 0.0276)$. To prevent rounding errors, it's probably safest to output the midpoint of the interval. If rounding were not a concern 0.027 would be shorter. We transmit **0.0273** and the length **6**.

```
Decode
c 0.26431398
cb 0.214379933333
cbb 0.0479331111111
cbba 0.239665555556
cbbab 0.132218518519
cbbaba 0.661092592593
cbbabac 0.322185185185
cbbabacb 0.407283950618
cbbabacbb 0.690946502059
cbbabacbbc 0.381893004117
```

To decode we rescale the interval at each step and end up with **cbbabacbbc**.

**Problem 7**

**Forward Result**

S: BCBaCHtHg, B: aE, C: tg, D: ag, E: ac, H: DDEa

**Backward Result**

S: ADEDgEGgtG, A: ga, D: FAA, E: ta, F: ca, G: Fa

**Step-by-step output of my code**

```
Add character a
S: a
Add character a
S: aa
Add character c
S: aac
Add character t
S: aact
Add character g
S: aactg
Add character a
S: aactga
Add character a
S: aactgaa
Enforce digram uniqueness for aa
```

```
S: ActgA, A: aa
Add character c
S: ActgAc, A: aa
Enforce digram uniqueness for Ac
S: BtgB, A: aa, B: Ac
Enforce rule utility for A
S: BtgB, B: aac
Add character a
S: BtgBa, B: aac
Add character t
S: BtgBat, B: aac
Add character g
S: BtgBatg, B: aac
Enforce digram uniqueness for tg
S: BCBaC, B: aac, C: tg
Add character a
S: BCBaCa, B: aac, C: tg
Add character g
S: BCBaCag, B: aac, C: tg
Add character a
S: BCBaCaga, B: aac, C: tg
Add character g
S: BCBaCagag, B: aac, C: tg
Enforce digram uniqueness for ag
S: BCBaCDD, B: aac, C: tg, D: ag
Add character a
S: BCBaCDDa, B: aac, C: tg, D: ag
Add character c
S: BCBaCDDac, B: aac, C: tg, D: ag
Enforce digram uniqueness for ac
S: BCBaCDDE, B: aE, C: tg, D: ag, E: ac
Add character a
S: BCBaCDDEa, B: aE, C: tg, D: ag, E: ac
Add character t
S: BCBaCDDEat, B: aE, C: tg, D: ag, E: ac
Add character a
S: BCBaCDDEata, B: aE, C: tg, D: ag, E: ac
Add character g
S: BCBaCDDEatag, B: aE, C: tg, D: ag, E: ac
Enforce digram uniqueness for ag
S: BCBaCDDEatD, B: aE, C: tg, D: ag, E: ac
Add character a
S: BCBaCDDEatDa, B: aE, C: tg, D: ag, E: ac
Add character g
S: BCBaCDDEatDag, B: aE, C: tg, D: ag, E: ac
Enforce digram uniqueness for ag
S: BCBaCDDEatDD, B: aE, C: tg, D: ag, E: ac
Enforce digram uniqueness for DD
```

```
S: BCBaCFEatF, B: aE, C: tg, D: ag, E: ac, F: DD
Add character a
S: BCBaCFEatFa, B: aE, C: tg, D: ag, E: ac, F: DD
Add character c
S: BCBaCFEatFac, B: aE, C: tg, D: ag, E: ac, F: DD
Enforce digram uniqueness for ac
S: BCBaCFEatFE, B: aE, C: tg, D: ag, E: ac, F: DD
Enforce digram uniqueness for FE
S: BCBaCGatG, B: aE, C: tg, D: ag, E: ac, F: DD, G: FE
Enforce rule utility for F
S: BCBaCGatG, B: aE, C: tg, D: ag, E: ac, G: DDE
Add character a
S: BCBaCGatGa, B: aE, C: tg, D: ag, E: ac, G: DDE
Enforce digram uniqueness for Ga
S: BCBaCHtH, B: aE, C: tg, D: ag, E: ac, G: DDE, H: Ga
Enforce rule utility for G
S: BCBaCHtH, B: aE, C: tg, D: ag, E: ac, H: DDEa
Add character g
S: BCBaCHtHg, B: aE, C: tg, D: ag, E: ac, H: DDEa

Add character g
S: g
Add character a
S: ga
Add character c
S: gac
Add character a
S: gaca
Add character g
S: gacag
Add character a
S: gacaga
Enforce digram uniqueness for ga
S: AcaA, A: ga
Add character g
S: AcaAg, A: ga
Add character a
S: AcaAga, A: ga
Enforce digram uniqueness for ga
S: AcaAA, A: ga
Add character t
S: AcaAAt, A: ga
Add character a
S: AcaAAta, A: ga
Add character c
S: AcaAAtac, A: ga
Add character a
S: AcaAAtaca, A: ga
```

