Previous Next

271. Encode and Decode Strings

July 6, 2019 | 18.3K views

★★★★★
Average Rating: 4.50 (12 votes)

Design an algorithm to encode **a list of strings** to **a string**. The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

```
string encode(vector<string> strs) {
  // ... your code
  return encoded_string;
}
```

Machine 2 (receiver) has the function:

```
vector<string> decode(string s) {
  //... your code
  return strs;
}
```

So Machine 1 does:

```
string encoded_string = encode(strs);
```

and Machine 2 does:

```
vector<string> strs2 = decode(encoded_string);
```

`strs2` in Machine 2 should be the same as `strs` in Machine 1.

Implement the `encode` and `decode` methods.

Note:

- The string may contain any possible characters out of 256 valid ascii characters. Your algorithm should be generalized enough to work on any possible characters.
- Do not use class member/global/static variables to store states. Your encode and decode algorithms should be stateless.
- Do not rely on any library method such as `eval` or serialize methods. You should implement your own encode/decode algorithm.

Solution

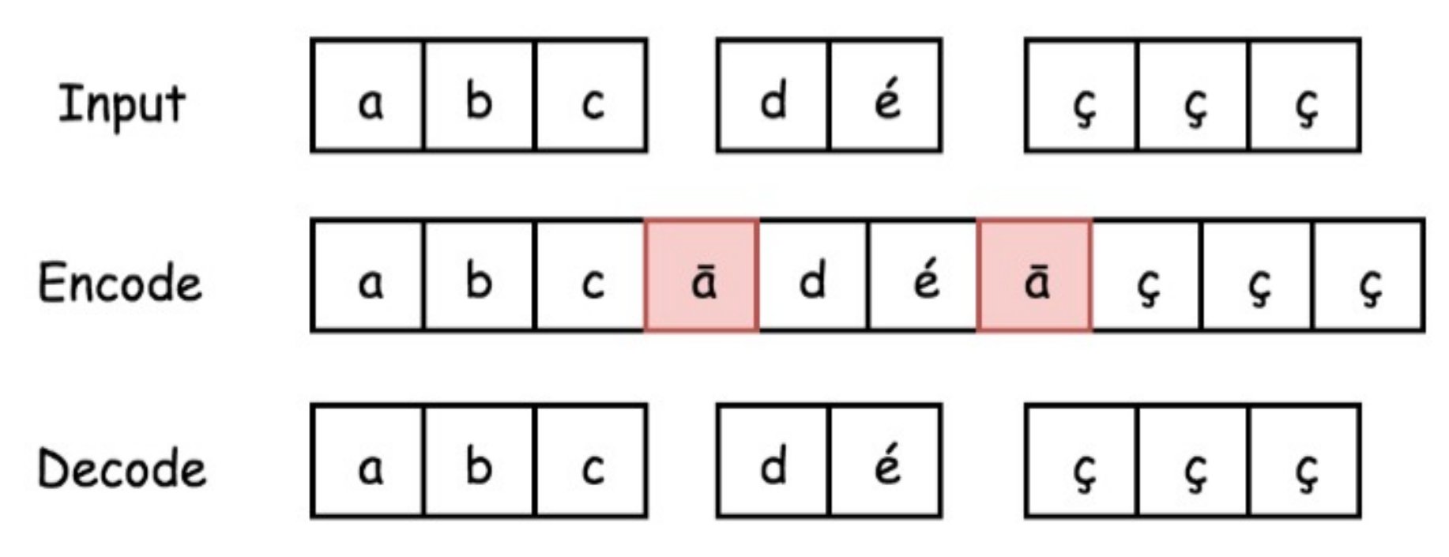
Approach 1: Non-ASCII Delimiter

Intuition

Naive solution here is to join strings using delimiters.

> What to use as a delimiter? Each string may contain any possible characters out of 256 valid ascii characters.

Seems like one has to use non-ASCII unichar character, for example `unichr(257)` in Python and `Character.toString((char)257)` in Java (it's character `ā`).



Here it's convenient to use two different non-ASCII characters, to distinguish between situations of "empty array" and of "array of empty strings".

Implementation

Use `split` in Java with a second argument `-1` to make it work as `split` in Python.

