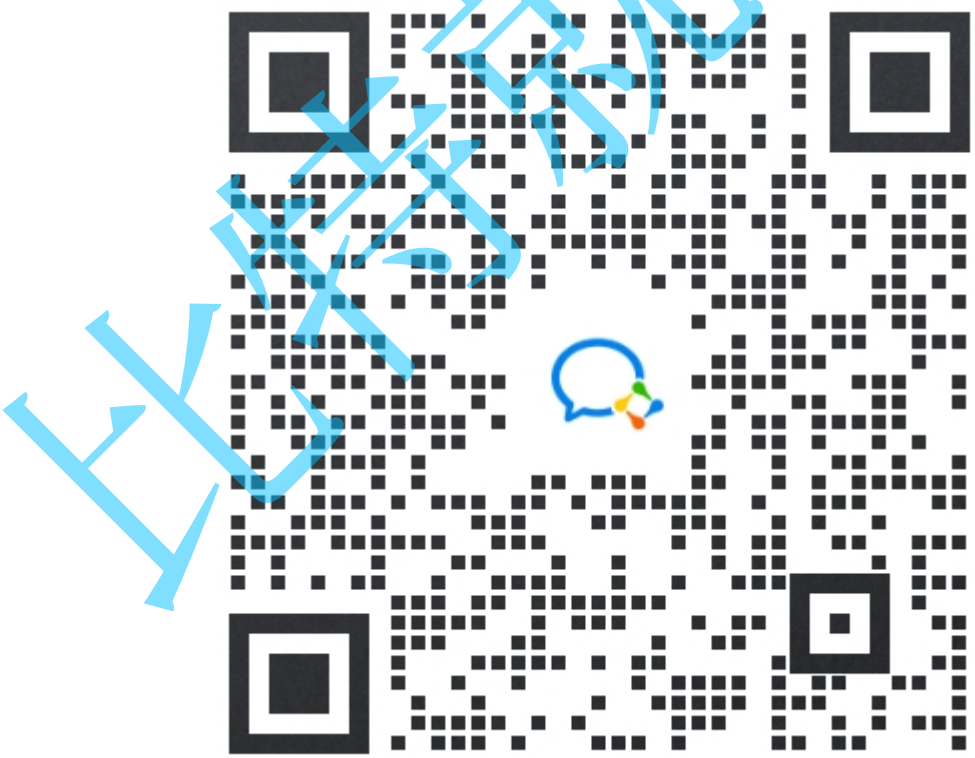
笔试强训第 06 周

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

Day31

1. 小红的口罩（贪心 + 堆）

（题号：2283787；类型：贪心 + 堆模拟）

1. **题目链接：** 小红的口罩

2. **题目描述：**

> 题目描述：疫情来了，小红网购了 n 个口罩。
> 众所周知，戴口罩是很不舒服的。小红每个口罩戴一天的初始不舒适度为 a_i。
> 小红有时候会将口罩重复使用（注：这是非常不卫生的！），每次重复使用时，该口罩的不舒适度会翻倍！
> 小红想知道，自己在不舒适度总和不超过 k 的情况下，最多能用现有的口罩度过多少天？
>
> 输入描述：第一行输入两个正整数 n 和 k，分别代表口罩的总数、以及小红最多能忍受的不舒适度总和。
> 第二行输入 n 个正整数 a_i，用空格隔开。分别代表每个口罩初始的不舒适度。
> $1 \leq n \leq 10^5, 1 \leq a_i, k \leq 10^9$
>
> 输出描述：一个整数，代表小红最多能度过的天数。
>
> 补充说明：

> 示例1
> 输入：2 30
> 2 3
> 输出：5
> 说明：第一天用第一个口罩，不舒适度为2。
> 第二天用第一个口罩，不舒适度为4。
> 第三天用第二个口罩，不舒适度为3。
> 第四天用第二个口罩，不舒适度为6。
> 第五天用第二个口罩，不舒适度为12。
> 总不舒适度为2+4+3+6+12=27，没有超过30。
> 可以证明，无论怎样分配，都无法度过6天且不舒适度总和不超过30

3. **解法：**

算法思路：

小贪心 + 堆，没难度~

C++ 算法代码：

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

using namespace std;

int n, k;

```

```cpp
int main()
{
    cin >> n >> k;
    priority_queue<int, vector<int>, greater<int>> heap;
    for(int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        heap.push(x);
    }

    int sum = 0, count = 0;
    while(true)
    {
        int t = heap.top();
        heap.pop();
        sum += t;
        heap.push(t * 2);
        count++;
        if(sum > k)
        {
            cout << count - 1 << endl;
            break;
        }
    }

    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);
        int n = in.nextInt(), k = in.nextInt();
        PriorityQueue<Integer> heap = new PriorityQueue<>();
        for(int i = 0; i < n; i++)
        {
            int x = in.nextInt();
            heap.add(x);
```

