67. Add Binary

Oct. 27, 2019 | 47.5K views

Average Rating: 4.79 (34 votes)

Given two binary strings, return their sum (also a binary string).

The input strings are both **non-empty** and contains only characters `1` or `0`.

Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

Example 2:

```
Input: a = "1010", b = "1011"
Output: "10101"
```

Constraints:

- Each string consists only of `'0'` or `'1'` characters.
- `1 <= a.length, b.length <= 10^4`
- Each string is either `"0"` or doesn't contain any leading zero.

Solution

Overview

There is a simple way using built-in functions:

- Convert a and b into integers.
- Compute the sum.
- Convert the sum back into binary form.

Java | Python

```python
class Solution:
    def addBinary(self, a, b) -> str:
        return '{0:b}'.format(int(a, 2) + int(b, 2))
```

The overall algorithm has $\mathcal{O}(N + M)$ time complexity and has two drawbacks which could be used against you during the interview.

> 1 . In Java this approach is limited by the length of the input strings a and b. Once the string is long enough, the result of conversion into integers will not fit into Integer, Long or BigInteger:

- 33 1-bits - and b doesn't fit into Integer.
- 65 1-bits - and b doesn't fit into Long.
- 500000001 1-bits - and b doesn't fit into BigInteger.

To fix the issue, one could use standard Bit-by-Bit Computation approach which is suitable for quite long input strings.

> 2 . This method has quite low performance in the case of large input numbers.

One could use Bit Manipulation approach to speed up the solution.

