

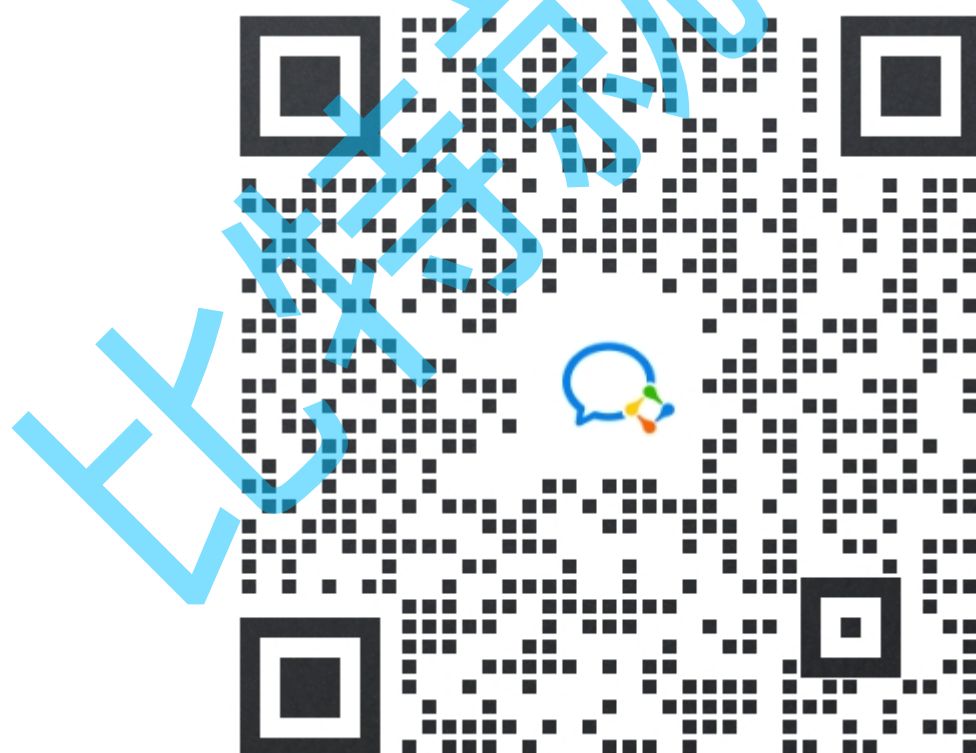
笔试强训第 02 周

版权说明

版权说明

本“**比特就业课**”笔试强训第 02 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 02 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>

Day07

1. 在字符串中找出连续最长的数字串（模拟 + 双指针）

（题号：10055169）

1. 题目链接： [在字符串中找出连续最长的数字串](#)

2. 题目描述：

题目描述：现有一个字符串str，输出字符串str中的最长的数字子串。

输入描述：一个包含字母和数字的字符串，长度不超过255。
保证最少有一个字符是数字，且只有一个最长的数字子串。

输出描述：最长的数字子串。

补充说明：

示例1

输入：abcd12345ed125ss123058789

输出：123058789

说明：

3. 解法：

算法思路：

双指针：

遍历整个字符串，遇到数字的时候，用双指针找出这段连续的数字子串，根据此时的长度更新起始位置和长度。

C++ 算法代码：

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     string s;
8     cin >> s;
9
10    int begin = -1, len = 0;
11    for(int i = 0; i < s.size(); i++)
12    {
13        if(s[i] >= '0' && s[i] <= '9')
```

```

14     {
15         int j = i;
16         while(j < s.size() && s[j] >= '0' && s[j] <= '9') j++;
17         if(j - i > len)
18         {
19             begin = i;
20             len = j - i;
21         }
22         i = j;
23     }
24 }
25 if(begin == -1) cout << "" << endl;
26 else cout << s.substr(begin, len) << endl;
27
28 return 0;
29 }

```

Java 算法代码:

```

1 import java.util.Scanner;
2 import java.io.*;
3
4 public class Main
5 {
6     public static void main(String[] args) throws Exception
7     {
8         BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
9         char[] s = br.readLine().toCharArray();
10
11         int begin = 0, len = 0;
12         for(int i = 0; i < s.length; i++)
13         {
14             if(s[i] >= '0' && s[i] <= '9')
15             {
16                 int j = i;
17                 while(j < s.length && s[j] >= '0' && s[j] <= '9') j++;
18                 if(j - i > len)
19                 {
20                     begin = i;
21                     len = j - i;
22                 }
23                 i = j;
24             }

```

```
25     }
26
27     for(int i = begin; i < begin + len; i++)
28     {
29         System.out.print(s[i]);
30     }
31 }
32 }
```

2. 岛屿数量 (BFS / DFS)

(题号: 1024684)

1. 题目链接: [NC109 岛屿数量](#)

2. 题目描述:

题目描述: 给一个01矩阵, 1代表是陆地, 0代表海洋, 如果两个1相邻, 那么这两个1属于同一个岛。我们只考虑上下左右为相邻。

岛屿: 相邻陆地可以组成一个岛屿 (相邻: 上下左右) 判断岛屿个数。

例如:

输入

```
[
[1,1,0,0,0],
[0,1,0,1,1],
[0,0,0,1,1],
[0,0,0,0,0],
[0,0,1,1,1]
]
```

对应的输出为3

(注: 存储的01数据其实是字符'0','1')

补充说明: 01矩阵范围 $\leq 200 \times 200$

示例1

输入: `[[1,1,0,0,0],[0,1,0,1,1],[0,0,0,1,1],[0,0,0,0,0],[0,0,1,1,1]]`

输出: 3

说明:

示例2

输入: `[[0]]`

输出: 0

说明:

3. 解法:

算法思路:

经典的 floodfill 算法。用 dfs 或者是 bfs 找出一个联通的区域, 并且标记上。看看一共能找出几个联通块。

C++ 算法代码：

