

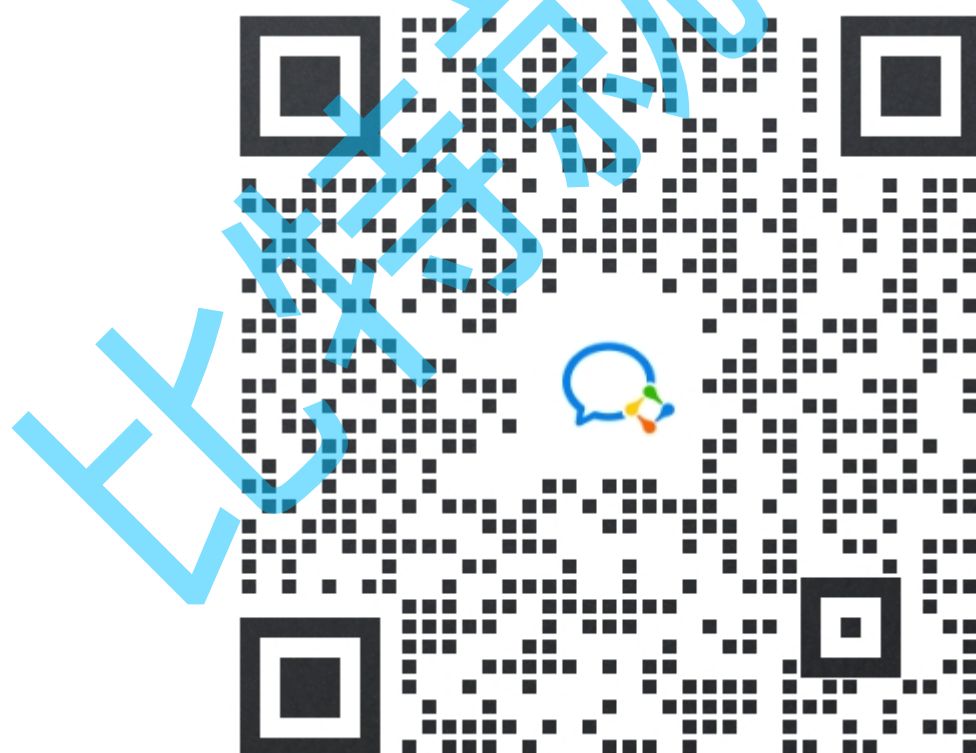
# 笔试强训第 01 周

## 版权说明

### 版权说明

本“**比特就业课**”笔试强训第 01 周（以下简称“本笔试强训”）的所有内容，包括但不限于文字、图片、音频、视频、软件、程序、数据库、设计、布局、界面等，均由本笔试强训的开发者或授权方拥有版权。我们鼓励个人学习者使用本笔试强训进行学习和研究。在遵守相关法律法规的前提下，个人学习者可以下载、浏览、学习本笔试强训的内容，并为了个人学习、研究或教学目的而使用其中的材料。但请注意，**未经我们明确授权，个人学习者不得将本笔试强训的内容用于任何商业目的**，包括但不限于销售、转让、许可或以其他方式从中获利。此外，个人学习者也不得擅自修改、复制、传播、展示、表演或制作本笔试强训内容的衍生作品。任何未经授权的使用均属侵权行为，我们将依法追究法律责任。如果您希望以其他方式使用本笔试强训的内容，包括但不限于引用、转载、摘录、改编等，请事先与我们联系，获取书面授权。感谢您对“比特就业课”笔试强训第 01 周的关注与支持，我们将持续努力，为您提供更好的学习体验。特此说明。比特就业课版权所有方。

对比特算法感兴趣，可以联系这个微信。



## 板书链接

<https://gitee.com/wu-xiaozhe/written-exam-training>

# Day01

## 1. 数字统计（数学 + 模拟）

（题号：170397）

1. 题目链接：[BC153 \[NOIP2010\]数字统计](#)

2. 题目描述：

题目描述：请统计某个给定范围[L, R]的所有整数中，数字2出现的次数。

比如给定范围[2, 22]，数字2在数2中出现了1次，在数12中出现1次，在数20中出现1次，在数21中出现1次，在数22中出现2次，所以数字2在该范围内一共出现了6次。

输入描述：输入共1行，为两个正整数L和R，之间用一个空格隔开。

输出描述：输出共1行，表示数字2出现的次数。

补充说明：1≤L≤R≤10000。

示例1

输入：2 22

输出：6

说明：

示例2

输入：2 100

输出：20

说明：

3. 解法：

算法思路：

常规操作：

循环提取末尾，然后干掉末尾~

C++ 算法代码：

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int l, r;
8     cin >> l >> r;
9
10    int ret = 0;
11    for(int i = l; i <= r; i++)
```

```

12     {
13         int tmp = i;
14         while(tmp)
15         {
16             if(tmp % 10 == 2) ret++;
17             tmp /= 10;
18         }
19     }
20     cout << ret << endl;
21
22     return 0;
23 }

```

Java 算法代码:

```

1  import java.util.Scanner;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9          int l = in.nextInt(), r = in.nextInt();
10
11          int ret = 0;
12          for(int i = l; i <= r; i++)
13          {
14              int tmp = i;
15              while(tmp != 0)
16              {
17                  if(tmp % 10 == 2) ret++;
18                  tmp /= 10;
19              }
20          }
21          System.out.println(ret);
22      }
23 }

```

## 2. 两个数组的交集 (哈希)

(题号: 2381092)

1. 题目链接: [NC313 两个数组的交集](#)

2. 题目描述:

题目描述: 给定两个整数数组分别为 $nums1$ ,  $nums2$ , 找到它们的公共元素并按返回。

数据范围:

$1 \leq nums1.length, nums2.length \leq 1000$

$1 \leq nums1[i], nums2[i] \leq 1000$

补充说明:

示例1

输入:  $[1, 2], [2, 2, 2, 2]$

输出:  $[2]$

说明: 两个数组的公共元素只有2

示例2

输入:  $[1, 2, 3], [8, 2, 2, 3, 8]$

输出:  $[2, 3]$

说明: 两个数组的公共元素为2和3, 返回 $[3, 2]$ 也是一个正确的答案

3. 解法:

- 将其中一个数组丢进哈希表中;
- 遍历另一个数组的时候, 在哈希表中看看就好了。

C++ 算法代码:

```
1 class Solution
2 {
3     bool hash[1010] = { 0 };
4
5 public:
6     vector<int> intersection(vector<int>& nums1, vector<int>& nums2)
7     {
8         vector<int> ret;
9         for(auto x : nums1)
10         {
11             hash[x] = true;
12         }
13
14         for(auto x : nums2)
15         {
```

```

16         if(hash[x])
17         {
18             ret.push_back(x);
19             hash[x] = false;
20         }
21     }
22
23     return ret;
24 }
25 };

```

## Java 算法代码：

