0: Decompressing Huffman Codes

```
codes = {"00":32,"01000":100,"010010":109,"010011":119,"0101":97,"01100":46,"011010":44,"
    01101100":84,"011011010":83,"011011011":48,"0110111":73,"0111":111,"10000":104,"10001"
    :115,"10010":110,"1001100":33,"100110100":76,"100110101":122,"10011011":65,"100111":99,"
    10100":108,"10101":105,"1011":101,"11000":114,"110010000":79,"1100100010":113,"1100100011
    ":78,"11001001":72,"11001010":63,"11001011000":56,"11001011001":68,"11001011010":69,"
    11001011011":85,"11001011100":86,"11001011101":120,"11001011110":70,"11001011111":77,"
    1100110":39,"1100111":112,"110100":103,"11010100":89,"1101010100":49,"1101010101":106,"
    110101011":87,"1101011":98,"110110":121,"1101110":107,"1101111":102,"1110":10,"11110"
    :116,"111110":117,"11111100":118,"11111101":66,"1111111":45}

hex_msg = "9b9e77c22b2d0f38b4a1ba4e9c8a2a71e7de2fe557d57935dc1de85e2cac49389 \\
aec29aeefa307fa8835d8bae9679cdf4d99dc1cdfd113acb488b8b3cee4525a50f39a5fbabe8 \\
9bd7c52bef52cd6e8d87dfbcf42cd30ff490cee0e6c859aeeda1f793bfd88db46fa56e22cb64 \\
d76ee0ed773afd1b35ddc443cb3789d717c9c17c43e496513d0acb7158954b3bc635d74b6770 \\
76a5d28e9b46ba2cfb986a5d28e9b46ba2cfb99da974a3a6d1ae8b3ee61a974a3a6d1ae8b3ee \\
67707641e0d1ae8b3ee164836ba51d399d34bf6689182eeb2be3ece294afbd4b677077eae31b \\
4c3f716bbb7ac7c56262d51b4cee0ec83a5596933b83b2565a0f2148dcef2433b83b26e947ca \\
ee0efe7eae31b6577f26d0a96577077f320b2367f8d774adfe59e85625624173e7b9565a6577 \\
f1b93acb379837cd4a19eb3ee367f8fb3b3b83a68efbab2d08c1719d9dc1d96e367a1647cb5c \\
4586a3fd837afa178eceb541a3ba049e57b1bdf07a0f1b67707369f18d9837cc9179767afad0 \\
cee0ec9bc5e689e859a6151f2fad9dabbcd8b3f2f1b19f0ff207de6cff3445e4cee0e9b33de3 \\
7dcfe3179df1e275c21a16943f73459dc1d972f1b19f0ff219dc1d97d660dcd1fc29e8565a1a \\
3d65a10bc5b3b83bf9a8ff1a3f8a565e1e46cff60dff4d66acefe64139836417d9a225733b83 \\
bf9ab5fcb3ce7d1986aef35286bb2b230e99aa6a9961a6db6db6cefe7eed733b83bf5718d9a1 \\
b9e7a20d9fe6ba3e7ce6fa6d59685647a1641fe8dccee0efe64decd13685499dfcc9bd9a3f57 \\
18d99dc1dfc6f13d05f1bfe9acd4d2e995dfc9b295f7a96c1b59ee7aad042ecd1a6151f2faaf \\
2677076477f9783d07fb48786f99024bbacaf99dc1cd90c5d885da2698545919e0ef46bbd0c5 \\
d885da242ba408cf077a33b839b20f1b13bc58a9ee4d70867707364317621768 93bd297a5b06 \\
f99234a2a06e79df772885d85920857d3c215dd68170ff4907a1642bf96cee0ed47f845504ee \\
959ad67dc22bbf7de2e5e677077f326ad1fab8c6ccefe4de3d5b68d30e9d65a0a25f47cb3c2f \\
289a3bee89a3ba19dc1dfcc9b5c0badfef197b0b2dd5bcaefe6a5ac19dc1dfcd47f8d1eb2d0f \\
39e859bfe95e16995dfcc8dda1be64893f8d1eb2d19dc1d96bf2f1b6b746c6e93a716ba3eacb \\
422e95bcada367f8456d9dc1d965e59bc26b1f595f0f2148e45f5718ba339b5f4f01483fa16a \\
8677073734fae312164eff510417f2fb55fa3e347f7ae43fde1f79e8589aeccee0e6e51ff2cf \\
42b12b29df19df0beacb4e590a97d1b4ae5e33ae3715979c7fb021766770736417d9a2270d84 \\
ec43cb37895da0fc59d7d579459dc1dfafb34645f57d629aebe1f799dd2cec633be922f07a16 \\
4f5c4ff251f2ca7b8b3b83bf9abba51a26d0a93479d0bb09d8b8b12b933bf9abb171733b83bf \\
9fb77fd2b933bf93695ca6770764ad2974fb55584cee"

hex_msg_int = int(hex_msg, 16)
hex_msg_bin = bin(hex_msg_int)[2:]

key = ""
decoded_msg = ""

for char in hex_msg_bin:
   key += char
   if key in codes:
      decoded_msg += chr(codes[key])
      key = ""

print(decoded_message)
```

According to all known laws
of aviation,

there is no way a bee
should be able to fly.

