0: Decompressing Huffman Codes

```python
def decode(msg, huff_dict):
  msg = msg.replace("\n", "")
  msg = bin(int(msg, 16))[2:]
  result = ""
  current = ""

  for bit in msg:
     current += bit
     if current in huff_dict:
        result += chr(huff_dict[current])
        current = ""

  return result

# Huffman Dictionary
huff_dict = {"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}

# Message
msg = """9b9e77c22b2d0f38b4a1ba4e9c8a2a71e7de2fe557d57935dc1de85e2cac49389aec29
aeefa307fa8835d8bae9679cdf4d99dc1cdfd113acb488b8b3cee4525a50f39a5fbabe
89bd7c52bef52cd6e8d87dfbcf42cd30ff490cee0e6c859aeeda1f793bfd88db46fa56
e22cb64d76ee0ed773afd1b35ddc443cb3789d717c9c17c43e496513d0acb7158954b3
bc635d74b677076a5d28e9b46ba2cfb986a5d28e9b46ba2cfb99da974a3a6d1ae8b3ee
61a974a3a6d1ae8b3ee67707641e0d1ae8b3ee164836ba51d399d34bf6689182eeb2be
3ece294afbd4b677077eae31b4c3f716bbb7ac7c56262d51b4cee0ec83a5596933b83b
2565a0f2148dcef2433b83b26e947caee0efe7eae31b6577f26d0a96577077f320b236
7f8d774adfe59e85625624173e7b9565a6577f1b93acb379837cd4a19eb3ee367f8fb3
b3b83a68efbab2d08c1719d9dc1d96e367a1647cb5c4586a3fd837afa178eceb541a3b
a049e57b1bdf07a0f1b67707369f18d9837cc9179767afad0cee0ec9bc5e689e859a61
51f2fad9dabbcd8b3f2f1b19f0ff207de6cff3445e4cee0e9b33de37dcfe3179df1e27
5c21a16943f73459dc1d972f1b19f0ff219dc1d97d660dcd1fc29e8565a1a3d65a10bc
5b3b83bf9a8ff1a3f8a565e1e46cff60dff4d66acefe64139836417d9a225733b83bf9
ab5fcb3ce7d1986aef35286bb2b230e99aa6a9961a6db6db6cefe7eed733b83bf5718d
9a1b9e7a20d9fe6ba3e7ce6fa6d59685647a1641fe8dccee0efe64decd13685499dfcc
9bd9a3f5718d99dc1dfc6f13d05f1bfe9acd4d2e995dfc9b295f7a96c1b59ee7aad042
ecd1a6151f2faaf2677076477f9783d07fb48786f99024bbacaf99dc1cd90c5d885da2
698545919e0ef46bbd0c5d885da242ba408cf077a33b839b20f1b13bc58a9ee4d70867
70736431762176893bd297a5b06f99234a2a06e79df772885d85920857d3c215dd6817
0ff4907a1642bf96cee0ed47f845504ee959ad67dc22bbf7de2e5e677077f326ad1fab
8c6ccefe4de3d5b68d30e9d65a0a25f47cb3c2f289a3bee89a3ba19dc1dfcc9b5c0bad
```

```
fef197b0b2dd5bcaefe6a5ac19dc1dfcd47f8d1eb2d0f39e859bfe95e16995dfcc8dda
1be64893f8d1eb2d19dc1d96bf2f1b6b746c6e93a716ba3eacb422e95bcada367f8456
d9dc1d965e59bc26b1f595f0f2148e45f5718ba339b5f4f01483fa16a8677073734fae
312164eff510417f2fb55fa3e347f7ae43fde1f79e8589aeccee0e6e51ff2cf42b12b2
9df19df0beacb4e590a97d1b4ae5e33ae3715979c7fb021766770736417d9a2270d84e
c43cb37895da0fc59d7d579459dc1dfafb34645f57d629aebe1f799dd2cec633be922f
07a164f5c4ff251f2ca7b8b3b83bf9abba51a26d0a93479d0bb09d8b8b12b933bf9abb
171733b83bf9fb77fd2b933bf93695ca6770764ad2974fb55584cee"""

# Decode
decoded = decode(msg, huff_dict)
print(decoded)
```

```
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) Combining Huffman Coding with LZ might lead to better compression to reduce redundancy and computation complexity. First, you would apply LZ to compress the input string into numbers. Using this string, we can then apply Huffman Coding by treating each of the numbers as unique characters. Using the Huffman Code, we can generate an even more compressed phrase that would require a mapping for all characters. This might lead to better compression because LZ can more efficiently compress repeated phrases while Huffman Coding is more efficient with unique elements.

