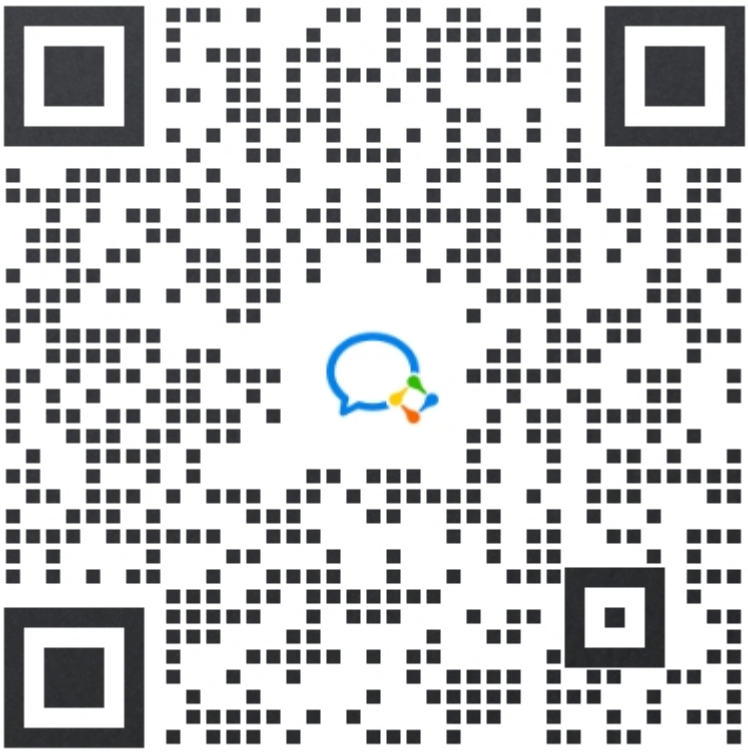
笔试强训第 02 周

版权说明

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：

```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
    string s;
    cin >> s;

    int begin = -1, len = 0;
    for(int i = 0; i < s.size(); i++)
    {
        if(s[i] >= '0' && s[i] <= '9')
```

```
        {
            int j = i;
            while(j < s.size() && s[j] >= '0' && s[j] <= '9') j++;
            if(j - i > len)
            {
                begin = i;
                len = j - i;
            }
            i = j;
        }
    }
    if(begin == -1) cout << "" << endl;
    else cout << s.substr(begin, len) << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;
import java.io.*;

public class Main
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        char[] s = br.readLine().toCharArray();

        int begin = 0, len = 0;
        for(int i = 0; i < s.length; i++)
        {
            if(s[i] >= '0' && s[i] <= '9')
            {
                int j = i;
                while(j < s.length && s[j] >= '0' && s[j] <= '9') j++;
                if(j - i > len)
                {
                    begin = i;
                    len = j - i;
                }
                i = j;
            }
```

```
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<=200*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：

```
class Solution
{
  public:
    int solve(vector<vector<char>>& grid)
    {
        int ret = 0; // 记录最终结果
        int m = grid.size(), n = grid[0].size();

        for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (grid[i][j] == '1')
                {
                    ret++; // 发现一个岛屿
                    dfs(grid, i, j); // 把这个岛屿全部修改成 '0'
                }
            }
        }
        return ret;
    }

    int dx[4] = {0, 1, -1, 0};
    int dy[4] = {1, 0, 0, -1};
    void dfs(vector<vector<char>>& grid, int i, int j)
    {
        grid[i][j] = '0'; // 把该位置先变成 '0'
        int m = grid.size(), n = grid[0].size();
        // 遍历相邻的所有位置
        for(int k = 0; k < 4; k++)
        {
            int x = i + dx[k], y = j + dy[k];
            if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1')
                dfs(grid, x, y);
        }
    }
};
```

Java 算法代码：

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