```
1 class Solution
2 {
3     public:
4         int solve(vector<vector<char>>& grid)
5         {
6             int ret = 0; // 记录最终结果
7             int m = grid.size(), n = grid[0].size();
8
9             for (int i = 0; i < m; i++)
10             {
11                 for (int j = 0; j < n; j++)
12                 {
13                     if (grid[i][j] == '1')
14                     {
15                         ret++; // 发现一个岛屿
16                         dfs(grid, i, j); // 把这个岛屿全部修改成 '0'
17                     }
18                 }
19             }
20             return ret;
21         }
22
23         int dx[4] = {0, 1, -1, 0};
24         int dy[4] = {1, 0, 0, -1};
25         void dfs(vector<vector<char>>& grid, int i, int j)
26         {
27             grid[i][j] = '0'; // 把该位置先变成 '0'
28             int m = grid.size(), n = grid[0].size();
29             // 遍历相邻的所有位置
30             for(int k = 0; k < 4; k++)
31             {
32                 int x = i + dx[k], y = j + dy[k];
33                 if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1')
34                     dfs(grid, x, y);
35             }
36         }
37     };
```

Java 算法代码：

```
1 import java.util.*;
```

```

2
3 public class Solution
4 {
5     int m, n;
6     int[] dx = {0, 0, 1, -1};
7     int[] dy = {1, -1, 0, 0};
8     boolean[][] vis = new boolean[210][210];
9
10    public int solve (char[][] grid)
11    {
12        // dfs
13        m = grid.length; n = grid[0].length;
14
15        int ret = 0;
16        for(int i = 0; i < m; i++)
17        {
18            for(int j = 0; j < n; j++)
19            {
20                if(grid[i][j] == '1' && !vis[i][j])
21                {
22                    ret++;
23                    dfs(grid, i, j);
24                }
25            }
26        }
27        return ret;
28    }
29
30    public void dfs(char[][] grid, int i, int j)
31    {
32        vis[i][j] = true;
33        for(int k = 0; k < 4; k++)
34        {
35            int x = i + dx[k], y = j + dy[k];
36            if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1' &&
!vis[x][y])
37            {
38                dfs(grid, x, y);
39            }
40        }
41    }
42 }

```

3. 拼三角（枚举 / dfs）

(题号: 1389509)

1. 题目链接: [拼三角](#)

2. 题目描述:

题目描述: 给出6根棍子, 能否在选出3根拼成一个三角形的同时剩下的3根也能组成一个三角形?

输入描述: 首先在一行中给出一个 $t, 1 \leq t \leq 10^3$, 代表测试数据的组数
接下来 t 行, 每行给出6个数字代表棍子长度, 棍子长度为正且小于 10^9

输出描述: 在一行中输出 "Yes" or "No"

补充说明:

示例1

输入: 2

1 1 1 1 1 1

1 2 3 4 5 6

输出: Yes

No

说明:

3. 解法:

算法思路:

简单枚举, 不过有很多种枚举方法, 我们这里之间用简单粗暴的枚举方式。

C++ 算法代码:

```
1 #include <iostream>
2 #include <algorithm>
3
4 using namespace std;
5
6 int t;
7 int arr[6];
8
9 int main()
10 {
11     cin >> t;
12     while(t--)
13     {
14         for(int i = 0; i < 6; i++) cin >> arr[i];
15         sort(arr, arr + 6);
16         if(arr[0] + arr[1] > arr[2] && arr[3] + arr[4] > arr[5] ||
17            arr[0] + arr[2] > arr[3] && arr[1] + arr[4] > arr[5] ||
18            arr[0] + arr[3] > arr[4] && arr[1] + arr[2] > arr[5] ||
19            arr[0] + arr[4] > arr[5] && arr[1] + arr[2] > arr[3])
```

```

20     {
21         cout << "Yes" << endl;
22     }
23     else cout << "No" << endl;
24 }
25
26 return 0;
27 }

```

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 t = in.nextInt();
9          int[] arr = new int[6];
10
11          while(t-- != 0)
12          {
13              for(int i = 0; i < 6; i++) arr[i] = in.nextInt();
14              Arrays.sort(arr);
15
16              if(arr[0] + arr[1] > arr[2] && arr[3] + arr[4] > arr[5] ||
17                 arr[0] + arr[2] > arr[3] && arr[1] + arr[4] > arr[5] ||
18                 arr[0] + arr[3] > arr[4] && arr[1] + arr[2] > arr[5] ||
19                 arr[0] + arr[4] > arr[5] && arr[1] + arr[2] > arr[3])
20              {
21                  System.out.println("Yes");
22              }
23              else
24              {
25                  System.out.println("No");
26              }
27          }
28      }
29  }

```


Day08

1. 求最小公倍数（数学）

（题号：10055186）

1. 题目链接： [HJ108 求最小公倍数](#)

2. 题目描述：

题目描述：正整数 a 和正整数 b 的最小公倍数，是指能被 a 和 b 整除的最小的正整数。请你求 a 和 b 的最小公倍数。
比如输入5和7，5和7的最小公倍数是35，则需要返回35。

输入描述：输入两个正整数。

$$1 \leq a, b \leq 100000$$

输出描述：输出最小公倍数。

补充说明：

示例1

输入：5 7

输出：35

说明：

示例2

输入：4 6

输出：12

说明：

示例3

输入：6 12

输出：12

说明：

3. 解法：

算法思路：

A 和 B 的最小公倍数 = $A * B /$ 两者的最大公约数。

最大公约数：辗转相除法。

C++ 算法代码：

```
1 #include <iostream>
```

```
2 using namespace std;
3
4 int gcd(int a, int b)
5 {
6     if(b == 0) return a;
7     return gcd(b, a % b);
8 }
9
10 int main()
11 {
12     int a, b;
13     cin >> a >> b;
14
15     cout << (a * b / gcd(a, b)) << endl;
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 int gcd(int a, int b)
7     {
8         if(b == 0) return a;
9         return gcd(b, a % b);
10    }
11
12    public static void main(String[] args)
13    {
14        Scanner in = new Scanner(System.in);
15        int a = in.nextInt(), b = in.nextInt();
16
17        System.out.println(a * b / gcd(a, b));
18    }
19 }
```

