1980. Find Unique Binary String

Given an array of strings `nums` containing `n` **unique** binary strings each of length `n`, return *a binary string of length* `n` *that* ***does not appear*** *in* `nums`. *If there are multiple answers, you may return* ***any*** *of them.*

Example 1:

```
Input: nums = ["01","10"]
Output: "11"
Explanation: "11" does not appear in nums. "00" would also be correct.
```

Example 2:

```
Input: nums = ["00","01"]
Output: "11"
Explanation: "11" does not appear in nums. "10" would also be correct.
```

Example 3:

```
Input: nums = ["111","011","001"]
Output: "101"
Explanation: "101" does not appear in nums. "000", "010", "100", and "110" would also be correct.
```

Constraints:

- `n == nums.length`
- `1 <= n <= 16`
- `nums[i].length == n`
- `nums[i]` is either `'0'` or `'1'`.
- All the strings of `nums` are **unique**.

1980. Find Unique Binary String

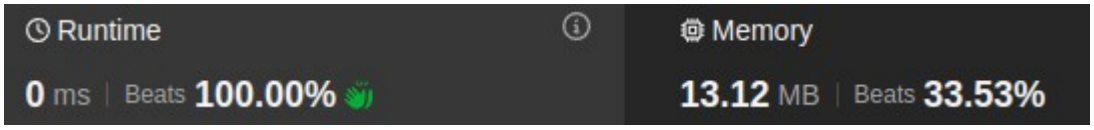
/*

Generate all binary strings

Time Complexity: $O(n+n2^n)=O(n.2^n)$

Space complexity: $O(2n)$

*/




```
class Solution {
public:
    std::string findDifferentBinaryString(std::vector<std::string>& nums) {
        int n=nums.size();

        // Mark all binary strings in nums
        std::unordered_map<std::string,bool> exists;
        for(auto& bin: nums) exists[bin]=true;
```

```
        // Function to generate al binary string from 0 to 2^n - 1
        auto generate=[&](int i,std::string& bin,std::string& ans,auto& self)->void{
            // If size of generated binary string is equal to n
            if(i==n){
                // Look it up, if does not exists
                if(!exists[bin]){
                    ans=bin; // Take it as an answer
                    return;
                }
                return;
            }

            // First assign "0" at i-th position
            // and try for all other permutations
            // for remaining positions
            bin[i]='0';
            self(i+1,bin,ans,self);

            // If we found an answer, stop the process
            if(!ans.empty()) return;

            // And then assign "1" at ith position
            // and try for all other permutations
            // for remaining positions
            bin[i]='1';
            self(i+1,bin,ans,self);

            // If we found an answer, stop the process
            if(!ans.empty()) return;
        };

        std::string bin(n,0),ans;
        generate(0,bin,ans,generate);

        return ans;
    }
};
```

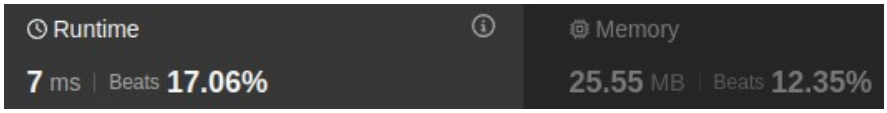
1980. Find Unique Binary String

/*

Convert to integers equivalents

Time Complexity: $O\left(n^2+\log_{10}\left(number\right)*2^n\right)$

Space complexity: $O\left(2^n\right)$



*/

```
class Solution {
   public:
      std::string findDifferentBinaryString(std::vector<std::string>& nums) {
         int n=nums.size();

         // 2^n−1 : maximum number of n bits
         int m=1<<n;

         // Array of size to 2^n , to store numbers from 0 to 2^n−1
         std::vector<int> exists(m,0);

         // For each binary string bin
         for(auto& bin: nums){
            // Converted to its equivalent decimal number
            int p=1<<(n-1);
            int number=0;
            for(auto& bit: bin){
               number+=(bit=='1'?1:0)*p;
               p/=2;
            }

            // Mark that number as exists
            exists[number]=1;
         }
```

```
        // For each number from 0 to 2^n-1
        for(int number=0;number<m;++number){
            // If it does not exist
            if(!exists[number]){
                // Converted to a binary string
                std::string ans(n,'0');
                int i=n-1;
                while(number!=0){
                    ans[i--]=number%2+'0';
                    number/=2;
                }

                // Return it
                return ans;
            }
        }

        return ""; // Never reached, because it exists at most 1 answer
    }
};
```

1980. Find Unique Binary String

Approach 4: Cantor's Diagonal Argument

Intuition

Cantor's diagonal argumentis a proof in set theory.

While we do not need to fully understand the proof and its consequences, this approach uses very similar ideas.

We start by initializing the answer ans to an empty string. To build ans, we need to assign either $'0'$ or $'1'$ to each index i for indices 0 to $n-1$. How do we assign them such that ans is guaranteed to be different from every string in $nums$? We know that two strings are different, as long as they differ by at least one character. We can intentionally construct our ans based on this fact.

For each index i, we will check the $i-th$ character of the $i-th$ string in $nums$. That is, we check $nums[i][i]$. We then assign $ans[i]$ to the opposite of $nums[i][i]$. That is, if $nums[i][i]=='0'$, we assign $ans[i]='1'$. If $nums[i][i]=='1'$, we assign $ans[i]='0'$.

What is the point of this strategy? ans will differ from every string in **at least** one position. More specifically:

- ans differs from $nums[0]$ in $nums[0][0]$.
- ans differs from $nums[1]$ in $nums[1][1]$.
- ans differs from $nums[2]$ in $nums[2][2]$.
- ...
- ans differs from $nums[n-1]$ in $nums[n-1][n-1]$.

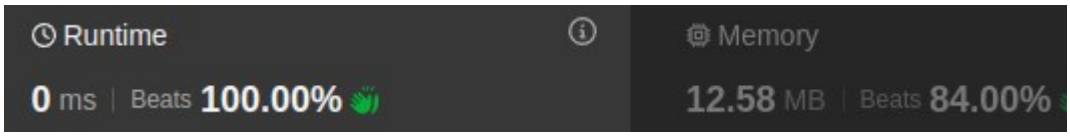
Thus, it is guaranteed that ans does not appear in $nums$ and is a valid answer.

> This strategy is applicable because both the length of ans and the length of each string in $nums$ are larger than or equal to n, the number of strings in $nums$. Therefore, we can find one unique position for each string in $nums$.

Algorithm

1. Initialize the answer ans . Note that you should build the answer in an efficient manner according to the programming language you're using.
2. Iterate i over the indices of `nums`:
    - If $nums[i][i]=='0'$, add $‘1’$ to ans . Otherwise, add $‘0’$ to ans .
3. Return ans .

```
/*
   ***Cantor's diagonal argument***
   Time complexity: O(n)
   Space complexity: O(1)
*/
```




```
class Solution {
public:
   std::string findDifferentBinaryString(std::vector<std::string>& nums) {
      int n=nums.size();
      std::string ans;
      for(int i=0;i<n;++i){
         ans+=nums[i][i]=='1'?'0':'1';
      }
      return ans;
   }
};
```