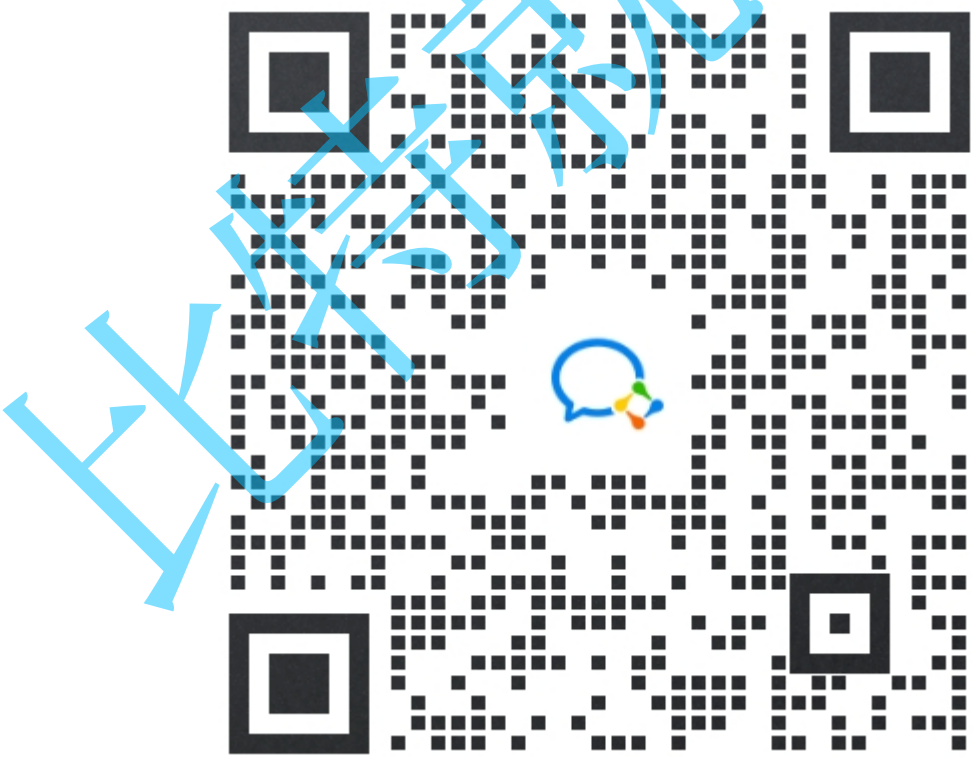
笔试强训第 04 周

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

Day19

1. 小易的升级之路（数学 + 模拟）

（题号：2600154）

1. **题目链接：** WY3 小易的升级之路

2. **题目描述：**

> 题目描述：小易准备打怪升级，他将依次遇到n个怪物，每个怪物的战斗力是a_i。小易初始的战斗力是x。当小易的战力不小于当前遇到怪物的战斗力时，小易会获得该战斗力的数值的战斗力。否则，小易会获得自身战斗力和怪物战斗力的最大公约数的战斗力。小易想知道，自己最终的战斗力是多少？
>
> 输入描述：第一行输入两个正整数n和x，代表怪物的数量、小易输初始的战斗力。
> 第二行输入n个正整数a_i，代表每个怪物的战斗力。
> $1 \leq n \leq 10^5$
> $1 \leq x, a_i \leq 10^9$
>
> 输出描述：一个正整数，代表小易最终的战斗力。
>
> 补充说明：
>
> 示例1
> 输入：3 50
> 50 105 200
> 输出：110
> 说明：小易初始战斗力是50。
> 第一个怪物战斗力不大于小易，所以小易的战斗力变成50+50=100
> 第二个怪物战斗力大于此时小易战斗力，所以小易的战斗力变成100+gcd(100,105)=100+5=105
> 第三个怪物战斗力大于此时小易战斗力，所以小易的战斗力变成105+gcd(105,200)=105+5=110

3. **解法：**

算法思路：

根据题意模拟即可。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

int n, a;

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

```
12
13 int main()
14 {
15     while(cin >> n >> a)
16     {
17         int b;
18         for(int i = 0; i < n; i++)
19         {
20             cin >> b;
21             if(b <= a)
22             {
23                 a += b;
24             }
25             else
26             {
27                 a += gcd(a, b);
28             }
29         }
30         cout << a << endl;
31     }
32
33     return 0;
34 }
```

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         while(in.hasNext())
16         {
17             int n = in.nextInt();
18             int a = in.nextInt();
```

```
            int b = 0;
            for(int i = 0; i < n; i++)
            {
                b = in.nextInt();
                if(b <= a)
                {
                    a += b;
                }
                else
                {
                    a += gcd(a, b);
                }
            }
            System.out.println(a);
        }
    }
}
```

2. 礼物的最大价值（动态规划 - 路径问题）

（题号：2276652）

1. **题目链接：** JZ47 礼物的最大价值

2. **题目描述：**

题目描述：在一个$m \times n$的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？
如输入这样的一个二维数组，
[
[1,3,1],
[1,5,1],
[4,2,1]
]
那么路径 1→3→5→2→1 可以拿到最多价值的礼物，价值为12

补充说明：
- $0 < \text{grid.length} \le 200$
- $0 < \text{grid}[0].\text{length} \le 200$

示例1
输入：[[1,3,1],[1,5,1],[4,2,1]]
输出：12

3. **解法：**

算法思路：

简单路径类 dp 问题。

C++ 算法代码：