```java

public class Solution
{
    int m, n;
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};
    boolean[][] vis = new boolean[210][210];

    public int solve (char[][] grid)
    {
        // dfs
        m = grid.length; n = grid[0].length;

        int ret = 0;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                if(grid[i][j] == '1' && !vis[i][j])
                {
                    ret++;
                    dfs(grid, i, j);
                }
            }
        }
        return ret;
    }

    public void dfs(char[][] grid, int i, int j)
    {
        vis[i][j] = true;
        for(int k = 0; k < 4; k++)
        {
            int x = i + dx[k], y = j + dy[k];
            if(x >= 0 && x < m && y >= 0 && y < n && grid[x][y] == '1' &&
!vis[x][y])
            {
                dfs(grid, x, y);
            }
        }
    }
}
```

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：

```
#include <iostream>
#include <algorithm>

using namespace std;

int t;
int arr[6];

int main()
{
    cin >> t;
    while(t--)
    {
        for(int i = 0; i < 6; i++) cin >> arr[i];
        sort(arr, arr + 6);
        if(arr[0] + arr[1] > arr[2] && arr[3] + arr[4] > arr[5] ||
           arr[0] + arr[2] > arr[3] && arr[1] + arr[4] > arr[5] ||
           arr[0] + arr[3] > arr[4] && arr[1] + arr[2] > arr[5] ||
           arr[0] + arr[4] > arr[5] && arr[1] + arr[2] > arr[3])
```

```
        {
            cout << "Yes" << endl;
        }
        else cout << "No" << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        int[] arr = new int[6];

        while(t-- != 0)
        {
            for(int i = 0; i < 6; i++) arr[i] = in.nextInt();
            Arrays.sort(arr);

            if(arr[0] + arr[1] > arr[2] && arr[3] + arr[4] > arr[5] ||
               arr[0] + arr[2] > arr[3] && arr[1] + arr[4] > arr[5] ||
               arr[0] + arr[3] > arr[4] && arr[1] + arr[2] > arr[5] ||
               arr[0] + arr[4] > arr[5] && arr[1] + arr[2] > arr[3])
            {
                System.out.println("Yes");
            }
            else
            {
                System.out.println("No");
            }
        }
    }
}
```

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：

```cpp
#include <iostream>
```

```cpp
using namespace std;

int gcd(int a, int b)
{
    if(b == 0) return a;
    return gcd(b, a % b);
}

int main()
{
    int a, b;
    cin >> a >> b;

    cout << (a * b / gcd(a, b)) << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static int gcd(int a, int b)
    {
        if(b == 0) return a;
        return gcd(b, a % b);
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int a = in.nextInt(), b = in.nextInt();

        System.out.println(a * b / gcd(a, b));
    }
}
```

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 $O(nlogn)$
>
> 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：

```cpp
class Solution
{
public:
    int MLS(vector<int>& arr)
    {
        sort(arr.begin(), arr.end());

        int n = arr.size(), ret = 0;
        for(int i = 0; i < n; )
        {
            int j = i + 1, count = 1;
            while(j < n)
            {
```

```
                if(arr[j] - arr[j - 1] == 1)
                {
                    count++;
                    j++;
                }
                else if(arr[j] - arr[j - 1] == 0)
                {
                    j++;
                }
                else
                {
                    break;
                }
            }
            ret = max(ret, count);
            i = j;
        }
        return ret;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int MLS (int[] arr)
    {
        Arrays.sort(arr);

        int n = arr.length, ret = 0;
        for(int i = 0; i < n; )
        {
            int j = i + 1, count = 1;
            while(j < n)
            {
                if(arr[j] - arr[j - 1] == 1)
                {
                    count++;
                    j++;
                }
                else if(arr[j] - arr[j - 1] == 0)
                {
```

```
                    j++;
                }
                else
                {
                    break;
                }
            }
            ret = Math.max(ret, count);
            i = j;
        }
        return ret;
    }
}
```

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，她最多能获取多少分?

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：