```
        }

        int sum = 0, count = 0;
        while(true)
        {
            int t = heap.poll();
            sum += t;
            count++;
            heap.add(t * 2);
            if(sum > k)
            {
                System.out.println(count - 1);
                break;
            }
        }
    }
}
```

2. 春游（模拟 - 分情况讨论）

（题号：1389158）

1. **题目链接：** 春游
2. **题目描述：**

题目描述：盼望着，盼望着，东风来了，春天脚步近了。

值此大好春光，老师组织了同学们出去划船，划船项目收费如下：

双人船最多坐两人，也可以坐一人，收费a元

三人船最多坐三人，也可以坐两人或者一人，收费b元

本次出游加上带队老师共n人，如何安排能使得花费最小呢？

输入描述：第一行给出一个正整数$T(1 \leq T \leq 1000)$，代表测试数据的组数。

接下来T行每行给出三个正整数$n, a, b, 1 \leq n, a, b \leq 10^9$，含义如题。

输出描述：每组输入输出一行，代表最小的花费

补充说明：

示例1

输入：2
2 20 200
3 20 20

输出：20
20

3. **解法：**

算法思路：

贪心 + 分情况讨论。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;
typedef long long LL;

LL t;
LL n, a, b;

LL fun()
{
    // 边界情况
    if(n <= 2) return min(a, b);
    LL ret = 0;
    if(a * 3 < b * 2) // 尽可能的选择双人船
    {
        ret += n / 2 * a;
```

```
        n %= 2;
        if(n) ret += min(min(a, b), b - a);
    }
    else // 尽可能的选择三人船
    {
        ret += n / 3 * b;
        n %= 3;
        if(n == 1) ret += min(min(a, b), 2 * a - b);
        if(n == 2) ret += min(min(a, b), 3 * a - b);
    }
    return ret;
}

int main()
{
    cin >> t;
    while(t--)
    {
        cin >> n >> a >> b;
        cout << fun() << endl;
    }

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static void main(String[] s)
    {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        while(t-- != 0)
        {
            long n = in.nextLong(), a = in.nextLong(), b = in.nextLong();
            long ret = 0;
            if(n <= 2) // 边界情况
            {
                ret = Math.min(a, b);
            }
            else
```

```
            {
                if(a * 3 < b * 2) // 尽可能的选择双人船
                {
                    ret += n / 2 * a;
                    n %= 2;
                    if(n == 1)
                    {
                        ret += Math.min(Math.min(a, b), b - a);
                    }
                }
                else // 尽可能的选择三人船
                {
                    ret += n / 3 * b;
                    n %= 3;
                    if(n == 1) ret += Math.min(Math.min(a, b), 2 * a - b);
                    if(n == 2) ret += Math.min(Math.min(a, b), 3 * a - b);
                }
            }
            System.out.println(ret);
        }
    }
}
```

3. 数位染色（动态规划 - 01背包）

（题号：1815295）

1. **题目链接：** DP59 数位染色

2. **题目描述：**

题目描述：小红拿到了一个正整数 x 。她可以将其中一些数位染成红色。然后她想让所有染红的数位数字之和等于没染色的数位数字之和。她不知道能不能达成目标。你能告诉她吗？

输入描述：一个正整数 x ，$1 \le x \le 10^{18}$

输出描述：如果小红能按要求完成染色，输出"Yes"。否则输出"No"。

补充说明：

示例1
输入：1234567
输出：Yes
说明：将3、4、7染成红色即可，这样3+4+7=1+2+5+6

示例2
输入：23
输出：No
说明：显然无论如何都不能完成染色。

3. **解法：**

算法思路：

01 背包应用题~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 20, M = N * 9;

long long x;
int n, sum; // 这个数有多少位
int arr[N];
bool dp[M];

bool fun()
{
    if(sum % 2 == 1) return false;
    sum /= 2;
    dp[0] = true;
    for(int i = 0; i < n; i++)
    {
        for(int j = sum; j >= arr[i]; j--)
        {
            dp[j] = dp[j] || dp[j - arr[i]];
        }
    }
    return dp[sum];
}

int main()
{
    cin >> x;
    while(x)
    {
        arr[n++] = x % 10;
        sum += x % 10;
        x /= 10;
    }

    if(fun()) cout << "Yes" << endl;
    else cout << "No" << endl;
```

```
39
40      return 0;
41  }
```

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);
        long x = in.nextLong();
        int[] arr = new int[20];
        int n = 0, sum = 0;
        while(x != 0)
        {
            int t = (int)(x % 10);
            arr[n++] = t;
            sum += t;
            x /= 10;
        }
        if(sum % 2 == 1)
        {
            System.out.println("No");
        }
        else
        {
            sum /= 2;
            boolean[] dp = new boolean[sum + 1];
            dp[0] = true;
            for(int i = 0; i < n; i++)
            {
                for(int j = sum; j >= arr[i]; j--)
                {
                    dp[j] = dp[j] || dp[j - arr[i]];
                }
            }
            if(dp[sum]) System.out.println("Yes");
            else System.out.println("No");
        }
    }
```

```
39 }
```

Day32

1. 素数回文（模拟 + 数学）

（题号：140152）

1. **题目链接：** **BC157 素数回文**

2. **题目描述：**

> 题目描述：现在给出一个素数，这个素数满足两点：
> 1、 只由1-9组成，并且每个数只出现一次，如13,23,1289。
> 2、 位数从高到低为递减或递增，如2459，87631。
> 请你判断一下，这个素数的回文数是否为素数（13的回文数是131,127的回文数是12721）。
>
> 输入描述：输入只有1行。
> 第1行输入一个整数t，保证t为素数。
> 数据保证：$9<t<10^9$
>
> 输出描述：输出一行字符串，如果t的回文数仍是素数，则输出“prime”，否则输出"noprime"。
>
> 补充说明：素数定义为在大于1的自然数中，除了1和它本身以外不再有其他因数。
> 素数的回文数为题意中的定义，1331不是素数的回文数。

```
示例1
输入：13
输出：prime
说明：13的回文数是131，131是素数

示例2
输入：17
输出：noprime
说明：17的回文数是171,171不是素数（因子有3）
```

3. **解法：**

算法思路：

模拟题，注意数据范围~

C++ 算法代码：

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

using namespace std;
```

```cpp

long long change(string s)
{
    for(int i = s.size() - 2; i >= 0; i--)
    {
        s += s[i];
    }
    return stol(s);
}

bool isprim(long long x)
{
    if(x <= 1) return false;
    for(long long i = 2; i <= sqrt(x); i++)
    {
        if(x % i == 0) return false;
    }
    return true;
}

int main()
{
    string s;
    cin >> s;
    long long x = change(s);
    if(isprim(x)) cout << "prime" << endl;
    else cout << "noprime" << 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 long change(String s)
    {
        StringBuffer tmp = new StringBuffer(s);
        for(int i = s.length() - 2; i >= 0; i--)
        {
            tmp.append(s.charAt(i));
```