```cpp
class Solution
{
    int dp[210][210] = { 0 };

public:
    int maxValue(vector<vector<int> >& grid)
    {
        int m = grid.size(), n = grid[0].size();

        for(int i = 1; i <=m; i++)
        {
            for(int j = 1; j <= n; j++)
            {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]) + grid[i - 1][j - 1];
            }
        }

        return dp[m][n];
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{
    int[][] dp = new int[210][210];

    public int maxValue (int[][] grid)
    {
        int m = grid.length, n = grid[0].length;

        for(int i = 1; i <= m; i++)
        {
            for(int j = 1; j <= n; j++)
            {
                dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]) + grid[i - 1][j - 1];
            }
```

```
        }

        return dp[m][n];
    }
}
```

3. 对称之美（字符串 + 哈希）

（题号：1218157）

1. **题目链接：** 对称之美

2. **题目描述：**

题目描述：给出n个字符串，从第1个字符串一直到第n个字符串每个串取一个字母来构成一个新字符串，新字符串的第i个字母只能从第i行的字符串中选出，这样就得到了一个新的长度为n的字符串，请问这个字符串是否有可能为回文字符串？

输入描述：第一行一个数字 $t, 1 \leq t \leq 50$,代表测试数据的组数

每组测试数据先给出一个数字 n，然后接下来n行每行一个只由小写字母组成的字符串 s_i

$1 \leq n \leq 100, 1 \leq |s_i| \leq 50$

输出描述：在一行中输出 “Yes” or “No”

补充说明：

示例1

输入：
```
2
1
a
3
a
b
c
```

输出：
```
Yes
No
```

说明：

3. **解法：**

算法思路：

左右指针判断回文串。

判断左右指针相等的时候，应该看看两个字符串中有没有相同的字符。

C++ 算法代码：

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

```cpp

using namespace std;

int t, n;
string s;
bool vis[110][26];

bool check(int left, int right)
{
    for(int i = 0; i < 26; i++)
    {
        if(vis[left][i] && vis[right][i]) return true;
    }
    return false;
}

int main()
{
    cin >> t;
    while(t--)
    {
        memset(vis, 0, sizeof vis); // 清空之前的数据
        cin >> n;
        for(int i = 0; i < n; i++)
        {
            cin >> s;
            for(auto ch : s)
            {
                vis[i][ch - 'a'] = true;
            }
        }

        int left = 0, right = n - 1;
        while(left < right)
        {
            if(!check(left, right)) break;
            left++; right--;
        }
        if(left < right) cout << "No" << endl;
        else cout << "Yes" << endl;
    }

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static boolean check(boolean[][] hash, int left, int right)
    {
        for(int i = 0; i < 26; i++)
        {
            if(hash[left][i] && hash[right][i]) return true;
        }
        return false;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        while(t-- != 0)
        {
            int n = in.nextInt();
            // hash[i][j] 表示第 i 个字符串，j 字符是否出现过
            boolean[][] hash = new boolean[n][26];
            for(int i = 0; i < n; i++)
            {
                char[] s = in.next().toCharArray();
                for(char ch : s)
                {
                    hash[i][ch - 'a'] = true;
                }
            }

            int left = 0, right = n - 1;
            while(left < right)
            {
                if(!check(hash, left, right)) break;
                left++; right--;
            }

            if(left < right) System.out.println("No");
            else System.out.println("Yes");
        }
    }
}
```

Day20

1. 经此一役小红所向无敌（模拟）

（题号：1907586）

1. **题目链接：** 经此一役小红所向无敌

2. **题目描述：**

> 题目描述：经过重重困难，对立和光终于来到魔王城，和最终的大魔王——小红进行决战。
> 已知小红的血量是 $10^{999999999}$ 。
> 对立的攻击力是 a ，血量是 h 。
> 光的攻击力是 b ，血量是 k 。
> 每回合光先对小红发起攻击，然后对立对小红发起攻击，然后小红展开幻术，令光和对立同时互相攻击。
> 每次攻击后，受击者的血量会减掉攻击者的攻击力。
> 当光和对立其中一人死亡后，另一人会悲痛欲绝，对小红发出自己攻击力*10的伤害的大招，然后自杀。（若两人同时死亡，则两人都无法发出大招）
> 小红想知道，弱小的光和对立，她们能对自己造成多少点伤害？
>
> 输入描述：一行 4 个正整数 a , h , b , k ，用空格隔开。
>
> $$1 \leq a, b, h, k \leq 10^9$$
>
> 输出描述：一个正整数，代表小红受到的伤害。
>
> 补充说明：
>
> 示例1
> 输入：2 3 1 3
> 输出：26
> 说明：第一回合，小红受到了对立和光的攻击，并让她们互相攻击。第一回合结束时，小红共受到 3 点伤害。这时对立血量为 2 ，光的血量为 1 。
> 第二回合，小红受到了对立和光的攻击，并让她们互相攻击。这时对立血量为 1 ，光的血量为 0 死亡。对立放出大招后自杀。本回合小红共受到 23 点伤害。
> 小红受到的总伤害为 $3 + 23 = 26$ 。

3. **解法：**

算法思路：

根据数据直接计算出结果。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

LL a, h, b, k;
```

```cpp

int main()
{
    cin >> a >> h >> b >> k;

    LL ret = 0;
    // 1. 计算互砍多少回合
    LL n = min(h / b, k / a);
    ret += n * (a + b);

    // 2. 计算剩余血量
    h -= n * b;
    k -= n * a;

    // 3. 判断是否都还活着
    if(h > 0 && k > 0)
    {
        h -= b;
        k -= a;
        ret += a + b;
    }

    // 4. 判断是否会放大
    if(h > 0 || k > 0)
    {
        ret += 10 * (h > 0 ? a : b);
    }

    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 a = in.nextInt(), h = in.nextInt(), b = in.nextInt(), k =
in.nextInt();
```

```
9
10          long ret = 0;
11          // 1. 计算能够互砍多少回合
12          long n = Math.min(h / b, k / a);
13          ret += n * (a + b);
14
15          // 2. 计算剩余血量
16          h -= n * b;
17          k -= n * a;
18
19          // 3. 判断是否都还活着
20          if(h > 0 && k > 0)
21          {
22              h -= b;
23              k -= a;
24              ret += a + b;
25          }
26
27          // 4. 判断是否会放大
28          if(h > 0 || k > 0)
29          {
30              ret += 10 * (h > 0 ? a : b);
31          }
32
33          System.out.println(ret);
34      }
35  }
```