```cpp
#include <iostream>

using namespace std;

const int N = 510;

char g[N][N];
int dp[N][N];
int m, n;

int main()
{
    cin >> m >> n;
    for(int i = 1; i <= m; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> g[i][j];
        }
    }

    for(int i = 1; i <= m; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            int t = 0;
            if(g[i][j] == 'l') t = 4;
            else if(g[i][j] == 'o') t = 3;
            else if(g[i][j] == 'v') t = 2;
            else if(g[i][j] == 'e') t = 1;
            dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + t;
        }
    }

    cout << dp[m][n] << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), m = in.nextInt();
        char[][] arr = new char[n + 1][m + 1];

        for(int i = 1; i <= n; i++)
        {
            char[] s = in.next().toCharArray();
            for(int j = 1; j <= m; j++)
            {
                arr[i][j] = s[j - 1];
            }
        }

        int[][] dp = new int[n + 1][m + 1];
        for(int i = 1; i <= n; i++)
        {
            for(int j = 1; j <= m; j++)
            {
                int t = 0;
                if(arr[i][j] == 'l') t = 4;
                else if(arr[i][j] == 'o') t = 3;
                else if(arr[i][j] == 'v') t = 2;
                else if(arr[i][j] == 'e') t = 1;
                dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]) + t;
            }
        }

        System.out.println(dp[n][m]);
    }
}
```

Day09

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<=N<=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≤n≤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：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;
    cin >> s;
    string ret;

    int n = s.size();
    for(int i = 0; i < n; i++)
    {
        ret += s[i];
        if((n - i - 1) % 3 == 0 && i != n - 1) ret += ',';
    }

```

```
    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;
import java.io.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String s = br.readLine();
        int n = s.length();

        for(int i = 0; i < n; i++)
        {
            System.out.print(s.charAt(i));
            if((n - i - 1) % 3 == 0 && i != n - 1)
            {
                System.out.print(',');
            }
        }
    }
}
```

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：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;

    int a = 1, b = 1, c;
    for(int i = 2; i <= n; i++)
    {
        c = a + b;
        a = b;
        b = c;
    }

    if(n == 0 || n == 1) cout << n << endl;
    else cout << c << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();

        int a = 1, b = 1, c = 0;
        for(int i = 2; i <= n; i++)
        {
            c = a + b;
            a = b;
            b = c;
        }

        if(n == 0 || n == 1)
        {
            System.out.println(n);
        }
        else
        {
            System.out.println(c);
        }
    }
}
```

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 $O(nlogn)$，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 <= 4`

C++ 算法代码：

```
class Solution
{
    bool hash[14] = { 0 };

public:
    bool IsContinuous(vector<int>& numbers)
    {
        int maxVal = 0, minVal = 14;
        for(auto x : numbers)
        {
            if(x)
            {
                if(hash[x]) return false;
```

```
                hash[x] = true;
                maxVal = max(maxVal, x);
                minVal = min(minVal, x);
            }
        }
        return maxVal - minVal <= 4;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public boolean IsContinuous (int[] numbers)
    {
        boolean[] hash = new boolean[14];

        int maxVal = 0, minVal = 14;
        for(int x : numbers)
        {
            if(x != 0)
            {
                if(hash[x]) return false;
                hash[x] = true;
                maxVal = Math.max(maxVal, x);
                minVal = Math.min(minVal, x);
            }
        }
        return maxVal - minVal <= 4;
    }
}
```

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：

```cpp
class Solution
{
public:
    int getLongestPalindrome(string s)
    {
        int ret = 1, n = s.size();
        for(int i = 1; i < n; i++)
        {
            // 当长度是奇数的时候
            int left = i - 1, right = i + 1;
            while(left >= 0 && right < n && s[left] == s[right])
            {
                left--;
                right++;
            }
            ret = max(ret, right - left - 1);
            // 当长度是偶数的时候
            left = i - 1, right = i;
```