```

1  import java.util.*;
2
3  public class Solution
4  {
5      public ArrayList<Integer> intersection (ArrayList<Integer> nums1,
        ArrayList<Integer> nums2)
6      {
7          boolean[] hash = new boolean[1010];
8
9          for(int x : nums1)
10         {
11             hash[x] = true;
12         }
13
14         ArrayList<Integer> ret = new ArrayList<>();
15         for(int x : nums2)
16         {
17             if(hash[x])
18             {
19                 ret.add(x);
20                 hash[x] = false;
21             }
22         }
23         return ret;
24     }
25 }

```

### 3. 点击消除（栈）

(题号：952218)

#### 1. 题目链接：[AB5 点击消除](#)

#### 2. 题目描述：

题目描述：牛牛拿到了一个字符串。

他每次“点击”，可以把字符串中相邻两个相同字母消除，例如，字符串“abbc”点击后可以生成“ac”。

但相同而不相邻、不相同的相邻字母都是不可以被消除的。

牛牛想把字符串变得尽可能短。他想知道，当他点击了足够多次之后，字符串的最终形态是什么？

输入描述：一个字符串，仅由小写字母组成。（字符串长度不大于300000）

输出描述：一个字符串，为“点击消除”后的最终形态。若最终的字符串为空串，则输出0。

补充说明：

示例1

输入：abbc

输出：ac

说明：

示例2

输入：abba

输出：0

说明：

示例3

输入：bbbbbb

输出：b

说明：

#### 3. 解法：

##### 算法思路：

用栈来模拟消除的过程。

##### C++ 算法代码：

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main()
7 {
8     string s, st;
9     cin >> s;
10
11     for(auto ch : s)
```

```

12     {
13         if(st.size() && st.back() == ch) st.pop_back();
14         else st += ch;
15     }
16
17     cout << (st.size() == 0 ? "0" : st) << endl;
18
19     return 0;
20 }

```

## Java 算法代码:

```

1  import java.util.Scanner;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9          char[] s = in.next().toCharArray();
10
11          StringBuilder st = new StringBuilder();
12          for(int i = 0; i < s.length; i++)
13          {
14              char ch = s[i];
15              if(st.length() != 0 && ch == st.charAt(st.length() - 1))
16              {
17                  // 出栈
18                  st.deleteCharAt(st.length() - 1);
19              }
20              else
21              {
22                  // 进栈
23                  st.append(ch);
24              }
25          }
26
27          System.out.println(st.length() == 0 ? 0 : st.toString());
28      }
29 }

```

## Day02

### 1. 牛牛的快递（模拟）

（题号：2368469）

1. 题目链接：[BC64 牛牛的快递](#)

2. 题目描述：

题目描述：牛牛正在寄快递，他了解到快递在 1kg 以内的按起步价 20 元计算，超出部分按每 kg 1元计算，不足 1kg 部分按 1kg计算。如果加急的话要额外付五元，请问牛牛总共要支付多少快递费

输入描述：第一行输入一个单精度浮点数 a 和一个字符 b，a 表示牛牛要寄的快递的重量，b表示牛牛是否选择加急，'y' 表示加急，'n' 表示不加急。

输出描述：输出牛牛总共要支付的快递费用

补充说明：

示例1

输入：1.5 y

输出：26

说明：

示例2

输入：0.7 n

输出：20

说明：

3. 解法：

算法思路：

模拟：分情况讨论即可。

扩展两个库函数：`ceil` 和 `floor`（天花板和地板）

C++ 算法代码：

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main()
6 {
7     double a;
8     char b;
9     cin >> a >> b;
10
11     int ret = 0;
```



```

12     if(a <= 1)
13     {
14         ret += 20;
15     }
16     else
17     {
18         ret += 20;
19         a -= 1;
20         ret += ceil(a);
21     }
22
23     if(b == 'y') ret += 5;
24
25     cout << ret << endl;
26
27     return 0;
28 }

```

## Java 算法代码:

```

1  import java.util.Scanner;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9          double a = in.nextDouble();
10         char b = in.next().charAt(0);
11
12         int ret = 0;
13         if(a <= 1)
14         {
15             ret = 20;
16         }
17         else
18         {
19             ret = 20 + (int)Math.ceil(a - 1);
20         }
21         if(b == 'y') ret += 5;
22
23         System.out.println(ret);
24     }

```

## 2. 最小花费爬楼梯（动态规划 - 线性 dp）

（题号：2366449）

### 1. 题目链接：DP4 最小花费爬楼梯

### 2. 题目描述：

题目描述：给定一个整数数组  $cost$ ，其中  $cost[i]$  是从楼梯第  $i$  个台阶向上爬需要支付的费用，下标从 0 开始。一旦你支付此费用，即可选择向上爬一个或者两个台阶。

你可以选择从下标为 0 或下标为 1 的台阶开始爬楼梯。

请你计算并返回达到楼梯顶部的最低花费。

数据范围：数组长度满足  $1 \leq n \leq 10^5$ ，数组中的值满足  $1 \leq cost_i \leq 10^4$

输入描述：第一行输入一个正整数  $n$ ，表示数组  $cost$  的长度。

第二行输入  $n$  个正整数，表示数组  $cost$  的值。

输出描述：输出最低花费

补充说明：

示例1

输入：3

2 5 20

输出：5

说明：你将从下标为1的台阶开始，支付5，向上爬两个台阶，到达楼梯顶部。总花费为5

### 3. 解法：

#### 算法思路：

简单线性 dp。

#### C++ 算法代码：

```
1 #include <iostream>
2 using namespace std;
3
4 const int N = 1e5 + 10;
5
6 int n;
7 int cost[N];
8 int dp[N];
9
10 int main()
11 {
```

```

12     cin >> n;
13     for(int i = 0; i < n; i++) cin >> cost[i];
14
15     for(int i = 2; i <= n; i++)
16     {
17         dp[i] = min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);
18     }
19
20     cout << dp[n] << endl;
21
22     return 0;
23 }

```

### Java 算法代码:

```

1  import java.util.Scanner;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9          int n = in.nextInt();
10         int[] cost = new int[n];
11         int[] dp = new int[n + 1];
12
13         for(int i = 0; i < n; i++)
14         {
15             cost[i] = in.nextInt();
16         }
17
18         for(int i = 2; i <= n; i++)
19         {
20             dp[i] = Math.min(dp[i - 1] + cost[i - 1], dp[i - 2] + cost[i - 2]);
21         }
22
23         System.out.println(dp[n]);
24     }
25 }

```

### 3. 数组中两个字符串的最小距离（模拟 + 贪心）

（题号：343447）

1. 题目链接：[\[编程题\]数组中两个字符串的最小距离](#)

2. 题目描述：

题目描述：给定给定两个字符串str1和str2，再一个字符串数组strs，返回在strs中str1和str2的最小距离，如果str1或str2为null，或不在strs中，返回-1。

输入描述：输入包含有多行，第一输入一个整数 $n$ （ $1 \leq n \leq 10^5$ ），代表数组strs的长度，第二行有两个字符串分别代表str1和str2，接下来 $n$ 行，每行一个字符串，代表数组strs (保证题目中出现的所有字符串长度均小于等于10)。

输出描述：输出一行，包含一个整数，代表返回的值。

补充说明：时间复杂度 $O(n)$ ，额外空间复杂度 $O(1)$

示例1

输入：1

CD AB

CD

输出：-1

说明：strs数组为["CD"]，由于"AB"不在strs里，返回-1

示例2

输入：5

QWER 666

QWER

1234

qwe

666

QWER

输出：1

说明：strs数组为["QWER","1234","qwe","666","QWER"]，其中"QWER"和"666"的最短距离为1（选择第四个和第五个字符串）

3. 解法：

算法思路：

小贪心，或者是小dp：

- 用 `prev1` 标记 `i` 位置之前最近一次出现的第一个字符串的下标；
- 用 `prev2` 标记 `i` 位置之前最近一次出现的第二个字符串的下标。

C++ 算法代码：

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main()
7 {
8     int n;
```

```

9     string s1, s2;
10    string s;
11
12    cin >> n;
13    cin >> s1 >> s2;
14
15    int prev1 = -1, prev2 = -1, ret = 0x3f3f3f3f;
16    for(int i = 0; i < n; i++)
17    {
18        cin >> s;
19        if(s == s1) // 去前面找最近的 s2
20        {
21            if(prev2 != -1)
22            {
23                ret = min(ret, i - prev2);
24            }
25            prev1 = i;
26        }
27        else if(s == s2) // 去前面找 s1
28        {
29            if(prev1 != -1)
30            {
31                ret = min(ret, i - prev1);
32            }
33            prev2 = i;
34        }
35    }
36
37    if(ret == 0x3f3f3f3f) cout << -1 << endl;
38    else cout << ret << endl;
39
40    return 0;
41 }

```

## Java 算法代码：

```

1  import java.util.*;
2  import java.io.*;
3
4  // 注意类名必须为 Main, 不要有任何 package xxx 信息
5  public class Main
6  {
7      public static void main(String[] args) throws Throwable
8      {

```

```

9      BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
10     int n = Integer.parseInt(reader.readLine());
11     String[] str = reader.readLine().split(" ");
12     String s1 = str[0], s2 = str[1];
13
14     int prev1 = -1, prev2 = -1, ret = 0x3f3f3f3f;
15     for(int i = 0; i < n; i++)
16     {
17         String s = reader.readLine();
18         if(s.equals(s1)) // 去前面找最近的 s2
19         {
20             if(prev2 != -1)
21             {
22                 ret = Math.min(ret, i - prev2);
23             }
24             prev1 = i;
25         }
26         else if(s.equals(s2)) // 去前面找最近的 s1
27         {
28             if(prev1 != -1)
29             {
30                 ret = Math.min(ret, i - prev1);
31             }
32             prev2 = i;
33         }
34     }
35
36     System.out.println(ret == 0x3f3f3f3f ? -1 : ret);
37 }
38 }

```

## Day03

### 1. 简写单词 (模拟)

(题号: 989791)

1. 题目链接: [BC149 简写单词](#)

2. 题目描述:

题目描述: 规定一种对于复合词的简写方式为只保留每个组成单词的首字母, 并将首字母大写后再连接在一起  
比如 "College English Test" 可以简写成 "CET", "Computer Science" 可以简写为 "CS", "I am Bob" 简写为 "IAB"  
输入一个长复合词 (组成单词数  $sum, sum \geq 1$  且  $sum \leq 100$ , 每个单词长度  $len, len \geq 1$  且  $len \leq 50$ ), 请你输出它的简写

输入描述: 输入一个复合词

输出描述: 输出一行, 表示复合词的简写

补充说明:

示例1

输入: College English Test

输出: CET

说明:

### 3. 解法:

#### 算法思路:

简单模拟题, 主要是处理一下输入的问题。

#### C++ 算法代码:

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main()
7 {
8     string s;
9     while(cin >> s) // 自动跳过空格
10    {
11        if(s[0] <= 'z' && s[0] >= 'a') cout << (char)(s[0] - 32);
12        else cout << s[0];
13    }
14
15    return 0;
16 }
```

#### Java 算法代码:

```
1 import java.util.Scanner;
2
3 // 注意类名必须为 Main, 不要有任何 package xxx 信息
4 public class Main
```

```

5 {
6     public static void main(String[] args)
7     {
8         Scanner in = new Scanner(System.in);
9
10        while(in.hasNext())
11        {
12            char ch = in.next().charAt(0);
13            if(ch >= 'a' && ch <= 'z') System.out.print((char)(ch - 32));
14            else System.out.print(ch);
15        }
16    }
17 }

```

## 2. dd爱框框（滑动窗口）

(题号: 1698733)

### 1. 题目链接: [dd爱框框](#)

### 2. 题目描述:

题目描述: 读入 $n$ ,  $x$ , 给出 $n$ 个数 $a[1], a[2], \dots, a[n]$ , 求最小的区间 $[l, r]$ , 使 $a[l] + a[l+1] + \dots + a[r] \geq x$ , 若存在相同长度区间, 输出 $l$ 最小的那个

输入描述: 第一行两个数,  $n(1 \leq n \leq 100000000), x(1 \leq x \leq 10000)$   
第二行 $n$ 个数 $a[i](1 \leq a[i] \leq 1000)$

输出描述: 输出符合条件 $l, r$ (保证有解)

补充说明:

示例1

输入: 10 20

1 1 6 10 9 3 3 5 3 7

输出: 3 5

说明:

### 3. 解法:

#### 算法思路:

基础同向双指针算法。

#### C++ 算法代码:

```

1 #include <iostream>
2
3 using namespace std;

```



```

4
5 const int N = 1e7 + 10;
6
7 int arr[N];
8 int n, x;
9
10 int main()
11 {
12     cin >> n >> x;
13     for(int i = 1; i <= n; i++) cin >> arr[i];
14
15     int left = 0, right = 0, sum = 0;
16     int retLen = N, retLeft = -1, retRight = -1;
17
18     while(right <= n)
19     {
20         sum += arr[right];
21         while(sum >= x)
22         {
23             if(right - left + 1 < retLen)
24             {
25                 retLeft = left;
26                 retRight = right;
27                 retLen = right - left + 1;
28             }
29             sum -= arr[left++];
30         }
31         right++;
32     }
33
34     cout << retLeft << " " << retRight << endl;
35
36     return 0;
37 }

```

## Java 算法代码:

```

1 import java.util.*;
2 import java.io.*;
3
4 public class Main
5 {
6     public static void main(String[] args) throws IOException
7     {

```

```

8      Read in = new Read();
9      int n = in.nextInt(), x = in.nextInt();
10     int[] arr = new int[n + 1];
11     for(int i = 1; i <= n; i++)
12     {
13         arr[i] = in.nextInt();
14     }
15
16     int left = 1, right = 1, sum = 0;
17     int retLeft = -1, retRight = -1, retLen = n;
18
19     while(right <= n)
20     {
21         // 进窗口
22         sum += arr[right];
23         while(sum >= x)
24         {
25             // 更新结果
26             if(right - left + 1 < retLen)
27             {
28                 retLeft = left;
29                 retRight = right;
30                 retLen = right - left + 1;
31             }
32             sum -= arr[left++];
33         }
34         right++;
35     }
36
37     System.out.println(retLeft + " " + retRight);
38 }
39 }
40
41 class Read // 自定义快读 Read
42 {
43     StringTokenizer st = new StringTokenizer("");
44     BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));
45     String next() throws IOException
46     {
47         while(!st.hasMoreTokens())
48         {
49             st = new StringTokenizer(bf.readLine());
50         }
51         return st.nextToken();
52     }
53
54     String nextLine() throws IOException

```

```

55     {
56         return bf.readLine();
57     }
58
59     int nextInt() throws IOException
60     {
61         return Integer.parseInt(next());
62     }
63
64     long nextLong() throws IOException
65     {
66         return Long.parseLong(next());
67     }
68
69     double nextDouble() throws IOException
70     {
71         return Double.parseDouble(next());
72     }
73 }

```

### 3. 除2! (贪心 + 堆)

(题号: 1089261)

#### 1. 题目链接: [除2!](#)

#### 2. 题目描述:

题目描述: 给一个数组, 一共有  $n$  个数。  
 你能进行最多  $k$  次操作。每次操作可以进行以下步骤:

- 选择数组中的一个偶数  $a_i$ , 将其变成  $a_i/2$ 。

现在你进行不超过  $k$  次操作后, 让数组中所有数之和尽可能小。请输出这个最小的和。

输入描述: 第一行输入两个正整数  $n$  和  $k$ , 用空格隔开  
 第二行输入  $n$  个正整数  $a_i$   
 数据范围:

$$1 \leq n \leq 100000, \quad 1 \leq k \leq 10^9$$

$$1 \leq a_i \leq 10^9$$

输出描述: 一个正整数, 代表和的最小值。

补充说明:

示例1

输入: 5 3

2 4 8 10 11

输出: 24

说明: 对8操作2次, 对10操作1次, 最后的数组是2 4 2 5 11。可以证明这样的操作是最优的。

### 3. 解法:

#### 算法思路:

搞一个堆模拟一下就好了~

#### C++ 算法代码:

```
1  #include <iostream>
2  #include <queue>
3
4  using namespace std;
5
6  typedef long long LL;
7
8  LL n, k;
9  priority_queue<LL> heap;
10
11 int main()
12 {
13     cin >> n >> k;
14     LL sum = 0, x;
15
16     while(n--)
17     {
18         cin >> x;
19         sum += x;
20         if(x % 2 == 0) heap.push(x);
21     }
22
23     while(heap.size() && k--)
24     {
25         LL t = heap.top() / 2;
26         heap.pop();
27         sum -= t;
28         if(t % 2 == 0) heap.push(t);
29     }
30
31     cout << sum << endl;
32
33     return 0;
34 }
```

## Java 算法代码：

```
1 import java.util.*;
2
3 public class Main
4 {
5     public static void main(String[] args)
6     {
7         Scanner in = new Scanner(System.in);
8         int n = in.nextInt(), k = in.nextInt();
9         PriorityQueue<Integer> heap = new PriorityQueue<>((a, b) -> {
10             return b - a;
11         });
12
13         long sum = 0, x;
14         for(int i = 0; i < n; i++)
15         {
16             x = in.nextLong();
17             sum += x;
18             if(x % 2 == 0)
19             {
20                 heap.add((int)x);
21             }
22         }
23
24         while(!heap.isEmpty() && k-- != 0)
25         {
26             long t = heap.poll() / 2;
27             sum -= t;
28             if(t % 2 == 0)
29             {
30                 heap.add((int)t);
31             }
32         }
33
34         System.out.println(sum);
35     }
36 }
```

## Day04

### 1. Fibonacci数列（Fib 数列）

(题号: 45791)

1. 题目链接: [WY22 Fibonacci数列](#)

2. 题目描述:

题目描述: Fibonacci数列是这样定义的:

$$f_i = \begin{cases} 0 & i = 0 \\ 1 & i = 1 \\ f_{i-2} + f_{i-1} & i \geq 2 \end{cases}$$

因此, Fibonacci数列: 0, 1, 1, 2, 3, 5, 8, 13 ... 在Fibonacci数列中的数称为Fibonacci数。

现在有一个  $n$ , 需要将它变为一个Fibonacci数, 每一步可以当前数字加 1 或者减 1。

求最少需要多少步将  $n$  变为Fibonacci数?

输入描述: 输入为一个正整数  $n$ 。

$$1 \leq n \leq 10^6$$

输出描述: 输出一个整数, 代表将  $n$  变为Fibonacci数的最小步数。

补充说明:

示例1

输入: 15

输出: 2

说明:

3. 解法:

算法思路:

求斐波那契数列的过程中, 判断一下: 何时  $n$  会在两个 fib 数之间。

C++ 算法代码:

```
1 #include <iostream>
2 using namespace std;
3
4 int n;
5
6 int main()
7 {
8     cin >> n;
9     int a = 0, b = 1, c = 1;
10
11     while(n > c)
12     {
13         a = b;
14         b = c;
```

```
15         c = a + b;
16     }
17
18     cout << min(c - n, n - b) << endl;
19
20     return 0;
21 }
```

## Java 算法代码：

```
1  import java.util.Scanner;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9
10         int n = in.nextInt();
11         int a = 0, b = 1, c = 1;
12
13         while(n > c)
14         {
15             a = b;
16             b = c;
17             c = a + b;
18         }
19
20         System.out.println(Math.min(c - n, n - b));
21     }
22 }
```

## 2. 单词搜索（搜索）

（题号：2315888）

1. 题目链接：[NC242 单词搜索](#)

2. 题目描述：

给出一个二维字符数组和一个单词，判断单词是否在数组中出现，单词由相邻单元格的字母连接而成，相邻单元指的是上下左右相邻。同一单元格的字母不能多次使用。

数据范围：

0 < 行长度 ≤ 100

0 < 列长度 ≤ 100

0 < 单词长度 ≤ 1000

例如：

给出的数组为["XYZE","SFZS","XDEE"]时，

对应的二维字符数组为：

X	Y	Z	E
S	F	Z	S
X	D	E	E

若单词为"XYZZED"时，应该返回 true，

也即：

X	Y	Z	E
S	F	Z	S
X	D	E	E

若单词为"SEE"时，应该返回 true，

也即：

X	Y	Z	E
S	F	Z	S
X	D	E	E

若单词为"XYZY"时，应该返回 false。

### 3. 解法：

算法思路：

简单深搜应用题~

C++ 算法代码：