2. 连续子数组最大和（动态规划 - 线性dp）

（题号：2225856）

1. **题目链接：** DP6 连续子数组最大和
2. **题目描述：**

题目描述：给定一个长度为 n 的数组，数组中的数为整数。
请你选择一个**非空连续**子数组，使该子数组所有数之和尽可能大。求这个最大值。

输入描述：第一行为一个正整数 n ，代表数组的长度。 $1 \leq n \leq 10^5$
第二行为 n 个整数 a_i ，用空格隔开，代表数组中的每一个数。 $|a_i| \leq 10^9$

输出描述：连续子数组的最大之和。

补充说明：

示例1
输入：3
3 -4 5
输出：5
说明：选择 [5] 这个子数组即可。

示例2
输入：3
4 -3 5
输出：6
说明：选择 [4,-3,5] 这个子数组。

3. **解法：**

算法思路：

简单线性 dp。

i. 状态表示：`dp[i]` 表示：以 i 位置为结尾的所有子数组中，最大和是多少。

ii. 状态转移方程：`dp[i] = max(dp[i - 1] + arr[i], arr[i])`

C++ 算法代码：

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

const int N = 2e5 + 10;

int n;
int dp[N];
int arr[N];

int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> arr[i];

    int ret = -101;
    for(int i = 1; i <= n; i++)
```

```
    {
        dp[i] = max(dp[i - 1], 0) + arr[i];
        ret = max(ret, dp[i]);
    }

    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 + 1];
        int[] dp = new int[n + 1];

        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextInt();
        }

        int ret = -101;
        for(int i = 1; i <= n; i++)
        {
            dp[i] = Math.max(dp[i - 1], 0) + arr[i];
            ret = Math.max(ret, dp[i]);
        }

        System.out.println(ret);
    }
}
```

3. 非对称之美（规律）

（题号：1218323）

1. **题目链接：** 非对称之美

2. **题目描述：**

> 题目描述：给出一个字符串，求最长非回文子字符串的长度
>
> 输入描述：在一行中给出一个字符串 $s, 1 \leq |s| \leq 10^7$
>
> 输出描述：一个整数
>
> 补充说明：
>
> 示例1
> 输入：meow
> 输出：4
> 说明：

3. **解法：**

算法思路：

找到规律就很好做了。

C++ 算法代码：

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

using namespace std;

int n;
string s;

int fun()
{
    // 1. 判断是否全都是相同字符
    bool flag = false;
    for(int i = 1; i < n; i++)
    {
        if(s[i] != s[0])
        {
            flag = true;
            break;
        }
    }
    if(flag == false) return 0;
```

```
22
23      // 2. 判断本身是否是回文
24      flag = true;
25      int left = 0, right = n - 1;
26      while(left < right)
27      {
28          if(s[left] == s[right])
29          {
30              left++;
31              right--;
32          }
33          else
34          {
35              flag = false;
36              break;
37          }
38      }
39
40      if(flag) return n - 1;
41      else return n;
42  }
43
44  int main()
45  {
46      cin >> s;
47      n = s.size();
48
49      cout << fun() << endl;
50
51      return 0;
52  }
```

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          char[] s = in.next().toCharArray();
9          int n = s.length;
10
```

```
        boolean flag = false;
        for(int i = 1; i < n; i++)
        {
            if(s[i] != s[0])
            {
                flag = true;
                break;
            }
        }

        if(flag == true) // 不是相同的字符
        {
            flag = false;
            // 判断本身是不是回文
            int left = 0, right = n - 1;
            while(left < right)
            {
                if(s[left] == s[right])
                {
                    left++;
                    right--;
                }
                else
                {
                    flag = true;
                    break;
                }
            }
            if(flag == true) System.out.println(n);
            else System.out.println(n - 1);
        }
        else // 是相同字符
        {
            System.out.println(0);
        }
    }
}
```


Day21

1. 爱丽丝的人偶（贪心 + 构造）

（题号：1113894）

1. **题目链接：** 爱丽丝的人偶

2. **题目描述：**

> 题目描述：爱丽丝有n 个人偶，每个人偶的身高依次是1、 2、 $3.....n$
> 现在她要将这n 个人偶摆成一排。
> 但是人偶被设置了魔法。假设对一个非两端的（不在队首也不在队尾）人偶x 而言，她相邻的两个人偶，一个比x 高、一个比x 矮，那么x 就会爆炸。
> 爱丽丝想找到一种摆法，使得所有人偶都不会爆炸。你能帮帮她吗?
>
> 输入描述：一个正整数n $(3 \leq n \leq 100000)$
>
> 输出描述：满足要求的一种摆法。如果有多解，输出任意一种摆法即可。
>
> 补充说明：
>
> 示例1
> 输入：3
> 输出：1 3 2
> 说明：对于第二个人偶，她两边的两个人偶都比她矮，满足要求。
> 另外，[3 1 2]、 [2 1 3] 、[2 3 1]这三种摆法也都满足要求。输出这三种摆法也视为正确。

3. **解法：**

算法思路：

放个小的之后，再放个大的~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

int n;

int main()
{
    cin >> n;

    int left = 1, right = n;

    while(left <= right)
    {
        cout << left << " ";
        left++;
        if(left <= right)
        {
            cout << right << " ";
            right--;
        }
```

```
    }

    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();

        int left = 1, right = n;
        while(left <= right)
        {
            System.out.print(left + " ");
            left++;
            if(left <= right)
            {
                System.out.print(right + " ");
                right--;
            }
        }
    }
}
```