```
            while(left >= 0 && right < n && s[left] == s[right])
            {
                left--;
                right++;
            }
            ret = max(ret, right - left - 1);
        }
        return ret;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int getLongestPalindrome (String s)
    {
        // 中心扩展算法
        int n = s.length();

        int ret = 0;
        for(int i = 0; i < n; i++) // 枚举所有的中点
        {
            // 当长度为奇数的时候
            int left = i - 1, right = i + 1;
            while(left >= 0 && right < n && s.charAt(left) == s.charAt(right))
            {
                left--;
                right++;
            }
            ret = Math.max(ret, right - left - 1);
            // 当长度为偶数的时候
            left = i; right = i + 1;
            while(left >= 0 && right < n && s.charAt(left) == s.charAt(right))
            {
                left--;
                right++;
            }
            ret = Math.max(ret, right - left - 1);
        }
        return ret;
    }
```

```
32 }
```

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次股票，并非每天都可以买入或卖出一次，总共只能买入和卖出一次，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：

```cpp
#include <iostream>
using namespace std;
```

```cpp

const int N = 1e5 + 10;

int n;
int arr[N];

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++) cin >> arr[i];

    int ret = 0, prevMin = arr[0];
    for(int i = 1; i < n; i++)
    {
        prevMin = min(arr[i], prevMin);
        ret = max(ret, arr[i] - prevMin);
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);

        int n = in.nextInt();
        int ret = 0, prevMin = in.nextInt();
        int x = 0;
        for(int i = 1; i < n; i++)
        {
            x = in.nextInt();
            ret = Math.max(ret, x - prevMin);
            prevMin = Math.min(prevMin, x);
        }

```

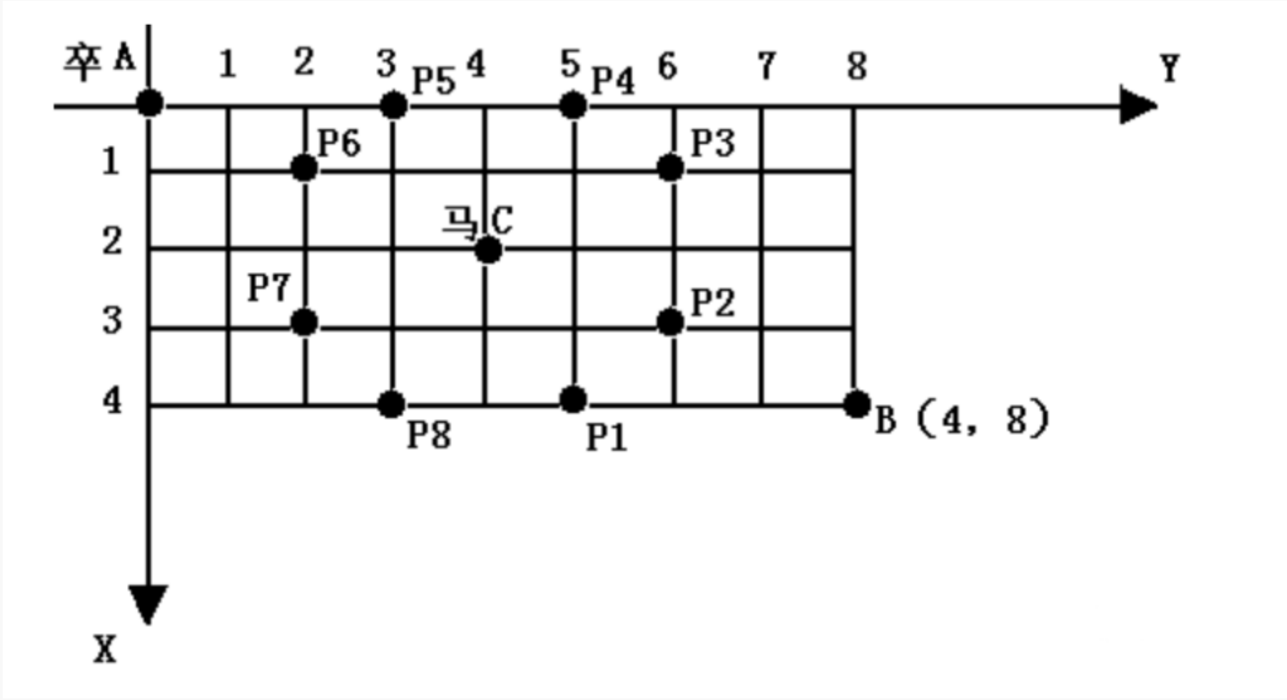
```
20          System.out.println(ret < 0 ? 0 : ret);
21      }
22  }
```

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：

