只用暴力，稳拿省一系列。

只允许使用：

判断，循环，数组，函数，语言自带函数，素数算法，gcd，lcm 算法，一维前缀和，一维差分，简单递归，子集枚举（二进制枚举 or dfs）。

超出范围的，或需要大量动脑的，我们都不做，只尝试输出样例

穿越时空之门

```cpp
#include <iostream>
using namespace std;
int calc(int x,int k){
  int sum=0;
  while(x!=0){
    sum=sum+x%k;
    x/=k;
  }
  return sum;
}
int main()
{
  int ans=0;
  for(int i=1;i<=2024;i++){
    if(calc(i,2)==calc(i,4)){
      ans++;
    }
  }
  printf("%d",ans);
  return 0;
}
```

```java
public class Main {
    public static int calc(int x, int k) {
        int sum = 0;
        while (x != 0) {
            sum += x % k;
            x /= k;
        }
        return sum;
    }

    public static void main(String[] args) {
        int ans = 0;
        for (int i = 1; i <= 2024; i++) {
            if (calc(i, 2) == calc(i, 4)) {
                ans++;
            }
        }
        System.out.println(ans);
    }
}

```

```python
def calc(x, k):
    total = 0
    while x != 0:
        total += x % k
        x //= k
    return total

ans = sum(1 for i in range(1, 2025) if calc(i, 2) == calc(i, 4))

print(ans)

```

数字串个数

```cpp
#include <iostream>
using namespace std;
const int MOD = 1e9 + 7;

// 逐步累乘并取模
long long brute_force_pow(long long base, int exp) {
    long long result = 1;
    for (int i = 0; i < exp; i++) {
        result = (result * base) % MOD;
    }
    return result;
}

int main() {
    long long term1 = brute_force_pow(9, 10000);
    long long term2 = (2 * brute_force_pow(8, 10000)) % MOD;
    long long term3 = brute_force_pow(7, 10000);

    long long ans = (term1 - term2 + term3 + MOD) % MOD;
    cout << ans << endl;

    return 0;
}

```

```java
public class Main {
    static final int MOD = (int) 1e9 + 7;

    // 逐步累乘并取模
    public static long bruteForcePow(long base, int exp) {
        long result = 1;
        for (int i = 0; i < exp; i++) {
            result = (result * base) % MOD;
        }
        return result;
    }

    public static void main(String[] args) {
        long term1 = bruteForcePow(9, 10000);
        long term2 = (2 * bruteForcePow(8, 10000)) % MOD;
        long term3 = bruteForcePow(7, 10000);

        long ans = (term1 - term2 + term3 + MOD) % MOD;
        System.out.println(ans);
    }
}

```

```python
MOD = int(1e9 + 7)

# 逐步累乘并取模
def brute_force_pow(base, exp):
    result = 1
    for _ in range(exp):
        result = (result * base) % MOD
    return result

term1 = brute_force_pow(9, 10000)
term2 = (2 * brute_force_pow(8, 10000)) % MOD
term3 = brute_force_pow(7, 10000)

ans = (term1 - term2 + term3 + MOD) % MOD
print(ans)

```

连连看

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
int ri[2010][2010];
int le[2010][2010];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;
    vector<vector<int>> g(n, vector<int>(m));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> g[i][j];
        }
    }

    ll ans = 0;


    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int x = g[i][j];
            // 累加右对角线和左对角线的计数
            ans += ri[i + j][x] + le[j - i + 1000][x];
            // 更新右对角线和左对角线的计数
            ri[i + j][x]++;
            le[j - i + 1000][x]++;
        }
    }

    cout << ans * 2 << "\n"; // 输出结果
    return 0;
}
```

```java
import java.util.Scanner;

public class Main {
    static final int OFFSET = 1000; // 用于平移负索引
    static int[][] ri = new int[2010][2010]; // 右对角线计数
    static int[][] le = new int[2010][2010]; // 左对角线计数

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(), m = scanner.nextInt();
        int[][] g = new int[n][m];

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

        long ans = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                int x = g[i][j];
                ans += ri[i + j][x] + le[j - i + OFFSET][x];
                ri[i + j][x]++;
                le[j - i + OFFSET][x]++;
            }
        }