2. 集合（排序）

（题号：105620）

1. **题目链接：** JD7 集合
2. **题目描述：**

题目描述：给你两个集合，要求{A} + {B}。 注：同一个集合中不会有两个相同的元素。

输出时按数字升序输出。

数据范围：$1 \leq n, m \leq 10000$，集合中的元素满足 $1 \leq val \leq 10^5$

输入描述：每组输入数据分为三行,第一行有两个数字n,m，分别表示集合A和集合B的元素个数。后两行分别表示集合A和集合B。每个元素为不超过int范围的整数,每个元素之间有个空格隔开。

输出描述：针对每组数据输出一行数据，表示合并后的集合，要求从小到大输出，每个元素之间有一个空格隔开,行末无空格。

补充说明：

示例1
输入：3 3
1 3 5
2 4 6
输出：1 2 3 4 5 6
说明：

示例2
输入：2 2
1 2
1 2
输出：1 2
说明：

3. 解法：

算法思路：

什么？笔试题？我直接 set 走起！

C++ 算法代码：

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

using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    int x;
    set<int> s;
    for(int i = 0; i < n; i++)
    {
        cin >> x;
        s.insert(x);
    }
    for(int i = 0; i < m; i++)
    {
```

```
        cin >> x;
        s.insert(x);
    }

    for(auto x : s)
    {
        cout << x << " ";
    }

    return 0;
}
```

Java 算法代码：

```
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();

        TreeSet<Integer> set = new TreeSet<>();
        int x;
        while(n-- != 0)
        {
            x = in.nextInt();
            set.add(x);
        }
        while(m-- != 0)
        {
            x = in.nextInt();
            set.add(x);
        }

        for(int a : set)
        {
            System.out.print(a + " ");
        }
    }
}
```

3. 最长回文子序列（动态规划 - 区间 dp）

（题号：2363349）

1. **题目链接：** DP22 最长回文子序列
2. **题目描述：**

> 题目描述：给定一个字符串，找到其中最长的回文子序列，并返回该序列的长度。
>
> 注：回文序列是指这个序列无论从左读还是从右读都是一样的。
> 本题中子序列字符串任意位置删除k（len(s)>=k>=0）个字符后留下的子串。
>
> 数据范围：字符串长度满足 $1 \leq n \leq 1000$
> 进阶：空间复杂度 $O(n^2)$ ，时间复杂度 $O(n^2)$
>
> 输入描述：输入一个字符串
>
> 输出描述：输出最长回文子序列
>
> 补充说明：
>
> 示例1
> 输入：abccsb
> 输出：4
> 说明：分别选取第2、3、4、6位上的字符组成"bccb"子序列是最优解
>
> 示例2
> 输入：abcdewa
> 输出：3
> 说明：分别选取第一个和最后一个a，再取中间任意一个字符就是最优解

3. **解法：**

算法思路：

基础的区间 dp 问题：

1. 状态表示：`dp[i][j]` 表示：字符串 `[i, j]` 范围内的最长回文子序列的长度；
2. 状态转移方程：
   - 当 `i == j` 的时候，只有一个字符，长度为 1；
   - 当 `i < j` 的时候，分情况讨论：
     - `s[i] == s[j]: dp[i][j] = dp[i + 1][j - 1];`
     - `s[i] != s[j]: dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);`
3. 返回值：`dp[0][n - 1]`。

C++ 算法代码：

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

using namespace std;

int dp[1010][1010];

int main()
{
    string s;
    cin >> s;
    int n = s.size();

    for(int i = n - 1; i >= 0; i--)
    {
        dp[i][i] = 1;
        for(int j = i + 1; j < n; j++)
        {
            if(s[i] == s[j]) dp[i][j] = dp[i + 1][j - 1] + 2;
            else dp[i][j] = max(dp[i + 1][j], dp[i][j - 1]);
        }
    }

    cout << dp[0][n - 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);
        char[] s = in.next().toCharArray();
        int n = s.length;
        int[][] dp = new int[n][n];
```

```
12
13          for(int i = n - 1; i >= 0; i--)
14          {
15              dp[i][i] = 1;
16              for(int j = i + 1; j < n; j++)
17              {
18                  if(s[i] == s[j]) dp[i][j] = dp[i + 1][j - 1] + 2;
19                  else dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);
20              }
21          }
22
23          System.out.println(dp[0][n - 1]);
24      }
25  }
```

Day22

1. 添加字符（字符串）

（题号：100344）

1. **题目链接：** [编程题]添加字符

2. **题目描述：**

题目描述：牛牛手里有一个字符串A，羊羊的手里有一个字符串B，B串的长度大于等于A串，所以牛牛想把A串变得和B串一样长，这样羊羊就愿意和牛牛一起玩了。
而且A串的长度增加到和B串一样长的时候，对应的每一位相等的越多，羊羊就越喜欢。比如"abc"和"abd"对应相等的位数为2，为前两位。
牛牛可以在A串的开头或者结尾添加任意字符，使得长度和B串一样。现在问牛牛对A串添加完字符之后，不相等的位数最少有多少位?