```cpp
#include <iostream>
using namespace std;

int n, m, x, y;
long long dp[25][25];

int main()
{
    cin >> n >> m >> x >> y;

    x += 1; y += 1;
    dp[0][1] = 1;
    for(int i = 1; i <= n + 1; i++)
    {
        for(int j = 1; j <= m + 1; j++)
        {
            if(i != x && j != y && abs(i - x) + abs(j - y) == 3 || (i == x && j == y))
            {
                dp[i][j] = 0;
            }
            else
            {
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
            }
        }
    }

    cout << dp[n + 1][m + 1] << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
```

```
        int n = in.nextInt(), m = in.nextInt(), x = in.nextInt(), y = in.nextInt();

        long[][] dp = new long[n + 2][m + 2];
        dp[0][1] = 1;
        x += 1; y += 1;

        for(int i = 1; i <= n + 1; i++)
        {
            for(int j = 1; j <= m + 1; j++)
            {
                if(i != x && j != y && Math.abs(i - x) + Math.abs(j - y) == 3 || (i == x && j == y))
                {
                    dp[i][j] = 0;
                }
                else
                {
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
                }
            }
        }

        System.out.println(dp[n + 1][m + 1]);
    }
}
```

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：

```cpp
#include <iostream>

using namespace std;

long long n, m, a, b;

int main()
{
    cin >> n >> m >> a >> b;

    long long ret = 0;
    for(long long x = 0; x <= min(n / 2, m); x++) // 枚举 1 号礼包的个数
    {
        long long y = min(n - x * 2, (m - x) / 2); // 计算 2 号礼包的个数
        ret = max(ret, a * x + b * y);
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        long n = in.nextInt();
        long m = in.nextInt();
        long a = in.nextInt();
        long b = in.nextInt();

        long ret = 0;
        for(long x = 0; x <= Math.min(n / 2, m); x++) // 枚举 1 号礼包的个数
        {
            long y = Math.min(n - x * 2, (m - x) / 2); // 计算 2 号礼包的个数
            ret = Math.max(ret, a * x + b * y);
        }
        System.out.println(ret);
    }
}
```

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：

```cpp
#include <iostream>
using namespace std;

const int N = 1e5 + 10;

int n;
int arr[N];

```

```
int main()
{
    cin >> n;
    for(int i = 0; i < n; i++) cin >> arr[i];

    int ret = 0;
    for(int i = 1; i < n; i++)
        if(arr[i] > arr[i - 1])
            ret += arr[i] - arr[i - 1];

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++)
        {
            arr[i] = in.nextInt();
        }

        int ret = 0;
        for(int i = 1; i < n; i++)
        {
            if(arr[i] > arr[i - 1])
            {
                ret += arr[i] - arr[i - 1];
            }
        }
        System.out.println(ret);
    }
}
```

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：

```cpp
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

int main()
{
    string s;
    getline(cin, s);

    reverse(s.begin(), s.end());

    int left = 0, n = s.size();
    while(left < n)
    {
        int right = left;
```

```
        while(right < n && s[right] != ' ') // 找单词
        {
            right++;
        }
        reverse(s.begin() + left, s.begin() + right);
        while(right < n && s[right] == ' ') right++;
        left = right;
    }

    cout << s << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;
import java.io.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void Reverse(char[] s, int left, int right)
    {
        int l = left, r = right;
        while(l < r)
        {
            char ch = s[l];
            s[l] = s[r];
            s[r] = ch;
            l++;
            r--;
        }
    }

    public static void main(String[] args) throws Throwable
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        char[] s = br.readLine().toCharArray();
        int n = s.length;
        Reverse(s, 0, n - 1);