```java
        }
        return Long.parseLong(tmp.toString());
    }

    public static boolean isprim(long x)
    {
        if(x <= 1) return false;
        for(long i = 2; i <= Math.sqrt(x); i++)
        {
            if(x % i == 0) return false;
        }
        return true;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        String s = in.next();
        long x = change(s);
        if(isprim(x)) System.out.println("prime");
        else System.out.println("noprime");
    }
}
```

2. 活动安排（贪心 - 区间）

（题号：2373697）

1. **题目链接：** AB31 活动安排
2. **题目描述：**

> 题目描述：给定n个活动，每个活动安排的时间为$[a_i, b_i)$。求最多可以选择多少个活动，满足选择的活动时间两两之间没有重合。
>
> 输入描述：第一行输入一个整数n ($1 \leq n \leq 2 \cdot 10^5$)，表示可选活动个数。
> 接下来的n行，每行输入两个整数a_i, b_i ($0 \leq a_i < b_i \leq 10^9$)，表示第$i$个活动的时间。
>
> 输出描述：输出一行一个整数，表示最多可选择的活动数。
>
> 补充说明：
>
> 示例1
> 输入：3
> 1 4
> 1 3
> 3 5
> 输出：2
> 说明：

3. **解法：**

算法思路：

区间问题的贪心：排序，然后分情况讨论，看看是合并还是求交集

C++ 算法代码：

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

typedef pair<int, int> PII;

const int N = 2e5 + 10;

int n;
PII arr[N];

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++) cin >> arr[i].first >> arr[i].second;
    sort(arr, arr + n);

    int ret = 0, r = arr[0].second;
    for(int i = 1; i < n; i++)
    {
        if(arr[i].first < r) // 有重叠
        {
            r = min(r, arr[i].second);
        }
        else // 没有重叠
        {
            ret++;
            r = arr[i].second;
        }
    }
    cout << ret + 1 << 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();
        int[][] arr = new int[n][2];
        for(int i = 0; i < n; i++)
        {
            arr[i][0] = in.nextInt();
            arr[i][1] = in.nextInt();
        }
        Arrays.sort(arr, (a, b) ->
        {
            return a[0] <= b[0] ? -1 : 1;
        });

        int ret = 0, r = arr[0][1];
        for(int i = 1; i < n; i++)
        {
            if(arr[i][0] < r) // 有重叠
            {
                r = Math.min(r, arr[i][1]);
            }
            else // 没有重叠
            {
                ret++;
                r = arr[i][1];
            }
        }
        System.out.println(ret + 1);
    }
}
```


3. 合唱团（动态规划 - 线性 dp）

（题号：2600642）

1. **题目链接：** WY6 合唱团
2. **题目描述：**

题目描述：有n个学生站成一排，每个学生有一个能力值a_i，牛牛想从这n个学生中按照顺序选取k名学生，要求相邻两个学生的位置的差不超过d。
牛牛想要使得这k个学生的能力值的乘积最大，你能返回最大的乘积吗？

输入描述：第一行包含三个整数n，k和d $(1 \le n \le 1000, 1 \le k \le 10, 1 \le d \le n)$，如题意所示。
接下来的一行，包含n个整数，按顺序表示每个学生的能力值 $a_i (-50 \le a_i \le 50)$。

输出描述：输出一行表示最大的乘积。

补充说明：

示例1
输入：3 2 3
7 4 7
输出：49
说明：选取第一个和第三个学生乘积最大，为49。

3. 解法：

算法思路：

线性 dp。

C++ 算法代码：

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

typedef long long LL;

const int N = 55, M = 15;
const LL INF = 0x3f3f3f3f3f3f3f3f;

int n, k, d;
LL arr[N];
LL f[N][M], g[N][M];

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

    // 初始化放在填表中进行了

    for(int i = 1; i <= n; i++) // 填写每一行
    {
        g[i][1] = f[i][1] = arr[i];
        for(int j = 2; j <= min(i, k); j++) // 挑选几个人
        {
            f[i][j] = -INF; // 初始化
```

```
            g[i][j] = INF;  // 初始化
            for(int prev = max(i - d, j - 1); prev <= i - 1; prev++) // 前面挑选的最后一个位置
            {
                f[i][j] = max(max(f[prev][j - 1] * arr[i], g[prev][j - 1] * arr[i]), f[i][j]);
                g[i][j] = min(min(f[prev][j - 1] * arr[i], g[prev][j - 1] * arr[i]), g[i][j]);
            }
        }
    }

    LL ret = -INF;
    for(int i = k; i <= n; i++) ret = max(ret, f[i][k]);
    cout << ret << endl;

    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 int N = 55, M = 15;
    // public static long INF = 0x3f3f3f3f3f3f3f3f; // 报错

    public static int n, k, d;
    public static long[] arr = new long[N];
    public static long[][] f = new long[N][M];
    public static long[][] g = new long[N][M];

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        for(int i = 1; i <= n; i++)
        {
            arr[i] = in.nextLong();
        }
        k = in.nextInt(); d = in.nextInt();

```

```
        // 初始化放在填表中进行

        for(int i = 1; i <= n; i++) // 填写每一行
        {
            f[i][1] = g[i][1] = arr[i];
            for(int j = 2; j <= Math.min(k, i); j++) // 挑选 j 个人
            {
                f[i][j] = Long.MIN_VALUE; // 初始化
                g[i][j] = Long.MAX_VALUE; // 初始化
                for(int prev = Math.max(i - d, j - 1); prev <= i - 1; prev++)
                {
                    f[i][j] = Math.max(Math.max(f[prev][j - 1] * arr[i],
g[prev][j - 1] * arr[i]), f[i][j]);
                    g[i][j] = Math.min(Math.min(f[prev][j - 1] * arr[i],
g[prev][j - 1] * arr[i]), g[i][j]);
                }
            }
        }

        long ret = Long.MIN_VALUE;
        for(int i = k; i <= n; i++) ret = Math.max(ret, f[i][k]);

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

Day33

1. 跳台阶扩展问题（规律）

（题号：2361300）

1. **题目链接：** DP3 跳台阶扩展问题
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……它也可以跳上n级。求该青蛙跳上一个n级的台阶(n为正整数)总共有多少种跳法。

数据范围：$1 \leq n \leq 20$
进阶：空间复杂度 $O(1)$ ，时间复杂度 $O(1)$

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 级台阶的跳法

补充说明：

示例1
输入：3
输出：4
说明：

示例2
输入：1
输出：1
说明：

3. 解法：

算法思路：

没想到吧，居然是一道规律题~

C++ 算法代码：

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

int main()
{
    int n;
    cin >> n;
    cout << (1 << (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
{
```