2. 数组中的最长连续子序列（排序 + 模拟）

（题号：1008752）

1. 题目链接：NC95 数组中的最长连续子序列

2. 题目描述：

题目描述：给定无序数组arr，返回其中最长的连续序列的长度(要求值连续，位置可以不连续,例如 3,4,5,6为连续的自然数)

数据范围： $1 \leq n \leq 10^5$ ，数组中的值满足 $1 \leq val \leq 10^8$

要求：空间复杂度 $O(n)$ ，时间复杂度 $O(n\log n)$

补充说明： $1 \leq n \leq 10^5$

$1 \leq arr_i \leq 10^8$

示例1

输入：[100,4,200,1,3,2]

输出：4

说明：

示例2

输入：[1,1,1]

输出：1

说明：

3. 解法：

算法思路：

排序 + 模拟

但是要注意处理数字相同的情况！

C++ 算法代码：

```
1 class Solution
2 {
3 public:
4     int MLS(vector<int>& arr)
5     {
6         sort(arr.begin(), arr.end());
7
8         int n = arr.size(), ret = 0;
9         for(int i = 0; i < n; )
10        {
11            int j = i + 1, count = 1;
12            while(j < n)
13            {
```

```

14         if(arr[j] - arr[j - 1] == 1)
15         {
16             count++;
17             j++;
18         }
19         else if(arr[j] - arr[j - 1] == 0)
20         {
21             j++;
22         }
23         else
24         {
25             break;
26         }
27     }
28     ret = max(ret, count);
29     i = j;
30 }
31 return ret;
32 }
33 };

```

Java 算法代码:

```

1 import java.util.*;
2
3 public class Solution
4 {
5     public int MLS (int[] arr)
6     {
7         Arrays.sort(arr);
8
9         int n = arr.length, ret = 0;
10        for(int i = 0; i < n; )
11        {
12            int j = i + 1, count = 1;
13            while(j < n)
14            {
15                if(arr[j] - arr[j - 1] == 1)
16                {
17                    count++;
18                    j++;
19                }
20                else if(arr[j] - arr[j - 1] == 0)
21                {

```

```

22         j++;
23     }
24     else
25     {
26         break;
27     }
28 }
29 ret = Math.max(ret, count);
30 i = j;
31 }
32 return ret;
33 }
34 }

```

3. 字母收集（动态规划 - 路径问题）

（题号：1714951）

1. 题目链接：DP39 字母收集

2. 题目描述：

题目描述：有一个 $n * m$ 的矩形方阵，每个格子上面写了一个小写字母。

小红站在矩形的左上角，她每次可以向右或者向下走，走到某个格子上就可以收集这个格子的字母。

小红非常喜欢 "love" 这四个字母。她拿到一个 l 字母可以得 4 分，拿到一个 o 字母可以得 3 分，拿到一个 v 字母可以得 2 分，拿到一个 e 字母可以得 1 分。

她想知道，在最优的选择一条路径的情况下，她最多能获得多少分？

输入描述： $1 \leq n, m \leq 500$

接下来的 n 行 每行一个长度为 m 的、仅有小写字母构成的字符串，代表矩形方阵。

输出描述：小红最大可能的得分。

补充说明：

示例1

输入：3 2

ab

cd

ef

输出：1

说明：选择下、下、右）这条路径即可，可以收集到 acef 这四个字母各一次，获得 $0+0+1+0=1$ 分。

示例2

输入：2 3

lle

ove

输出：11

说明：

3. 解法：

算法思路：

基础的路径问题的 dp 模型。

C++ 算法代码：

```
1  #include <iostream>
2
3  using namespace std;
4
5  const int N = 510;
6
7  char g[N][N];
8  int dp[N][N];
9  int m, n;
10
11 int main()
12 {
13     cin >> m >> n;
14     for(int i = 1; i <= m; i++)
15     {
16         for(int j = 1; j <= n; j++)
17         {
18             cin >> g[i][j];
19         }
20     }
21
22     for(int i = 1; i <= m; i++)
23     {
24         for(int j = 1; j <= n; j++)
25         {
26             int t = 0;
27             if(g[i][j] == 'l') t = 4;
28             else if(g[i][j] == 'o') t = 3;
29             else if(g[i][j] == 'v') t = 2;
30             else if(g[i][j] == 'e') t = 1;
31             dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + t;
32         }
33     }
34
35     cout << dp[m][n] << endl;
36
37     return 0;
38 }
```

Java 算法代码：

```
1 import java.util.*;
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(), m = in.nextInt();
10        char[][] arr = new char[n + 1][m + 1];
11
12        for(int i = 1; i <= n; i++)
13        {
14            char[] s = in.next().toCharArray();
15            for(int j = 1; j <= m; j++)
16            {
17                arr[i][j] = s[j - 1];
18            }
19        }
20
21        int[][] dp = new int[n + 1][m + 1];
22        for(int i = 1; i <= n; i++)
23        {
24            for(int j = 1; j <= m; j++)
25            {
26                int t = 0;
27                if(arr[i][j] == 'l') t = 4;
28                else if(arr[i][j] == 'o') t = 3;
29                else if(arr[i][j] == 'v') t = 2;
30                else if(arr[i][j] == 'e') t = 1;
31                dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]) + t;
32            }
33        }
34
35        System.out.println(dp[n][m]);
36    }
37 }
```

1. 添加逗号（模拟）

(题号: 144140)

1. 题目链接: [BC146 添加逗号](#)

2. 题目描述:

题目描述: 对于一个较大的整数 $N(1 \leq N \leq 2,000,000,000)$

比如 980364535, 我们常常需要一位一位数这个数字是几位数, 但是如果在这个数字每三位加一个逗号, 它会变得更加易于朗读。因此, 这个数字加上逗号成如下的模样: 980,364,535 请写一个程序帮她完成这件事情

输入描述: 一行一个整数 N

输出描述: 一行一个字符串表示添加完逗号的结果

补充说明: $1 \leq n \leq 2,000,000,000$

示例1

输入: 980364535

输出: 980,364,535

说明:

3. 解法:

算法思路:

可以从后往前遍历这个数, 每提取三个数字的时候, 加一个逗号。最后处理一下边界情况即可。

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      cin >> s;
10     string ret;
11
12     int n = s.size();
13     for(int i = 0; i < n; i++)
14     {
15         ret += s[i];
16         if((n - i - 1) % 3 == 0 && i != n - 1) ret += ',';
17     }
18 }
```



```
19     cout << ret << endl;
20
21     return 0;
22 }
```

Java 算法代码:

```
1  import java.util.Scanner;
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 Exception
8      {
9          BufferedReader br = new BufferedReader(new
10 InputStreamReader(System.in));
11          String s = br.readLine();
12          int n = s.length();
13
14          for(int i = 0; i < n; i++)
15          {
16              System.out.print(s.charAt(i));
17              if((n - i - 1) % 3 == 0 && i != n - 1)
18              {
19                  System.out.print(',');
20              }
21          }
22 }
```

2. 跳台阶 (动态规划)

(题号: 2357966)

1. 题目链接: [DP2 跳台阶](#)

2. 题目描述:

题目描述：一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个 n 级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

数据范围： $0 \leq n \leq 40$

要求：时间复杂度： $O(n)$ ，空间复杂度： $O(1)$

输入描述：本题输入仅一行，即一个整数 n

输出描述：输出跳上 n 级台阶有多少种跳法

补充说明：

示例1

输入：2

输出：2

说明：青蛙要跳上两级台阶有两种跳法，分别是：先跳一级，再跳一级或者直接跳两级。因此答案为2

示例2

输入：7

输出：21

说明：

3. 解法：

算法思路：

最入门的动态规划问题，不必多说.....

C++ 算法代码：

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int n;
8     cin >> n;
9
10    int a = 1, b = 1, c;
11    for(int i = 2; i <= n; i++)
12    {
13        c = a + b;
14        a = b;
15        b = c;
16    }
17
18    if(n == 0 || n == 1) cout << n << endl;
19    else cout << c << endl;
20
21    return 0;
22 }
```

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
11         int a = 1, b = 1, c = 0;
12         for(int i = 2; i <= n; i++)
13         {
14             c = a + b;
15             a = b;
16             b = c;
17         }
18
19         if(n == 0 || n == 1)
20         {
21             System.out.println(n);
22         }
23         else
24         {
25             System.out.println(c);
26         }
27     }
28 }
```

3. 扑克牌顺子（排序）

（题号：23252）

1. 题目链接：[扑克牌顺子](#)

2. 题目描述：

题目描述：现在有2副扑克牌，从扑克牌中随机五张扑克牌，我们需要来判断一下是不是顺子。

有如下规则：

1. A为1，J为11，Q为12，K为13，A不能视为14
2. 大、小王为 0，0 可以看作任意牌
3. 如果给出的五张牌能组成顺子（即这五张牌是连续的）就输出true，否则就输出false。
4. 数据保证每组5个数字，每组最多含有4个零，数组的数取值为 [0, 13]

要求：空间复杂度 $O(1)$ ，时间复杂度 $O(n\log n)$ ，本题也有时间复杂度 $O(n)$ 的解法

补充说明：

示例1

输入：[6,0,2,0,4]

输出：true

说明：中间的两个0一个看作3，一个看作5。即：[6,3,2,5,4] 这样这五张牌在[2,6]区间连续，输出true

示例2

输入：[0,3,2,6,4]

输出：true

说明：

示例3

输入：[1,0,0,1,0]

输出：false

说明：

3. 解法：

规律：

如果能够构成顺子的话，所有的非零元素应该满足下面两个条件：

a. 不能出现重复元素；

b. $\max - \min \leq 4$

C++ 算法代码：

```
1 class Solution
2 {
3     bool hash[14] = { 0 };
4
5 public:
6     bool IsContinuous(vector<int>& numbers)
7     {
8         int maxVal = 0, minVal = 14;
9         for(auto x : numbers)
10         {
11             if(x)
12             {
13                 if(hash[x]) return false;
```

```

14         hash[x] = true;
15         maxVal = max(maxVal, x);
16         minVal = min(minVal, x);
17     }
18 }
19 return maxVal - minVal <= 4;
20 }
21 };

```

Java 算法代码:

```

1 import java.util.*;
2
3 public class Solution
4 {
5     public boolean IsContinuous (int[] numbers)
6     {
7         boolean[] hash = new boolean[14];
8
9         int maxVal = 0, minVal = 14;
10        for(int x : numbers)
11        {
12            if(x != 0)
13            {
14                if(hash[x]) return false;
15                hash[x] = true;
16                maxVal = Math.max(maxVal, x);
17                minVal = Math.min(minVal, x);
18            }
19        }
20        return maxVal - minVal <= 4;
21    }
22 }

```

Day10

1. 最长回文子串（回文串）

(题号: 25269)

1. 题目链接: [OR26 最长回文子串](#)

2. 题目描述:

题目描述: 对于长度为 n 的一个字符串A (仅包含数字, 大小写英文字母), 请设计一个高效算法, 计算其中最长回文子串的长度。

数据范围: $1 \leq n \leq 1000$

要求: 空间复杂度 $O(1)$, 时间复杂度 $O(n^2)$

进阶: 空间复杂度 $O(n)$, 时间复杂度 $O(n)$

补充说明:

示例1

输入: "ababc"

输出: 3

说明: 最长的回文子串为"aba"与"bab", 长度都为3

示例2

输入: "abbba"

输出: 5

说明:

示例3

输入: "b"

输出: 1

说明:

3. 解法:

算法思路:

枚举所有的中心点, 然后向两边扩散。

C++ 算法代码:

```
1 class Solution
2 {
3 public:
4     int getLongestPalindrome(string s)
5     {
6         int ret = 1, n = s.size();
7         for(int i = 1; i < n; i++)
8         {
9             // 当长度是奇数的时候
10            int left = i - 1, right = i + 1;
11            while(left >= 0 && right < n && s[left] == s[right])
12            {
13                left--;
14                right++;
15            }
16            ret = max(ret, right - left - 1);
17            // 当长度是偶数的时候
18            left = i - 1, right = i;
```