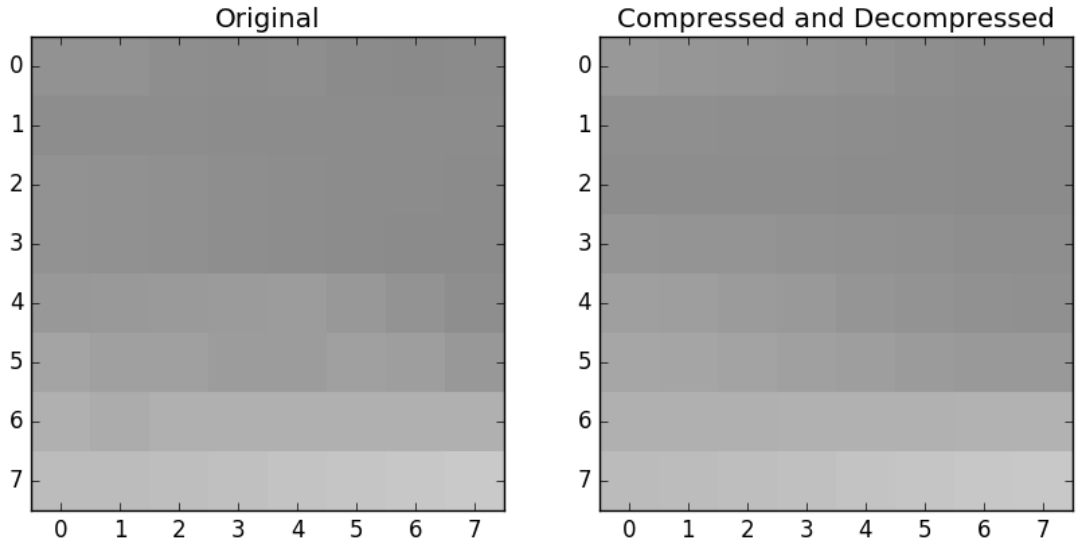
```
Enforce digram uniqueness for ca
S: ABAAtaB, A: ga, B: ca
Add character g
S: ABAAtaBg, A: ga, B: ca
Add character a
S: ABAAtaBga, A: ga, B: ca
Enforce digram uniqueness for ga
S: ABAAtaBA, A: ga, B: ca
Enforce digram uniqueness for BA
S: ACAtaC, A: ga, B: ca, C: BA
Enforce rule utility for B
S: ACAtaC, A: ga, C: caA
Add character g
S: ACAtaCg, A: ga, C: caA
Add character a
S: ACAtaCga, A: ga, C: caA
Enforce digram uniqueness for ga
S: ACAtaCA, A: ga, C: caA
Enforce digram uniqueness for CA
S: ADtaD, A: ga, C: caA, D: CA
Enforce rule utility for C
S: ADtaD, A: ga, D: caAA
Add character g
S: ADtaDg, A: ga, D: caAA
Add character t
S: ADtaDgt, A: ga, D: caAA
Add character a
S: ADtaDgta, A: ga, D: caAA
Enforce digram uniqueness for ta
S: ADEDgE, A: ga, D: caAA, E: ta
Add character c
S: ADEDgEc, A: ga, D: caAA, E: ta
Add character a
S: ADEDgEca, A: ga, D: caAA, E: ta
Enforce digram uniqueness for ca
S: ADEDgEF, A: ga, D: FAA, E: ta, F: ca
Add character a
S: ADEDgEFa, A: ga, D: FAA, E: ta, F: ca
Add character g
S: ADEDgEFag, A: ga, D: FAA, E: ta, F: ca
Add character t
S: ADEDgEFagt, A: ga, D: FAA, E: ta, F: ca
Add character c
S: ADEDgEFagtc, A: ga, D: FAA, E: ta, F: ca
Add character a
S: ADEDgEFagtca, A: ga, D: FAA, E: ta, F: ca
Enforce digram uniqueness for ca
S: ADEDgEFagtF, A: ga, D: FAA, E: ta, F: ca
```

```
Add character a
S: ADEDgEFagtFa, A: ga, D: FAA, E: ta, F: ca
Enforce digram uniqueness for Fa
S: ADEDgEGgtG, A: ga, D: FAA, E: ta, F: ca, G: Fa
```