```java
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        System.out.println(1 << (n - 1));
    }
}
```

2. 包含不超过两种字符的最长子串（滑动窗口）

（题号：2454794）

1. **题目链接：** **NC402 包含不超过两种字符的最长子串**
2. **题目描述：**

> 题目描述：给定一个长度为 n 的字符串，找出最多包含两种字符的最长子串 t，返回这个最长的长度。
>
> 数据范围：$1 \leq n \leq 10^5$ ，字符串种仅包含小写英文字母
>
> 输入描述：仅一行，输入一个仅包含小写英文字母的字符串
>
> 输出描述：输出最长子串的长度
>
> 补充说明：
>
> 示例1
> 输入：nowcoder
> 输出：2
> 说明：
>
> 示例2
> 输入：nooooow
> 输出：6
> 说明：

3. **解法：**

算法思路：

简单滑动窗口~

C++ 算法代码：

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

```cpp

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

    int left = 0, right = 0, n = s.size();
    int hash[26] = { 0 }; // 统计窗口内每种字符出现了多少次
    int count = 0; // 统计窗口内一共有多少种字符
    int ret = 0;

    while(right < n)
    {
        if(hash[s[right] - 'a']++ == 0) count++; // 0->1, 窗口内多了一种字符
        while(count > 2)
        {
            if(hash[s[left++] - 'a']-- == 1) count--; // 1->0, 窗口内少了一种字符
        }
        ret = max(ret, right - left + 1);
        right++;
    }

    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[] s = in.next().toCharArray();

        int left = 0, right = 0, n = s.length;
        int count = 0; // 统计窗口内有多少种字符
        int[] hash = new int[26]; // 统计窗口内每种字符出现的次数
        int ret = 0;

```

```java
        while(right < n)
        {
            if(hash[s[right] - 'a']++ == 0) // 0->1，窗口内多了一种字符
            {
                count++;
            }
            while(count > 2)
            {
                if(hash[s[left++] - 'a']-- == 1) // 1->0，窗口内少了一种字符
                {
                    count--;
                }
            }
            ret = Math.max(ret, right - left + 1);
            right++;
        }
        System.out.println(ret);
    }
}
```

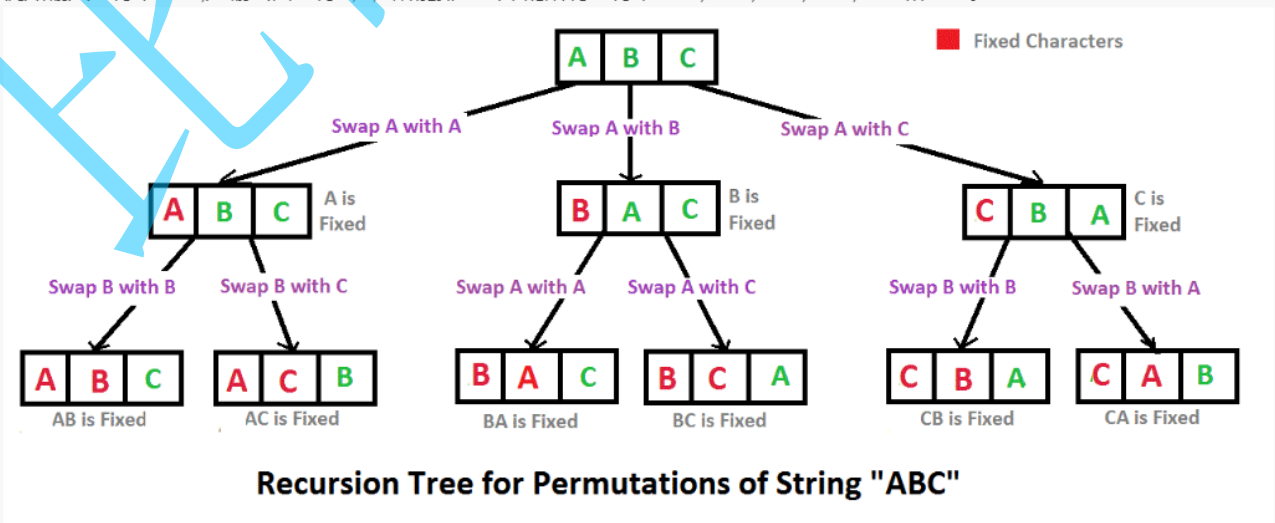
3. 字符串的排列（DFS 枚举）

（题号：23291）

1. **题目链接：** JZ38 字符串的排列

2. **题目描述：**

题目描述：输入一个长度为 n 字符串，打印出该字符串中字符的所有排列，你可以以任意顺序返回这个字符串数组。
例如输入字符串ABC,则输出由字符A,B,C所能排列出来的所有字符串ABC,ACB,BAC,BCA,CBA和CAB。



Recursion Tree for Permutations of String "ABC"

数据范围：$n < 10$
要求：空间复杂度 $O(n!)$，时间复杂度 $O(n!)$

3. **解法：**

算法思路：

递归实现全排列，注意剪枝~

C++ 算法代码：

```cpp
class Solution {
public:
    vector<string> ret; // 收集叶子结点
    string path; // 记录路径的信息
    bool vis[11] = { 0 }; // 标记当前位置时候已经使用过
    int n;
    string s;

    vector<string> Permutation(string str)
    {
        n = str.size();
        sort(str.begin(), str.end());
        s = str;

        dfs(0);
        return ret;
    }

    void dfs(int pos)
    {
        if(pos == n)
        {
            ret.push_back(path);
            return;
        }

        // 填 pos 位置
        for(int i = 0; i < n; i++)
        {
            if(!vis[i])
            {
                if(i > 0 && s[i] == s[i - 1] && !vis[i - 1]) continue;
                path.push_back(s[i]);
                vis[i] = true;
                dfs(pos + 1);
                // 恢复现场
                vis[i] = false;
                path.pop_back();
            }
```

```
        }
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{
    boolean[] vis = new boolean[15]; // 标记当前位置是否已经使用过
    StringBuffer path = new StringBuffer();
    ArrayList<String> ret = new ArrayList<>(); // 收集叶子结点
    char[] s;
    int n;

    public ArrayList<String> Permutation (String str)
    {
        n = str.length();
        s = str.toCharArray();
        Arrays.sort(s);

        dfs(0);
        return ret;
    }

    public void dfs(int pos) // 填哪个位置
    {
        if(pos == n) // 收集结果
        {
            ret.add(path.toString());
            return;
        }