```
1 class Solution
2 {
3     int m, n;
4     bool vis[101][101] = { 0 };
5 }
```



```

6     int dx[4] = {0, 0, 1, -1};
7     int dy[4] = {1, -1, 0, 0};
8
9 public:
10    bool exist(vector<string>& board, string word)
11    {
12        m = board.size(), n = board[0].size();
13
14        for(int i = 0; i < m; i++)
15        {
16            for(int j = 0; j < n; j++)
17            {
18                if(board[i][j] == word[0])
19                {
20                    if(dfs(board, i, j, word, 0)) return true;
21                }
22            }
23        }
24
25        return false;
26    }
27
28    bool dfs(vector<string>& board, int i, int j, string& word, int pos)
29    {
30        if(pos == word.size() - 1)
31        {
32            return true;
33        }
34
35        vis[i][j] = true;
36        for(int k = 0; k < 4; k++)
37        {
38            int a = i + dx[k], b = j + dy[k];
39            if(a >= 0 && a < m && b >= 0 && b < n && !vis[a][b] && board[a][b]
== word[pos + 1])
40            {
41                if(dfs(board, a, b, word, pos + 1)) return true;
42            }
43        }
44        vis[i][j] = false;
45        return false;
46    }
47 };

```

## Java 算法代码：

```
1 import java.util.*;
2
3
4 public class Solution
5 {
6     int m, n;
7     int[] dx = {0, 0, 1, -1};
8     int[] dy = {1, -1, 0, 0};
9     boolean[][] vis;
10    char[] word;
11
12    public boolean exist (String[] board, String _word)
13    {
14        m = board.length;
15        n = board[0].length();
16        vis = new boolean[m][n];
17        word = _word.toCharArray();
18
19        for(int i = 0; i < m; i++)
20        {
21            for(int j = 0; j < n; j++)
22            {
23                if(board[i].charAt(j) == word[0])
24                {
25                    if(dfs(board, i, j, 0) == true) return true;
26                }
27            }
28        }
29
30        return false;
31    }
32
33    public boolean dfs(String[] board, int i, int j, int pos)
34    {
35        if(pos == word.length - 1)
36        {
37            return true;
38        }
39
40        vis[i][j] = true;
41        for(int k = 0; k < 4; k++)
42        {
43            int x = i + dx[k], y = j + dy[k];
```

```

44         if(x >= 0 && x < m && y >= 0 && y < n && !vis[x][y] &&
board[x].charAt(y) == word[pos + 1])
45         {
46             if(dfs(board, x, y, pos + 1)) return true;
47         }
48     }
49     vis[i][j] = false;
50
51     return false;
52 }
53 }

```

### 3. 杨辉三角（动态规划）

(题号：618638)

#### 1. 题目链接：BC140 杨辉三角

#### 2. 题目描述：

题目描述：KiKi知道什么叫杨辉三角之后对杨辉三角产生了浓厚的兴趣，他想知道杨辉三角的前n行，请编程帮他解答。杨辉三角，本质上是二项式 $(a+b)^n$ 的n次方展开后各项的系数排成的三角形。其性质包括：每行的端点数为1，一个数也为1；每个数等于它左上方和上方的两数之和。

输入描述：第一行包含一个整数n。(1≤n≤30)

输出描述：包含n行，为杨辉三角的前n行，每个数输出域宽为5。

补充说明：

示例1

输入：6

输出：

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1

```

说明：

#### 3. 解法：

##### 算法思路：

最基础的 dp 模型，按照规律模拟出来杨辉三角即可。

##### C++ 算法代码：

```

1 #include <iostream>
2

```

```

3 using namespace std;
4
5 int dp[31][31];
6
7 int main()
8 {
9     int n;
10    cin >> n;
11
12    dp[1][1] = 1;
13    for(int i = 2; i <= n; i++)
14    {
15        for(int j = 1; j <= i; j++)
16        {
17            dp[i][j] = dp[i - 1][j] + dp[i - 1][j - 1];
18        }
19    }
20
21    for(int i = 1; i <= n; i++)
22    {
23        for(int j = 1; j <= i; j++)
24        {
25            printf("%5d", dp[i][j]);
26        }
27        printf("\n");
28    }
29
30    return 0;
31 }

```

#### Java 算法代码:

```

1 import java.util.Scanner;
2
3 // 注意类名必须为 Main, 不要有任何 package xxx 信息
4 public class Main
5 {
6     public static void main(String[] args)
7     {
8         Scanner in = new Scanner(System.in);
9         int n = in.nextInt();
10        int[][] dp = new int[n + 1][n + 1];
11
12        dp[1][1] = 1;

```

```

13     for(int i = 2; i <= n; i++)
14     {
15         for(int j = 1; j <= i; j++)
16         {
17             dp[i][j] = dp[i - 1][j] + dp[i - 1][j - 1];
18         }
19     }
20
21     for(int i = 1; i <= n; i++)
22     {
23         for(int j = 1; j <= i; j++)
24         {
25             // dp[i][j]
26             StringBuffer ret = new StringBuffer();
27             int len = Integer.toString(dp[i][j]).length();
28             for(int k = 0; k < 5 - len; k++)
29             {
30                 ret.append(" ");
31             }
32             System.out.print(ret.toString() + dp[i][j]);
33         }
34         System.out.println();
35     }
36 }
37 }

```

## Day05

### 1. 游游的you (贪心 + 模拟)

(题号: 10274330)

1. 题目链接: [\[编程题\]游游的you](#)

2. 题目描述:

题目描述: 游游现在有  $a$  个 'y',  $b$  个 'o',  $c$  个 'u', 他想用这些字母拼成一个字符串。  
三个相邻的字母是 "you" 可以获得 2 分, 两个相邻的字母是 "oo", 可以获得 1 分。  
问最多可以获得多少分?