### Problem 8

```
Compress
Shift by 128
[[ -4  -3  -6  -8  -6  -9 -11 -10]
 [ -8  -8  -8  -9  -9  -8  -8  -8]
 [ -3  -4  -5  -6  -7  -8  -9 -10]
 [ -3  -4  -5  -6  -7  -8  -9 -10]
 [  2   3   4   5   6   2  -2  -6]
 [ 12   9   9   5   5   9   7   2]
 [ 22  19  22  22  22  22  22  22]
 [ 32  32  34  36  40  42  44  47]]

DCT
[[  40.62    5.58   -1.59    3.01   -0.62   -0.11    1.06    1.12]
 [-111.32   11.43    0.4    -0.77   -0.1    -2.01   -1.34   -0.47]
 [  53.87  -12.25    4.16   -1.77    1.4    -1.61    0.33    0.22]
 [ -19.71    8.63   -2.11    2.39   -1.86   -0.12    0.34    0.48]
 [  12.37   -2.15   -2.87   -0.48    1.62   -2.56   -1.38    0.2 ]
 [  -7.81    5.52    2.07   -0.55    1.02    0.26    0.27    0.12]
 [   2.99    2.19    2.33    1.62   -1.53   -0.38    0.09   -0.1 ]
 [   6.19    2.09   -5.01   -0.49    1.89   -0.78   -0.8     0.41]]

Quantized
[[ 3  1  0  0  0  0  0  0]
 [-9  1  0  0  0  0  0  0]
 [ 4 -1  0  0  0  0  0  0]
 [-1  1  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0]]

Decompress
Unquantized
[[  48   11    0    0    0    0    0    0]
 [-108   12    0    0    0    0    0    0]
 [  56  -13    0    0    0    0    0    0]
 [ -14   17    0    0    0    0    0    0]
 [  18    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0]]
```

```
IDCT
[[  2   1   0  -2  -4  -6  -8  -9]
 [ -6  -6  -7  -7  -8  -9 -10 -10]
 [ -8  -8  -8  -8  -8  -9  -9  -9]
 [  0  -1  -1  -3  -4  -5  -6  -6]
 [  8   7   5   3   0  -2  -4  -5]
 [ 14  13  11   9   7   5   3   3]
 [ 22  22  22  23  23  23  24  24]
 [ 31  32  34  37  40  42  44  46]]

Shift by 128
[[130 129 128 126 124 122 120 119]
 [122 122 121 121 120 119 118 118]
 [120 120 120 120 120 119 119 119]
 [128 127 127 125 124 123 122 122]
 [136 135 133 131 128 126 124 123]
 [142 141 139 137 135 133 131 131]
 [150 150 150 151 151 151 152 152]
 [159 160 162 165 168 170 172 174]]
```




JPEG appears to compress well by reducing most components to zero, but visibly blurs the sharp edges in the original block.