Java | Python | Copy

```
class Codec:
    def encode(self, strs):
        """Encodes a list of strings to a single string.
        :type strs: List[str]
        :rtype: str
        """
        if len(strs) == 0:
            return unichr(258)

        # encode here is a workaround to fix BE CodecDriver error
        return unichr(257).join(x.encode('utf-8') for x in strs)


    def decode(self, s):
        """Decodes a single string to a list of strings.
        :type s: str
        :rtype: List[str]
        """
        if s == unichr(258):
            return []
        return s.split(unichr(257))
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ both for encode and decode, where N is a number of strings in the input array.
- Space complexity : $\mathcal{O}(1)$ for encode to keep the output, since the output is one string. $\mathcal{O}(N)$ for decode keep the output, since the output is an array of strings.

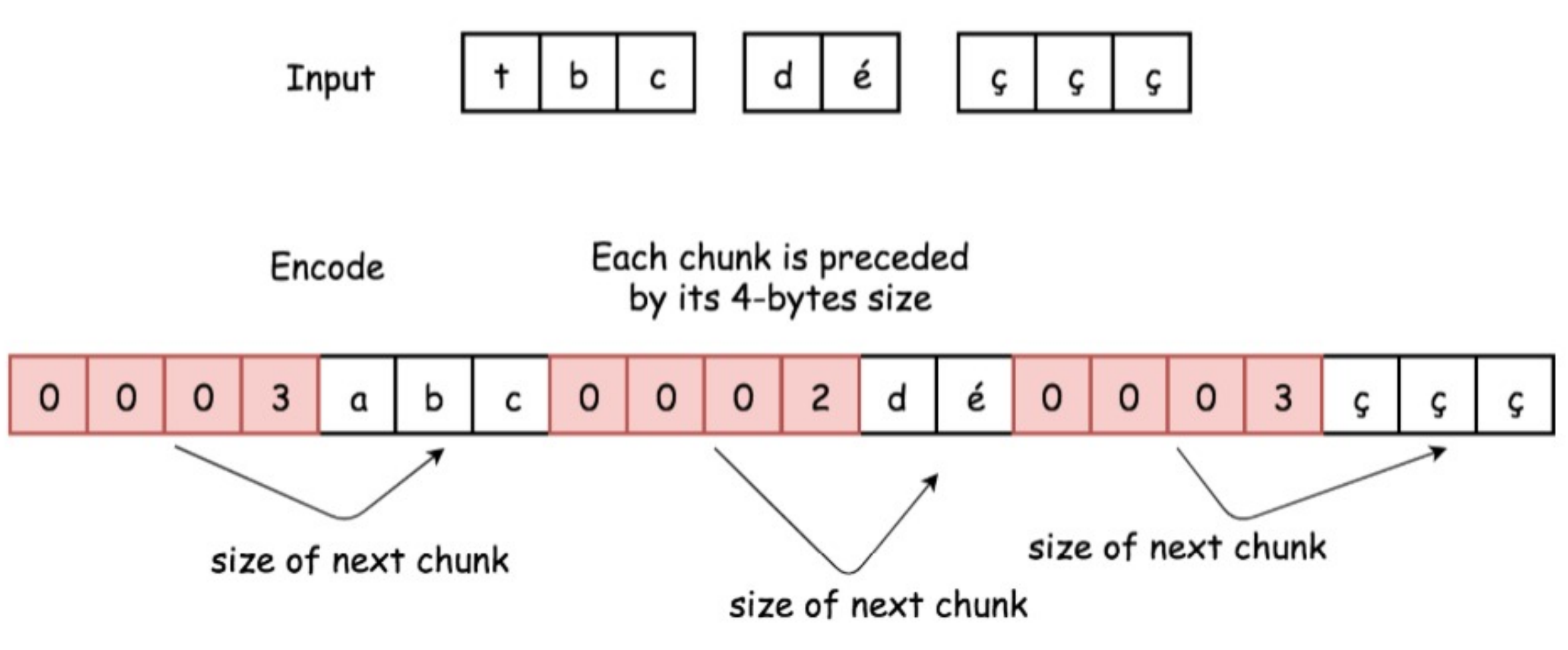
Approach 2: Chunked Transfer Encoding

Pay attention to this approach because last year Google likes to ask that sort of low-level optimisation. Serialize and deserialize BST problem is a similar example.