Its wings are too small to get
its fat little body off the ground.

The bee, of course, flies anyway

because bees don’t care
what humans think is impossible.

Yellow, black. Yellow, black.
Yellow, black. Yellow, black.

Ooh, black and yellow!
Let’s shake it up a little.

Barry! Breakfast is ready!

Ooming!

Hang on a second.

Hello?

- Barry?
- Adam?

- Oan you believe this is happening?
- I can’t. I’ll pick you up.

Looking sharp.

Use the stairs. Your father
paid good money for those.

Sorry. I’m excited.

Here’s the graduate.
We’re very proud of you, son.

A perfect report card, all B’s.

Very proud.

Ma! I got a thing going here.

- You got lint on your fuzz.
- Ow! That’s me!

- Wave to us! We’ll be in row 118,000.
- Bye!

Barry, I told you,
stop flying in the house!

- Hey, Adam.
- Hey, Barry.

- Is that fuzz gel?
- A little. Special day, graduation.

Never thought I’d make it.

Three days grade school,
three days high school.

Those were awkward.

Three days college. I’m glad I took
a day and hitchhiked around the hive.

You did come back different.

- Hi, Barry.
- Artie, growing a mustache? Looks good.

- Hear about Frankie?
- Yeah.

- You going to the funeral?
- No, I’m not going.

Everybody knows,
sting someone, you die.

Don’t waste it on a squirrel.
Such a hothead.

I guess he could have
just gotten out of the way.

I love this incorporating
an amusement park into our day.

That’s why we don’t need vacations.

Boy, quite a bit of pomp...
under the circumstances.

- Well, Adam, today we are men.
- We are!

- Bee-men.
- Amen!

Hallelujah!

1: Lempel-Ziv

(a) We would combine Huffman with LZ by first performing LZ on a string and then applying Huffman coding to it. This might lead to better compression than using any one by itself because LZ reduces the repetition in the string while Huffman coding reduces the average number of symbols in the string. By first performing LZ, we are able to reduce the length of the string by condesing the repetition. Since we assign the repeated pharases to numbers, LZ returns just another string which can be compressed using Huffman. This would lead to a shorter string than applying just Huffman or LZ by itself.

(b) To minimize the number of phrases, we want to maximize the length of the phrases as having shorter phrases will increase our phrase count. By the construction of LZ, to create a new phrase with a greater length, we either repeat the phrase or we add one new character to the current phrase with the longest length. Put another way, in a string n, if the longest phrase has a length of k, then the only possible phrase longer than it has length of k + 1. Once LZ parses the string up to that phrase, the next longest phrase has length k + 2 and so on. As such, the string which minimizes the number of phrases is one where each phrase is one longer than the previous phrase.

An example of such a string is "aaaaaaaaaaaaaaa" which LZ will parse as "a|aa|aaa|aaaa|aaaaa". We see that each longest phrase is one longer than the previous longest phrase. The length of the string is the sum of length of the phrases from 1 to p where p is the number of phrases. We perform the following calculations:

$$\begin{aligned}
n &= \frac{p(p+1)}{2} && \text{[length of string by our construction of the minimal string]} \\
p^2 + p &= 2n \\
p^2 + p - 2n &= 0 \\
p &= \frac{-1 \pm \sqrt{1+8n}}{2} && \text{[quadratic formula]} \\
p &= \frac{\sqrt{1+8n} - 1}{2} && \text{[only consider principal root because number of phrases is non-negative]}
\end{aligned}$$

The string we presented above was an ideal example, but if our string ends in the "middle" of our last longest phrase, then we still want to count that last phrase. As such, our formula for the number of phrases changes to $\boxed{p = \frac{\lceil\sqrt{1+8n}\rceil - 1}{2}}$

(c) For any arbitrary compression scheme, it must be able to take all string of lenght n and map them to a unique strong shorter than n. This mapping must be injective to ensure that the original message is recoverable. We will show that such a mapping is not injective by showing the total number of strings with a lenght less than n is smaller than the number of strings with length n i.e. there aren't enough encoding strings available for an arbitary compression scheme.