Approach 1: Bit-by-Bit Computation

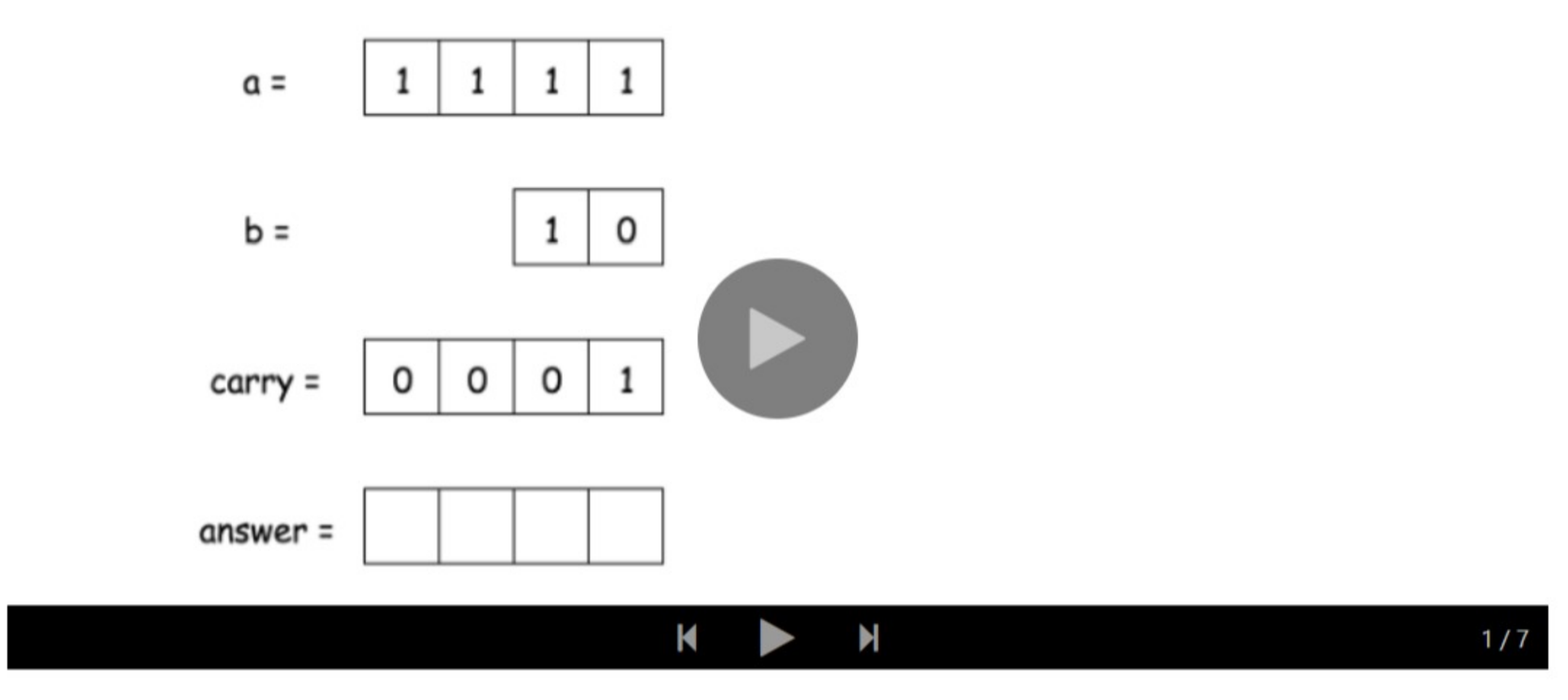
Algorithm

That's a good old classical algorithm, and there is no conversion from binary string to decimal and back here. Let's consider the numbers bit by bit starting from the lowest one and compute the carry this bit will add.

Start from carry = 0. If number a has 1-bit in this lowest bit, add 1 to the carry. The same for number b: if number b has 1-bit in the lowest bit, add 1 to the carry. At this point the carry for the lowest bit could be equal to $(00)_2$, $(01)_2$, or $(10)_2$.

Now append the lowest bit of the carry to the answer, and move the highest bit of the carry to the next order bit.

Repeat the same steps again, and again, till all bits in a and b are used up. If there is still nonzero carry to add, add it. Now reverse the answer string and the job is done.



