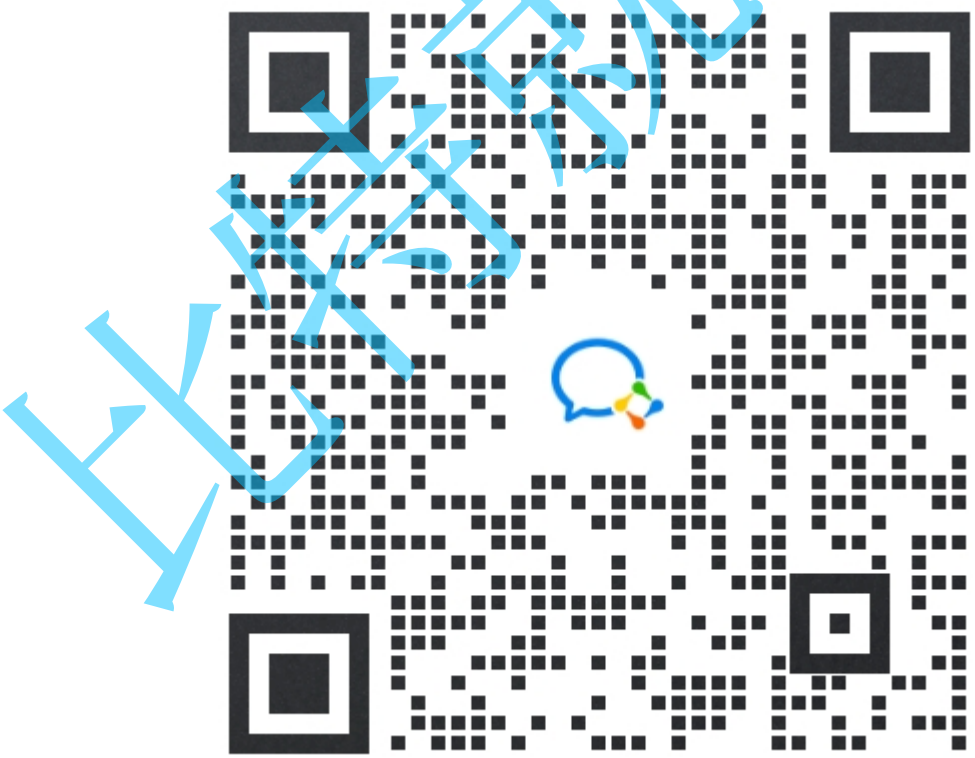
笔试强训第 08 周

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

Day01

1. kotori和抽卡（二）（概率 - 数学期望）

（题号：500566）

1. **题目链接：** kotori和抽卡（二）

2. **题目描述：**

> 题目描述：kotori最近喜欢上了lovelive这个游戏，因为她发现自己居然也是里面的一个人物。
> lovelive有个抽卡系统。共有R、SR、SSR、UR四个稀有度，每次单抽对应稀有度的概率分别是80%，15%，4%，1%。
> 然而，kotori抽了很多次卡还没出一张UR，反而出了一大堆R，气得她想删游戏了。她想知道n次单抽正好出m张R卡的概率是多少?
>
> 输入描述：两个正整数n和m（1<=m<=n<=50）
>
> 输出描述：n次单抽正好出m张R的概率。保留四位小数。
>
> 补充说明：
>
> 示例1
> 输入：1 1
> 输出：0.8000
> 说明：

3. **解法：**

算法思路：

直接代入高中求概率的公式~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    double ret = 1.0;
    for(int i = n; i >= n - m + 1; i--) ret *= i;
    for(int i = m; i >= 2; i--) ret /= i;
    for(int i = 0; i < m; i++) ret *= 0.8;
    for(int i = 0; i < n - m; i++) ret *= 0.2;

    printf("%.4lf", ret);

```

```
    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(), m = in.nextInt();
        double ret = 1.0;
        for(int i = n; i >= n - m + 1; i--) ret *= i;
        for(int i = m; i >= 2; i--) ret /= i;
        for(int i = 0; i < m; i++) ret *= 0.8;
        for(int i = 0; i < n - m; i++) ret *= 0.2;

        System.out.printf("%.4f", ret);
    }
}
```

2. ruby和薯条（排序 + 二分 / 双指针）

（题号：375038）

1. **题目链接：** ruby和薯条
2. **题目描述：**

题目描述：ruby很喜欢吃薯条。
有一天，她拿出了n根薯条。第i根薯条的长度为ai。
ruby认为，若两根薯条的长度之差在l和r之间，则认为这两根薯条有“最萌身高差”。
用数学语言描述，即若l≤|ai-aj|≤r，则第i根薯条和第j根薯条有“最萌身高差”。
ruby想知道，这n根薯条中，存在多少对薯条有“最萌身高差”？
注：次序不影响统计，即认为(ai,aj)和(aj,ai)为同一对。

输入描述：第一行三个正整数n,l,r，含义见题目描述。 （1≤n≤200000,1≤l≤r≤1e9）
第二行n个正整数ai，分别代表每根薯条的长度。 （1≤ai≤1e9）

输出描述：一个正整数，代表，代表“最萌身高差”的薯条对数。

补充说明：

示例1
输入：5 2 3
3 1 6 2 5
输出：4
说明： (3,1) (3,6) (3,5) (2,5) 共4对

3. **解法：**

算法思路：

解法一：排序 + 二分。

先排序，然后枚举较大值，在 `[1, i - 1]` 区间找差值的左右端点即可。

解法二：排序 + 前缀和 + 双指针。

先排序；

求差值在 `[L, R]` 区间内数对的个数，可以转化成求 `[0, R]` 区间内的个数 - `[0, L]` 区间内的个数。

其中求 `[0, X]` 区间内数对的个数，可以用双指针快速统计出以 `arr[right]` 为结尾的数对有多少个。

C++ 算法代码：

```cpp
// 解法一：排序 + 二分
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 2e5 + 10;

```

```cpp
int n, l, r;
int arr[N];

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

    long long ret = 0;
    for(int i = 2; i <= n; i++)
    {
        int L, R;
        // 找左端点
        int left = 1, right = i - 1;
        while(left < right)
        {
            int mid = (left + right) / 2;
            if(arr[mid] >= arr[i] - r) right = mid;
            else left = mid + 1;
        }
        if(arr[left] >= arr[i] - r) L = left;
        else L = left + 1;
        // 找右端点
        left = 1, right = i - 1;
        while(left < right)
        {
            int mid = (left + right + 1) / 2;
            if(arr[mid] <= arr[i] - l) left = mid;
            else right = mid - 1;
        }
        if(arr[left] <= arr[i] - l) R = left;
        else R = left - 1;

        if(R >= L) ret += R - L + 1;
    }

    cout << ret << endl;

    return 0;
}

// 解法二：排序 + 前缀和 + 滑动窗口
#include <iostream>
#include <algorithm>

using namespace std;
```

```cpp

const int N = 2e5 + 10;

int n, l, r;
int arr[N];