输入描述：第一行一个字符串，表示字符串 A。
第二行一个字符串，表示字符串 B。
字符均为小写字母。

$$1 \leq len(A) \leq len(B) \leq 50$$

输出描述：输出一个整数，表示A串添加完字符之后，不相等的位数最少有多少位?

补充说明：

示例1
输入：abe
cabc
输出：1
说明：

3. **解法：**

算法思路：

枚举所有字符串 a 与字符串 b 相对应的位置。

C++ 算法代码：

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

using namespace std;

string a, b;

int main()
{
    cin >> a >> b;
    int m = a.size(), n = b.size();
    int ret = m;

    for(int i = 0; i <= n - m; i++) // 枚举 b 的起始位置
    {
        int tmp = 0;
        for(int j = 0; j < m; j++)
        {
            if(a[j] != b[i + j])
            {
                tmp++;
            }
        }
        ret = min(tmp, ret);
    }

    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);
        char[] a = in.next().toCharArray();
        char[] b = in.next().toCharArray();
        int m = a.length, n = b.length;
        int ret = m;

        for(int i = 0; i <= n - m; i++) // 枚举 b 的起始位置
        {
            int tmp = 0;
            for(int j = 0; j < m; j++)
            {
                if(a[j] != b[i + j])
                {
                    tmp++;
                }
            }
            ret = Math.min(ret, tmp);
        }

        System.out.println(ret);
    }
}
```

2. 数组变换（贪心 + 位运算）

（题号：100345）

1. **题目链接：** [编程题]数组变换
2. **题目描述：**

题目描述：牛牛有一个数组，里面的数可能不相等，现在他想进行一些操作，使数组的所有数相等。问是否可行？
牛牛可以进行的操作：将数组中的任意一个数改为这个数的两倍。该操作可以进行任意次。

输入描述：第一行输入一个正整数 n，代表数组的长度。
第二行输入 n 个正整数 a_i，代表数组的每一个数。

$$1 \leq n \leq 50$$
$$1 \leq a_i \leq 10^9$$

输出描述：如果使得 n 个数都相等，输出"YES"，否则输出"NO"。

补充说明：

示例1
输入：2
1 2
输出：YES
说明：

3. **解法：**

算法思路：

如果能够变换成功，那么最大的数除以剩下的数的商，一定都是 2 的 n 次方。

C++ 算法代码：

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

int b;
int n;
int arr[51];

bool fun()
{
    for(int i = 0; i < n; i++)
    {
        if(b % arr[i]) return false;
        int x = b / arr[i];
        if(x - (x & -x)) return false;
    }
    return true;
}

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

```
    {
        cin >> arr[i];
        b = max(b, arr[i]);
    }

    if(fun()) cout << "YES" << endl;
    else cout << "NO" << 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];
        int b = 0;

        for(int i = 0; i < n; i++)
        {
            arr[i] = in.nextInt();
            b = Math.max(b, arr[i]);
        }

        boolean flag = true;
        for(int i = 0; i < n; i++)
        {
            if(b % arr[i] != 0)
            {
                flag = false;
                break;
            }
            int x = b / arr[i];
            if((x - (x & -x)) != 0)
            {
                flag = false;
                break;
```

```
            }
        }

        if(flag) System.out.println("YES");
        else System.out.println("NO");
    }
}
```

3. 装箱问题（动态规划 - 01 背包）

（题号：170605）

1. **题目链接：** [NOIP2001]装箱问题

2. **题目描述：**

> 题目描述：有一个箱子容量为V（正整数，0 ≤ V ≤ 20000），同时有n个物品（0 < n ≤ 30），每个物品有一个体积（正整数）。
> 要求n个物品中，任取若干个装入箱内，使箱子的剩余空间为最小。
>
> 输入描述：1个整数，表示箱子容量
> 1个整数，表示有n个物品
> 接下来n行，分别表示这n个物品的各自体积
>
> 输出描述：1个整数，表示箱子剩余空间。
>
> 补充说明：
>
> 示例1
> 输入：24
> 6
> 8
> 3
> 12
> 7
> 9
> 7
> 输出：0
> 说明：

3. **解法：**

算法思路：

01 背包简单应用。

C++ 算法代码：