Implementation

Java | Python

```python
class Solution:
    def addBinary(self, a, b) -> str:
        n = max(len(a), len(b))
        a, b = a.zfill(n), b.zfill(n)

        carry = 0
        answer = []
        for i in range(n - 1, -1, -1):
            if a[i] == '1':
                carry += 1
            if b[i] == '1':
                carry += 1

            if carry % 2 == 1:
                answer.append('1')
            else:
                answer.append('0')

            carry //= 2

        if carry == 1:
            answer.append('1')
        answer.reverse()

        return ''.join(answer)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(\max(N, M))$, where N and M are lengths of the input strings a and b.
- Space complexity: $\mathcal{O}(\max(N, M))$ to keep the answer.

Approach 2: Bit Manipulation

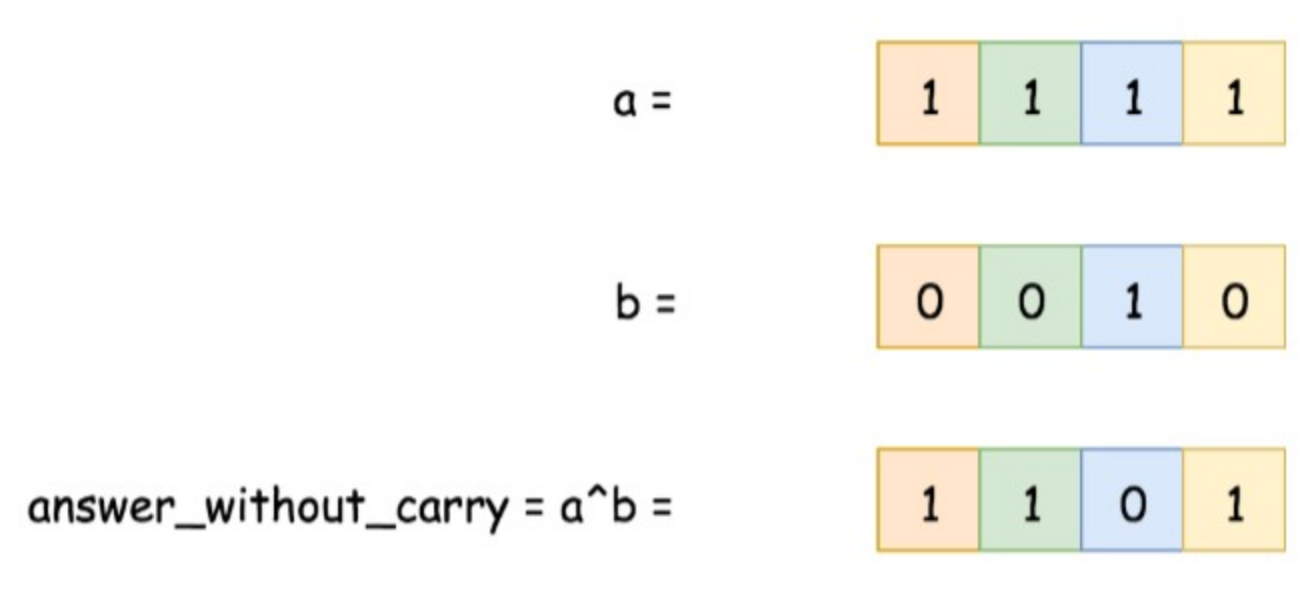
Intuition

Here the input is more adapted to push towards Approach 1, but there is popular Facebook variation of this problem when interviewer provides you two numbers and asks to sum them up without using addition operation.

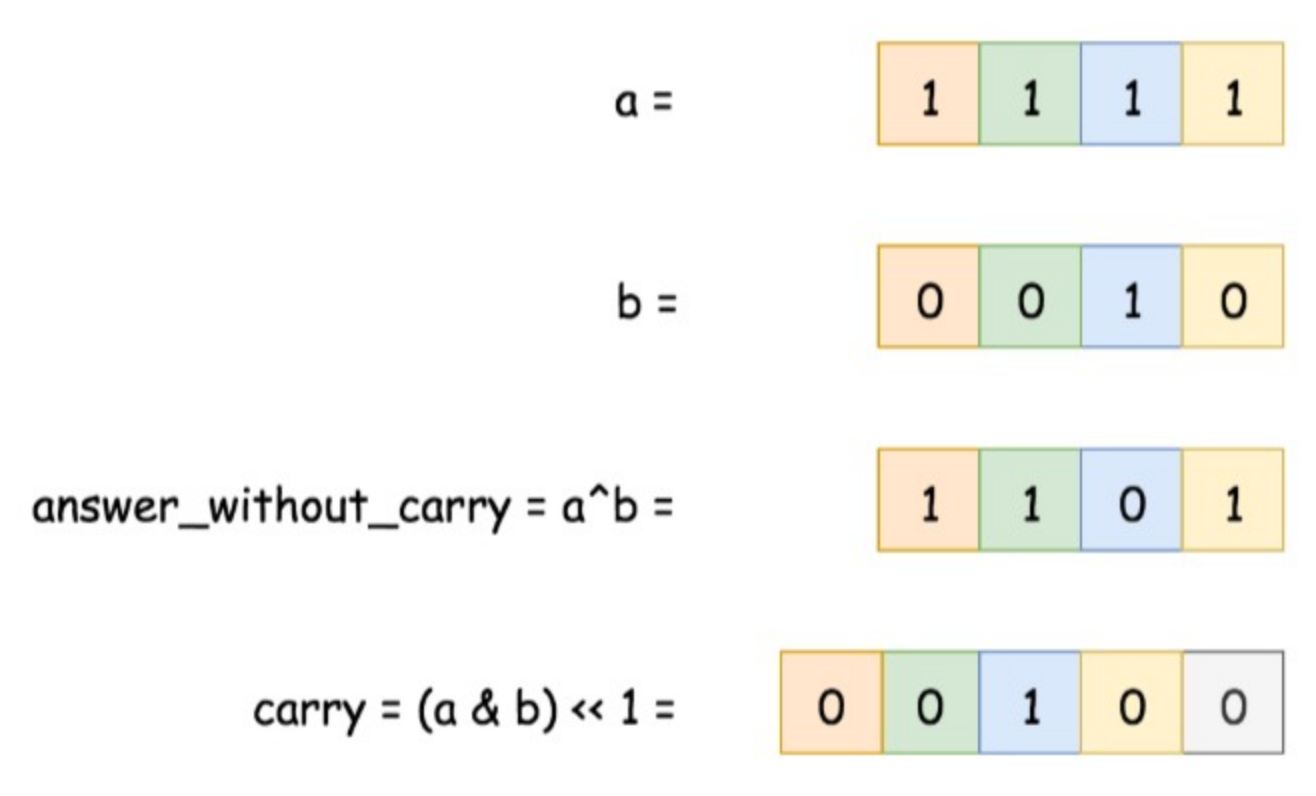
> No addition? OK, bit manipulation then.

How to start? There is an interview tip for bit manipulation problems: if you don't know how to start, start from computing XOR for your input data. Strangely, that helps to go out for quite a lot of problems, Single Number II, Single Number III, Maximum XOR of Two Numbers in an Array, Repeated DNA Sequences, Maximum Product of Word Lengths, etc.

Here XOR is a key as well, because it's a sum of two binaries without taking carry into account.



To find current carry is quite easy as well, it's AND of two input numbers, shifted one bit to the left.



Now the problem is reduced: one has to find the sum of answer without carry and carry. It's the same problem - to sum two numbers, and hence one could solve it in a loop with the condition statement "while carry is not equal to zero".

Algorithm

- Convert a and b into integers x and y, x will be used to keep an answer, and y for the carry.
- While carry is nonzero: `y != 0`:
  - Current answer without carry is XOR of x and y: `answer = x^y`.
  - Current carry is left-shifted AND of x and y: `carry = (x & y) << 1`.
  - Job is done, prepare the next loop: `x = answer`, `y = carry`.
- Return x in the binary form.

Implementation

Java | Python