// 找出差值在 [0, x] 之间一共有多少对
long long find(int x)
{
    int left = 0, right = 0;
    long long ret = 0;
    while(right < n)
    {
        while(arr[right] - arr[left] > x) left++;
        ret += right - left;
        right++;
    }

    return ret;
}

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

    cout << find(r) - find(l - 1) << endl;

    return 0;
}
```

Java 算法代码：

```java
// 解法一：排序 + 二分
import java.util.*;

public class Main
{
    public static int n, l, r;
    public static int[] arr;

    public static void main(String[] args)
    {
```

```java
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); l = in.nextInt(); r = in.nextInt();
        arr = new int[n];
        for(int i = 0; i < n; i++) arr[i] = in.nextInt();
        Arrays.sort(arr);

        long ret = 0;
        for(int i = 1; i < n; i++)
        {
            int L, R;
            // 找左端点
            int left = 0, right = i - 1;
            while(left < right)
            {
                int mid = (left + right) / 2;
                if(arr[mid] >= arr[i] - r) right = mid;
                else left = mid + 1;
            }
            if(arr[left] >= arr[i] - r) L = left;
            else L = left + 1;
            // 找右端点
            left = 0; right = i - 1;
            while(left < right)
            {
                int mid = (left + right + 1) / 2;
                if(arr[mid] <= arr[i] - l) left = mid;
                else right = mid - 1;
            }
            if(arr[left] <= arr[i] - l) R = left;
            else R = left - 1;

            if(R >= L) ret += R - L + 1;
        }

        System.out.println(ret);
    }
}

// 解法二：排序 + 前缀和 + 滑动窗口
import java.util.*;

public class Main
{
    public static int n, l, r;
    public static int[] arr;

    // 找出差值在 [0, x] 区间内一共有多少对
```

```java
    public static long find(int x)
    {
        int left = 0, right = 0;
        long ret = 0;
        while(right < n)
        {
            while(arr[right] - arr[left] > x) left++;
            ret += right - left;
            right++;
        }
        return ret;
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt(); l = in.nextInt(); r = in.nextInt();
        arr = new int[n];
        for(int i = 0; i < n; i++) arr[i] = in.nextInt();
        Arrays.sort(arr);

        System.out.println(find(r) - find(l - 1));
    }
}
```

3. 循环汉诺塔（动态规划）

（题号：1116945）

1. **题目链接：** AB27 循环汉诺塔
2. **题目描述：**

> 题目描述：Eli最近迷上了汉诺塔。她玩了传统汉诺塔后突发奇想，发明了一种新的汉诺塔玩法。
> 有A、B、C三个柱子顺时针放置，移动的次序为A仅可以到B，B仅可以到C、C仅可以到A。即只可顺时针移动，不可逆时针移动。当然，汉诺塔的普适规则是适用的：每次移动后，大金片必须在小金片的下面。
> 现在A柱子上有n 个金片。Eli想知道，她把这些全部移动到B或C，分别要多少次操作?
>
> 输入描述：一个正整数n 。$(n < 10^7)$
>
> 输出描述：两个整数，分别代表A移到B和A移到C的最少操作数。由于该数可能过大，你需要对1000000007取模。
>
> 补充说明：
>
> 示例1
> 输入：2
> 输出：5 7
> 说明：A移到B的5步：A->B B->C A->B C->A A->B
> A移到C的7步：A->B B->C A->B C->A B->C A->B B->C

3. **解法：**

算法思路：

动态规划 + 空间优化。

C++ 算法代码：

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

const int MOD = 1e9 + 7;

int n;

int main()
{
    cin >> n;
    int x = 1, y = 2;
    for(int i = 2; i <= n; i++)
    {
        int xx = x, yy = y;
        x = (2 * yy + 1) % MOD;
        y = ((2 * yy) % MOD + 2 + xx) % MOD;
    }

    cout << x << " " << y << 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 x = 1, y = 2;
        int MOD = 1000000007;

        for(int i = 2; i <= n; i++)
        {
            int xx = x, yy = y;
            x = (2 * yy + 1) % MOD;
            y = ((2 * yy + 2) % MOD + xx) % MOD;
        }

        System.out.println(x + " " + y);
    }
}
```

Day02

1. 差值（排序）

（题号：2156174，题目可能不一样，但是考察的内容一致）

1. 题目链接： 最小差值

2. 题目描述：

题目描述：擂台赛要开始了，现在有 n 名战士，其中第 i 名战士的战斗力为 a_i。现在准备从这些战士中挑两名战士进入擂台赛进行对战，由于观众们更喜欢看势均力敌的比赛，所以我们也要挑选两个战斗力尽可能相近的战士进行参赛。那么现在请问，战斗力最接近的两名战士，战斗力之差为多少？

输入描述：第一行输入一行一个正整数 n 表示战士的数量。

第二行输入 n 个正整数表示每名战士的战斗力。$(1 \leq n \leq 10^5, 1 \leq a_i \leq 10^9)$

输出描述：输出一行一个正整数表示答案。

补充说明：

示例1
输入：3
3 5 5
输出：0
说明：选择两名战斗力为 5 的战士，战斗力之差为 0。

示例2
输入：5
1 10 4 9 6
输出：1
说明：选择战斗力为 10 和 9 两名战士，战斗力的差值为 1。

3. 解法：

算法思路：

排序，然后计算相邻两个数之差的最小值即可。

C++ 算法代码：

```cpp
class Solution
{
public:
    int minDifference(vector<int>& arr)
    {
        // INT_MIN ~ INT_MAX
        sort(arr.begin(), arr.end());
        long long ret = 1e16 + 10;
        for(int i = 1; i < arr.size(); i++)
        {
            ret = min(ret, (long long)arr[i] - arr[i - 1]);
        }
        return ret;
    }
};
```

Java 算法代码：

```java
import java.util.*;

public class Solution
{
    public int minDifference (int[] a)
    {
        // INT_MIN ~ INT_MAX
        Arrays.sort(a);
        long ret = (long)1e16 + 10;
        for(int i = 1; i < a.length; i++)
        {
            ret = Math.min(ret, (long)a[i] - a[i - 1]);
        }
        return (int)ret;
    }
}
```

2. kotori和素因子（DFS）

（题号：500564）

1. **题目链接：** kotori和素因子

2. **题目描述：**

题目描述：kotori拿到了一些正整数。她决定从每个正整数取出一个素因子。但是，kotori有强迫症，她不允许两个不同的正整数取出相同的素因子。
她想知道，最终所有取出的数的和的最小值是多少?
注：若a%k==0，则称k是a的因子。若一个数有且仅有两个因子，则称其是素数。显然1只有一个因子，不是素数。

输入描述：第一行一个正整数n，代表kotori拿到正整数的个数。

第二行共有n个数ai，表示每个正整数的值。

保证不存在两个相等的正整数。

1<=n<=10

2<=ai<=1000

输出描述：一个正整数，代表取出的素因子之和的最小值。若不存在合法的取法，则输出-1。

补充说明：1<=n<=10

2<=ai<=1000

示例1
输入：4
12 15 28 22
输出：17
说明：分别取3，5，7，2，可保证取出的数之和最小

示例2
输入：5
4 5 6 7 8
输出：-1
说明：

3. **解法：**

算法思路：

递归型枚举所有的情况。

C++ 算法代码：

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

using namespace std;

const int N = 15, M = 1010;
```

```cpp

int n, arr[N];
bool use[M]; // 记录路径中用了哪些值
int path; // 记录当前路径中所有元素的和
int ret = 0x3f3f3f3f; // 统计最终结果

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

void dfs(int pos)
{
    if(pos == n)
    {
        ret = min(ret, path);
        return;
    }

    // 枚举 arr[pos] 的所有没有使用过的素因子
    for(int i = 2; i <= arr[pos]; i++)
    {
        if(arr[pos] % i == 0 && isPrim(i) && !use[i])
        {
            path += i;
            use[i] = true;
            dfs(pos + 1);
            // 回溯 - 恢复现场
            path -= i;
            use[i] = false;
        }
    }
}

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

    dfs(0);

    if(ret == 0x3f3f3f3f) cout << -1 << endl;
```

```
    else cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static int n;
    public static int[] arr;
    public static boolean[] use = new boolean[1010]; // 记录路径里面选了哪些元素
    public static int path; // 记录路径里面所有元素的和
    public static int ret = 0x3f3f3f3f; // 记录最终结果