```
#include <iostream>

using namespace std;
```

```
4
5  const int N = 35, M = 2e4 + 10;
6
7  int n, v;
8  int arr[N];
9  int dp[N][M];
10
11 int main()
12 {
13     cin >> v >> n;
14     for(int i = 1; i <= n; i++)
15     {
16         cin >> arr[i];
17     }
18
19     for(int i = 1; i <= n; i++)
20     {
21         for(int j = 0; j <= v; j++)
22         {
23             dp[i][j] = dp[i - 1][j];
24             if(j >= arr[i])
25             {
26                 dp[i][j] = max(dp[i][j], dp[i - 1][j - arr[i]] + arr[i]);
27             }
28         }
29     }
30
31     cout << (v - dp[n][v]) << 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 v = in.nextInt();
9          int n = 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[][] dp = new int[n + 1][v + 1];
17          for(int i = 1; i <= n; i++)
18          {
19              for(int j = 0; j <= v; j++)
20              {
21                  dp[i][j] = dp[i - 1][j];
22                  if(j >= arr[i])
23                  {
24                      dp[i][j] = Math.max(dp[i][j], dp[i - 1][j - arr[i]] +
    arr[i]);
25                  }
26              }
27          }
28
29          System.out.println(v - dp[n][v]);
30      }
31  }
```

Day23

1. 打怪（模拟）

（题号：848858）

1. **题目链接：** 打怪
2. **题目描述：**

题目描述：　你是一个勇士，现在你准备去森林刷毛球怪，你有两个属性（血量，攻击力），毛球怪也有这两个属性。当你遭遇一只毛球怪时你们会进入战斗，然后你和毛球怪轮流攻击（你先手），每次使对方的血量减去自己攻击力的数值，当一方的血量小于等于 0 时死亡。现在你想知道在自己活着的前提下最多杀死几只毛球怪。

输入描述：第一行一个正整数t，代表测试数据组数。
第二行四个正整数h，a，H，A，代表你的血量和攻击力以及毛球怪的血量和攻击力。
所有整数大小不超过1000。

输出描述：共 t 行，每行一个整数x，代表最多能杀死多少毛球怪。如果能杀死无数只，输出-1。

补充说明：

示例1
输入：1
5 1 2 1
输出：4
说明：

3. **解法：**

算法思路：

根据题意模拟，注意一下细节就好了。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

int t;
int h, a, H, A;

int fun()
{
    if(a >= H) return -1;

    int m = (H / a) + (H % a != 0 ? 1 : 0); // 怪物能抗几次
    int n = m - 1; // 玩家被攻击几次
    int x = n * A; // 杀死一只怪物的时候，玩家会掉多少血
    int ret = h / x - (h % x == 0 ? 1 : 0);
    return ret;
}

int main()
{
    cin >> t;
    while(t--)
    {
        cin >> h >> a >> H >> A;
        cout << fun() << endl;
    }

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

```

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

        int h, a, H, A;
        while(t-- != 0)
        {
            h = in.nextInt();
            a = in.nextInt();
            H = in.nextInt();
            A = in.nextInt();

            if(a >= H) System.out.println(-1);
            else
            {
                int m = H / a + (H % a != 0 ? 1 : 0); // 怪物能抗一下
                int n = m - 1; // 杀死一只怪物的时候，玩家被攻击几下
                int x = n * A; // 杀死一只怪物的时候，玩家掉的血量
                int ret = h / x - (h % x == 0 ? 1 : 0);
                System.out.println(ret);
            }
        }
    }
}
```

2. 字符串的分类（哈希 / 排序）

（题号：10055183）

1. **题目链接：** **[编程题]字符串分类**
2. **题目描述：**

题目描述：有 n 个字符串，将这些字符串分类，两个字符串A和B属于同一类需要满足以下条件：
A中交换任意位置的两个字符，最终可以得到B，交换的次数不限。比如：abc与bca就是同一类字符串。
这 n 个字符串可以分成几类?

输入描述：首先输入一个正整数 n，接下来输入N个字符串，每个字符串长度不超过50。

$$1 \leq n \leq 50$$

输出描述：输出一个整数，表示分类的个数。

补充说明：

示例1
输入：4
abcd
abdc
dabc
bacd
输出：1
说明：

3. 解法：

算法思路：

将字符串排序后，丢进能去重的哈希表里面就好了。

C++ 算法代码：

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

using namespace std;

int n;
string s;

int main()
{
    cin >> n;
    unordered_set<string> hash;

    while(n--)
    {
        cin >> s;
        sort(s.begin(), s.end());
        hash.insert(s);
    }

```

```
    cout << hash.size() << 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();
        HashSet<String> set = new HashSet<>();

        while(n-- != 0)
        {
            char[] s = in.next().toCharArray();
            Arrays.sort(s);
            set.add(new String(s));
        }

        System.out.println(set.size());
    }
}
```


3. 城市群数量（联通块）

（题号：2392883）

1. **题目链接：** NC345 城市群数量
2. **题目描述：**

题目描述：给定一个 n 个节点的邻接矩阵 m。 节点定义为城市，如果 a 城市与 b 城市相连， b 与 c 城市相连，尽管 a 与 c 并不直接相连，但可以认为 a 与 c 相连，定义 a,b,c 是一个城市群。
矩阵 m[i][j] = 1 表示第 i 个城市和第 j 个城市直接相连，否则表示不相连。
请你找出共有多少个城市群。

数据范围： $1 \leq n \leq 200$ ， 矩阵中只包含0 和1

补充说明：

示例1
输入：[[1,1,0],[1,1,0],[0,0,1]]
输出：2
说明：1 2 相连，3 独立，因此是 3 个城市群。

示例2
输入：[[1,1,0,0],[1,1,1,0],[0,1,1,0],[0,0,0,1]]
输出：2
说明：1 ， 2相连 ； 2 ， 3相连，4独立，因此是 1,2,3 属于一个城市群，4属于一个城市群。

3. 解法：

算法思路：

经典 `floodfill` 算法，可以用 `dfs` 或者 `bfs` 解决。

C++ 算法代码：

```
class Solution
{
public:
    bool vis[210] = { 0 }; // 用来标记当前位置是否已经搜索过

    int citys(vector<vector<int> >& m)
    {
        int n = m.size();

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

    void dfs(vector<vector<int> >& m, int pos)
    {
        vis[pos] = true;
```

```
25
26          for(int i = 0; i < m.size(); i++)
27          {
28              if(!vis[i] && m[pos][i])
29              {
30                  dfs(m, i);
31              }
32          }
33      }
34  };
```