输入描述: 第一行一个整数  $q$ , 代表询问次数。  
接下来  $q$  行, 每行三个正整数  $a, b, c$ , 用空格隔开。

$$1 \leq q \leq 10^5$$
$$1 \leq a, b, c \leq 10^9$$

输出描述: 输出  $q$  行, 代表每次询问的答案。

补充说明:

示例1

输入: 3

1 1 1

2 3 2

1 5 2

输出: 2

4

5

说明: 第一次询问, 可以拼出 "you", 获得 2 分。

第二次询问, 可以拼出 "oyouyou", 获得 4 分。

第三次询问, 可以拼出 "uoooooyou", 获得 5 分。

### 3. 解法:

#### 算法思路:

由题意得:

- you 和 oo 是相互独立的;
- 但是 you 的分值更高, 因此我们应该优先去拼凑 you, 然后再考虑 oo

#### C++ 算法代码:

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     int q;
7     int a, b, c;
8     cin >> q;
9
10    while(q--)
11    {
12        cin >> a >> b >> c;
13        int x = min(a, min(b, c));
14        cout << (x * 2 + max(b - x - 1, 0)) << endl;
15    }
```

```
16
17     return 0;
18 }
```

## Java 算法代码：

```
1 import java.util.Scanner;
2
3 // 注意类名必须为 Main, 不要有任何 package xxx 信息
4 public class Main
5 {
6     public static void main(String[] args)
7     {
8         Scanner in = new Scanner(System.in);
9
10        int q = in.nextInt();
11        int a, b, c;
12
13        while(q-- != 0)
14        {
15            a = in.nextInt(); b = in.nextInt(); c = in.nextInt();
16            int x = Math.min(a, Math.min(b, c));
17            System.out.println(x * 2 + Math.max(b - x - 1, 0));
18        }
19    }
20 }
```

## 2. 腐烂的苹果（多源 BFS）

（题号：2426947）

1. 题目链接：[NC398 腐烂的苹果](#)

2. 题目描述：

题目描述: 给定一个  $n \times m$  的网格, 其中每个单元格中可能有三种值中的一个 0, 1, 2。

其中 0 表示这个格子为空、1 表示这个格子有一个完好的苹果, 2 表示这个格子有一个腐烂的苹果。

腐烂的苹果每分钟会向上下左右四个方向的苹果传播一次病菌, 并导致相邻的苹果腐烂。请问经过多少分钟, 网格中不存在完好的苹果。如果有苹果永远不会腐烂则返回 -1。

数据范围:  $1 \leq n, m \leq 1000$ , 网格中的值满足  $0 \leq val \leq 2$

补充说明:

示例1

输入: `[[2,1,1],[1,0,1],[1,1,1]]`

输出: 4

说明:

示例2

输入: `[[2,1,0],[1,0,1],[0,0,0]]`

输出: -1

说明:

### 3. 解法:

#### 算法思路:

多源 BFS 问题, 固定套路~

#### C++ 算法代码:

```
1 class Solution
2 {
3     int m, n;
4     int dx[4] = {0, 0, 1, -1};
5     int dy[4] = {1, -1, 0, 0};
6     bool vis[1010][1010] = { 0 };
7
8 public:
9     int rotApple(vector<vector<int>> &grid)
10    {
11        m = grid.size(), n = grid[0].size();
12
13        queue<pair<int, int>> q;
14        for(int i = 0; i < m; i++)
15            for(int j = 0; j < n; j++)
16                if(grid[i][j] == 2)
17                    q.push({i, j});
18
19        int ret = 0;
20        while(q.size())
21        {
22            int sz = q.size();
23            ret++;
24            while(sz--)
```



```

25         {
26             auto [a, b] = q.front();
27             q.pop();
28             for(int i = 0; i < 4; i++)
29             {
30                 int x = a + dx[i], y = b + dy[i];
31                 if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == 1
&& !vis[x][y])
32                 {
33                     vis[x][y] = true;
34                     q.push({x, y});
35                 }
36             }
37         }
38     }
39
40     for(int i = 0; i < m; i++)
41         for(int j = 0; j < n; j++)
42             if(grid[i][j] == 1 && !vis[i][j])
43                 return -1;
44
45     return ret - 1;
46 }
47 };

```

Java 算法代码:

```

1 import java.util.*;
2
3 public class Solution
4 {
5     int[] dx = {0, 0, 1, -1};
6     int[] dy = {1, -1, 0, 0};
7
8     public int rotApple (ArrayList<ArrayList<Integer>> grid)
9     {
10         int m = grid.size();
11         int n = grid.get(0).size();
12         boolean[][] vis = new boolean[m][n];
13
14         Queue<int[]> q = new LinkedList<>();
15
16         for(int i = 0; i < m; i++)
17         {

```

```

18         for(int j = 0; j < n; j++)
19         {
20             if(grid.get(i).get(j) == 2)
21             {
22                 q.add(new int[]{i, j});
23             }
24         }
25     }
26
27     int ret = 0;
28     while(!q.isEmpty())
29     {
30         int sz = q.size();
31         while(sz-- != 0)
32         {
33             int[] t = q.poll();
34             int a = t[0], b = t[1];
35             for(int i = 0; i < 4; i++)
36             {
37                 int x = a + dx[i], y = b + dy[i];
38                 if(x >= 0 && x < m && y >= 0 && y < n && vis[x][y] == false
&& grid.get(x).get(y) == 1)
39                 {
40                     vis[x][y] = true;
41                     q.add(new int[]{x, y});
42                 }
43             }
44         }
45         ret++;
46     }
47
48     // 判断剩余的苹果
49     for(int i = 0; i < m; i++)
50     {
51         for(int j = 0; j < n; j++)
52         {
53             if(grid.get(i).get(j) == 1 && !vis[i][j])
54             {
55                 return -1;
56             }
57         }
58     }
59
60     return ret - 1;
61
62 }
63 }

```

### 3. 孩子们的的游戏（约瑟夫环）

（题号：23265）

1. 题目链接：[JZ62 孩子们的的游戏\(圆圈中最后剩下的数\)](#)

2. 题目描述：

题目描述： 每年六一儿童节，牛客都会准备一些小礼物和小游戏去看望孤儿院的孩子们。其中，有个游戏是这样的：首先，让  $n$  个小朋友们围成一个大圈，小朋友们的编号是  $0 \sim n-1$ 。然后，随机指定一个数  $m$ ，让编号为  $0$  的小朋友开始报数。每次喊到  $m-1$  的那个小朋友要出列唱首歌，然后可以在礼品箱中任意的挑选礼物，并且不再回到圈中，从他的下一个小朋友开始，继续  $0 \dots m-1$  报数....这样下去....直到剩下最后一个小朋友，可以不用表演，并且拿到牛客礼品，请你试着想下，哪个小朋友会得到这份礼品呢？

例如， $0、1、2、3、4$  这  $5$  个数字组成一个圆圈（如图 6.3 所示），从数字  $0$  开始每次删除第  $3$  个数字，则删除的前  $4$  个数字依次是  $2、0、4、1$ ，因此最后剩下的数字是  $3$ 。

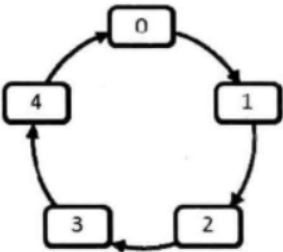


图 6.3 由  $0 \sim 4$  这  $5$  个数字组成的圆圈

数据范围： $1 \leq n \leq 5000, 1 \leq m \leq 10000$   
要求：空间复杂度  $O(1)$ ，时间复杂度  $O(n)$

补充说明：

示例1  
输入：5, 3  
输出：3  
说明：

示例2  
输入：2, 3  
输出：1  
说明：有2个小朋友编号为0, 1，第一次报数报到3的是0号小朋友，0号小朋友出圈，1号小朋友得到礼物

示例3  
输入：10, 17  
输出：2  
说明：

3. 解法：

算法思路：

解法一：模拟。这里不多赘述，用数组或者链表模拟均可。但是数据量大的话会超时.....

解法二：数学规律。通过画图，可以找到相邻两次删除坐标的规律。通过递推关系，求出剩下的最后一个数是哪个。

C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     int LastRemaining_Solution(int n, int m)
5     {
6         int f = 0;
7         for(int i = 2; i <= n; i++) f = (f + m) % i;
8         return f;
9     }
10 };
```