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

    public static void dfs(int pos)
    {
        if(pos == n)
        {
            ret = Math.min(ret, path);
            return;
        }

        // 枚举 arr[pos] 里面所有的素因子
        for(int i = 2; i <= arr[pos]; i++)
        {
            if(arr[pos] % i == 0 && !use[i] && isPrim(i))
            {
                path += i;
                use[i] = true;
                dfs(pos + 1);
                // 回溯 - 恢复现场
```

```
                use[i] = false;
                path -= i;
            }
        }
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        n = in.nextInt();
        arr = new int[n];
        for(int i = 0; i < n; i++) arr[i] = in.nextInt();

        dfs(0);

        if(ret == 0x3f3f3f3f) System.out.println(-1);
        else System.out.println(ret);
    }
}
```

3. dd爱科学1.0（最长上升子序列 - 贪心 + 二分）

（题号：1714894）

1. **题目链接：** dd爱科学1.0

2. **题目描述：**

> 题目描述：大科学家dd最近在研究转基因白菜，白菜的基因序列由一串大写英文字母构成，dd经过严谨的推理证明发现，只有当白菜的基因序列呈按位非递减形式时，这株白菜的高附加值将达到最高，于是优秀的dd开始着手修改白菜的基因序列，dd每次修改基因序列的任意位需要的代价是1，dd想知道，修改白菜的基因序列使其高附加值达到最高，所需要的最小代价的是多少。
>
> 输入描述：第一行一个正整数n(1≤n≤1000000)
> 第二行一个长度为n的字符串，表示所给白菜的基因序列
> 保证给出字符串中有且仅有大写英文字母
>
> 输出描述：输出一行，表示最小代价
>
> 补充说明：

```
示例1
输入：5
      ACEBF
输出：1
说明：改成ACEEF或者ACEFF，都只用改动一个字符，所需代价最小为1
```

3. **解法：**

算法思路：

要想改动最小，就应该在最长非下降子序列的基础上，对不是最长的部分进行更换。

因为这道题的数据范围比较大，所以应该用贪心 + 二分求出最长非下降子序列的长度。

C++ 算法代码：

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

using namespace std;

const int N = 1e6 + 10;

int n;
string s;

char dp[N]; //  dp[i] 表示：长度为 i 的所有的子序列中，最小的末尾是多少
int ret;

int main()
{
    cin >> n >> s;
    for(int i = 0; i < n; i++)
    {
        char ch = s[i];
        // 找出 ch 应该放在哪个位置
        if(ret == 0 || ch >= dp[ret])
        {
            dp[++ret] = ch;
        }
        else
        {
            // 二分出 ch 应该放的位置
            int left = 1, right = ret;
            while(left < right)
            {
                int mid = (left + right) / 2;
                if(dp[mid] > ch) right = mid;
                else left = mid + 1;
            }
            dp[left] = ch;
        }
    }

    cout << n - 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);
        int n = in.nextInt();
        char[] s = in.next().toCharArray();

        char[] dp = new char[n + 1]; // dp[i] 表示：长度为 i 的所有的子序列中，最小的末尾是多少
        int ret = 0;

        for(int i = 0; i < n; i++)
        {
            char ch = s[i];
            // 把 ch 放在什么位置上
            if(ret == 0 || ch >= dp[ret])
            {
                dp[++ret] = ch;
            }
            else
            {
                // 二分出 ch 的插入位置
                int left = 1, right = ret;
                while(left < right)
                {
                    int mid = (left + right) / 2;
                    if(dp[mid] > ch) right = mid;
                    else left = mid + 1;
                }
                dp[left] = ch;
            }
        }

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

Day03

1. kanan和高音（模拟 + 双指针）

（题号：375043）

1. **题目链接：** kanan和高音

2. **题目描述：**

> 题目描述：kanan唱歌经常高音上不去，为此她非常苦恼。
> 后来她发现了一个规律，一段连续的音符，只要后一个音比前一个音的高度差不大于8那么她就能唱上去。
> 但如果一个音过高，比前一个音高9以上，kanan就无法发出这个音了。
> 而低音则没有这个限制。如“1 5 10 1 2”她就能完整唱完，但“1 10 5 1 2”她就不能完整唱完。
> 现在有一段音符。她想截取其中一个连续的片段把它唱完。她想知道，这个片段长度的最大值是多少？
>
> 输入描述：第一行一个正整数n，代表音符的数量。（1≤n≤200000）
> 第二行有n个正整数ai，代表每个音符的音高。（1≤ai≤100）
>
> 输出描述：一个正整数，代表连续片段长度的最大值。
>
> 补充说明：
>
> 示例1
> 输入：5
> 1 10 5 1 2
> 输出：4
> 说明：截取（10 5 1 2）这个片段即可。

3. **解法：**

算法思路：

从前往后遍历，用双指针找出一段能唱完的区域，然后更新指针继续找下一段。

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

const int N = 2e5 + 10;

int n;
int arr[N];

int main()
{
```

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

    int ret = 1;
    for(int i = 0; i < n; )
    {
        int j = i;
        while(j + 1 < n && arr[j + 1] - arr[j] <= 8) j++;
        ret = max(ret, j - i + 1);
        i = j + 1;
    }

    cout << ret << 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();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++) arr[i] = in.nextInt();

        int ret = 1;
        for(int i = 0; i < n; )
        {
            int j = i;
            while(j + 1 < n && arr[j + 1] - arr[j] <= 8) j++;
            ret = Math.max(ret, j - i + 1);
            i = j + 1;
        }

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

2. 拜访（BFS）

（题号：2323703）

1. 题目链接： MT3 拜访

2. 题目描述：

题目描述：现在有一个城市销售经理，需要从公司出发，去拜访市内的某位商家，已知他的位置以及商家的位置，但是由于城市道路交通的原因，他每次移动只能在左右中选择一个方向 或 在上下中选择一个方向，现在问他有多少种最短方案到达商家地址。
给定一个地图 CityMap 及它的 行长度 n 和 列长度 m ，其中1代表经理位置， 2 代表商家位置， -1 代表不能经过的地区， 0 代表可以经过的地区，请返回方案数，保证一定存在合法路径。保证矩阵的长宽都小于等于 10。
注意：需保证所有方案的距离都是最短的方案

数据范围：$2 \leq n, m \leq 10$

例如当输入为[[2,0,0,0],[0,-1,-1,0],[0,-1,1,0],[0,0,0,0]],4,4时，对应的4行4列CityMap如下图所示：

2	0	0	0
0	−1	−1	0
0	−1	1	0
0	0	0	0

经理的位置在(2,2)，商家的位置在(0,0)，经分析，经理到达商家地址的最短方案有两种，分别为：
(2,2)->(2,3)->(1,3)->(0,3)->(0,2)->(0,1)->(0,0)
和
(2,2)->(3,2)->(3,1)->(3,0)->(2,0)->(1,0)->(0,0)，所以对应的返回值为2