        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：

```cpp
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s, t;
    getline(cin, s);
    getline(cin, t);

    bool hash[300] = { 0 };
    for(char ch : t) hash[ch] = true;

    string ret;
    for(auto ch : s)
    {
        if(!hash[ch])
        {
            ret += ch;
        }
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.Scanner;
import java.io.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        String s = br.readLine();
        String t = br.readLine();

        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）

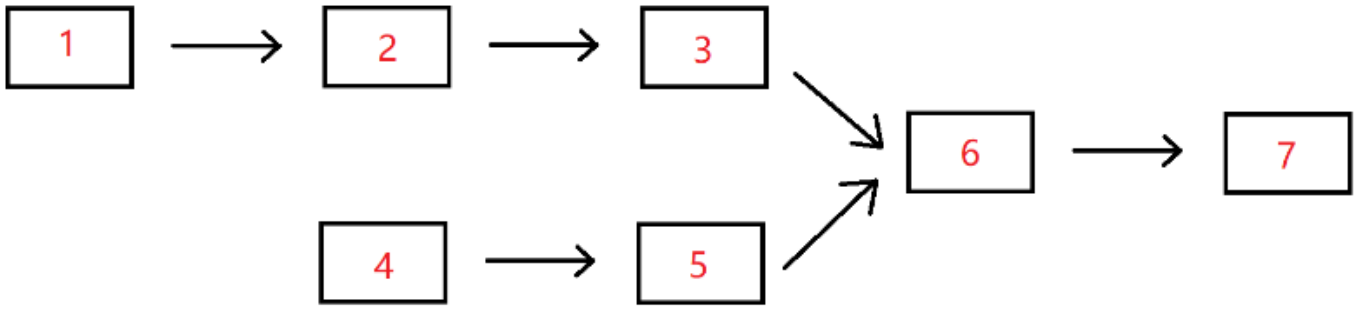
1. **题目链接：** **JZ52 两个链表的第一个公共结点**
2. **题目描述：**

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：

```
/*
struct ListNode {
        int val;
        struct ListNode *next;
        ListNode(int x) :
                        val(x), next(NULL) {
        }
};*/
class Solution
{
public:
    ListNode* FindFirstCommonNode( ListNode* pHead1, ListNode* pHead2)
        {
        ListNode* cur1 = pHead1, *cur2 = pHead2;
                while(cur1 != cur2)
                {
                        cur1 = cur1 != NULL ? cur1->next : pHead2;
                        cur2 = cur2 != NULL ? cur2->next : pHead1;
                }
                return cur1;
    }
};
```

Java 算法代码：

```
import java.util.*;
/*
public class ListNode {
    int val;
    ListNode next = null;

    ListNode(int val) {
        this.val = val;
    }
}*/
public class Solution
{
    public ListNode FindFirstCommonNode(ListNode pHead1, ListNode pHead2)
    {
        ListNode cur1 = pHead1, cur2 = pHead2;
        while(cur1 != cur2)
        {
            cur1 = cur1 != null ? cur1.next : pHead2;
            cur2 = cur2 != null ? cur2.next : pHead1;
```

```
        }
        return cur1;
    }
}
```

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≤n≤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
> sshhyyau
> 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：

```
#include <iostream>
#include <string>

using namespace std;

int n;
string str;

int main()
```

```
{
    cin >> n >> str;

    long long s = 0, h = 0, y = 0;
    for(int i = 0; i < n; i++)
    {
        char ch = str[i];
        if(ch == 's') s++;
        else if(ch == 'h') h += s;
        else if(ch == 'y') y += h;
    }

    cout << y << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        char[] str = in.next().toCharArray();

        long s = 0, h = 0, y = 0;
        for(int i = 0; i < n; i++)
        {
            char ch = str[i];
            if(ch == 's') s += 1;
            else if(ch == 'h') h += s;
            else if(ch == 'y') y += h;
        }
        System.out.println(y);
    }
}
```