(b) To find a string which minimizes the number of phrases the output could have, we must first prove a Lemma.
Lemma 1: We want to prove an upper bound for the length of the string S_n, and we claim that this upper bound is $\frac{k(k+1)}{2}$. The maximum length of string with k phrases is one where the length of each phrase is also maximized. This happens when each subsequent phrase is exactly one character longer than the previous phrase since LZ splits the string at the smallest unique prefix. Starting at length 1 and each subsequent string being 1 more than the previous, we have the total length defined by $1 + 2 + ... + k = \frac{k(k+1)}{2}$.

Let k be the number of phrases in S_n. k is at its minimum when S_n is a string containing only one character repeated ("aaa...", "bbb...", etc.). This is because LZ splits the string at the smallest unique prefix. Since the next phrase will always begin with the previous one, it minimizes the total number of phrases. By **Lemma 1**, we know that the upper bound for S_n is $\frac{k(k+1)}{2}$.

$$
\begin{aligned}
n &\leq \frac{k(k+1)}{2} && \text{Above}\\
2n &\leq k^2 + k && \text{Multiply both sides by 2}\\
2n + \frac{1}{4} &\leq k^2 + k + \frac{1}{4} && \text{Add } \frac{1}{4} \text{ to both sides}\\
2n + \frac{1}{4} &\leq (k + \frac{1}{2})^2 && \text{Simplification}\\
\sqrt{2n + \frac{1}{4}} &\leq k + \frac{1}{2} && \text{Square Root both sides}\\
\sqrt{2n + \frac{1}{4}} - \frac{1}{2} &\leq k && \text{Subtract } \frac{1}{2} \text{ from both sides}
\end{aligned}
$$

Thus, the minimum number of phrases for a string of length n is the smallest value for k such that $k \geq \sqrt{2n + \frac{1}{4}} - \frac{1}{2}$. It is minimal since it is the smallest value that also satisfies **Lemma 1**.

(c) Claim: For an arbitrary alphabet Σ with $|\Sigma| \geq 2$ and an arbitrary compression scheme, it is not possible to compress all strings of length n.

We will do this proof by contradiction. Let's assume that for an arbitrary alphabet $|\Sigma|$ and an arbitrary compression scheme, every string can be compressed. For all strings of length n, there are a total of $|\Sigma|^n$ strings. Since by definition a compressed string must have less characters than the original string, the output must have a length that is at most $n-1$. Therefore, adding all possible string lengths that are at most $n-1$, we have $|\Sigma|^1 + |\Sigma|^2 + ... + |\Sigma|^{n-1} = \sum_{i=1}^{n-1} |\Sigma|^{n-1}$ total outputs.

$$\begin{aligned}
\sum_{i=1}^{n-1} |\Sigma|^{n-1} &= \sum_{i=0}^{n-1} |\Sigma|^{n-1} - |\Sigma|^0 && \text{Subtract first term} \\
&= \frac{1-|\Sigma|^n}{1-|\Sigma|} - 1 && \text{Def. of Finite Geometric Series} \\
&= \frac{|\Sigma|^n - |\Sigma|}{|\Sigma| - 1} && \text{Algebra}
\end{aligned}$$

Since we know that $|\Sigma| \geq 2$, we know that $\frac{|\Sigma|^n - |\Sigma|}{|\Sigma|-1} \leq |\Sigma|^n - |\Sigma| < |\Sigma|^n$. Thus, we are left with the total number of possible compressions being less than the total possible strings ($|\Sigma|$). It follows from this that every possible string can no longer be represented by a unique compression. Since this is not possible, there is a contradiction. As a result, we have proven that for an arbitrary alphabet Σ with $|\Sigma| \geq 2$ and an arbitrary compression scheme, it is not possible to compress all strings of length n.

(d) We are given that $\Sigma = \{0, 1\}$. To find the string, S_k, in this set that maximizes the number of phrases LZ will use, it must be constructed such that it contains all possible phrases of length $\leq k$. Therefore, we can set S_k to be the combined string of all strings of length $n \in \{1, 2, ...k\}$. Using the alphabet that is given, the combined string for $n = 1$ is '01' ('0' + '1'). Similarly, the combined string for $n = 2$ is '00011011' ('00' + '01' + '10' + '11'). Thus, if k had been 2, S_k would be the combined string '01' + '00011011' = '0100011011'.