**Problem 9**

```
LZ77
(0, 0, 'a')
(0, 0, 'b')
(0, 0, 'r')
(3, 1, 'c')
(2, 1, 'd')
(2, 1, 'b')
(0, 0, 'r')
(3, 1, 'a')
(3, 1, 'b')
(3, 1, 'd')
(2, 1, 'c')
(2, 1, 'r')
(0, 0, 'b')
(3, 1, None)

LZ78
(0, 'a')
(0, 'b')
(0, 'r')
(1, 'c')
(1, 'd')
(1, 'b')
(3, 'a')
(1, 'r')
(2, 'a')
(0, 'd')
(4, 'a')
(3, 'b')
(1, None)

LZW
[0, 1, 4, 0, 2, 0, 3, 5, 7, 0, 4, 1, 10, 8, 14, 16]
```

For LZ77, if multiple matches of the same length occur in the window, I'm taking the smallest offset (rightmost) match, which I believe is the match that the original paper used and could potentially be compressed more. The example on Wikipedia appears to take the opposite convention of taking the first (leftmost) match. Following that convention, you would get the following output.

```
(0, 0, 'a')
(0, 0, 'b')
(0, 0, 'r')
(3, 1, 'c')
(5, 1, 'd')
(4, 1, 'b')
(0, 0, 'r')
(5, 1, 'a')
```

```
(3, 1, 'b')
(4, 1, 'd')
(6, 1, 'c')
(4, 1, 'r')
(0, 0, 'b')
(5, 1, None)
```

**Problem 10**

We have enough information to reconstruct the numbers because we can find each number by adding the difference to the previous number $a_i = a_{i-1} + d_{i-1,i}$ and we have $a_0$ as the base case to start the chain.

```
bits used: 210014 k: 21
bits used: 200015 k: 20
bits used: 210014 k: 21
bits used: 200015 k: 20
bits used: 200015 k: 20
bits used: 210014 k: 21
bits used: 210014 k: 21
bits used: 210014 k: 21
bits used: 210014 k: 21
bits used: 210014 k: 21
```

Consider $m = 1$. The left half contains only 0 and 1, and is very easy to compress. We could just transmit the count of zeros and ones to recover the left half, because the entries are sorted. However, this scheme is very inefficient for the right half, since we would need 29 bits each.

We will extend the idea of transmitting counts of each possible left half. The optimal number of bins appears to be around $m = 13$ where the number of bins $2^{13} = 8192$ is approximately the data size 10000. Because the numbers are random, with high probability they contain a small number of entries and require only $\lceil \log_2 c \rceil$ bits each where $c$ is the maximum count of a bin. This is enough information to recover the left half because we have the run length of every possible left half in sequence. We can combine the left and right halves to recover all original numbers. For the case of random inputs, transmitting all $2^{13}$ counts turns out to be more efficient than transmitting pairs of (value, run length) because most of the $2^{13}$ possible values have non-zero counts.

The total number of bits required is $n(30-m)+2^m\lceil \log_2 c \rceil)$, plus or minus a few bits, depending on the exact implementation. You could transmit the bits per bin or some end-of-file symbol to recover the bits per bin. You could also get away with not transmitting one of the counts if you know the total data size.

```
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 202782 bits per bin: 4
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
bits used: 194590 bits per bin: 3
```

**Problem 1 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2

import subprocess
import resource
import os

#files to test
path = "./testfiles"
files = os.listdir(path)

#commands
commands = ["gzip {}", "ls -l *.gz", "gunzip *.gz",
            "bzip2 {}", "ls -l *.bz2", "bunzip2 *.bz2",
            "xz {}", "ls -l *.xz", "xz -d *.xz",
            "7z a temp.7z {} -m0=ppmd", "ls -l temp.7z", "7z e temp.7z -y; rm temp.7z"]

for testfile in files:
  results = [testfile]
  for command in commands:
    info = resource.getrusage(resource.RUSAGE_CHILDREN)
    start = info.ru_utime + info.ru_stime #add user and system time
    output = subprocess.check_output(command.format(testfile), cwd=path, shell=True).split()
    info = resource.getrusage(resource.RUSAGE_CHILDREN)
    end = info.ru_utime + info.ru_stime
    elapsed = end - start

    if command[:2] == "ls": #ls command, get size
      results.append(output[4])
    else: #time command
      results.append(str(elapsed))

  print ", ".join(results)
```

**Problem 5 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2

#generate all cyclic shifts
s = "wabbawabbawoo|"
n = len(s)
ss = s + s
shifts = [ss[i:i+n] for i in range(n)]
print "Shifts"
print "\n".join(shifts)

print "Sorted Shifts"
shifts.sort()
print "\n".join(shifts)

print "Last Column"
last = [s[-1] for s in shifts]
print "".join(last)
print last.index("|")

print "First Column"
first = [s[0] for s in shifts]
print "".join(first)

#move-to-front encode
alphabet = list(set(c for c in s))
alphabet.sort()

output = []
for c in last:
  i = alphabet.index(c)
  output.append(i)
  print alphabet, output
  #shift down
  while i:
    alphabet[i] = alphabet[i-1]
    i -= 1
  alphabet[0] = c


#print pairs of last, first
print "Pairs"
for l, f in zip(last, first):
  print l + f
```