        // 填 pos 位置
        for(int i = 0; i < n; i++)
        {
            if(vis[i]) continue;
            if(i > 0 && s[i] == s[i - 1] && !vis[i - 1]) continue;
            path.append(s[i]);
            vis[i] = true;
            dfs(pos + 1);
            // 恢复现场
            vis[i] = false;
```

```
39                 path.deleteCharAt(path.length() - 1);
40             }
41         }
42 }
```

Day34

1. ISBN号码（模拟）

（题号：170470）

1. **题目链接：** BC76 [NOIP2008]ISBN号码

2. **题目描述：**

> 题目描述：每一本正式出版的图书都有一个ISBN号码与之对应，ISBN码包括9位数字、1位识别码和3位分隔符，其规定格式如“x-xxx-xxxxx-x”，其中符号“-”是分隔符（键盘上的减号），最后一位是识别码，例如0-670-82162-4就是一个标准的ISBN码。ISBN码的首位数字表示书籍的出版语言，例如0代表英语；第一个分隔符“-”之后的三位数字代表出版社，例如670代表维京出版社；第二个分隔之后的五位数字代表该书在出版社的编号；最后一位为识别码。
> 识别码的计算方法如下：
> 首位数字乘以1加上次位数字乘以2……以此类推，用所得的结果mod 11，所得的余数即为识别码，如果余数为10，则识别码为大写字母X。例如ISBN号码0-670-82162-4中的识别码4是这样得到的：对067082162这9个数字，从左至右，分别乘以1，2，…，9，再求和，即0×1+6×2+……+2×9=158，然后取158 mod 11的结果4作为识别码。
> 你的任务是编写程序判断输入的ISBN号码中识别码是否正确，如果正确，则仅输出“Right”；如果错误，则输出你认为是正确的ISBN号码。
>
> 输入描述：只有一行，是一个字符序列，表示一本书的ISBN号码（保证输入符合ISBN号码的格式要求）。
>
> 输出描述：共一行，假如输入的ISBN号码的识别码正确，那么输出“Right”，否则，按照规定的格式，输出正确的ISBN号码（包括分隔符“-”）。
>
> 补充说明：

```
示例1
输入：0-670-82162-4
输出：Right
说明：

示例2
输入：0-670-82162-0
输出：0-670-82162-4
说明：
```

3. **解法：**

算法思路：

模拟模拟，首先读懂题意，其次注意细节~

C++ 算法代码：

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

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

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

    int sum = 0, count = 1, n = s.size();
    for(int i = 0; i < n - 1; i++)
    {
        if(s[i] >= '0' && s[i] <= '9')
        {
            sum += (s[i] - '0') * count;
            count++;
        }
    }
    sum %= 11;

    if(sum == s[n - 1] - '0' || (sum == 10 && s[n - 1] == 'X'))
    {
        cout << "Right" << endl;
    }
    else
    {
        s[n - 1] = sum == 10 ? 'X' : sum + '0';
        cout << s << 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();

```

```
11          int sum = 0, count = 1, n = s.length;
12          for(int i = 0; i < n - 1; i++)
13          {
14              if(s[i] >= '0' && s[i] <= '9')
15              {
16                  sum += (s[i] - '0') * count;
17                  count++;
18              }
19          }
20          sum %= 11;
21
22          if(sum == s[n - 1] - '0' || sum == 10 && s[n - 1] == 'X')
23          {
24              System.out.println("Right");
25          }
26          else
27          {
28              s[n - 1] = sum == 10 ? 'X' : (char)(sum + '0');
29              for(int i = 0; i < n; i++)
30              {
31                  System.out.print(s[i]);
32              }
33          }
34
35      }
36  }
```

2. kotori和迷宫（BFS / DFS）

（题号：500543）

1. **题目链接：** kotori和迷宫
2. **题目描述：**

题目描述：kotori在一个n*m迷宫里，迷宫的最外层被岩浆淹没，无法涉足，迷宫内有k个出口。kotori只能上下左右四个方向移动。她想知道有多少出口是她能到达的，最近的出口离她有多远?

输入描述：第一行为两个整数n和m，代表迷宫的行和列数 (1≤n,m≤30)

后面紧跟着n行长度为m的字符串来描述迷宫。'k'代表kotori开始的位置，'.'代表道路，'*'代表墙壁，'e'代表出口。保证输入合法。

输出描述：若有出口可以抵达，则输出2个整数，第一个代表kotori可选择的出口的数量，第二个代表kotori到最近的出口的步数。（注意，kotori到达出口一定会离开迷宫）

若没有出口可以抵达，则输出-1。

补充说明：

示例1

输入：
```
6 8
e.*.*e.*
.**.*.*e
..*k**..
***.*.e*
.**.*.**
*......e
```

输出：2 7

说明：可供选择坐标为[4,7]和[6,8]，到kotori的距离分别是8和7步。

3. 解法：

算法思路：

迷宫问题小扩展~

C++ 算法代码：

```cpp
#include <iostream>
#include <cstring>
#include <queue>

using namespace std;

const int N = 35;

int x1, y1; // 标记起点位置
int n, m;
char arr[N][N];
int dist[N][N];
queue<pair<int, int>> q;

int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};

void bfs()
{
    memset(dist, -1, sizeof dist);
```

```
    dist[x1][y1] = 0;
    q.push({x1, y1});

    while(q.size())
    {
        auto [x2, y2] = q.front();
        q.pop();
        for(int i = 0; i < 4; i++)
        {
            int a = x2 + dx[i], b = y2 + dy[i];
            if(a >= 1 && a <= n && b >= 1 && b <= m && dist[a][b] == -1 &&
arr[a][b] != '*')
            {
                dist[a][b] = dist[x2][y2] + 1;
                if(arr[a][b] != 'e')
                {
                    q.push({a, b});
                }
            }
        }
    }
}

int main()
{
    cin >> n >> m;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            cin >> arr[i][j];
            if(arr[i][j] == 'k')
            {
                x1 = i, y1 = j;
            }
        }
    }

    bfs();

    int count = 0, ret = 1e9;
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            if(arr[i][j] == 'e' && dist[i][j] != -1)
            {
```

```
                count++;
                ret = min(ret, dist[i][j]);
            }
        }
    }
    if(count == 0) cout << -1 << endl;
    else cout << count << " " << ret << endl;

    return 0;
}