This approach is based on the encoding used in HTTP v1.1. It doesn't depend on the set of input characters, and hence is more versatile and effective than Approach 1.

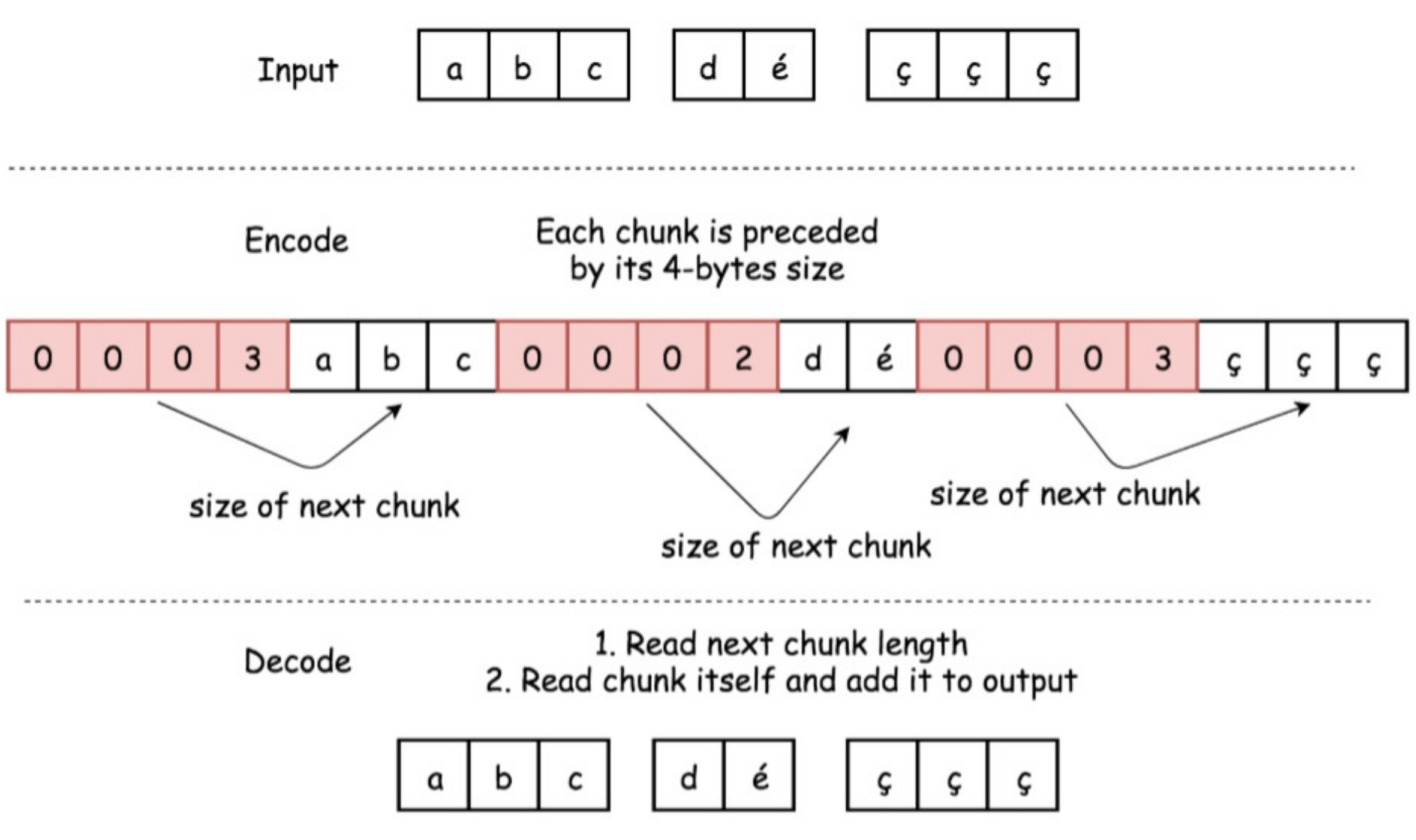
> Data stream is divided into chunks. Each chunk is preceded by its size in bytes.

Encoding Algorithm



- Iterate over the array of chunks, i.e. strings.
  - For each chunk compute its length, and convert that length into 4-bytes string.
  - Append to encoded string :
    - 4-bytes string with information about chunk size in bytes.
    - Chunk itself.
- Return encoded string.

Decoding Algorithm



- Iterate over the encoded string with a pointer `i` initiated as 0. While `i < n`:
  - Read 4 bytes `s[i: i + 4]`. It's chunk size in bytes. Convert this 4-bytes string to integer `length`.
  - Move the pointer by 4 bytes `i += 4`.
  - Append to the decoded array string `s[i: i + length]`.
  - Move the pointer by `length` bytes `i += length`.
- Return decoded array of strings.

Implementation

Java | Python | Copy