补充说明：

示例1
输入：[[0,1,0],[2,0,0]],2,3
输出：2
说明：

示例2
输入：[[2,0,0,0],[0,-1,-1,0],[0,-1,1,0],[0,0,0,0]],4,4
输出：2
说明：

3. 解法：

算法思路：

在层序遍历的过程中，维护额外的信息。

C++ 算法代码：

```cpp
class Solution
{
    int n, m;
    int x1, y1, x2, y2;
    int dist[15][15] = { 0 };
    int cnt[15][15] = { 0 };
    int dx[4] = {0, 0, 1, -1};
    int dy[4] = {1, -1, 0, 0};

    int bfs(vector<vector<int> >& CityMap)
    {
        memset(dist, -1, sizeof dist);
        queue<pair<int, int>> q;
        q.push({x1, y1});
        dist[x1][y1] = 0;
        cnt[x1][y1] = 1;

        while(q.size())
        {
            auto [a, b] = q.front();
            q.pop();
            for(int i = 0; i < 4; i++)
            {
                int x = a + dx[i], y = b + dy[i];
                if(x >= 0 && x < n && y >= 0 && y < m && CityMap[x][y] != -1)
                {
                    if(dist[x][y] == -1) // 第一次到这个位置
                    {
                        dist[x][y] = dist[a][b] + 1;
                        cnt[x][y] += cnt[a][b];
                        q.push({x, y});
                    }
                    else // 不是第一次到这个位置
                    {
                        if(dist[a][b] + 1 == dist[x][y]) // 是不是最短路
                        {
                            cnt[x][y] += cnt[a][b];
                        }
                    }
                }
            }
```

```
        }

        return cnt[x2][y2];
    }

public:

    int countPath(vector<vector<int> >& CityMap, int _n, int _m)
    {
        n = _n, m = _m;
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                if(CityMap[i][j] == 1)
                {
                    x1 = i, y1 = j;
                }
                else if(CityMap[i][j] == 2)
                {
                    x2 = i, y2 = j;
                }
            }
        }

        return bfs(CityMap);
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    int n, m;
    int x1, y1, x2, y2;
    int[][] dist = new int[15][15];
    int[][] cnt = new int[15][15];
    int[] dx = {0, 0, 1, -1};
    int[] dy = {1, -1, 0, 0};

    int bfs(int[][] CityMap)
    {
```

```
14          Queue<int[]> q = new LinkedList<>();
15          q.offer(new int[]{x1, y1});
16          dist[x1][y1] = 0;
17          cnt[x1][y1] = 1;
18
19          while(!q.isEmpty())
20          {
21              int[] t = q.poll();
22              int a = t[0], b = t[1];
23              for(int i = 0; i < 4; i++)
24              {
25                  int x = a + dx[i], y = b + dy[i];
26                  if(x >= 0 && x < n && y >= 0 && y < m && CityMap[x][y] != -1)
27                  {
28                      if(dist[x][y] == -1) // 第一次到这个位置
29                      {
30                          dist[x][y] = dist[a][b] + 1;
31                          cnt[x][y] += cnt[a][b];
32                          q.offer(new int[]{x, y});
33                      }
34                      else // 不是第一次到这个位置
35                      {
36                          if(dist[a][b] + 1 == dist[x][y]) // 判断是否是最短路到达
    [x, y]
37                          {
38                              cnt[x][y] += cnt[a][b];
39                          }
40                      }
41                  }
42              }
43          }
44
45          return cnt[x2][y2];
46      }
47
48      public int countPath (int[][] CityMap, int _n, int _m)
49      {
50          n = _n; m = _m;
51          for(int i = 0; i < n; i++)
52          {
53              for(int j = 0; j < m; j++)
54              {
55                  dist[i][j] = -1;
56                  if(CityMap[i][j] == 1)
57                  {
58                      x1 = i; y1 = j;
59                  }
```

```
60                 else if(CityMap[i][j] == 2)
61                 {
62                     x2 = i; y2 = j;
63                 }
64             }
65         }
66 
67         return bfs(CityMap);
68     }
69 }
```

3. 买卖股票的最好时机(四)（动态规划）

（题号：2364648）

1. **题目链接：** **DP33 买卖股票的最好时机(四)**
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. 你最多可以对该股票有k笔交易操作，一笔交易代表着一次买入与一次卖出，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：
$1 \leq prices.length \leq 1000$
$0 \leq prices[i] \leq 1000$
$1 \leq k \leq 100$

输入描述：第一行输入一个正整数 n 和一个正整数 k。表示数组 prices 的长度和 交易笔数
第二行输入 n 个正整数表示数组的所有元素值。

输出描述：输出最大收益

补充说明：

示例1
输入：6 3
8 9 3 5 1 3
输出：5
说明：第一天(股票价格=8)买进，
第二天(股票价格=9)卖出，收益为1
第三天(股票价格=3)买进，
第四天(股票价格=5)卖出，收益为2
第五天(股票价格=1)买进，
第六天(股票价格=3)卖出，收益为2
总收益为5。

示例2
输入：8 2
3 2 5 0 0 3 1 4
输出：7
说明：第二天(股票价格=2)买进，
第三天(股票价格=5)卖出，收益为3
第五天(股票价格=0)买进，
第八天(股票价格=4)卖出，收益为4
总收益为7

示例3
输入：4 4
9 8 4 1
输出：0
说明：

3. **解法：**

算法思路：

1. 状态表示：

   为了更加清晰的区分「买入」和「卖出」，我们换成「有股票」和「无股票」两个状态。

   `f[i][j]` 表示：第 `i` 天结束后，完成了 `j` 笔交易，此时处于「有股票」状态的最大收益；

   `g[i][j]` 表示：第 `i` 天结束后，完成了 `j` 笔交易，此时处于「无股票」状态的最大收益。

2. 状态转移方程：

对于 `f[i][j]`，我们也有两种情况能在第 `i` 天结束之后，完成 `j` 笔交易，此时手里「有股票」的状态：

i. 在 `i - 1` 天的时候，手里「有股票」，并且交易了 `j` 次。在第 i 天的时候，啥也不干。此时的收益为 `f[i - 1][j]`；

ii. 在 `i - 1` 天的时候，手里「没有股票」，并且交易了 `j` 次。在第 `i` 天的时候，买了股票。那么 `i` 天结束之后，我们就有股票了。此时的收益为 `g[i - 1][j] - prices[i]`；

上述两种情况，我们需要的是「最大值」，因此 f 的状态转移方程为：

```
f[i][j] = max(f[i - 1][j], g[i - 1][j] - prices[i])
```

对于 `g[i][j]`，我们有下面两种情况能在第 `i` 天结束之后，完成 `j` 笔交易，此时手里「没有股票」的状态：

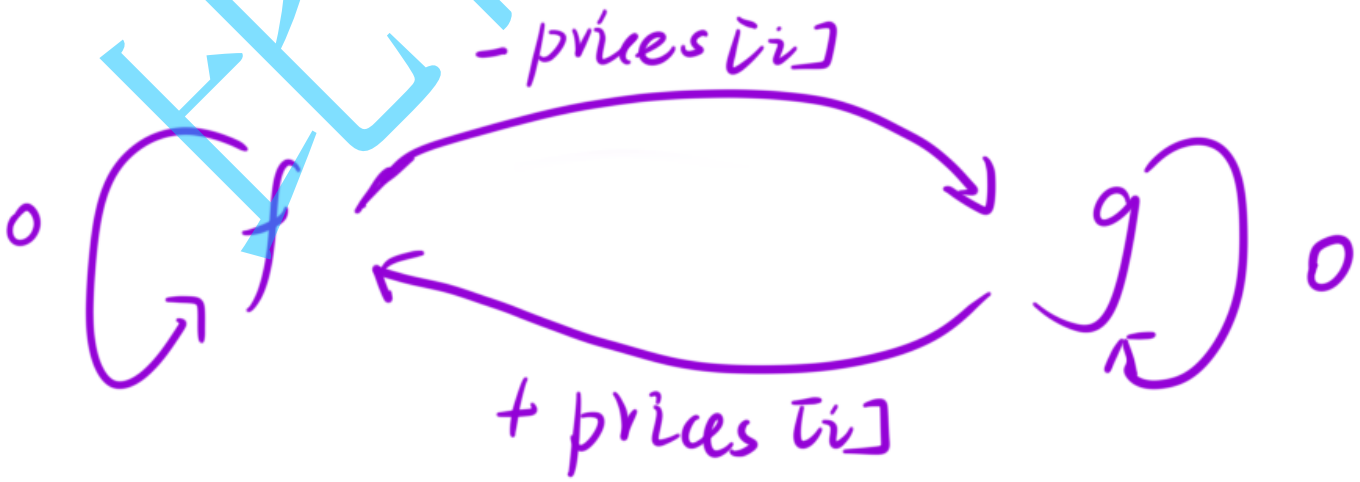
i. 在 `i - 1` 天的时候，手里「没有股票」，并且交易了 `j` 次。在第 i 天的时候，啥也不干。此时的收益为 `g[i - 1][j]`；

ii. 在 `i - 1` 天的时候，手里「有股票」，并且交易了 `j - 1` 次。在第 `i` 天的时候，把股票卖了。那么 `i` 天结束之后，我们就交易了 `j` 次。此时的收益为 `f[i - 1][j - 1] + prices[i]`；