(e) From part (d) we defined S_k to be the combined string of all possible strings $\leq k$ with the alphabet $\Sigma = \{0, 1\}$. Given that the alphabet is the binary set, for each individual string of length i, we have 2^i total combinations. With k total strings in S_k, it follows that $|S_k| = \sum_{i=1}^{k} i \cdot 2^i$. To prove that $\sum_{i=1}^{k} i \cdot 2^i$ is equivalent to $(k-1)2^{k+1} + 2$, we will use induction.

Claim: $\sum_{i=1}^{k} i \cdot 2^i = (k-1)2^{k+1} + 2$.
Base Case (k=1):

$$\begin{aligned}
\sum_{i=1}^{k} i \cdot 2^i &= \sum_{i=1}^{1} i \cdot 2^i && \text{k = 1} \\
&= 1 \cdot 2^1 && \text{Simplifaction} \\
&= 2 && \text{Algebra} \\
&= (0)(2^2) + 2 && \text{Algebra} \\
&= (k-1)(2^{k+1}) + 2 && \text{k = 1}
\end{aligned}$$

Therefore, the base case holds.
Induction Hypothesis: Suppose the claim holds for all strings of length $n \geq 1$.

Induction Step: We must show that the claim holds for strings of length $n+1$.

$$\begin{aligned}
\sum_{i=1}^{k+1} i \cdot 2^i &= \sum_{i=1}^{k} i \cdot 2^i + [(k+1) \cdot 2^{k+1}] && \text{Add last term separately} \\
&= [(k-1)2^{k+1} + 2] + [(k+1) \cdot 2^{k+1}] && \text{By I.H.} \\
&= 2^{k+1}[(k-1) + (k+1)] + 2 && \text{Simplification} \\
&= 2^{k+1}[2k] + 2 && \text{Algebra} \\
&= k2^{k+2} + 2 && \text{Algebra}
\end{aligned}$$

This is exactly P(k+1). Therefore, since the base case and induction step hold, we have shown by induction that $\sum_{i=1}^{k} i \cdot 2^i = (k-1)2^{k+1} + 2$. Thus, $|S_k| = (k-1)2^{k+1} + 2$.

(f) In our definition for S_k, it is defined that S_k is the combination of all phrases of lengths $\leq k$. Given that the alphabet is $\Sigma = \{0, 1\}$, the total number of phrases for some length $i \leq k$ is 2^i since that is the total number of unique combinations using the alphabet. Therefore, the total number of phrases for S_k is the summation of all phrases $c(S_k) = \sum_{i=1}^{k} 2^i$.

$$\begin{aligned}
\sum_{i=1}^{k} 2^i &= \sum_{i=0}^{k} 2^i - 2^0 && \text{Subtract first term} \\
&= \frac{1 - 2^{k+1}}{1 - 2} - 1 && \text{Def. of Finite Geometric Series} \\
&= 2^{k+1} - 1 - 1 && \text{Algebra} \\
&= 2^{k+1} - 2 && \text{Algebra}
\end{aligned}$$

Thus, we have shown that $c(S_k) = 2^{k+1} - 2$.

(g) From part (e), we know that $|S_k| = (k-1)2^{k+1} + 2$ and $c(S_k) = 2^{k+1} - 2$. Plugging these values into the given inequality, we have

$$\begin{aligned}
2^{k+1} - 2 &\leq \frac{(k-1)2^{k+1} + 2}{k-1} && \text{Plug in for } c(S_k) \text{ and } |S - k| \\
&= 2^{k+1} + \frac{2}{k-1} && \text{Algebra}
\end{aligned}$$

Since it is given that $k > 1$, we know that the term $\frac{2}{k-1}$ will be positive. Thus, subtracting 2^{k+1} from both sides we are left with $-2 \leq$ [Some positive value]. This is true and therefore, $c(S_k) \leq \frac{|S_k|}{k-1}$ is also true.

(h) When adding some string A to pad, we know that it must be at least length $k+1$ by the definition of S_k. We also know that each phrase within this new string will be of length $k+1$. Therefore, the total number of phrases with the additional string A, $c(A)$, must be less than or equal to the length of A divided by the length of each additional phrase. Thus, $c(A) \leq \frac{|A|}{k+1}$.