```
        :rtype: str
        """
        # encode here is a workaround to fix BE CodecDriver error
        return ''.join(self.len_to_str(x) + x.encode('utf-8') for x in strs)

    def str_to_int(self, bytes_str):
        """
        Decodes bytes string to integer.
        """
        result = 0
        for ch in bytes_str:
            result = result * 256 + ord(ch)
        return result

    def decode(self, s):
        """Decodes a single string to a list of strings.
        :type s: str
        :rtype: List[str]
        """
        i, n = 0, len(s)
        output = []
        while i < n:
            length = self.str_to_int(s[i: i + 4])
            i += 4
            output.append(s[i: i + length])
            i += length
        return output
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ both for encode and decode, where N is a number of strings in the input array.
- Space complexity : $\mathcal{O}(1)$ for encode to keep the output, since the output is one string. $\mathcal{O}(N)$ for decode keep the output, since the output is an array of strings.

Rate this article: ★★★★★

Previous Next

Comments: 20 Sort By

Type comment here... (Markdown is supported)

Preview Post

san_py ★ 249 February 29, 2020 10:22 PM Report

Understanding bytes = [chr(x >> (i * 8) & 0xff) for i in range(4)]

Read More

26 Share Reply

SHOW 2 REPLIES

jrhero ★ 9 September 9, 2019 2:39 AM

what if length of string exceeds the Integer.MAX_VALUE (which means 4 butes cannot hold the length)?

9 Share Reply

SHOW 3 REPLIES

willye ★ 878 September 9, 2019 11:32 AM

Chunked encoding is too clever for me... I dont think I can solve this in an interview lol

11 Share Reply

SHOW 2 REPLIES

xitrium ★ 3 November 19, 2019 12:29 AM Report

The space analysis is wrong here - just because a solution uses one string doesn't make it O(1) as strings have a variable size. Clearly the output of `encode` in both of these examples is linearly proportional to the size of the input.

3 Share Reply

cipone ★ 9 July 8, 2019 5:54 PM

HI, I am a little confused by this statement. **For each chunk compute its length, and convert that length into 4-bytes string.** It is actually a 8 bytes string since a char is 2 bytes (16 bits). And then the stored amount is sub-optimal, it uses 64 bits to store a 2**32 number.

4 Share Reply

SHOW 4 REPLIES

lenchen1112 ★ 1005 December 10, 2019 1:49 PM

Approach 1 by using escape character

```
class Codec:
    def encode(self, strs: [str]) -> str:
        return str(len(strs)) + '\t' + '\t'.join(strs)
```

Read More

2 Share Reply

Reputation

rocky-andre ★ 12 July 8, 2019 6:28 AM Report

what is "BE CodecDriver error? Python 3.* version do not have the issue atleast I do not see it. Also python 3.0 version uses `chr()` instead of `unichr`

2 Share Reply

SHOW 2 REPLIES

sin1080 ★ 44 July 6, 2019 9:49 PM

What about just implement the full HTTP Chunked Transfer Encoding and dump length as readable text representations. If you dump integer as bytes you can easily encounter endian issues between machines and things like strict pointer aliasing violation if you are using C/C++ and are not careful enough about dark corners of the language standards (E.g. if you used an int* to dereference data stored in a char* / string, you entered the land of undefined behaviour).

Read More

2 Share Reply

SHOW 4 REPLIES

m_2010 ★ 0 December 4, 2019 2:56 AM

Why no `Swift` implementation is possible?

0 Share Reply

leeteatsleep ★ 7 July 6, 2020 4:37 AM

Simpler Python Approach 1

```
class Codec:
    def encode(self, strs: [str]) -> str:
        return 'あ'.join(strs) if strs else None
```

Read More

0 Share Reply

‹ 1 2 ›