上述两种情况，我们需要的是「最大值」，因此 `g` 的状态转移方程为：

```
g[i][j] = max(g[i - 1][j], f[i - 1][j - 1] + prices[i])
```

如果画一个图的话，它们之间交易关系如下：



3. 初始化：

由于需要用到 `i = 0` 时的状态，因此我们初始化第一行即可。

- 当处于第 `0` 天的时候，只能处于「买入过一次」的状态，此时的收益为 `-prices[0]`，因此 `f[0][0] = - prices[0]`。
- 为了取 `max` 的时候，一些不存在的状态「起不到干扰」的作用，我们统统将它们初始化为 `-INF`（用 `INT_MIN` 在计算过程中会有「溢出」的风险，这里 `INF` 折半取 `0x3f3f3f3f`，足够小即可）

4. 填表顺序：

   从上往下填每一行，每一行从左往右，两个表一起填。

5. 返回值：

   返回处于卖出状态的最大值，但是我们也不知道是交易了几次，因此返回 `g` 表最后一行的最大值。

优化点：

我们的交易次数是不会超过整个天数的一半的，因此我们可以先把 `k` 处理一下，优化一下问题的规模：

`k = min(k, n / 2)`

C++ 算法代码：

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

const int N = 1010, M = 110;

int n, k, p[N];
int f[N][M], g[N][M];

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

    k = min(k, n / 2);
    for(int j = 0; j <= k; j++) f[0][j] = g[0][j] = -0x3f3f3f3f;
    f[0][0] = -p[0], g[0][0] = 0;
```

```
17
18      for(int i = 1; i < n; i++)
19      {
20          for(int j = 0; j <= k; j++)
21          {
22              f[i][j] = max(f[i - 1][j], g[i - 1][j] - p[i]);
23              g[i][j] = g[i - 1][j];
24              if(j >= 1) g[i][j] = max(g[i][j], f[i - 1][j - 1] + p[i]);
25          }
26      }
27
28      int ret = 0;
29      for(int j = 0; j <= k; j++) ret = max(ret, g[n - 1][j]);
30
31      cout << ret << endl;
32
33      return 0;
34  }
```

Java 算法代码：

```
1  import java.util.*;
2
3  // 注意类名必须为 Main, 不要有任何 package xxx 信息
4  public class Main
5  {
6      public static void main(String[] args)
7      {
8          Scanner in = new Scanner(System.in);
9          int n = in.nextInt(), k = in.nextInt();
10         int[] p = new int[n];
11         for(int i = 0; i < n; i++) p[i] = in.nextInt();
12
13         int[][] f = new int[n][k + 1];
14         int[][] g = new int[n][k + 1];
15
16         k = Math.min(k, n / 2);
17         for(int j = 0; j <= k; j++) f[0][j] = g[0][j] = -0x3f3f3f3f;
18         f[0][0] = -p[0]; g[0][0] = 0;
19
20         for(int i = 1; i < n; i++)
21         {
22             for(int j = 0; j <= k; j++)
23             {
```

```
                f[i][j] = Math.max(f[i - 1][j], g[i - 1][j] - p[i]);
                g[i][j] = g[i - 1][j];
                if(j >= 1) g[i][j] = Math.max(g[i][j], f[i - 1][j - 1] + p[i]);
            }
        }

        int ret = 0;
        for(int j = 0; j <= k; j++) ret = Math.max(ret, g[n - 1][j]);

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

Day04

1. AOE还是单体？（贪心）

（题号：938754）

1. **题目链接：** AOE还是单体?

2. **题目描述：**

题目描述：牛可乐准备和 n 个怪物厮杀。已知第 i 个怪物的血量为 a_i 。
牛可乐有两个技能：
第一个技能是蛮牛冲撞，消耗 1 mp ，可以对任意单体怪物造成 1 点伤害。
第二个技能是蛮牛践踏，消耗 x mp ，可以对全体怪物造成 1 点伤害。
牛可乐想知道，将这些怪物全部击杀，消耗 mp 的最小值的多少?

输入描述：第一行两个正整数 n 和 x ，分别代表怪物的数量、每次蛮牛践踏消耗的 mp 值。
（$1 \leq n \leq 200000$，$1 \leq x \leq 10^9$）
第二行 n 个正整数 a_i ，分别代表每个怪物的血量。（$1 \leq a_i \leq 10^9$）

输出描述：一个正整数，代表消耗 mp 的最小值。

补充说明：

示例1
输入：5 2
2 4 5 6 3
输出：11
说明：先对3号怪物用1次蛮牛冲撞，对4号怪物用2次蛮牛冲撞，此时消耗mp为3，怪物的血量是2，4，4，4，3。
然后用4次蛮牛践踏，击杀全部怪物，消耗的mp总量为11。

3. **解法：**

算法思路：

小贪心：

a. 如果使用一次 AOE 造成的伤害比消耗的蓝量多，那就使用；

b. 否则就一直使用单体伤害。

C++ 算法代码：

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

using namespace std;

const int N = 2e5 + 10;

int n, x;
int arr[N];

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

    sort(arr + 1, arr + 1 + n);
    long long ret = 0;
    int index = max(0, n - x); // 处理 x 过大的情况
    ret += arr[index] * x;
    for(int i = index + 1; i <= n; i++) ret += arr[i] - arr[index];

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```java
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
```

```java
        Scanner in = new Scanner(System.in);
        int n = in.nextInt(), x = in.nextInt();
        int[] arr = new int[n];
        for(int i = 0; i < n; i++)
        {
            arr[i] = in.nextInt();
        }

        Arrays.sort(arr);
        long ret = 0;
        int index = n - 1 - x;

        if(index < 0)
        {
            for(int i = 0; i < n; i++) ret += arr[i];
        }
        else
        {
            ret += arr[index] * x;
            for(int i = index + 1; i < n; i++)
            {
                ret += arr[i] - arr[index];
            }
        }

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


2. kotori和n皇后（哈希表）

（题号：500565）

1. **题目链接：** kotori和n皇后
2. **题目描述：**

题目描述：kotori最近在研究n皇后的问题。
所谓n皇后问题是这样的：一个n*n的地图，上面一共放n个皇后，保证任意两个皇后都不能互相攻击（每个皇后可以攻击同一行、同一列以及同一45度角斜线和135度角斜线上的所有其他皇后）。
kotori思考了很久都无法得出答案，整个人都变成琴梨了。她于是拿了一堆皇后在一个无穷大的棋盘上模拟，按照次序一共放了k个皇后。
但是，皇后的站位太复杂了，kotori甚至不知道是否存在两个皇后会互相攻击。于是她想问问聪明的你，在第i个皇后放置在棋盘上之后，是否存在两个皇后可以互相攻击?