$$\begin{aligned}
|\Sigma| &> 2 && \text{[given assumption]}\\
0 &< |\Sigma| - 2\\
0 &< |\Sigma|^n(|\Sigma| - 2)\\
0 &< |\Sigma|^n(|\Sigma| - 2) + |\Sigma|\\
0 &< |\Sigma|^{n+1} - 2 * |\Sigma|^n + |\Sigma|\\
|\Sigma|^n - 1 &< |\Sigma|^{n+1} + |\Sigma| - |\Sigma|^n - 1\\
|\Sigma|^n - 1 &< (|\Sigma| - 1)(|\Sigma|^n + 1)\\
\frac{|\Sigma|^n - 1}{|\Sigma| - 1} &< |\Sigma|^n + 1\\
\frac{1 - |\Sigma|^n}{1 - |\Sigma|} &< |\Sigma|^n + 1\\
\sum_{i=0}^{n-1} |\Sigma|^i &< |\Sigma|^n + 1 && \text{[summation of a finite geometric series]}\\
1 + |\Sigma|^1 + |\Sigma|^2 + \cdots + |\Sigma|^{n-1} &< |\Sigma|^n + 1\\
|\Sigma|^1 + |\Sigma|^2 + \cdots + |\Sigma|^{n-1} &< |\Sigma|^n
\end{aligned}$$

We see from the above result that there exist more strings of lenght n than strings of length less than n thus proving that no compression scheme can make every string shrink in length.

(d) To maximize the number of phrases, we have to generate as many unique phrases as possible. By the construction of LZ, we can acccomplish this by first writing all unique prefixes of lenght 1 followed by all unique prefixes of length 2 followed by all unique prefixes of length 3 and so on. By the above construction, one possible S_3 is as follows:

$$S_3 = 01011000110000010100111011101101111$$

$$S_3 = 0|1|01|10|00|11|000|001|010|011|101|110|110|111$$

(e) Let $P(k)$ be the statement "$|S_k| = (k-1)2^{k+1} + 2$" for $k > 1$. We prove $P(k)$ for all $k \in \mathbb{N}$ by induction on k. Before proceeding, we note to find the lenght of $S|k+1$, all we need to do is add the length of S_k to the lenght of all prefixes of length $k+1$. Since there are 2^{k+1} unique phrases each with length $k+1$, we can write that $S_{k+1} = |S_k| + (k+1)2^{k+1}$ Base Case: k = 1

$|S_1| = |01| = 2 = 0 + 2 = 0 * 2^{1+1}(1-1)2^{1+1} + 2$

Thus our base case holds.

Induction Hypothesis: Suppose $P(k)$ is true for some $k \in \mathbb{N}$.

$$\begin{aligned}
|S_{k+1}| &= |S_k| + (k+1)2^{k+1} && \text{[by explanation above]}\\
&= (k-1)2^{k+1} + 2 + (k+1)2^{k+1} && \text{[by I.H.]}\\
&= (k-1+k+1)2^{k+1} + 2\\
&= (2k)2^{k+1} + 2\\
&= k2^{k+2} + 2
\end{aligned}$$

So, the induction step holds and we have proven that $|S_k| = (k-1)2^{k+1} + 2$ by induction

(f) Let $P(k)$ be the statement "$c(S_k) = 2^{k+1} - 2$" for $k > 1$. We prove $P(k)$ for all $k \in \mathbb{N}$ by induction on k. Before proceeding, we note to find the the number of phrases in $S|k+1$, all we need to do is add the number of phrases in S_k to the number of phrases of length $k+1$. Since there are 2^{k+1} unique phrases with length $k+1$, we can write that $c(S_{k+1}) = c(S_k) + 2^{k+1}$ Base Case: k $= 1$

$|S_1| = |01| = 2 = 0 + 2 = 0 * 2^{1+1}(1-1)2^{1+1} + 2$

Thus our base case holds.

Induction Hypothesis: Suppose $P(k)$ is true for some $k \in \mathbb{N}$.

$$\begin{aligned}
c(S_{k+1}) &= c(S_k) + 2^{k+1} && \text{[by explanation above]}\\
&= 2^{k+1} - 2 + (k+1)2^{k+1} && \text{[by I.H.]}\\
&= 2 * 2^{k+1} - 2\\
&= 2^{k+2} + 2\\
&= k2^{k+2} + 2
\end{aligned}$$

So, the induction step holds and we have proven that $c(S_k) = 2^{k+1} - 2$ by induction

(g) We perform the following steps:

$$\begin{aligned} k &> 1 && \text{[by our assumption]} \\ k-1 > 0 \frac{2}{k-1} &> 0 > -2 \\ -2 &< \frac{2}{k-1} \\ 2^{k+1} - 2 &< 2^{k+1} + \frac{2}{k-1} \\ 2^{k+1} - 2 &< \end{aligned}$$

So, the induction step holds and we have proven that $c(S_k) = 2^{k+1} - 2$ by induction

(h)

(i)

(j)

(k)