```

19         while(left >= 0 && right < n && s[left] == s[right])
20         {
21             left--;
22             right++;
23         }
24         ret = max(ret, right - left - 1);
25     }
26     return ret;
27 }
28 };

```

Java 算法代码：

```

1  import java.util.*;
2
3  public class Solution
4  {
5      public int getLongestPalindrome (String s)
6      {
7          // 中心扩展算法
8          int n = s.length();
9
10         int ret = 0;
11         for(int i = 0; i < n; i++) // 枚举所有的中点
12         {
13             // 当长度为奇数的时候
14             int left = i - 1, right = i + 1;
15             while(left >= 0 && right < n && s.charAt(left) == s.charAt(right))
16             {
17                 left--;
18                 right++;
19             }
20             ret = Math.max(ret, right - left - 1);
21             // 当长度为偶数的时候
22             left = i; right = i + 1;
23             while(left >= 0 && right < n && s.charAt(left) == s.charAt(right))
24             {
25                 left--;
26                 right++;
27             }
28             ret = Math.max(ret, right - left - 1);
29         }
30         return ret;
31     }

```

2. 买卖股票的最好时机(一) (贪心)

(题号: 2364518)

1. 题目链接: DP30 买卖股票的最好时机(一)

2. 题目描述:

题目描述: 假设你有一个数组prices, 长度为n, 其中prices[i]是股票在第i天的价格, 请根据这个价格数组, 返回买卖股票能获得的最大收益

1. 你可以买入一次股票和卖出一一次股票, 并非每天都可以买入或卖出一一次, 总共只能买入和卖出一一次, 且买入必须在卖出的前面的某一天
2. 如果不能获取到任何利润, 请返回0
3. 假设买入卖出均无手续费

数据范围: $0 \leq n \leq 10^5, 0 \leq val \leq 10^4$

输入描述: 第一行输入一个正整数 n 表示数组的长度
第二行输入 n 个正整数, 表示股票在第 i 天的价格

输出描述: 输出只买卖一次的最高收益

补充说明:

示例1

输入: 7

8 9 2 5 4 7 1

输出: 5

说明: 在第3天(股票价格 = 2)的时候买入, 在第6天(股票价格 = 7)的时候卖出, 最大利润 = 7-2 = 5, 不能选择在第2天买入, 第3天卖出, 这样就亏损7了; 同时, 你也不能在买入前卖出股票。

示例2

输入: 3

2 4 1

输出: 2

说明:

3. 解法:

算法思路:

小贪心:

因为只能买卖一次, 因此, 对于第 i 天来说, 如果在这天选择卖出股票, 应该在 $[0, i]$ 天之内, 股票最低点买入股票, 此时就可以获得最大利润。

那么, 我们仅需维护一个前驱最小值的变量, 并且不断更新结果即可。

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 arr[N];
8
9 int main()
10 {
11     cin >> n;
12     for(int i = 0; i < n; i++) cin >> arr[i];
13
14     int ret = 0, prevMin = arr[0];
15     for(int i = 1; i < n; i++)
16     {
17         prevMin = min(arr[i], prevMin);
18         ret = max(ret, arr[i] - prevMin);
19     }
20
21     cout << ret << endl;
22
23     return 0;
24 }

```

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 ret = 0, prevMin = in.nextInt();
12        int x = 0;
13        for(int i = 1; i < n; i++)
14        {
15            x = in.nextInt();
16            ret = Math.max(ret, x - prevMin);
17            prevMin = Math.min(prevMin, x);
18        }
19

```

```
20         System.out.println(ret < 0 ? 0 : ret);  
21     }  
22 }
```

3. 过河卒（动态规划 - 路径问题）

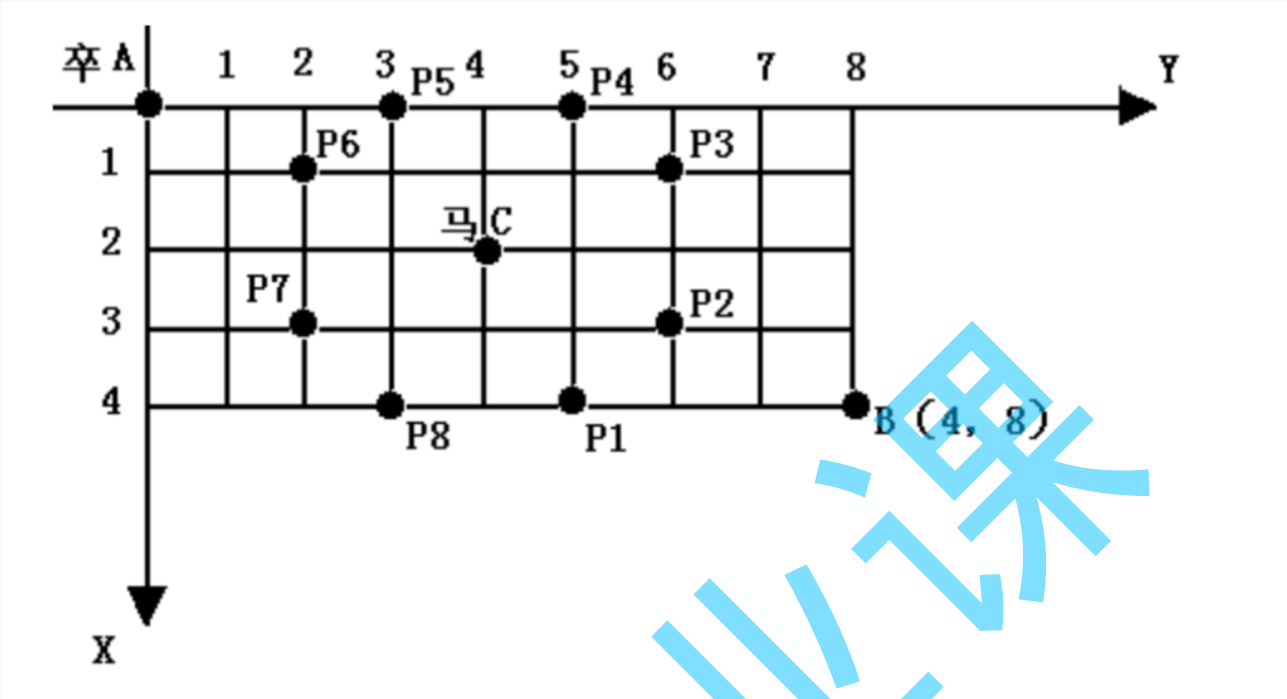
(题号: 2378812)

1. 题目链接: [DP13 \[NOIP2002 普及组\] 过河卒](#)
2. 题目描述:

题目描述：棋盘上 A 点有一个过河卒，需要走到目标 B 点。卒行走的规则：可以向下、或者向右。同时在棋盘上 C 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。

棋盘用坐标表示，A 点 (0, 0)、B 点(n,m)，同样马的位置坐标是需要给出的。

现在要求你计算出卒从 A 点能够到达 B 点的路径的条数，假设马的位置(x,y)是固定不动的，并不是卒走一步马走一步。



注：马一次跳跃到达的点(x1,y1)和马原坐标(x,y)的关系是 $|x1 - x| + |y1 - y| = 3$ ，且 $x1 \neq x, y1 \neq y$

数据范围： $1 \leq n, m \leq 20$ ，马的坐标 $0 \leq x, y \leq 20$
 $1 \leq a, b, c, d \leq 1000$

输入描述： 仅一行，输入 n,m,x,y 四个正整数。分别表示B点坐标和马的坐标

输出描述： 输出路径总数

补充说明：

示例1

输入： 6 6 3 3

输出： 6

说明：

示例2

输入： 5 4 2 3

输出： 3

说明：

示例3

输入： 2 5 3 5

输出： 1

说明：

3. 解法：

算法思路：

简单路径 dp 问题：相当于是有障碍物的路径类问题，标记走到障碍物上的方法数为 0 即可。

C++ 算法代码：