```

Java 算法代码：

```
 import java.util.*;

public class Main
{
    public static int N = 35;
    public static int x1, y1; // 记录起始位置
    public static int n, m;
    public static char[][] arr = new char[N][N];
    public static int[][] dist = new int[N][N];

    public static int[] dx = {0, 0, 1, -1};
    public static int[] dy = {1, -1, 0, 0};

    public static void bfs()
    {
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                dist[i][j] = -1;
            }
        }

        Queue<int[]> q = new LinkedList<>();
        q.add(new int[]{x1, y1});
        dist[x1][y1] = 0;

        while(!q.isEmpty())
        {
            int[] tmp = q.poll();
```

```
31              int a = tmp[0], b = tmp[1];
32              for(int i = 0; i < 4; i++)
33              {
34                  int x = a + dx[i], y = b + dy[i];
35                  if(x >= 0 && x < n && y >= 0 && y < m && arr[x][y] != '*' &&
    dist[x][y] == -1)
36                  {
37                      dist[x][y] = dist[a][b] + 1;
38                      if(arr[x][y] != 'e')
39                      {
40                          q.add(new int[]{x, y});
41                      }
42                  }
43              }
44          }
45      }
46
47      public static void main(String[] s)
48      {
49          Scanner in = new Scanner(System.in);
50          n = in.nextInt(); m = in.nextInt();
51          for(int i = 0; i < n; i++)
52          {
53              char[] tmp = in.next().toCharArray();
54              for(int j = 0; j < m; j++)
55              {
56                  arr[i][j] = tmp[j];
57                  if(arr[i][j] == 'k')
58                  {
59                      x1 = i; y1 = j;
60                  }
61              }
62          }
63
64          bfs();
65
66          // 更新结果
67          int count = 0, ret = 1000;
68          for(int i = 0; i < n; i++)
69          {
70              for(int j = 0; j < m; j++)
71              {
72                  if(arr[i][j] == 'e' && dist[i][j] != -1)
73                  {
74                      count++;
75                      ret = Math.min(ret, dist[i][j]);
76                  }
```

```
77              }
78          }
79
80          if(count == 0) System.out.println(-1);
81          else System.out.println(count + " " + ret);
82      }
83  }
```

3. 矩阵最长递增路径（记忆化搜索）

（题号：1076860）

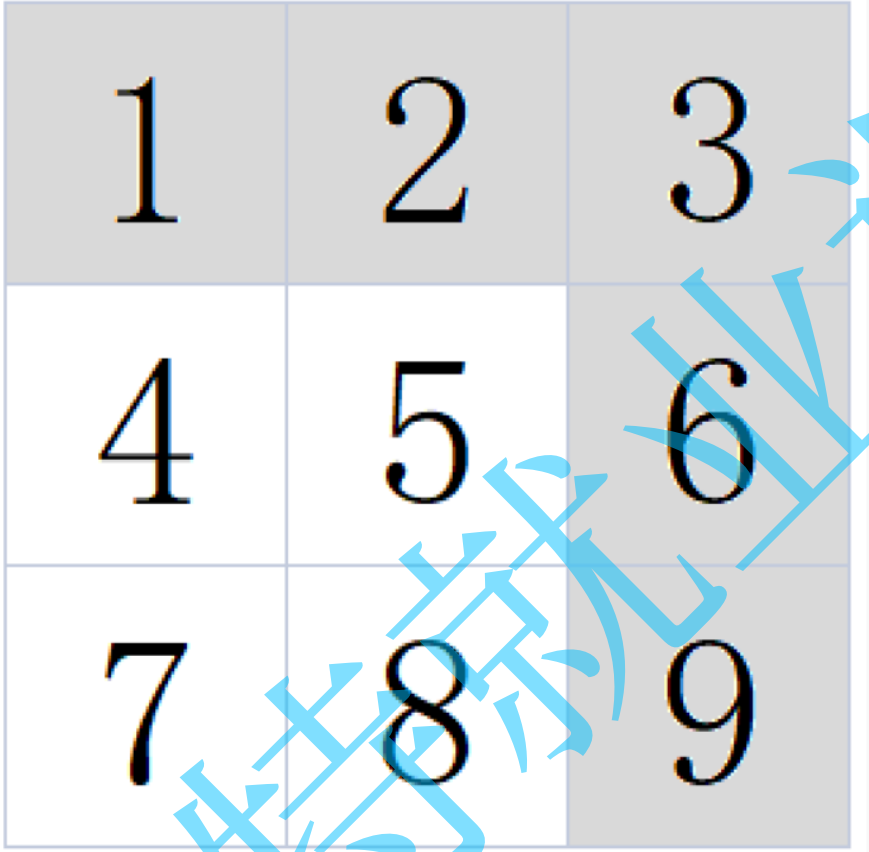
1. **题目链接：** **NC138 矩阵最长递增路径**
2. **题目描述：**

题目描述：给定一个 n 行 m 列矩阵 matrix ，矩阵内所有数均为非负整数。 你需要在矩阵中找到一条最长路径，使这条路径上的元素是递增的。并输出这条最长路径的长度。
这个路径必须满足以下条件：

1. 对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外。
2. 你不能走重复的单元格。即每个格子最多只能走一次。

数据范围：$1 \leq n, m \leq 1000$，$0 \leq matrix[i][j] \leq 1000$
进阶：空间复杂度 $O(nm)$ ，时间复杂度 $O(nm)$