**Problem 6 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2

def encode(text, intervals):
  n = len(text)
  window_start = 0.0
  window_end = 1.0
  for i in xrange(n):
    interval_start, interval_end = intervals[text[i]]
    new_start = window_start + (window_end - window_start) * interval_start
    new_end = window_start + (window_end - window_start) * interval_end
    window_start = new_start
    window_end = new_end
    print (window_start, window_end, i + 1)

def decode(code, n, intervals):
  result = []
  for i in xrange(n):
    for (char, (start, end)) in intervals.iteritems():
      if code >= start and code < end:
        result.append(char)
        code = (code - start) / (end - start)
        break
    print "".join(result), code #print result and rescaled code

intervals = {"a": (0.0, 0.2), "b": (0.2, 0.5), "c": (0.5, 1.0)}

print "Encode"
encode("aacbca", intervals)

print "Decode"
decode(0.63215699, 10, intervals)

#sanity check
#encode("cbbabacbbc", intervals)
```

### Problem 7 Code

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2
from collections import OrderedDict

#apply sequitur algorithm and print out steps
#note this is not a efficient (linear) implmentation - in each step it alternates applying
#digram uniqueness and rule utility constraints as many times as possible in a brute force mann
#as mentioned in the paper, multiple valid solutions satisfying the two constraints
#are possible for some inputs and this will only return one of them
#only allows up to rule R
def sequitur(text):
  rule_label_cnt = ord("A")
  rules = OrderedDict({"S": ""})
  for char in text:
    #add new character
    rules["S"] = rules["S"] + char
    print_step("Add character", char, rules)

    check_rules = True #flag to check rules again if any actually gets applied
    while check_rules:
      check_rules = False

      #enforce digram uniqueness
      duplicate_digram = find_duplicate_digram(rules)
      if duplicate_digram:
        check_rules = True
        #attempt to find existing rule
        new_label = None
        for label, rule in rules.iteritems():
          if rule == duplicate_digram:
            new_label = label
            break
        #create new rule
        if not new_label:
          new_label = chr(rule_label_cnt)
          rule_label_cnt += 1
        #apply rule
        for label, rule in rules.iteritems():
          rules[label] = rule.replace(duplicate_digram, new_label)
        rules[new_label] = duplicate_digram
        print_step("Enforce digram uniqueness for", duplicate_digram, rules)
        duplicate_digram = find_duplicate_digram(rules)

      #enforce rule utility
      unused_rule = find_unused_rule(rules)
      if unused_rule:
```

```
        check_rules = True
        #delete unused rule
        for label, rule in rules.iteritems():
          rules[label] = rule.replace(unused_rule, rules[unused_rule])
        del rules[unused_rule]
        print_step("Enforce rule utility for", unused_rule, rules)
        unused_rule = find_unused_rule(rules)

  return rules

#find digram occuring more than once
def find_duplicate_digram(rules):
  digrams = set()
  for rule in rules.values():
    for i in xrange(len(rule)-1):
      digram = rule[i:i+2]
      if digram in digrams:
        return digram
      else:
        digrams.add(digram)
  return None

#find label of rule occuring less than 2 times
def find_unused_rule(rules):
  used_chars = ""
  for rule in rules.values():
    used_chars += rule
  for label in rules.keys():
    if label != "S" and used_chars.count(label) < 2:
      return label
  return None

#print step
def print_step(action, obj, rules):
  print action, obj
  print ", ".join([label + ": " + rule for label, rule in rules.iteritems()])


sequitur("aactgaacatgagagacatagagacag")
print
sequitur("aactgaacatgagagacatagagacag"[::-1])

#example in paper
#sequitur("abcdbcabcd")
```