```

1 #include <iostream>
2 using namespace std;
3
4 int n, m, x, y;
5 long long dp[25][25];
6
7 int main()
8 {
9     cin >> n >> m >> x >> y;
10
11     x += 1; y += 1;
12     dp[0][1] = 1;
13     for(int i = 1; i <= n + 1; i++)
14     {
15         for(int j = 1; j <= m + 1; j++)
16         {
17             if(i != x && j != y && abs(i - x) + abs(j - y) == 3 || (i == x &&
j == y))
18             {
19                 dp[i][j] = 0;
20             }
21             else
22             {
23                 dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
24             }
25         }
26     }
27
28     cout << dp[n + 1][m + 1] << endl;
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(), m = in.nextInt(), x = in.nextInt(), y =
in.nextInt();
10
11         long[][] dp = new long[n + 2][m + 2];
12         dp[0][1] = 1;
13         x += 1; y += 1;
14
15         for(int i = 1; i <= n + 1; i++)
16         {
17             for(int j = 1; j <= m + 1; j++)
18             {
19                 if(i != x && j != y && Math.abs(i - x) + Math.abs(j - y) == 3
|| (i == x && j == y))
20                 {
21                     dp[i][j] = 0;
22                 }
23                 else
24                 {
25                     dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
26                 }
27             }
28         }
29
30         System.out.println(dp[n + 1][m + 1]);
31     }
32 }

```

Day11

1. 游游的水果大礼包（枚举）

（题号：10274354）

1. 题目链接：[游游的水果大礼包](#)

2. 题目描述：

题目描述：游游有 n 个苹果， m 个桃子。她可以把2个苹果和1个桃子组成价值 a 元的一号水果大礼包，也可以把1个苹果和2个桃子组成价值 b 元的二号水果大礼包。游游想知道，自己最多能组成多少价值总和的大礼包？

输入描述：四个正整数 n, m, a, b ，用空格隔开。分别代表苹果的数量、桃子的数量、一号大礼包价值、二号大礼包价值。
 $1 \leq n, m, a, b \leq 10^6$

输出描述：一个整数，代表大礼包的最大价值总和。

补充说明：

示例1

输入：3 4 1 2

输出：4

说明：组成两个二号水果大礼包，使用了2个苹果和4个桃子。总价值为4。

示例2

输入：1 1 5 6

输出：0

说明：显然无法组合成任意一个大礼包

3. 解法：

算法思路：

很容易想到贪心，但是很不幸，贪心是错的。

正确的解法应该是枚举所有的情况~

C++ 算法代码：

```
1 #include <iostream>
2
3 using namespace std;
4
5 long long n, m, a, b;
6
7 int main()
8 {
9     cin >> n >> m >> a >> b;
10
11     long long ret = 0;
12     for(long long x = 0; x <= min(n / 2, m); x++) // 枚举 1 号礼包的个数
13     {
14         long long y = min(n - x * 2, (m - x) / 2); // 计算 2 号礼包的个数
15         ret = max(ret, a * x + b * y);
16     }
17
18     cout << ret << endl;
19
20     return 0;
21 }
```

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         long n = in.nextInt();
9         long m = in.nextInt();
10        long a = in.nextInt();
11        long b = in.nextInt();
12
13        long ret = 0;
14        for(long x = 0; x <= Math.min(n / 2, m); x++) // 枚举 1 号礼包的个数
15        {
16            long y = Math.min(n - x * 2, (m - x) / 2); // 计算 2 号礼包的个数
17            ret = Math.max(ret, a * x + b * y);
18        }
19        System.out.println(ret);
20    }
21 }
```

2. 买卖股票的最好时机(二) (贪心)

(题号: 2364576)

1. 题目链接: [DP31 买卖股票的最好时机\(二\)](#)

2. 题目描述:

题目描述: 假设你有一个数组prices, 长度为n, 其中prices[i]是某只股票在第i天的价格, 请根据这个价格数组, 返回买卖股票能获得的最大收益

1. 你可以多次买卖该只股票, 但是再次购买前必须卖出之前的股票
2. 如果不能获取收益, 请返回0
3. 假设买入卖出均无手续费

数据范围: $0 \leq n \leq 1 \times 10^5$, $1 \leq prices[i] \leq 10^4$

要求: 空间复杂度 $O(n)$, 时间复杂度 $O(n)$

进阶: 空间复杂度 $O(1)$, 时间复杂度 $O(n)$

输入描述: 第一行输入一个正整数 n, 表示数组 prices 的长度

第二行输入 n 个正整数, 表示数组中prices的值

输出描述: 输出最大收益

补充说明:

示例1

输入: 7

8 9 2 5 4 7 1

输出: 7

说明: 在第1天(股票价格=8)买入, 第2天(股票价格=9)卖出, 获利9-8=1
在第3天(股票价格=2)买入, 第4天(股票价格=5)卖出, 获利5-2=3
在第5天(股票价格=4)买入, 第6天(股票价格=7)卖出, 获利7-4=3
总获利1+3+3=7, 返回7

示例2

输入: 5

5 4 3 2 1

输出: 0

说明: 由于每天股票都在跌, 因此不进行任何交易最优。最大收益为0。

示例3

输入: 5

1 2 3 4 5

输出: 4

说明: 第一天买进, 最后一天卖出最优。中间的当天买进当天卖出不影响最终结果。最大收益为4。

3. 解法:

算法思路:

小贪心:

因为可以无限次交易, 因此, 只要股票的价格有上升, 就统统把利润拿到手。

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 arr[N];
8
```



```

9  int main()
10 {
11     cin >> n;
12     for(int i = 0; i < n; i++) cin >> arr[i];
13
14     int ret = 0;
15     for(int i = 1; i < n; i++)
16         if(arr[i] > arr[i - 1])
17             ret += arr[i] - arr[i - 1];
18
19     cout << ret << endl;
20
21     return 0;
22 }

```

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[] arr = new int[n];
11         for(int i = 0; i < n; i++)
12         {
13             arr[i] = in.nextInt();
14         }
15
16         int ret = 0;
17         for(int i = 1; i < n; i++)
18         {
19             if(arr[i] > arr[i - 1])
20             {
21                 ret += arr[i] - arr[i - 1];
22             }
23         }
24         System.out.println(ret);
25     }
26 }

```

3. 倒置字符串（字符串）

（题号：10055179）

1. 题目链接：[OR62 倒置字符串](#)

2. 题目描述：

题目描述：将一句话的单词进行倒置，标点不倒置。比如 "I like beijing."，经过处理后变为："beijing. like I"。
字符串长度不超过100。

输入描述：输入一个仅包含小写字母、空格、'.' 的字符串，长度不超过100。
'.' 只出现在最后一个单词的末尾。

输出描述：依次输出倒置之后的字符串，以空格分割。

补充说明：

示例1

输入：I like beijing.

输出：beijing. like I

说明：

3. 解法：

算法思路：

找到规律反转字符串即可。

C++ 算法代码：

```
1 #include <iostream>
2 #include <string>
3 #include <algorithm>
4
5 using namespace std;
6
7 int main()
8 {
9     string s;
10    getline(cin, s);
11
12    reverse(s.begin(), s.end());
13
14    int left = 0, n = s.size();
15    while(left < n)
16    {
17        int right = left;
```

```

18     while(right < n && s[right] != ' ') // 找单词
19     {
20         right++;
21     }
22     reverse(s.begin() + left, s.begin() + right);
23     while(right < n && s[right] == ' ') right++;
24     left = right;
25 }
26
27 cout << s << endl;
28
29 return 0;
30 }

```

Java 算法代码：

```

1  import java.util.Scanner;
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 Reverse(char[] s, int left, int right)
8      {
9          int l = left, r = right;
10         while(l < r)
11         {
12             char ch = s[l];
13             s[l] = s[r];
14             s[r] = ch;
15             l++;
16             r--;
17         }
18     }
19
20     public static void main(String[] args) throws Throwable
21     {
22         BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
23         char[] s = br.readLine().toCharArray();
24         int n = s.length;
25         Reverse(s, 0, n - 1);
26
27         int left = 0;

```

```

28         while(left < n)
29         {
30             int right = left;
31             while(right < n && s[right] != ' ') right++; // 找单词
32             Reverse(s, left, right - 1);
33             while(right < n && s[right] == ' ') right++; // 跳过空格
34             left = right;
35         }
36
37         for(char ch : s) System.out.print(ch);
38     }
39 }

```

Day12

1. 删除公共字符（哈希）

（题号：10055174）

1. 题目链接：OR63 删除公共字符

2. 题目描述：

题目描述：输入两个字符串，从第一个字符串中删除第二个字符串中所有的字符。

例如：第一个字符串是"They are students."，第二个字符串是"aeiou"。删除之后的第一个字符串变成"Thy r stdnts."。
保证两个字符串的长度均不超过100。

输入描述：输入两行，每行一个字符串。

输出描述：输出删除后的字符串。

补充说明：

示例1

输入：They are students.
aeiou

输出：Thy r stdnts.

说明：

3. 解法：

算法思路：

用哈希表记录一下字符串的字符信息即可。

C++ 算法代码：

```

1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main()
7 {
8     string s, t;
9     getline(cin, s);
10    getline(cin, t);
11
12    bool hash[300] = { 0 };
13    for(char ch : t) hash[ch] = true;
14
15    string ret;
16    for(auto ch : s)
17    {
18        if(!hash[ch])
19        {
20            ret += ch;
21        }
22    }
23
24    cout << ret << endl;
25
26    return 0;
27 }

```

Java 算法代码：

```

1 import java.util.Scanner;
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 IOException
8     {
9         BufferedReader br = new BufferedReader(new
10         InputStreamReader(System.in));
11         String s = br.readLine();
12         String t = br.readLine();
13
14         boolean[] hash = new boolean[300];

```

```
14     for(int i = 0; i < t.length(); i++)
15     {
16         hash[t.charAt(i)] = true;
17     }
18
19     for(int i = 0; i < s.length(); i++)
20     {
21         if(!hash[s.charAt(i)])
22         {
23             System.out.print(s.charAt(i));
24         }
25     }
26 }
27 }
```