例如：当输入为[[1,2,3],[4,5,6],[7,8,9]]时，对应的输出为5，
其中的一条最长递增路径如下图所示：



补充说明：矩阵的长和宽均不大于1000，矩阵内每个数不大于1000

示例1
输入：[[1,2,3],[4,5,6],[7,8,9]]
输出：5
说明：1->2->3->6->9即可。当然这种递增路径不是唯一的。

示例2
输入：[[1,2],[4,3]]
输出：4
说明： 1->2->3->4

3. 解法：

算法思路：

递归 -> 记忆化搜索。

C++ 算法代码：

```cpp
class Solution
{
    int m, n;
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};
    int memo[1010][1010];

    int dfs(vector<vector<int> >& matrix, int i, int j)
    {
        if(memo[i][j] != -1) return memo[i][j];

        int len = 1;
        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 && matrix[x][y] > matrix[i][j])
            {
                len = max(len, 1 + dfs(matrix, x, y));
            }
        }

        memo[i][j] = len;
        return len;
    }

public:

    int solve(vector<vector<int> >& matrix)
    {
        m = matrix.size(), n = matrix[0].size();
        memset(memo, -1, sizeof memo);

        int ret = 1;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                ret = max(ret, dfs(matrix, i, j));
            }
        }
        return ret;
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{
    int m, n;
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};
    int[][] memo = new int[1010][1010];

    public int dfs(int[][] matrix, int i, int j)
    {
        if(memo[i][j] != -1) return memo[i][j];

        int len = 1;
        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 && matrix[x][y] > matrix[i][j])
            {
                len = Math.max(len, 1 + dfs(matrix, x, y));
            }
        }

        memo[i][j] = len;
        return len;
    }

    public int solve (int[][] matrix)
    {
        m = matrix.length; n = matrix[0].length;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
            {
                memo[i][j] = -1;
            }
        }

        int ret = 1;
        for(int i = 0; i < m; i++)
        {
            for(int j = 0; j < n; j++)
```

```
43                 {
44                     ret = Math.max(ret, dfs(matrix, i, j));
45                 }
46             }
47             return ret;
48         }
49 }
```

Day35

1. 奇数位丢弃（模拟 - 规律）

（题号：26166）

1. **题目链接：** MT8 奇数位丢弃
2. **题目描述：**

> 题目描述：对于一个由 0..n 的所有数按升序组成的序列，我们要进行一些筛选，每次我们丢弃去当前所有数字中第奇数位个的数。重复这一过程直到最后剩下一个数。请求出最后剩下的数字。
>
> 数据范围： $1 \leq n \leq 1000$ ，本题有多组输入
>
> 输入描述：每组数据一行一个数字，为题目中的n(n小于等于1000)。
>
> 输出描述：每一行输出最后剩下的数字。
>
> 补充说明：
>
> 示例1
> 输入：500
> 输出：255
> 说明：

3. **解法：**

算法思路：

通过一两个例子的模拟，我们发现，每次起始删除的下标都是 2 的次方。

根据这个规律，找到最后一次删除的起始位置的下标即可。

C++ 算法代码：

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

```
    int n;
    while(cin >> n) // 多组输入
    {
        int ret = 1;
        while(ret - 1 <= n) ret *= 2;
        cout << ret / 2 - 1 << 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);
        while(in.hasNext())
        {
            int n = in.nextInt();
            int ret = 1;
            while(ret - 1 <= n) ret *= 2;
            System.out.println(ret / 2 - 1);
        }
    }
}
```


2. 求和（dfs）

（题号：10055177）

1. **题目链接：** [编程题]求和
2. **题目描述：**

题目描述：输入两个整数 n 和 m，从数列1，2，3.....n 中随意取几个数,使其和等于 m，要求将其中所有的可能组合列出来。

输入描述：输入两个正整数，n和m。
其中n,m均不大于10

输出描述：按每个组合的字典序排列输出，每行输出一种组合。

补充说明：

示例1
输入：5 5
输出：1 4
2 3
5
说明：

3. **解法：**

算法思路：

递归型枚举。

C++ 算法代码：

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

int n, m;
bool choose[11]; // 标记路径中选了哪些数
int sum; // 标记选了的数的和

void dfs(int x)
{
    if(sum == m)
    {
        for(int i = 1; i <= n; i++)
        {
            if(choose[i]) cout << i << " ";
        }
        cout << endl;
        return;
    }
    if(sum > m || x > n) return;

    // 选
    sum += x;
    choose[x] = true;
    dfs(x + 1);
    sum -= x;
```

```
    choose[x] = false;

    // 不选
    dfs(x + 1);
}

int main()
{
    cin >> n >> m;
    dfs(1);

    return 0;
}
```

Java 算法代码：

```
import java.util.Scanner;

public class Main
{
    public static int n, m;
    public static boolean[] choose = new boolean[11]; // 标记路径中选了谁
    public static int sum; // 标记路径中选择的元素的和

    public static void dfs(int x)
    {
        if(sum == m)
        {
            for(int i = 1; i <= n; i++)
            {
                if(choose[i])
                {
                    System.out.print(i + " ");
                }
            }
            System.out.println("");
            return;
        }
        if(sum > m || x > n) return;