**Problem 8 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2
import numpy as np
import matplotlib.pyplot as plt
np.set_printoptions(precision=2, linewidth=150, suppress=True)

block = np.array([
  [124 ,125 ,122 ,120 ,122 ,119 ,117 ,118],
  [120 ,120 ,120 ,119 ,119 ,120 ,120 ,120],
  [125 ,124 ,123 ,122 ,121 ,120 ,119 ,118],
  [125 ,124 ,123 ,122 ,121 ,120 ,119 ,118],
  [130 ,131 ,132 ,133 ,134 ,130 ,126 ,122],
  [140 ,137 ,137 ,133 ,133 ,137 ,135 ,130],
  [150 ,147 ,150 ,150 ,150 ,150 ,150 ,150],
  [160 ,160 ,162 ,164 ,168 ,170 ,172 ,175]
])

quant = np.array([
  [16 ,11 ,10 ,16 ,24 ,40 ,51 ,61],
  [12 ,12 ,14 ,19 ,26 ,58 ,60 ,55],
  [14 ,13 ,16 ,24 ,40 ,57 ,69 ,56],
  [14 ,17 ,22 ,29 ,51 ,87 ,80 ,62],
  [18 ,22 ,37 ,56 ,68 ,109 ,103 ,77],
  [24 ,35 ,55 ,64 ,81 ,104 ,113 ,92],
  [49 ,64 ,78 ,87 ,103 ,121 ,120 ,101],
  [72 ,92 ,95 ,98 ,112 ,100 ,103 ,99]
])

#wikipedia example
wikiblock = np.array([
  [52, 55, 61, 66, 70, 61, 64, 73],
  [63, 59, 55, 90, 109, 85, 69, 72],
  [62, 59, 68, 113, 144, 104, 66, 73],
  [63, 58, 71, 122, 154, 106, 70, 69],
  [67, 61, 68, 104, 126, 88, 68, 70],
  [79, 65, 60, 70, 77, 68, 58, 75],
  [85, 71, 64, 59, 55, 61, 65, 83],
  [87, 79, 69, 68, 65, 76, 78, 94]
])


translate = block - 128

#discrete cosine transform of 8x8 block
def dct(g):
  result = np.empty((8, 8))
```

```
  for u in xrange(8):
    for v in xrange(8):
      total = 0
      for x in xrange(8):
        for y in xrange(8):
          total += g[x, y] * np.cos((2 * x + 1) * u * np.pi / 16) * np.cos((2 * y + 1) * v * np
      scale = 0.25
      if u == 0:
        scale *= 1 / np.sqrt(2)
      if v == 0:
        scale *= 1 / np.sqrt(2)
      result[u, v] = scale * total
  return result

def compress(block, quant):
  print "Compress"

  t = block - 128
  print "Shift by 128"
  print t
  print

  g = dct(t)
  print "DCT"
  print g
  print

  q = np.round(g / quant).astype(np.int)
  print "Quantized"
  print q
  print

  return q


def idct(f):
  result = np.empty((8, 8))
  for x in xrange(8):
    for y in xrange(8):
      total = 0
      for u in xrange(8):
        for v in xrange(8):
          scale = 0.25
          if u == 0:
            scale *= 1 / np.sqrt(2)
          if v == 0:
            scale *= 1 / np.sqrt(2)
          total += scale * f[u, v] * np.cos((2 * x + 1) * u * np.pi / 16) * np.cos((2 * y + 1)
```

```
      result[x, y] = total
  return result

def decompress(block, quant):
  print "Decompress"

  f = block * quant
  print "Unquantized"
  print f
  print

  t = np.round(idct(f)).astype(np.int)
  print "IDCT"
  print t
  print

  s = t + 128
  print "Shift by 128"
  print s
  print

  return s


def test_block(block, quant):
  compressed = decompress(compress(block, quant), quant)
  plt.figure(figsize=(10, 5))
  # plt.subplot(221)
  # plt.title("Original (Range from Min to Max)")
  # plt.imshow(block, cmap="Greys_r", interpolation="none")
  plt.subplot(121)
  plt.title("Original")
  plt.imshow(block, cmap="Greys_r", interpolation="none", vmin=0, vmax=255)
  # plt.subplot(223)
  # plt.title("Compressed and Decompressed")
  # plt.imshow(compressed, cmap="Greys_r", interpolation="none")
  plt.subplot(122)
  plt.title("Compressed and Decompressed")
  plt.imshow(compressed, cmap="Greys_r", interpolation="none", vmin=0, vmax=255)
  #plt.show()
  plt.savefig("jpegtest.png")