(i)

$$\begin{aligned}\lg(c(Q)) &= \lg(c(S_k)+c(A)) \\ &\leq \lg(2^{k+1}-2+\frac{|A|}{k+1}) \qquad (2)\end{aligned}$$

We know that k is the largest natural number such that $|S_k| \leq n$. Since k is the largest number such that this is true, it follows that $|S_{k+1}| > n$. We know that $n = |S_k| + |A|$, so substituting this into the inequality, we have $|S_{k+1}| > |S_k| + |A|$. Subtracting $|S_k|$ from both sides and rearranging, we have $|A| < |S_{k+1}| - |S_k|$. From part (e), we can substitute values for $|S_n|$. Therefore $|A| < [(k)2^{k+2}+2] - [(k-1)2^{k+1}+2]$. We can use this to substitute back into Eq. 2 above.

$$\begin{aligned}
\lg(2^{k+1}-2+\frac{|A|}{k+1}) &\leq \lg(2^{k+1}-2+\frac{[(k)2^{k+2}+2]-[(k-1)2^{k+1}+2]|}{k+1}) && \text{Above}\\
&= \lg(2^{k+1}-2+\frac{k2^{k+2}-(k-1)2^{k+1}}{k+1}) && \text{Simplification}\\
&= \lg(\frac{(k+1)(2^{k+1})}{k+1}-\frac{2(k+1)}{k+1}+\frac{k2^{k+2}-(k-1)2^{k+1}}{k+1}) && \text{Algebra}\\
&= \lg(\frac{(2^{k+1}(2)-2(k+1)+k(2^{k+2})}{k+1}) && \text{Simplification}\\
&= \lg(\frac{2^{k+2}+k(2^{k+2})-2(k+1)}{k+1}) && \text{Simplification}\\
&= \lg(\frac{(k+1)(2^{k+2})-2(k+1)}{k+1}) && \text{Simplification}\\
&= \lg(2^{k+2}-2) && \text{Algebra}\\
&< \lg(2^{k+2}) && \text{-2} < 0\\
&= k+2 && \text{Algebra}
\end{aligned}$$

Therefore, we have shown that $\lg(c(Q)) \leq k+2$.

We are given that the maximum possible value of $c(Q)$ will occur when Q starts with S_k and finishes with some string A to pad. Therefore, the maximum possible value for $c(Q) = c(S_k) + c(A)$.

$$\begin{aligned}
c(Q) &= c(S_k)+c(A) && \text{Above}\\
&\leq \frac{|S_k|}{k-1}+\frac{|A|}{k+1} && \text{parts (g) and (h)}\\
&\leq \frac{|S_k|}{k-1}+\frac{|A|}{k-1} && \text{k} > 1\text{, Algebra}\\
&= \frac{|S_k|+|A|}{k-1} && \text{Algebra}\\
&= \frac{n}{k-1} && n = |S_k|+|A|\\
&\leq \frac{n}{\lg(c(Q))-3} && \lg(c(Q)) \leq k+2
\end{aligned}$$

(j) When LZ processes a string, it splits it and encodes it into binary. Since the encoding is in binary and the number of bits needed to encode each phrase depends on the total number of phrases, $c(Q)$, each phrase requires $\lg(c(Q))$ bits to encode the phrase in a unique way. As it processes the string, it already has some encoding associated with the prefix. The remainder of the string has a maximum length of 1 (since it is either 0 or 1 with the given $\Sigma = \{0, 1\}$. Thus, to process the unseen part of the string, it requires at most $\lg(c(Q)) + 1$ bits. Given that there are a total of $c(Q)$ phrases, the largest number of bits Q can be compressed to by LZ is $c(Q) \cdot (\lg(\mathrm{c(Q)}) + 1) = c(Q)\lg(\mathrm{c(Q)}) + c(Q)$

(k) We know that the largest number of bits that Q can be compressed to is $c(Q)\lg(c(Q)) + c(Q)$ from (j). We also know that $c(Q) \leq \frac{n}{\lg(c(Q))-3}$ from (i). Therefore, we know that the largest number of bits that Q can be compressed to is $\leq \frac{n}{(\lg(c(Q))-3}(\lg(c(Q)) + 1)$.

$$\begin{aligned}
\frac{n}{\lg(c(Q)) - 3}(\lg(c(Q)) + 1) &= \frac{n(\lg(c(Q)) - 3 + 4)}{\lg(c(Q)) - 3} && \text{Algebra} \\
&= n + \frac{4n}{\lg(c(Q)) - 3} && \text{Algebra} \\
&= n + \frac{4n}{\lg(c(Q)/8)} && \text{Simplification} \\
&\leq n + \frac{4n}{\lg(n)} && n >> c(Q)/8
\end{aligned}$$

Therefore, if we let $l = 4$ (some natural number), we have that the largest number of bits that Q can be compressed into is $n + l\frac{n}{\lg(n)}$.