        // 选 x
        sum += x;
        choose[x] = true;
        dfs(x + 1);
```

```
29          sum -= x;
30          choose[x] = false;
31
32          // 不选 x
33          dfs(x + 1);
34      }
35
36      public static void main(String[] args)
37      {
38          Scanner in = new Scanner(System.in);
39          n = in.nextInt(); m = in.nextInt();
40          dfs(1);
41      }
42  }
```

3. 计算字符串的编辑距离（动态规划）

（题号：36876）

1. **题目链接：** **HJ52 计算字符串的编辑距离**

2. **题目描述：**

题目描述：Levenshtein 距离，又称编辑距离，指的是两个字符串之间，由一个转换成另一个所需的最少编辑操作次数。许可的编辑操作包括**将一个字符替换成另一个字符，插入一个字符，删除一个字符**。编辑距离的算法是首先由俄国科学家 Levenshtein 提出的，故又叫 Levenshtein Distance 。
例如：
字符串A: abcdefg
字符串B: abcdef
通过增加或是删掉字符 "g" 的方式达到目的。这两种方案都需要一次操作。把这个操作所需要的次数定义为两个字符串的距离。
要求：
给定任意两个字符串，写出一个算法计算它们的编辑距离。

数据范围：给定的字符串长度满足 $1 \leq len(str) \leq 1000$

输入描述：每组用例一共2行，为输入的两个字符串

输出描述：每组用例输出一行，代表字符串的距离

补充说明：

示例1
输入：abcdefg
abcdef
输出：1
说明：

3. **解法：**

算法思路：

经典二维线性 dp。

C++ 算法代码：

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

using namespace std;

const int N = 1010;

string a, b;
int dp[N][N];

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

    for(int j = 0; j <= m; j++) dp[0][j] = j;
    for(int i = 0; i <= n; i++) dp[i][0] = i;

    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= m; j++)
        {
            if(a[i - 1] == b[j - 1]) dp[i][j] = dp[i - 1][j - 1];
            else dp[i][j] = min(min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;
        }
    }

    cout << dp[n][m] << 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
```

```java
{
    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        char[] a = in.next().toCharArray();
        char[] b = in.next().toCharArray();
        int n = a.length, m = b.length;

        int[][] dp = new int[n + 1][m + 1];
        for(int j = 1; j <= m; j++) dp[0][j] = j;
        for(int i = 1; i <= n; i++) dp[i][0] = i;

        for(int i = 1; i <= n; i++)
        {
            for(int j = 1; j <= m; j++)
            {
                if(a[i - 1] == b[j - 1]) dp[i][j] = dp[i - 1][j - 1];
                else dp[i][j] = Math.min(Math.min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;
            }
        }

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

Day36

1. 提取不重复的整数（数学 + 模拟）

（题号：10055187）

1. **题目链接：** HJ9 提取不重复的整数
2. **题目描述：**

> 题目描述：输入一个int型整数，按照从右向左的阅读顺序，返回一个不含重复数字的新的整数。
>
> 输入描述：输入一个int型整数。
>
> 输出描述：按照从右向左的阅读顺序，返回一个不含重复数字的新的整数。
>
> 补充说明：
>
> 示例1
> 输入：33456799
> 输出：976543
> 说明：

3. **解法：**

算法思路：

模拟题~

C++ 算法代码：

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

using namespace std;

int main()
{
    string s;
    cin >> s;
    bool hash[10] = { 0 };

    for(int i = s.size() - 1; i >= 0; i--)
    {
        int x = s[i] - '0';
        if(!hash[x])
        {
            cout << x;
            hash[x] = true;
        }
    }

    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();
        boolean[] hash = new boolean[10];

        for(int i = s.length - 1; i >= 0; i--)
        {
            int x = s[i] - '0';
            if(!hash[x])
            {
                System.out.print(x);
                hash[x] = true;
            }
        }
    }
}
```

2. 【模板】哈夫曼编码（哈夫曼编码）

（题号：2371724）

1. **题目链接：** AB32 【模板】哈夫曼编码

2. **题目描述：**

> 题目描述：给出一个有n种字符组成的字符串，其中第i种字符出现的次数为a_i。请你对该字符串应用哈夫曼编码，使得该字符串的长度尽可能短，求编码后的字符串的最短长度。
>
> 输入描述：第一行输入一个整数n（$1 \leq n \leq 2 \cdot 10^5$），表示字符种数。
> 第二行输入n个整数a_i（$1 \leq a_i \leq 10^9$），表示每种字符的出现次数。
>
> 输出描述：输出一行一个整数，表示编码后字符串的最短长度。
>
> 补充说明：
>
> 示例1
> 输入：3
> 1 2 3
> 输出：9
> 说明：三种字符的哈夫曼编码分别为["00","01","1"]时，长度最短，最短长度为9。

3. **解法：**

算法思路：

哈夫曼编码模板题~

C++ 算法代码：

```
#include <iostream>
#include <queue>
```

```cpp
#include <vector>

using namespace std;

typedef long long LL;

int n;

int main()
{
    cin >> n;
    priority_queue<LL, vector<LL>, greater<LL>> heap;
    while(n--)
    {
        LL x;
        cin >> x;
        heap.push(x);
    }

    // 构建最优二叉树 / 构建哈夫曼树
    LL ret = 0;
    while(heap.size() > 1)
    {
        LL t1 = heap.top(); heap.pop();
        LL t2 = heap.top(); heap.pop();
        heap.push(t1 + t2);
        ret += t1 + t2;
    }

    cout << ret << 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();
        PriorityQueue<Long> heap = new PriorityQueue<>();
        while(n-- != 0)
        {
            long x = in.nextLong();
            heap.offer(x);
        }

        // 构建最优二叉树 / 构建哈夫曼树
        long ret = 0;
        while(heap.size() > 1)
        {
            long t1 = heap.poll();
            long t2 = heap.poll();
            heap.offer(t1 + t2);
            ret += t1 + t2;
        }

        System.out.println(ret);
    }
}

```

3. abb（动态规划）

（题号：1831946）

1. 题目链接： DP36 abb
2. 题目描述：

题目描述：leafee 最近爱上了 abb 型语句，比如“叠词词”、“恶心心”
leafee 拿到了一个只含有小写字母的字符串，她想知道有多少个 "abb" 型的子序列?
定义： abb 型字符串满足以下条件：
1. 字符串长度为 3 。
2. 字符串后两位相同。
3. 字符串前两位不同。

输入描述：第一行一个正整数 n
第二行一个长度为 n 的字符串（只包含小写字母）
$1 \leq n \leq 10^5$

输出描述："abb" 型的子序列个数。

补充说明：

示例1
输入：6
abcbcc
输出：8
说明：共有1个abb，3个acc，4个bcc

示例2
输入：4
abbb
输出：3
说明：

3. **解法：**

算法思路：

线性dp。

C++ 算法代码：

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

typedef long long LL;

const int N = 1e5 + 10;

int n;
char s[N];
LL f[26];
LL g[26];
```

```
12
13 int main()
14 {
15     cin >> n >> s;
16
17     LL ret = 0;
18     for(int i = 0; i < n; i++)
19     {
20         int x = s[i] - 'a';
21         ret += f[x];
22
23         f[x] = f[x] + i - g[x];
24         g[x] = g[x] + 1;
25     }
26     cout << ret << endl;
27
28     return 0;
29 }
```

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         char[] s = in.next().toCharArray();
11
12         long[] f = new long[26];
13         long[] g = new long[26];
14
15         long ret = 0;
16         for(int i = 0; i < n; i++)
17         {
18             int x = s[i] - 'a';
19             ret += f[x];
20
21             // 更新哈希表
22             f[x] = f[x] + i - g[x];
23             g[x] = g[x] + 1;
```

```
24          }
25          System.out.println(ret);
26      }
27  }
28
```