2. 两个链表的第一个公共结点（链表）

(题号: 23257)

1. 题目链接: [JZ52 两个链表的第一个公共结点](#)

2. 题目描述:

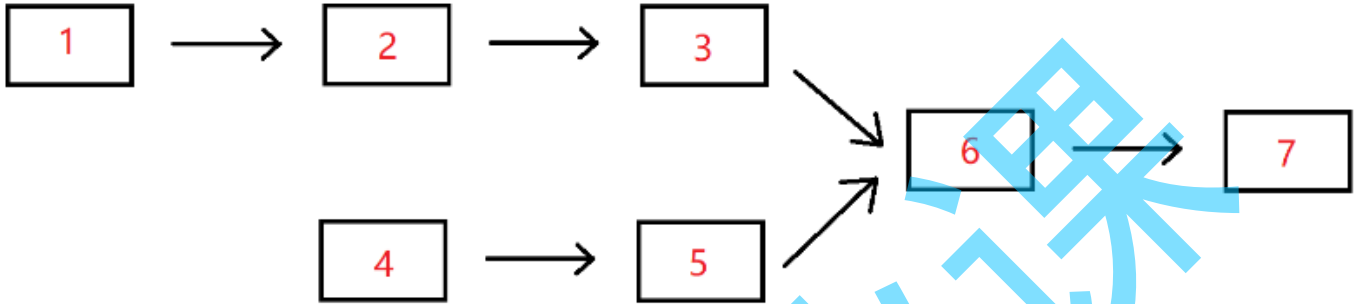
描述

输入两个无环的单向链表，找出它们的第一个公共结点，如果没有公共节点则返回空。（注意因为传入数据是链表，所以错误测试数据的提示是用其他方式显示的，保证传入数据是正确的）

数据范围： $n \leq 1000$

要求：空间复杂度 $O(1)$ ，时间复杂度 $O(n)$

例如，输入{1,2,3},{4,5},{6,7}时，两个无环的单向链表的结构如下图所示：



可以看到它们的第一个公共结点的结点值为6，所以返回结点值为6的结点。

输入描述：

输入分为是3段，第一段是第一个链表的非公共部分，第二段是第二个链表的非公共部分，第三段是第一个链表和第二个链表的公共部分。后台会将这3个参数组装为两个链表，并将这两个链表对应的头节点传入到函数FindFirstCommonNode里面，用户得到的输入只有pHead1和pHead2。

返回值描述：

返回传入的pHead1和pHead2的第一个公共结点，后台会打印以该节点为头节点的链表。

示例1

输入： {1,2,3},{4,5},{6,7}

复制

返回值： {6,7}

复制

说明： 第一个参数{1,2,3}代表是第一个链表非公共部分，第二个参数{4,5}代表是第二个链表非公共部分，最后的{6,7}表示的是2个链表的公共部分
这3个参数最后在后台会组装成为2个两个无环的单链表，且是有公共节点的

3. 解法：

算法思路：

根据两个链表走的路程相同，找到相交点。

C++ 算法代码：

```

1  /*
2  struct ListNode {
3      int val;
4      struct ListNode *next;
5      ListNode(int x) :
6          val(x), next(NULL) {
7          }
8  };*/
9  class Solution
10 {
11 public:
12     ListNode* FindFirstCommonNode( ListNode* pHead1, ListNode* pHead2)
13     {
14         ListNode* cur1 = pHead1, *cur2 = pHead2;
15         while(cur1 != cur2)
16         {
17             cur1 = cur1 != NULL ? cur1->next : pHead2;
18             cur2 = cur2 != NULL ? cur2->next : pHead1;
19         }
20         return cur1;
21     }
22 };

```

Java 算法代码：

```

1  import java.util.*;
2  /*
3  public class ListNode {
4      int val;
5      ListNode next = null;
6
7      ListNode(int val) {
8          this.val = val;
9      }
10 }*/
11 public class Solution
12 {
13     public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2)
14     {
15         ListNode cur1 = pHead1, cur2 = pHead2;
16         while(cur1 != cur2)
17         {
18             cur1 = cur1 != null ? cur1.next : pHead2;
19             cur2 = cur2 != null ? cur2.next : pHead1;

```



```
20     }
21     return cur1;
22 }
23 }
```

3. mari和shiny（动态规划 - 线性dp）

（题号：375040）

1. 题目链接：[mari和shiny](#)

2. 题目描述：

题目描述：mari每天都非常shiny。她的目标是把正能量传达到世界的每个角落！有一天，她得到了一个仅由小写字母组成的字符串。她想知道，这个字符串有多少个"shy"的子序列？（所谓子序列的含义见样例说明）

输入描述：第一行一个正整数n，代表字符串的长度。（ $1 \leq n \leq 300000$ ）第二行为一个长度为n，仅由小写字母组成的字符串。

输出描述：一个正整数，代表子序列"shy"的数量。

补充说明：mari大喊道：“是shiny不是shy！！！”

示例1

输入：8

sshhyau

输出：8

说明：假设字符串下标从1到8。共有 (135) (136) (145) (146) (235) (236) (245) (246) 八个"shy"子序列。

3. 解法：

算法思路：

简单线性 dp：

维护 i 位置之前，一共有多少个 "s" "sh"，然后更新 "shy" 的个数。

C++ 算法代码：

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

```

10 {
11     cin >> n >> str;
12
13     long long s = 0, h = 0, y = 0;
14     for(int i = 0; i < n; i++)
15     {
16         char ch = str[i];
17         if(ch == 's') s++;
18         else if(ch == 'h') h += s;
19         else if(ch == 'y') y += h;
20     }
21
22     cout << y << endl;
23
24     return 0;
25 }

```

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();
9         char[] str = in.next().toCharArray();
10
11         long s = 0, h = 0, y = 0;
12         for(int i = 0; i < n; i++)
13         {
14             char ch = str[i];
15             if(ch == 's') s += 1;
16             else if(ch == 'h') h += s;
17             else if(ch == 'y') y += h;
18         }
19         System.out.println(y);
20     }
21 }

```

比特就业课