题目描述

给一个数组，数组里面哦都是代表非负整数的字符串，将数组里所有的数值 排列组合 拼接起来组成一个数字，输出拼接成的最小的数字。

输入描述

一个数组，数组不为空，数组里面都是代表非负整数的字符串，可以是0开头，例如：["13", "045", "09", "56"]。

数组的大小范围：[1, 50]

数组中每个元素的长度范围：[1, 30]

输出描述

以字符串的格式输出一个数字，

- 如果最终结果是多位数字，要优先选择输出不是“0”开头的最小数字
- 如果拼接出来的数字都是“0”开头，则选取值最小的，并且把开头部分的“0”都去掉再输出
- 如果是单位数“0”，可以直接输出“0”

用例

输入	20 1
输出	120
说明	无

输入	08 10 2
输出	10082
说明	无

输入	01 02
输出	102
说明	无

题目解析

~~简单的全排列求解。~~

~~在求解全排列前，我们需要对输入的数字字符串数组进行字典序排序，这样所有数字字符串就会从小到大排列，如果不考虑前导0的特殊处理的话，此时拼接出来的数值就是最小的。~~

~~但是本题需要考虑前导0，因此按字典序排序后，还需要求解全排列，得到全排列后，将拼接字符串分为两类：含前导0的，和不含前导0的。~~

~~如果存在不含前导0的，则直接输出第一个。~~

~~如果不存在不含前导0的，则输出含前导0的第一个，但是按照题目要求，需要去除前导0，这里可以直接将字符串转为数字即可，比如Java的可以用Integer.parseInt，比如JS可以用Number函数或者parseInt。~~

> 2023.02.03 经网友指正，本题如果使用 全排列 求解的话会超时。原因是本题：数组的大小范围：[1, 50]，如果求解全排列的话，则有50!种排列，这是一个巨大的数量，因此会超时。

一般来说，如果数组中元素长度都相同的话，则可以直接将数组进行字典序升序，这样数组元素按顺序拼接得到的组合数就是最小，比如数组[45, 32, 11, 31] 按照字典序升序后变为 [11, 31, 32, 45]，然后进行拼接，得到的组合数为11313245就是最小的。

但是，如果数组中元素长度不全相同的话，则此时直接字典序升序，可能无法得到最小组合数，比如数组 [3, 32, 321]，按照字典序升序后变为 [3, 32, 321]，然后进行拼接，得到组合数332321，但是这个组合数显然不是最小的，最小的组合数应该是 321323。

而本题数组元素的长度是有可能不一样的，此时我们应该采用另一种排序规则：

即直接尝试对要组合的两个字符串a，b进行组合，此时有两种组合方式：

题目解析

~~简单的全排列求解。~~

~~在求解全排列前，我们需要对输入的数字字符串数组进行字典序排序，这样所有数字字符串就会从小到大排列，如果不考虑前导0的特殊处理的话，此时拼接出来的数值就是最小的。~~

~~但是本题需要考虑前导0，因此按字典序排序后，还需要求解全排列，得到全排列后，将拼接字符串分为两类：含前导0的，和不含前导0的。~~

~~如果存在不含前导0的，则直接输出第一个。~~

~~如果不存在不含前导0的，则输出含前导0的第一个，但是按照题目要求，需要去除前导0，这里可以直接将字符串转为数字即可，比如Java的可以用Integer.parseInt，比如JS可以用Number函数或者parseInt。~~

> 2023.02.03 经网友指正，本题如果使用 全排列 求解的话会超时。原因是本题：数组的大小范围：[1, 50]，如果求解全排列的话，则有50!种排列，这是一个巨大的数量，因此会超时。

一般来说，如果数组中元素长度都相同的话，则可以直接将数组进行字典序升序，这样数组元素按顺序拼接得到的组合数就是最小，比如数组[45, 32, 11, 31] 按照字典序升序后变为 [11, 31, 32, 45]，然后进行拼接，得到的组合数为11313245就是最小的。

但是，如果数组中元素长度不全相同的话，则此时直接字典序升序，可能无法得到最小组合数，比如数组 [3, 32, 321]，按照字典序升序后变为 [3, 32, 321]，然后进行拼接，得到组合数332321，但是这个组合数显然不是最小的，最小的组合数应该是 321323。