#test_block(wikiblock, quant)
test_block(block, quant)
```

**Problem 9 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2

def lz77(data, window_size, buffer_size):
  i = 0
  result = []
  while i < len(data):
    window_start = max(0, i - window_size)
    buffer_end = min(i + buffer_size, len(data))
    offset = 0
    length = 0
    next = data[i]
    #find match (for loop for simplicity: there are better substring matching algorithms)
    for j in xrange(i - 1, window_start - 1, -1):
    #for j in xrange(window_start, i):
      match_length = 0
      while i + match_length < buffer_end and data[j + match_length] == data[i + match_length]
        match_length = match_length + 1
      if match_length > length:
        length = match_length
        offset = i - j
        if i + match_length < len(data):
          next = data[i + match_length]
        else:
          next = None
    result.append((offset, length, next))
    i = i + length + 1
  return result

def lz78(data):
  next_index = 1
  dictionary = {"": 0}
  result = []
  i = 0
  while i < len(data):
    end = i
    #find longest match
    while end + 1 <= len(data) and data[i:end+1] in dictionary:
      end += 1
    match_index = dictionary[data[i:end]]
    next_char = None
    if end < len(data):
      #add to dictionary
      dictionary[data[i:end+1]] = next_index
      next_index += 1
      next_char = data[end]
```

```
    pair = (match_index, next_char)
    result.append(pair)
    # print pair
    # print dictionary
    # print
    i = end + 1
  return result

def lzw(data):
  next_index = 0
  dictionary = {}
  #add all characters
  for c in sorted(set(data)):
    dictionary[c] = next_index
    next_index += 1
  result = []
  i = 0
  while i < len(data):
    end = i
    #find longest match
    while end + 1 <= len(data) and data[i:end+1] in dictionary:
      end += 1
    match_index = dictionary[data[i:end]]
    if end < len(data):
      #add to dictionary
      dictionary[data[i:end+1]] = next_index
      next_index += 1
    #print dictionary
    #print match_index
    result.append((match_index))
    i = end
  return result


data = "abracadabraarbadacarba"
print "LZ77"
result = lz77(data, 6, 6)
for r in result:
  print r
print

print "LZ78"
result = lz78(data)
for r in result:
  print r
print

print "LZW"
```

```
print lzw(data)


#wikipedia examples
#print lz77("aacaacabcabaaac", 12, 9)
#print
#print lzw("TOBEORNOTTOBEORTOBEORNOT#ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

**Problem 10 Code**

```
#!/usr/bin/python
#Shawn Pan
#CS222 HW2
import numpy as np

#fix pseudorandom seed
np.random.seed(0)

#ceiling log 2
log2 = lambda x: int(np.ceil(np.log(x) / np.log(2)))

#create data sets
datasets = []
nsim = 10
n = 10000
for i in xrange(nsim):
  data = np.random.randint(0, 2**30, n)
  data.sort()
  datasets.append(data)

#original scheme
for data in datasets:
  diff = data[1:] - data[:-1]
  k = log2(np.max(diff))
  print "bits used:", 30 + 5 + k * (n - 1), "k:", k

#mask scheme
m = 13
bins = 2**m
mask_right = (1 << (30 - m)) - 1
mask_left = ((1 << 30) - 1) ^ mask_right

for data in datasets:
  right = np.bitwise_and(data, mask_right)
  left = np.bitwise_and(data, mask_left) >> (30 - m)

  counts = np.zeros(bins)
  for i in left:
    counts[i] += 1
    min_count = np.min(counts)
  bits_per_bin = log2(np.max(counts))
  print np.sum(counts==0)
  print "bits used:", n * (30 - m) + bins * bits_per_bin, "bits per bin:", bits_per_bin
```