输入描述：第一行输入一个正整数k，代表总共放置的皇后的个数。（1<=k<=1e5）

接下来的k行，每行两个正整数xi和yi，代表每个皇后的坐标。（1<=xi,yi<=1e9）

之后输入一个正整数t，代表t次询问。（1<=t<=1e5）

接下来的t行，每行一个正整数i，代表询问第i个皇后放置后，是否存在互相攻击的情况。（1<=i<=k）

保证不存在两个皇后放置的位置相同。

输出描述：共t行。每行对应当前的询问是否存在两个皇后可以互相攻击，若是则输出“Yes”，否则输出“No”

补充说明：

示例1
输入：5
1 2
2 5
3 1
6 7
4 8
2
2
4
输出：No
Yes
说明：第四个皇后放置后，第四个和第一个皇后可以互相攻击。

3. 解法：

算法思路：

使用哈希表标记**行列**以及**两个对角线**。

（坑爹题目，大家一定要好好读题，注意输出的时候提前判断一下。）

C++ 算法代码：

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

using namespace std;

typedef long long LL;

int k, t;
```

```
int ret = 1e5 + 10; // 第一次出现互相攻击的皇后的个数
unordered_set<LL> row; // 标记行 y
unordered_set<LL> col; // 标记列 x
unordered_set<LL> dig1; // 标记主对角线 y - x
unordered_set<LL> dig2; // 标记副对角线 y + x

int main()
{
    cin >> k;
    for(int i = 1; i <= k; i++)
    {
        int x, y;
        cin >> x >> y;
        if(ret != 1e5 + 10) continue;
        if(row.count(y) || col.count(x) || dig1.count(y - x) || dig2.count(y + x))
        {
            ret = i;
        }
        row.insert(y); col.insert(x); dig1.insert(y - x); dig2.insert(y + x);
    }

    cin >> t;
    while(t--)
    {
        int i;
        cin >> i;
        if(i >= ret) 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 k = in.nextInt();
```

```
9
10          Set<Long> row = new HashSet<>(); // 标记行 y
11          Set<Long> col = new HashSet<>(); // 标记列 x
12          Set<Long> dig1 = new HashSet<>(); // 标记主对角线 y - x
13          Set<Long> dig2 = new HashSet<>(); // 标记副对角线 y + x
14
15          int ret = (int)1e5 + 10; // 存一次哪个皇后第一次来的之后发生互相攻击
16          for(int i = 1; i <= k; i++)
17          {
18              long x = in.nextLong(), y = in.nextLong();
19              if(ret != (int)1e5 + 10) continue;
20              if(row.contains(y) || col.contains(x) || dig1.contains(y - x) ||
    dig2.contains(y + x))
21              {
22                  ret = i;
23              }
24              row.add(y); col.add(x); dig1.add(y - x); dig2.add(y + x);
25          }
26
27          int t = in.nextInt();
28          while(t-- != 0)
29          {
30              int i = in.nextInt();
31              if(i >= ret) System.out.println("Yes");
32              else System.out.println("No");
33          }
34      }
35  }
```

3. 取金币（动态规划 - 区间dp）

（题号：2433134）

1. **题目链接：** NC393 取金币
2. **题目描述：**

题目描述：给定一个长度为 n 的正整数数组 coins，每个元素表示对应位置的金币数量。
取位置 i 的金币时，假设左边一堆金币数量是L，右边一堆金币数量为R，则获得$L * cost[i] * R$的积分。如果左边或右边没有金币，则金币数量视为1。
请你计算最多能得到多少积分。

数据范围：数组长度满足 $12n2100$ ，数组中的元素满足 $1 \leq coins_i \leq 100$

补充说明：

示例1
输入：[5,6,4,8]
输出：480
说明：第一步取 4，得 6*4*8 = 192，余下 5 6 8。
第二步取 6，得 5*6*8 = 240，余下 5 8。
第三步取 5，得 5*8*1 = 40，余下 8。
最后取 8，得 1*8*1 = 8 。
最终积分为 192+240+40+8 = 480。

3. **解法：**

算法思路：

区间 dp：

为了方便能处理边界情况，将原数组前后添加一个 1，并不影响最后的结果。

1. 状态表示：`dp[i][j]` 表示：`[i, j]` 区间一共能获得多少金币。

C++ 算法代码：

```
class Solution
{
    int arr[110] = { 0 };
    int dp[110][110] = { 0 };

public:
    int getCoins(vector<int>& coins)
    {
        int n = coins.size();
        arr[0] = arr[n + 1] = 1;
        for(int i = 1; i <= n; i++) arr[i] = coins[i - 1];

        for(int i = n; i >= 1; i--)
        {
            for(int j = i; j <= n; j++)
            {
                for(int k = i; k <= j; k++)
                {
                    dp[i][j] = max(dp[i][j], dp[i][k - 1] + dp[k + 1][j] + arr[i - 1] * arr[k] * arr[j + 1]);
                }
```

```
            }
        }

        return dp[1][n];
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int getCoins (ArrayList<Integer> nums)
    {
        int n = nums.size();
        int[] arr = new int[n + 2];
        arr[0] = arr[n + 1] = 1;
        for(int i = 1; i <= n; i++)
        {
            arr[i] = nums.get(i - 1);
        }

        int[][] dp = new int[n + 2][n + 2];
        for(int i = n; i >= 1; i--)
        {
            for(int j = i; j <= n; j++)
            {
                for(int k = i; k <= j; k++)
                {
                    dp[i][j] = Math.max(dp[i][j], dp[i][k - 1] + dp[k + 1][j] + arr[i - 1] * arr[k] * arr[j + 1]);
                }
            }
        }

        return dp[1][n];
    }
}
```

Day05

1. 矩阵转置（数学）

（题号：618636）

1. **题目链接：** BC138 矩阵转置

2. **题目描述：**

> 题目描述：KiKi有一个矩阵，他想知道转置后的矩阵（将矩阵的行列互换得到的新矩阵称为转置矩阵），请编程帮他解答。
>
> 输入描述：第一行包含两个整数n和m，表示一个矩阵包含n行m列，用空格分隔。（1≤n≤10,1≤m≤10）
> 从2到n+1行，每行输入m个整数（范围-2^{31}~2^{31}-1），用空格分隔，共输入n*m个数，表示第一个矩阵中的元素。
>
> 输出描述：输出m行n列，为矩阵转置后的结果。每个数后面有一个空格。
>
> 补充说明：

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

3. **解法：**

算法思路：

观察转置前和转置后下标的关系即可。

C++ 算法代码：

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

const int N = 15;

int n, m;
int arr[N][N];

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

```
        for(int j = 0; j < m; j++)
        {
            cin >> arr[i][j];
        }
    }

    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < n; j++)
        {
            // ret[i][j] = arr[j][i]
            cout << arr[j][i] << " ";
        }
        cout << 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(), m = in.nextInt();
        int[][] arr = new int[n][m];

        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < m; j++)
            {
                arr[i][j] = in.nextInt();
            }
        }

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

```
                // ret[i][j] = arr[j][i]
                System.out.print(arr[j][i] + " ");
            }
            System.out.println("");
        }
    }
}
```