        System.out.println(ans * 2);
        scanner.close();
    }
}

```

```python
import sys

OFFSET = 1000  # 用于平移负索引
MAX_N = 2010   # 确保索引范围足够

def main():
    # 读取输入
    n, m = map(int, sys.stdin.readline().split())
    g = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]

    # 右对角线计数和左对角线计数
    ri = [{} for _ in range(MAX_N)]
    le = [{} for _ in range(MAX_N)]

    ans = 0

    # 遍历矩阵
    for i in range(n):
        for j in range(m):
            x = g[i][j]
            ans += ri[i + j].get(x, 0) + le[j - i + OFFSET].get(x, 0)
            ri[i + j][x] = ri[i + j].get(x, 0) + 1
            le[j - i + OFFSET][x] = le[j - i + OFFSET].get(x, 0) + 1

    print(ans * 2)

if __name__ == "__main__":
    main()

```

神奇闹钟

```cpp
#include <cstdio>

// 判断闰年
bool isLeap(int y) {
    return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
}

// 计算从1970年到目标时间的秒数
long long toSec(int y, int m, int d, int h, int min, int s) {
    int md[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (isLeap(y)) md[1] = 29;

    long long days = 0;
    for (int i = 1970; i < y; i++) days += isLeap(i) ? 366 : 365;
    for (int i = 1; i < m; i++) days += md[i - 1];
    days += d - 1;

    return days * 86400LL + h * 3600LL + min * 60LL + s;
}

// 将秒数转换为日期时间
void toDate(long long sec, int &y, int &m, int &d, int &h, int &min, int
&s) {
    long long days = sec / 86400LL;
    sec %= 86400LL;

    h = sec / 3600LL;
    sec %= 3600LL;
    min = sec / 60LL;
    s = sec % 60LL;

    y = 1970;
    while (true) {
        int cnt = isLeap(y) ? 366 : 365;
        if (days < cnt) break;
        days -= cnt;
        y++;
    }

    int md[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

```c
    if (isLeap(y)) md[1] = 29;

    m = 1;
    while (days >= md[m - 1]) {
        days -= md[m - 1];
        m++;
    }
    d = days + 1;
}

int main() {
    int T;
    scanf("%d", &T);

    while (T--) {
        int y, m, d, h, min, s, x;
        scanf("%d-%d-%d %d:%d:%d %d", &y, &m, &d, &h, &min, &s, &x);

        long long target = toSec(y, m, d, h, min, s);
        long long interval = x * 60LL;
        long long last = (target / interval) * interval;

        int aY, aM, aD, aH, aMin, aS;
        toDate(last, aY, aM, aD, aH, aMin, aS);

        printf("%04d-%02d-%02d %02d:%02d:%02d\n", aY, aM, aD, aH, aMin, aS);
    }

    return 0;
}
```

```java
import java.util.Scanner;

public class Main {
    // 判断闰年
    static boolean isLeap(int y) {
        return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
    }

    // 计算从1970年到目标时间的秒数
    static long toSec(int y, int m, int d, int h, int min, int s) {
        int[] md = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        if (isLeap(y)) md[1] = 29;

        long days = 0;
        for (int i = 1970; i < y; i++) days += isLeap(i) ? 366 : 365;
        for (int i = 1; i < m; i++) days += md[i - 1];
        days += d - 1;

        return days * 86400L + h * 3600L + min * 60L + s;
    }

    // 将秒数转换为日期时间
    static void toDate(long sec, int[] date) {
        long days = sec / 86400L;
        sec %= 86400L;

        date[3] = (int) (sec / 3600L);
        sec %= 3600L;
        date[4] = (int) (sec / 60L);
        date[5] = (int) sec;

        int y = 1970;
        while (true) {
            int cnt = isLeap(y) ? 366 : 365;
            if (days < cnt) break;
            days -= cnt;
            y++;
        }

        int[] md = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        if (isLeap(y)) md[1] = 29;

        int m = 1;
        while (days >= md[m - 1]) {
            days -= md[m - 1];
            m++;
        }
        date[0] = y;
        date[1] = m;
        date[2] = (int) (days + 1);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int T = scanner.nextInt();
        scanner.nextLine(); // 读取换行符

        while (T-- > 0) {
            String[] input = scanner.nextLine().split("[ :-]");
            int y = Integer.parseInt(input[0]);
            int m = Integer.parseInt(input[1]);
            int d = Integer.parseInt(input[2]);
            int h = Integer.parseInt(input[3]);
            int min = Integer.parseInt(input[4]);
            int s = Integer.parseInt(input[5]);
            int x = Integer.parseInt(input[6]);

            long target = toSec(y, m, d, h, min, s);
            long interval = x * 60L;
            long last = (target / interval) * interval;

            int[] result = new int[6];
            toDate(last, result);

            System.out.printf("%04d-%02d-%02d %02d:%02d:%02d\n",
                result[0], result[1], result[2], result[3], result[4], result[5]);
        }

        scanner.close();
    }
}