而本题数组元素的长度是有可能不一样的，此时我们应该采用另一种排序规则：

即直接尝试对要组合的两个字符串a，b进行组合，此时有两种组合方式：

1. a + b
2. b + a

然后，比较 (a + b) 和 （b+a）的数值谁小，如果a+b的数值更小，则保持当前a,b顺序不变，如果b+a的数值更小，则交换a,b位置。

比如 strs = [a, b]， a = "3"，b = "32"

a + b = "332"

b + a = "323"

可以发现 b+a更小，因此交换a,b位置，strs变为[32, 3]，然后进行元素拼接得到 323 最小组合数。

另外，本题，还有其他限定规则：

- 如果最终结果是多位数字，要优先选择输出不是“0”开头的最小数字
- 如果拼接出来的数字都是“0”开头，则选取值最小的，并且把开头部分的“0”都去掉再输出

即排序后，数组开头元素如果以“0”开头，此时分两种情况讨论：

- 数组全部元素都是以“0”开头，则此时应该将拼接后的组合数去除前导0
- 数组有不以“0”开头的元素，则此时不应该把“0”开头的元素放在数组头部。

我的解题思路如下：

首先，按照前面的规则将数组元素排序，排序后，检查数组头元素是否以“0”开头，如果是的话，则开始遍历数组后面的元素，直到找到一个不以“0”开头的元素x，然后将元素x取出，并插入到数组头部。如果一直找不到这样的x，则说明数组元素全部是以0开头的，此时直接拼接出组合数，然后去除前导0。

关于去除前导0，这里不建议使用parseInt，或者python的int，因为对应组合数非常有可能超出int范围，这里我们应该保持组合数为字符串，实现去除前导0。

对于Java,JS而言，可以是正则表达式 /^0+/ 来替换前导0为空字串。

对于Python而言，可以使用lstrip方法来去除左边0

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const strs = line.split(" ");
  console.log(getResult(strs));
});

function getResult(strs) {
  strs.sort((a, b) => {
    const s1 = a + b;
    const s2 = b + a;
    return s1 == s2 ? 0 : s1 > s2 ? 1 : -1;
  });

  if (strs[0][0] == "0") {
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const strs = line.split(" ");
  console.log(getResult(strs));
});

function getResult(strs) {
  strs.sort((a, b) => {
    const s1 = a + b;
    const s2 = b + a;
    return s1 == s2 ? 0 : s1 > s2 ? 1 : -1;
  });

  if (strs[0][0] == "0") {
    for (let i = 1; i < strs.length; i++) {
      if (strs[i][0] != "0") {
        strs[0] = strs[i] + strs[0];
        strs[i] = "";
        break;
      }
    }
  }

  return strs.join("").replace(/^0+/, "");
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String[] strs = sc.nextLine().split(" ");
    System.out.println(getResult(strs));
  }

  public static String getResult(String[] strs) {
    Arrays.sort(strs, (a, b) -> (a + b).compareTo(b + a));

    if (strs[0].charAt(0) == '0') {
      for (int i = 1; i < strs.length; i++) {
        if (strs[i].charAt(0) != '0') {
          strs[0] = strs[i] + strs[0];
          strs[i] = "";
          break;
        }
      }
    }

    StringBuilder sb = new StringBuilder();
    for (String str : strs) {
      sb.append(str);
    }

    return sb.toString().replaceAll("^0+", "");
  }
}
```

Python算法源码

```python
import functools

# 输入获取
strs = input().split()


# 算法入口
def cmp(a, b):
    s1 = a + b
    s2 = b + a
    return 0 if s1 == s2 else 1 if s1 > s2 else -1


def getResult(strs):
    strs.sort(key=functools.cmp_to_key(cmp))

    if strs[0][0] == '0':
        for i in range(1, len(strs)):
            if strs[i][0] != '0':
```

Python算法源码

```python
import functools

# 输入获取
strs = input().split()


# 算法入口
def cmp(a, b):
    s1 = a + b
    s2 = b + a
    return 0 if s1 == s2 else 1 if s1 > s2 else -1


def getResult(strs):
    strs.sort(key=functools.cmp_to_key(cmp))

    if strs[0][0] == '0':
        for i in range(1, len(strs)):
            if strs[i][0] != '0':
                strs[0] = strs[i] + strs[0]
                strs[i] = ""
                break

    return "".join(strs).lstrip("0")


# 算法调用
print(getResult(strs))
```