2. 四个选项（DFS + 剪枝 + 哈希表）

（题号：848875）

1. **题目链接：** 四个选项

2. **题目描述：**

> 题目描述：　众所周知，高考数学中有一个题目是给出12个单项选择，每一个选择的答案是 A，B，C，D 中的一个。
> 网上盛传答案存在某种规律，使得蒙对的可能性大大增加。于是今年老师想让你安排这12个题的答案。但是他有一些条件，首先四个选项的数量必须分别为 na，nb，nc，nd。其次有 m 个额外条件，分别给出两个数字 x，y，代表第 x 个题和第 y 个题的答案相同。 现在你的老师想知道，有多少种可行的方案安排答案。
>
> 输入描述：第一行五个非负整数na，nb，nc，nd，m，保证na+nb+nc+nd=12，0≤m≤1000。
> 接下来m行每行两个整数x，y（1≤ x,y ≤12），代表第x个题和第y个题答案必须一样。
>
> 输出描述：仅一行一个整数，代表可行的方案数。
>
> 补充说明：
>
> 示例1
> 输入：3 3 3 3 0
> 输出：369600
> 说明：

3. **解法：**

算法思路：

用递归枚举出所有的情况，注意剪枝。

C++ 算法代码：

```
#include <iostream>
#include <vector>

using namespace std;

int cnt[5]; // 用数组存每一个选项出现多少次
int m, x, y;
bool same[13][13]; // 存哪些题的答案是相同的
```

```cpp

int ret;
vector<int> path; // 记录路径里面选了哪些选项

bool isSame(int pos, int cur)
{
    for(int i = 1; i < pos; i++)
    {
        if(same[pos][i] && path[i] != cur) return false;
    }
    return true;
}

void dfs(int pos)
{
    if(pos > 12)
    {
        ret++;
        return;
    }

    for(int i = 1; i <= 4; i++)
    {
        if(cnt[i] == 0) continue; // 没有使用次数
        if(!isSame(pos, i)) continue; // 需要相同的位置，没有相同

        cnt[i]--;
        path.push_back(i);
        dfs(pos + 1);
        path.pop_back();
        cnt[i]++;
    }
}

int main()
{
    for(int i = 1; i <= 4; i++) cin >> cnt[i];
    cin >> m;
    while(m--)
    {
        cin >> x >> y;
        same[x][y] = same[y][x] = true;
    }

    path.push_back(0); // 先放进去一个占位符
    dfs(1);

```

```
    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static int[] cnt = new int[5]; // 统计一下每个选项出现的次数
    public static int m, x, y;
    public static boolean[][] same = new boolean[13][13];

    public static int ret;
    public static int[] path = new int[13]; // 记录当前路径填了哪些选项

    public static boolean isSame(int pos, int cur)
    {
        for(int i = 1; i < pos; i++)
        {
            if(same[pos][i] && path[i] != cur) return false;
        }
        return true;
    }

    public static void dfs(int pos)
    {
        if(pos > 12)
        {
            ret++;
            return;
        }

        for(int i = 1; i <= 4; i++)
        {
            if(cnt[i] == 0) continue; // 看看有没有剩余次数
            if(!isSame(pos, i)) continue; // 判断是否需要相同的题目，填了相同的选项

            path[pos] = i;
            cnt[i]--;
            dfs(pos + 1);
            cnt[i]++;
```

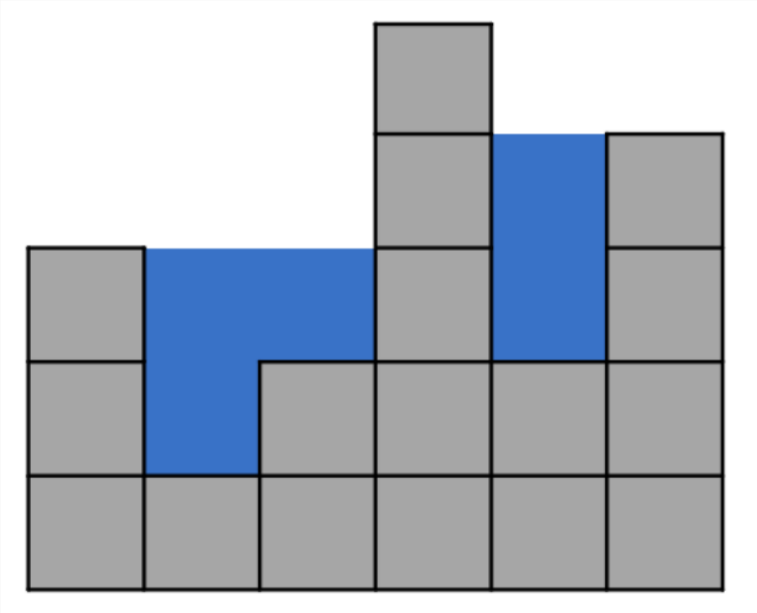
```
        }
    }

    public static void main(String[] args)
    {
        Scanner in = new Scanner(System.in);
        for(int i = 1; i <= 4; i++)
        {
            cnt[i] = in.nextInt();
        }
        m = in.nextInt();
        while(m-- != 0)
        {
            x = in.nextInt(); y = in.nextInt();
            same[x][y] = same[y][x] = true;
        }

        dfs(1);

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

3. 接雨水问题（双指针）

（题号：1002045）

1. **题目链接：** 接雨水
2. **题目描述：**

题目描述：给定一个整形数组arr，已知其中所有的值都是非负的，将这个数组看作一个柱子高度图，计算按此排列的柱子，下雨之后能接多少雨水。(数组以外的区域高度视为0)



数据范围：数组长度 $01n12 \times 10^5$，数组中每个值满足 $0 < val110^9$，保证返回结果满足 $01val110^9$
要求：时间复杂度 $O(n)$

补充说明：

示例1
输入：[3,1,2,5,2,4]
输出：5
说明：数组 [3,1,2,5,2,4] 表示柱子高度图，在这种情况下，可以接 5个单位的雨水，蓝色的为雨水 ，如题面图。

示例2
输入：[4,5,1,3,2]
输出：2
说明：

3. 解法：

算法思路：

考虑每一根柱子上方雨水的高度。

C++ 算法代码：

```
class Solution
{
public:
    int trap(vector<int>& height)
    {
        int n = height.size();
        vector<int> left(n);
        vector<int> right(n);

        // 预处理左侧最大值数组
```

```
        left[0] = height[0];
        for(int i = 1; i < n; i++) left[i] = max(left[i - 1], height[i]);

        // 预处理右侧最大值数组
        right[n - 1] = height[n - 1];
        for(int i = n - 2; i >= 0; i--) right[i] = max(right[i + 1],
height[i]);

        // 求结果
        int ret = 0;
        for(int i = 1; i < n - 1; i++)
        {
            ret += min(left[i], right[i]) - height[i];
        }

        return ret;
    }
};
```

Java 算法代码：

```
class Solution
{
    public int trap(int[] height)
    {
        int n = height.length;

        int[] left = new int[n];
        int[] right = new int[n];

        // 预处理左侧最大值数组
        left[0] = height[0];
        for(int i = 1; i < n; i++)
        {
            left[i] = Math.max(left[i - 1], height[i]);
        }

        // 预处理右侧最大值数组
        right[n - 1] = height[n - 1];
        for(int i = n - 2; i >= 0; i--)
        {
            right[i] = Math.max(right[i + 1], height[i]);
        }

```