Java 算法代码：

```
1  import java.util.*;
2
3  public class Solution
4  {
5      int n;
6      boolean[] vis = new boolean[210];
7
8      public int citys (ArrayList<ArrayList<Integer>> m)
9      {
10         n = m.size();
11
12         int ret = 0;
13         for(int i = 0; i < n; i++)
14         {
15             if(vis[i] == false)
16             {
17                 ret++;
18                 dfs(m, i);
19             }
20         }
21         return ret;
22     }
23
24     public void dfs(ArrayList<ArrayList<Integer>> m, int pos)
25     {
26         vis[pos] = true;
27
28         for(int i = 0; i < n; i++)
29         {
30             if(m.get(pos).get(i) == 1 && vis[i] == false)
31             {
```

```
32                  dfs(m, i);
33              }
34          }
35      }
36  }
```

Day24

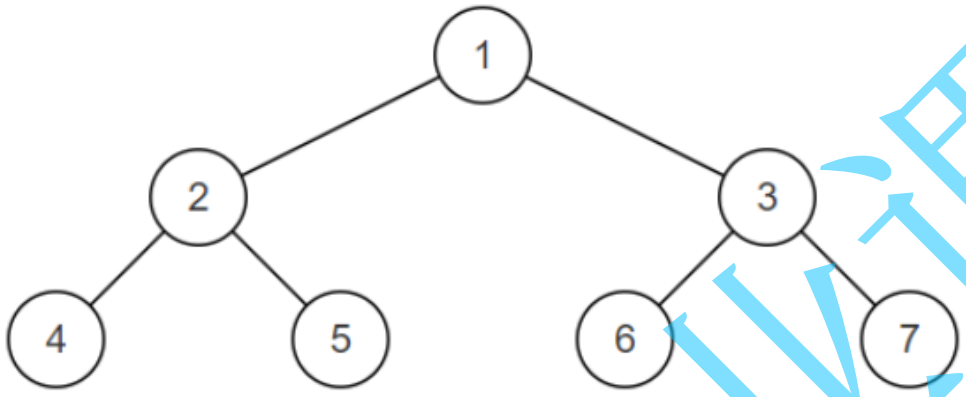
1. 判断是不是平衡二叉树（二叉树 + 递归）

（题号：23250）

1. **题目链接：** **JZ79 判断是不是平衡二叉树**
2. **题目描述：**

描述

输入一棵节点数为 n 二叉树，判断该二叉树是否是平衡二叉树。
在这里，我们只需要考虑其平衡性，不需要考虑其是不是排序二叉树
平衡二叉树（Balanced Binary Tree），具有以下性质：它是一棵空树或它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树。
样例解释：



样例二叉树如图，为一颗平衡二叉树
注：我们约定空树是平衡二叉树。

数据范围：$n \leq 100$,树上节点的val值满足$0 \leq n \leq 1000$
要求：空间复杂度$O(1)$，时间复杂度$O(n)$

输入描述：

输入一棵二叉树的根节点

返回值描述：

输出一个布尔类型的值

示例1

输入：`{1,2,3,4,5,6,7}` 复制

返回值：`true` 复制

3. **解法：**

算法思路：

递归即可。

利用返回值，判断左右子树的高度和左右子树是否是平衡二叉树。

C++ 算法代码：

```cpp
/**
 * struct TreeNode {
 *  int val;
 *  struct TreeNode *left;
 *  struct TreeNode *right;
 *  TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 * };
 */
class Solution
{
public:
    bool IsBalanced_Solution(TreeNode* pRoot)
    {
        return dfs(pRoot) != -1;
    }

    int dfs(TreeNode* root) // 返回值不是 -1 的话，其余的返回值表示的是树高
    {
        if(root == nullptr) return 0;
        int left = dfs(root->left);
        if(left == -1) return -1; // 剪枝
        int right = dfs(root->right);
        if(right == -1) return -1;
        return abs(left - right) <= 1 ? max(left, right) + 1 : -1;
    }
};
```

Java 算法代码：

```java
import java.util.*;

/*
 * public class TreeNode {
 *   int val = 0;
 *   TreeNode left = null;
 *   TreeNode right = null;
 *   public TreeNode(int val) {
 *     this.val = val;
 *   }
```

```
 * }
 */

public class Solution
{
    public boolean IsBalanced_Solution (TreeNode pRoot)
    {
        return dfs(pRoot) != -1;
    }

    public int dfs(TreeNode root) // 当返回值不是 -1 的时候，返回的是树的高度
    {
        if(root == null) return 0;
        int left = dfs(root.left);
        if(left == -1) return -1; // 剪枝
        int right = dfs(root.right);
        if(right == -1) return -1; // 剪枝
        return Math.abs(left - right) <= 1 ? Math.max(left, right) + 1 : -1;
    }
}

```

2. 最大子矩阵（二维前缀和）

（题号：23655）

1. **题目链接：** **DP10 最大子矩阵**

2. **题目描述：**

> 题目描述：已知矩阵的大小定义为矩阵中所有元素的和。给定一个矩阵，你的任务是找到最大的非空(大小至少是1 * 1)子矩阵。 比如，如下4 * 4的矩阵 0 -2 -7 0 9 2 -6 2 -4 1 -4 1 -1 8 0 -2 的最大子矩阵是 9 2 -4 1 -1 8 这个子矩阵的大小是15。
>
> 输入描述：输入是一个N * N的矩阵。输入的第一行给出N (0 < N <= 100)。 再后面的若干行中，依次（首先从左到右给出第一行的N个整数，再从左到右给出第二行的N个整数……）给出矩阵中的N2个整数，整数之间由空白字符分隔（空格或者空行）。 已知矩阵中整数的范围都在[-127, 127]。
>
> 输出描述：输出最大子矩阵的大小。
>
> 补充说明：