```python
class Solution:
    def addBinary(self, a, b) -> str:
        x, y = int(a, 2), int(b, 2)
        while y:
            answer = x ^ y
            carry = (x & y) << 1
            x, y = answer, carry
        return bin(x)[2:]
```

This solution could be written as 4-liner in Python

Python

```python
class Solution:
    def addBinary(self, a, b) -> str:
        x, y = int(a, 2), int(b, 2)
        while y:
            x, y = x ^ y, (x & y) << 1
        return bin(x)[2:]
```

Performance Discussion

Here we deal with input numbers which are greater than 2^{100}. That forces to use slow BigInteger in Java, and hence the performance gain will be present for the Python solution only. Provided here Java solution could make its best with Integers or Longs, but not with BigIntegers.

Complexity Analysis

- Time complexity : $\mathcal{O}(N + M)$, where N and M are lengths of the input strings a and b.
- Space complexity : $\mathcal{O}(\max(N, M))$ to keep the answer.

Rate this article: ★ ★ ★ ★ ★

Previous Next

Comments: 16 Sort By

Type comment here... (Markdown is supported)

Preview Post

lsheng_mel ★167 March 4, 2020 4:48 PM

yeah I get it, the bit manipulation is so elegant. `answer = x^y` may not be too difficult to figure out, but how am i supposed to know that `carry = (x & y) << 1` during an interview? I feel depressed here :-(

20 Share Reply

SHOW 8 REPLIES

sivas2019prep ★8 April 15, 2020 8:23 AM

the parseInt approach does not work for this input "1010000010010011011001000001010111101101100110111011111111110100000010111100111000111100001101" "110101001011101110001111100110001010100001101011101010000011011011001011101111001100000011011110011"

Read More

8 Share Reply

SHOW 1 REPLY

YagaBaba ★8 October 28, 2019 1:22 AM

Thanks for the article. Really appreciate it.

8 Share Reply

mango999 ★15 November 27, 2019 12:42 AM

Could you please explain why the time complexity of the bit manipulation approach is O(N+M). Shouldn't it just be O(max(M,N)) because for any set of inputs, the maximum number time while loop runs is max(N,M)+1.

8 Share Reply

SHOW 3 REPLIES

tfc ★3 March 23, 2020 7:54 AM

In the Bit-by-Bit Computation Java solution isn't `int L = Math.max(n, m);` on line 5 redundant since line 4 ensures `n` is always the longest?

3 Share Reply

AdrienF ★12 April 30, 2020 5:05 PM

The approach 2 is missing the point of the question IMO. I know integers don't overflow in python but that's a trick I wouldn't use in an itw. As for the java solution, BigIntegers are 20 bytes long and it doesn't say in the questions the sum will fit in 20 bytes.

Here is my solution, I'm not overly happy with its readability but it doesn't rely on a trick or non given

Read More

1 Share Reply

nishadkumar ★97 April 29, 2020 12:55 AM

Neither liked the question nor the answer :(

1 Share Reply

jordan34 ★265 3 days ago

For python at least, you can use the `bin` method and it greatly simplifies the logic. It also has the same time complexity as the big shifting solution and *seems* just as fast. I'm not sure about overflow issues tho but I think python handles this gracefully. The interviewer might not accept that tho.

`return bin(int(a, 2) + int(b, 2))[2:]`

Read More

0 Share Reply

xc359 ★0 July 6, 2020 2:23 AM

my one line solution: return bin(int(a,2) + int(b,2))[2:]

0 Share Reply

Ai7pfRu ★0 July 1, 2020 3:57 PM

two line code using inbuilt library bin python3
class Solution:
def addBinary(self, a: str, b: str) -> str:

Read More

0 Share Reply

< 1 2 >