Java 算法代码：

```
1 import java.util.*;
2
3 public class Solution
4 {
5     public int LastRemaining_Solution (int n, int m)
6     {
7         int f = 0;
8         for(int i = 2; i <= n; i++) f = (f + m) % i;
9         return f;
10    }
11 }
```

## Day06

### 1. 大数加法（高精度加法）

（题号：1061819）

1. 题目链接：[NC1 大数加法](#)

2. 题目描述：

题目描述：以字符串的形式读入两个数字，编写一个函数计算它们的和，以字符串形式返回。

数据范围： $s.length, t.length \leq 100000$ ，字符串仅由'0'~'9'构成

要求：时间复杂度  $O(n)$

补充说明：

示例1

输入："1","99"

输出："100"

说明：1+99=100

示例2

输入："114514",""

输出："114514"

说明：

### 3. 解法：

#### 算法思路：

模版类型的算法题，模拟加法列竖式运算的过程即可。

#### C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     string solve(string s, string t)
5     {
6         string ret;
7         int tmp = 0; // 进位 + 本次累加和
8         int i = s.size() - 1, j = t.size() - 1;
9
10        while(i >= 0 || j >= 0 || tmp) // 模拟加法
11        {
12            if(i >= 0) tmp += s[i--] - '0';
13            if(j >= 0) tmp += t[j--] - '0';
14            ret += '0' + tmp % 10;
15            tmp /= 10;
16        }
17
18        reverse(ret.begin(), ret.end()); // 别忘逆序
19        return ret;
20    }
21 };
```

## Java 算法代码：

```
1 import java.util.*;
2
3 public class Solution
4 {
5     public String solve (String s, String t)
6     {
7         StringBuffer ret = new StringBuffer();
8         int tmp = 0; // 标记进位 + 本次累加的结果
9         int i = s.length() - 1, j = t.length() - 1;
10
11         while(i >= 0 || j >= 0 || tmp > 0) // 模拟加法
12         {
13             if(i >= 0) tmp += s.charAt(i--)-'0';
14             if(j >= 0) tmp += t.charAt(j--)-'0';
15             ret.append((char)('0' + tmp % 10));
16             tmp /= 10;
17         }
18
19         return ret.reverse().toString(); // 别忘逆序
20     }
21 }
```

## 2. 链表相加（二）（链表 + 高精度加法）

（题号：1008772）

1. 题目链接：[NC40 链表相加\(二\)](#)

2. 题目描述：

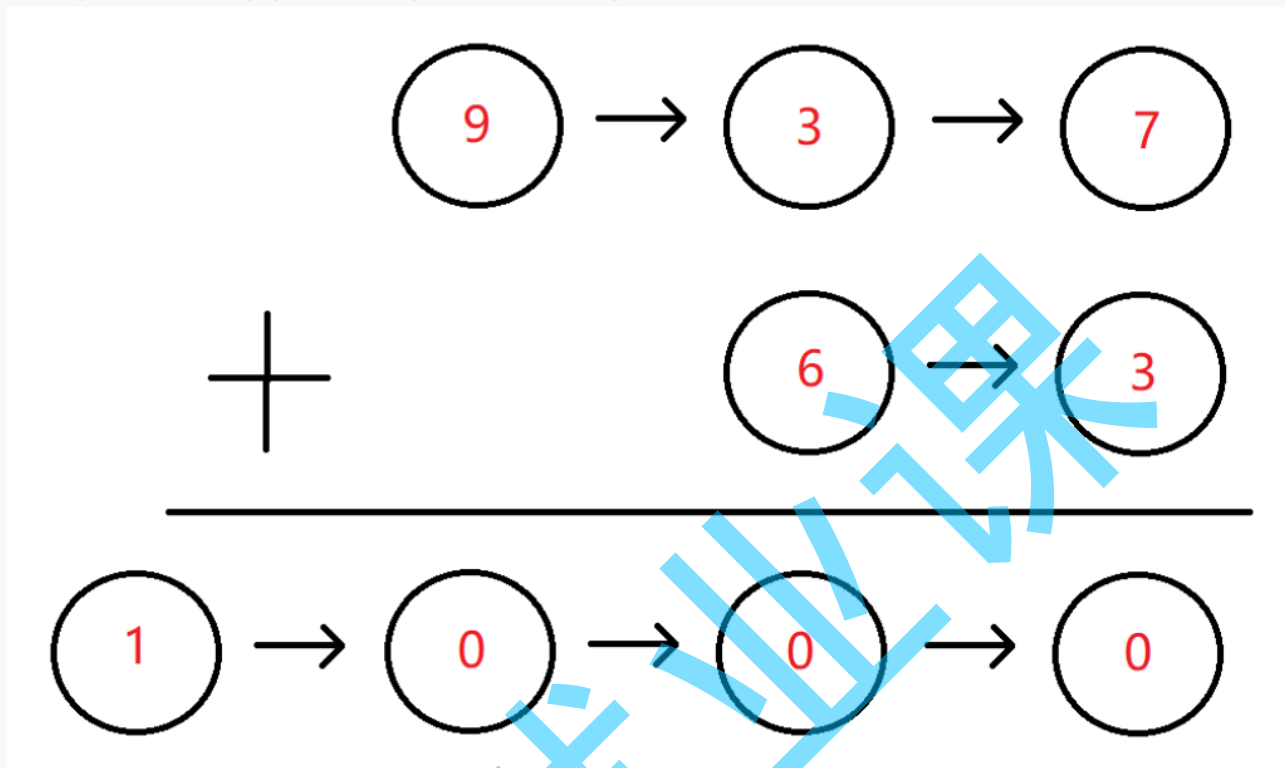
题目描述: 假设链表中每一个节点的值都在 0 - 9 之间, 那么链表整体就可以代表一个整数。

给定两个这种链表, 请生成代表两个整数相加值的结果链表。

数据范围:  $0 \leq n, m \leq 1000000$ , 链表任意值  $0 \leq val \leq 9$

要求: 空间复杂度  $O(n)$ , 时间复杂度  $O(n)$

例如: 链表 1 为 9->3->7, 链表 2 为 6->3, 最后生成新的结果链表为 1->0->0->0。



补充说明:  $1 \leq n, m \leq 10^6$   
 $0 \leq a_i, b_i \leq 9$

示例1

输入: [9, 3, 7], [6, 3]

输出: {1, 0, 0, 0}

说明: 如题面解释

### 3. 解法:

#### 算法思路:

模拟高精度加法的过程, 只不过是在链表中进行而已。

#### C++ 算法代码:

```
1 class Solution
2 {
3 public:
4     // 逆序链表
5     ListNode* reverse(ListNode* head)
6     {
7         ListNode* newHead = new ListNode(0);
8         ListNode* cur = head;
9     }
```

```

10     while(cur)
11     {
12         ListNode* next = cur->next;
13         cur->next = newHead->next;
14         newHead->next = cur;
15         cur = next;
16     }
17     cur = newHead->next;
18     delete newHead;
19     return cur;
20 }
21
22 ListNode* addInList(ListNode* head1, ListNode* head2)
23 {
24     // 1. 逆序
25     head1 = reverse(head1);
26     head2 = reverse(head2);
27
28     // 2. 高精度加法
29     int t = 0;
30     ListNode* cur1 = head1, *cur2 = head2;
31     ListNode* ret = new ListNode(0);
32     ListNode* prev = ret;
33     while(cur1 || cur2 || t)
34     {
35         if(cur1)
36         {
37             t += cur1->val;
38             cur1 = cur1->next;
39         }
40         if(cur2)
41         {
42             t += cur2->val;
43             cur2 = cur2->next;
44         }
45         prev = prev->next = new ListNode(t % 10);
46         t /= 10;
47     }
48
49     cur1 = ret->next;
50     ret->next = nullptr;
51     delete ret;
52     return reverse(cur1);
53 }
54 };
55

```



## Java 算法代码：

```
1 import java.util.*;
2
3 public class Solution
4 {
5     // 逆序链表
6     public ListNode reverse(ListNode head)
7     {
8         ListNode newHead = new ListNode(0);
9         ListNode cur = head;
10
11         while(cur != null)
12         {
13             ListNode next = cur.next;
14             cur.next = newHead.next;
15             newHead.next = cur;
16             cur = next;
17         }
18
19         return newHead.next;
20     }
21
22     public ListNode addInList (ListNode head1, ListNode head2)
23     {
24         // 1. 逆序
25         head1 = reverse(head1);
26         head2 = reverse(head2);
27
28         // 2. 高精度加法
29         ListNode cur1 = head1, cur2 = head2;
30         int t = 0;
31         ListNode ret = new ListNode(0), prev = ret;
32
33         while(cur1 != null || cur2 != null || t != 0)
34         {
35             if(cur1 != null)
36             {
37                 t += cur1.val;
38                 cur1 = cur1.next;
39             }
40             if(cur2 != null)
41             {
42                 t += cur2.val;
43                 cur2 = cur2.next;
```

```
44         }
45         prev = prev.next = new ListNode(t % 10);
46         t /= 10;
47     }
48
49     return reverse(ret.next);
50 }
51 }
52
```

### 3. 大数乘法（高精度乘法）

(题号: 1062527)

1. 题目链接: [NC10 大数乘法](#)

2. 题目描述:

题目描述: 以字符串的形式读入两个数字, 编写一个函数计算它们的乘积, 以字符串形式返回。

数据范围: 读入的数字大小满足  $0 \leq n \leq 10^{1000}$

要求: 空间复杂度  $O(m)$ , 时间复杂度  $O(m^2)$  (假设  $m$  是  $n$  的长度)

补充说明:

示例1

输入: "11", "99"

输出: "1089"

说明:  $11 \times 99 = 1089$

示例2

输入: "1", "0"

输出: "0"

说明:

3. 解法:

算法思路:

根据列竖式运算的过程模拟即可。

但是我们可以改进一下列竖式的过程:

- 先计算无进位相乘并且相加后的结果;
- 然后再处理进位。

## C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     string solve(string s, string t)
5     {
6         reverse(s.begin(), s.end());
7         reverse(t.begin(), t.end());
8         int m = s.size(), n = t.size();
9
10        vector<int> tmp(m + n);
11        // 1. 无进位相乘相加
12        for(int i = 0; i < m; i++)
13        {
14            for(int j = 0; j < n; j++)
15            {
16                tmp[i + j] += (s[i] - '0') * (t[j] - '0');
17            }
18        }
19
20        // 2. 处理进位
21        int c = 0;
22        string ret;
23        for(auto x : tmp)
24        {
25            c += x;
26            ret += c % 10 + '0';
27            c /= 10;
28        }
29        while(c)
30        {
31            ret += c % 10 + '0';
32            c /= 10;
33        }
34
35        // 3. 处理前导零
36        while(ret.size() > 1 && ret.back() == '0') ret.pop_back();
37
38        reverse(ret.begin(), ret.end());
39        return ret;
40    }
41 };
42
```

## Java 算法代码:

```
1 import java.util.*;
2
3 public class Solution
4 {
5     public String solve (String ss, String tt)
6     {
7         char[] s = new StringBuffer(ss).reverse().toString().toCharArray();
8         char[] t = new StringBuffer(tt).reverse().toString().toCharArray();
9         int m = s.length, n = t.length;
10        int[] tmp = new int[m + n];
11
12        // 1. 无进位相乘相加
13        for(int i = 0; i < m; i++)
14        {
15            for(int j = 0; j < n; j++)
16            {
17                tmp[i + j] += (s[i] - '0') * (t[j] - '0');
18            }
19        }
20
21        // 2. 处理进位
22        StringBuffer ret = new StringBuffer();
23        int c = 0;
24        for(int x : tmp)
25        {
26            c += x;
27            ret.append((char)(c % 10 + '0'));
28            c /= 10;
29        }
30        while(c != 0)
31        {
32            ret.append((char)(c % 10 + '0'));
33            c /= 10;
34        }
35
36        // 3. 处理前导零
37        while(ret.length() > 1 && ret.charAt(ret.length() - 1) == '0')
38        {
39            ret.deleteCharAt(ret.length() - 1);
40        }
41        return ret.reverse().toString();
42    }
43 }
```

比特就业课