示例1
输入：4
```
0 -2 -7 0
9 2 -6 2
-4 1 -4  1
-1 8  0 -2
```
输出：15
说明：

3. **解法：**

算法思路：

二维前缀和矩阵的应用。

a. 初始化二维前缀和矩阵；

b. 枚举所有的子矩阵，求出最大子矩阵。

C++ 算法代码：

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

const int N = 110;

int n;
int dp[N][N];

int main()
{
    int x;
    cin >> n;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            cin >> x;
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1] - dp[i - 1][j - 1] + x;
        }
    }

    int ret = -127 * N;
    for(int x1 = 1; x1 <= n; x1++)
    {
        for(int y1 = 1; y1 <= n; y1++)
        {
            for(int x2 = x1; x2 <= n; x2++)
            {
                for(int y2 = y1; y2 <= n; y2++)
                {
                    ret = max(ret, dp[x2][y2] - dp[x1 - 1][y2] - dp[x2][y1 - 1] + dp[x1 - 1][y1 - 1]);
                }
            }
        }
    }
```

```
36
37      cout << ret << endl;
38
39      return 0;
40  }
```

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 int n;
7      public static int[][] dp = new int[110][110];
8
9      public static void main(String[] args)
10     {
11         Scanner in = new Scanner(System.in);
12         n = in.nextInt();
13
14         for(int i = 1; i <= n; i++)
15         {
16             for(int j = 1; j <= n; j++)
17             {
18                 int x = in.nextInt();
19                 dp[i][j] = dp[i - 1][j] + dp[i][j - 1] - dp[i - 1][j - 1] + x;
20             }
21         }
22
23         int ret = -127;
24         for(int x1 = 1; x1 <= n; x1++)
25         {
26             for(int y1 = 1; y1 <= n; y1++)
27             {
28                 for(int x2 = x1; x2 <= n; x2++)
29                 {
30                     for(int y2 = y1; y2 <= n; y2++)
31                     {
32                         ret = Math.max(ret, dp[x2][y2] - dp[x1 - 1][y2] -
   dp[x2][y1 - 1] + dp[x1 - 1][y1 - 1]);
33                     }
34                 }
35             }
```

```
36          }
37          System.out.println(ret);
38      }
39  }
```

3. 小葱的01串（滑动窗口）

（题号：2310079）

1. **题目链接：** **小葱的01串**

2. **题目描述：**

> 题目描述：给定一个长度为偶数的**环形** 01 字符串。（环形指，第一个字符和最后一个字符是相邻的）
> 字符串初始每个字符都是白色。小葱想把一段**连续区间**染成红色，使得红色的字符'0'数量等于白色的字符'0'数量，红色的字符'1'数量等于白色的字符'1'数量。问有多少种不同的染色方法?
> 两个方案不同当且仅当存在一个某字符，在一个方案是染成红色，在另一个方案为白色。
>
> 输入描述：第一行输入一个正整数 n，代表字符串长度。
> 第二行输入一个长度为 n 的 01 字符串（仅由字符'0'和字符'1'组成的字符串）
> 数据范围：
> $2 \leq n \leq 300000$ 。保证 n 是偶数。
>
> 输出描述：合法的染色方案数。
>
> 补充说明：
>
> 示例1
> 输入：2
> 11
> 输出：2
> 说明：将第一个数字染红为一个方案。
> 将第二个数字染红为一个方案。
>
> 示例2
> 输入：4
> 0101
> 输出：4
> 说明：任意一个长度为2的区间染红均合法。

3. **解法：**

算法思路：

滑动窗口：

长度固定的滑动窗口，因为要想符合要求，必定是一半一半的。

C++ 算法代码：

```
1  #include <iostream>
2  #include <string>
```

```cpp

using namespace std;

int n;
string s;

int main()
{
    cin >> n >> s;
    int sum[2] = { 0 }; // 统计字符串中所有 0 和 1 的个数
    for(auto ch : s)
    {
        sum[ch - '0']++;
    }

    int left = 0, right = 0, ret = 0, half = n / 2;
    int count[2] = { 0 }; // 统计窗口内 0 和 1 的个数
    while(right < n - 1) // 细节问题
    {
        count[s[right] - '0']++;
        while(right - left + 1 > half)
        {
            count[s[left++] - '0']--;
        }
        if(right - left + 1 == half)
        {
            if(count[0] * 2 == sum[0] && count[1] * 2 == sum[1])
            {
                ret += 2;
            }
        }
        right++;
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
```

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

        int[] sum = new int[2]; // 统计字符串中所有 0 和 1 的个数
        for(int i = 0; i < n; i++)
        {
            sum[s[i] - '0']++;
        }

        int left = 0, right = 0, ret = 0, half = n / 2;
        int[] count = new int[2]; // 统计窗口内 0 和 1 的个数
        while(right < n - 1) // 细节问题
        {
            count[s[right] - '0']++;
            while(right - left + 1 > half)
            {
                count[s[left++] - '0']--;
            }
            if(right - left + 1 == half)
            {
                if(count[0] * 2 == sum[0] && count[1] * 2 == sum[1])
                {
                    ret += 2;
                }
            }
            right++;
        }

        System.out.println(ret);
    }
}
```