```
24          // 求结果
25          int ret = 0;
26          for(int i = 1; i < n - 1; i++)
27          {
28              ret += Math.min(left[i], right[i]) - height[i];
29          }
30
31          return ret;
32      }
33  }
```

Day06

1. 疯狂的自我检索者（贪心）

（题号：955142）

1. **题目链接：** 疯狂的自我检索者

2. **题目描述：**

> 题目描述：牛妹作为偶像乐队的主唱，对自己的知名度很关心。她平时最爱做的事就是去搜索引擎搜自己的名字，看看别人对自己的评价怎么样。
> 这天，她打开了一个“偶像评分系统”，上面有很多人给她打分。
> “偶像评分系统”的分数有1分、2分、3分、4分和5分。给牛妹评分的人有 n 个。但其中有 m 个人把分数隐藏了，牛妹并不能看到这些人给她打的分数。
> 牛妹想知道，已知这些信息的情况下，自己得到的平均分数的最大可能和最小可能分别是多少？
>
> 输入描述：第一行输入两个正整数 n 和 m （$1 \leq m \leq n \leq 200000$）
> 第二行输入 $n-m$ 个正整数 a_i，代表没有隐藏的分数。（$1 \leq a_i \leq 5$）
> 若 m 和 n 相等，则第二行为空。
>
> 输出描述：两个数，用空格隔开，分别代表最小可能平均分数和最大可能平均分数。如果你的输出和正确答案之间误差不超过 10^{-5}，则认为你的答案正确。
>
> 补充说明：

```
示例1
输入：5 1
     1 2 3 4
输出：2.20000 3.00000
说明：
```

3. **解法：**

算法思路：

小贪心~

C++ 算法代码：

```cpp
#include <iostream>

using namespace std;

int n, m;
int a;

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

    int sum = 0;
    for(int i = 0; i < n - m; i++)
    {
        cin >> a;
        sum += a;
    }

    printf("%.5lf %.5lf\n", (sum + m) * 1.0 / n, (sum + m * 5) * 1.0 / n);

    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(), m = in.nextInt();

        int sum = 0;
        for(int i = 0; i < n - m; i++)
        {
            int x = in.nextInt();
            sum += x;
        }

        double a = (sum + m) * 1.0 / n;
        double b = (sum + m * 5) * 1.0 / n;
        System.out.printf("%.5f %.5f", a, b);
```

```
    }
}
```

2. 栈和排序（栈 + 贪心）

（题号：1024794）

1. **题目链接：** NC115 栈和排序

2. **题目描述：**

> 题目描述：给你一个 1 到 n 的排列和一个栈，并按照排列顺序入栈
>
> 你要在不打乱入栈顺序的情况下，仅利用入栈和出栈两种操作，输出字典序最大的出栈序列
>
> 排列：指 1 到 n 每个数字出现且仅出现一次
>
> 数据范围： $1 \leq n \leq 5 \times 10^4$，排列中的值都满足 $1 \leq val \leq n$
>
> 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：[2,1,5,3,4]
> 输出：[5,4,3,1,2]
> 说明：操作 栈 结果 2 入栈；[2] [] 1 入栈；[2\1] [] 5 入栈；[2\1\5] [] 5 出栈；[2\1] [5] 3 入栈；[2\1\3] [5] 4 入栈；[2\1\3\4] [5] 4 出栈；[2\1\3] [5,4] 3 出栈；[2\1] [5,4,3] 1 出栈；[2] [5,4,3,1] 2 出栈；[] [5,4,3,1,2]
>
> 示例2
> 输入：[1,2,3,4,5]
> 输出：[5,4,3,2,1]
> 说明：

3. **解法：**

算法思路：

每次尽可能的先让当前需要的最大值弹出去。

C++ 算法代码：

```
class Solution
{
public:
    vector<int> solve(vector<int>& a)
    {
        int n = a.size();
        stack<int> st;
        bool hash[50010] = { 0 }; // 统计当前哪些元素已经进栈
```

```
        int aim = n;
        vector<int> ret;

        for(auto x : a)
        {
            st.push(x);
            hash[x] = true;

            // 先更新目标值
            while(hash[aim])
            {
                aim--;
            }

            // 出栈
            while(st.size() && st.top() >= aim)
            {
                ret.push_back(st.top());
                st.pop();
            }
        }

        return ret;
    }
};
```

Java 算法代码：

```
import java.util.*;

public class Solution
{
    public int[] solve (int[] a)
    {
        int n = a.length;
        Stack<Integer> st = new Stack<>();
        boolean[] hash = new boolean[n + 1]; // 标记哪些元素已经入栈
        int aim = n;
        int[] ret = new int[n];
        int i = 0;

        for(int x : a)
        {
            st.push(x);
```

```
17              hash[x] = true;
18
19              // 更新目标值
20              while(hash[aim])
21              {
22                  aim--;
23              }
24
25              // 出栈
26              while(!st.isEmpty() && st.peek() >= aim)
27              {
28                  ret[i++] = st.peek();
29                  st.pop();
30              }
31          }
32
33          return ret;
34      }
35  }
```

3. 加减（枚举 + 前缀和 + 滑动窗口 + 贪心）

（题号：1946143）

1. **题目链接：** 加减
2. **题目描述：**

题目描述：小红拿到了一个长度为 n 的数组。她每次操作可以让某个数加 1 或者某个数减 1 。
小红最多能进行 k 次操作。她希望操作结束后，该数组出现次数最多的元素次数尽可能多。
你能求出这个最大的次数吗?

输入描述：第一行两个正整数 n 和 k
第二行有 n 个正整数 a_i

$1 \le n \le 10^5$
$1 \le k \le 10^{12}$
$1 \le a_i \le 10^9$

输出描述：不超过 k 次操作之后，数组中可能出现最多次数元素的次数。

补充说明：

示例1
输入：5 3
6 3 20 8 1
输出：2
说明：共 3 次操作如下：
第一个数加一。
第二个数加一。
第四个数减一。
数组变成了 7 4 20 7 1 ，共有 2 个相同的数： 7 。
可以证明， 2 为最优解。另外，此上操作并不是唯一的操作。

3. **解法：**

算法思路：

转化问题：将原数组排序之后，选择一段区间，让他们在不超过 k 次的前提下，全部变的相等。

C++ 算法代码：

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

using namespace std;

typedef long long LL;

const int N = 1e5 + 10;

LL n, k;
LL arr[N];
LL sum[N]; // 前缀和数组

LL cal(int l, int r)
{
    int mid = (l + r) / 2;
```

```
    return (mid - l - r + mid) * arr[mid] - (sum[mid - 1] - sum[l - 1]) +
(sum[r] - sum[mid]);
}

int main()
{
    cin >> n >> k;
    for(int i = 1; i <= n; i++) cin >> arr[i];
    sort(arr + 1, arr + 1 + n);
    // 初始化前缀和数组
    for(int i = 1; i <= n; i++) sum[i] = sum[i - 1] + arr[i];

    int left = 1, right = 1, ret = 1;
    while(right <= n)
    {
        // 进窗口
        LL cost = cal(left, right);
        while(cost > k) // 判断
        {
            left++; // 出窗口
            cost = cal(left, right);
        }
        // 更新结果
        ret = max(ret, right - left + 1);
        right++;
    }

    cout << ret << endl;

    return 0;
}
```

Java 算法代码：

```
import java.util.*;

public class Main
{
    public static int n;
    public static long k;
    public static long[] arr;
    public static long[] sum;

    public static long cal(int l, int r)
```

```
    {
        int mid = (l + r) / 2;
        return (mid - l) * arr[mid] - (sum[mid - 1] - sum[l - 1]) + (sum[r] -
sum[mid]) - (r - mid) * arr[mid];
    }

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

        Arrays.sort(arr, 1, n + 1);

        // 初始化前缀和数组
        for(int i = 1; i <= n; i++)
        {
            sum[i] = sum[i - 1] + arr[i];
        }

        int left = 1, right = 1, ret = 1;
        while(right <= n)
        {
            // 进窗口
            long cost = cal(left, right);
            while(cost > k) // 判断
            {
                left++; // 出窗口
                cost = cal(left, right);
            }
            // 更新结果
            ret = Math.max(ret, right - left + 1);
            right++;
        }

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