```

```python
# 判断闰年
def is_leap(y):
    return (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0)

# 计算从1970年到目标时间的秒数
def to_sec(y, m, d, h, min, s):
    md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if is_leap(y):
        md[1] = 29

    days = 0
    for i in range(1970, y):
        days += 366 if is_leap(i) else 365
    for i in range(1, m):
        days += md[i - 1]
    days += d - 1

    return days * 86400 + h * 3600 + min * 60 + s

# 将秒数转换为日期时间
def to_date(sec):
    days = sec // 86400
    sec %= 86400

    h = sec // 3600
    sec %= 3600
    min = sec // 60
    s = sec % 60

    y = 1970
    while True:
        cnt = 366 if is_leap(y) else 365
        if days < cnt:
            break
        days -= cnt
        y += 1

    md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if is_leap(y):
        md[1] = 29

    m = 1
    while days >= md[m - 1]:
        days -= md[m - 1]
        m += 1
    d = days + 1

    return y, m, d, h, min, s

def main():
    T = int(input())

    for _ in range(T):
        date_time = input().strip()
        x = int(input())

        # 解析输入的时间
        y, m, d = map(int, date_time.split(' ')[0].split('-'))
        h, min, s = map(int, date_time.split(' ')[1].split(':'))

        target = to_sec(y, m, d, h, min, s)
        interval = x * 60
        last = (target // interval) * interval

        aY, aM, aD, aH, aMin, aS = to_date(last)

        print(f"{aY:04d}-{aM:02d}-{aD:02d} {aH:02d}:{aMin:02d}:{aS:02d}")

if __name__ == "__main__":
    main()
```

```python
# 判断闰年
def is_leap(y):
    return (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0)

# 计算从1970年到目标时间的秒数
def to_sec(y, m, d, h, min, s):
    md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if is_leap(y):
        md[1] = 29

    days = 0
    for i in range(1970, y):
        days += 366 if is_leap(i) else 365
    for i in range(1, m):
        days += md[i - 1]
    days += d - 1

    return days * 86400 + h * 3600 + min * 60 + s

# 将秒数转换为日期时间
def to_date(sec):
    days = sec // 86400
    sec %= 86400

    h = sec // 3600
    sec %= 3600
    min = sec // 60
    s = sec % 60

    y = 1970
    while True:
        cnt = 366 if is_leap(y) else 365
        if days < cnt:
            break
        days -= cnt
        y += 1

    md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    if is_leap(y):
        md[1] = 29

    m = 1
    while days >= md[m - 1]:
        days -= md[m - 1]
        m += 1
    d = days + 1

    return y, m, d, h, min, s

def main():
    T = int(input())

    for _ in range(T):
        date_time = input()

        # 解析输入的时间
        y, m, d = map(int, date_time.split(' ')[0].split('-'))
        h, min, s = map(int, date_time.split(' ')[1].split(':'))
        x = (int)(date_time.split(' ')[2])
        target = to_sec(y, m, d, h, min, s)
        interval = x * 60
        last = (target // interval) * interval

        aY, aM, aD, aH, aMin, aS = to_date(last)

        print(f"{aY:04d}-{aM:02d}-{aD:02d} {aH:02d}:{aMin:02d}:{aS:02d}")

if __name__ == "__main__":
    main()
```

蓝桥村的真相

```cpp
#include <iostream>
using namespace std;
void solve(int n){

    int total = 0; // 所有满足条件的组合中，说谎者的总数
    int cnt = 0;   // 满足条件的组合数量

    // 遍历所有可能的身份组合（2^n 种）
    for (int mask = 0; mask < (1 << n); mask++) {
        bool ok = true; // 当前组合是否满足所有村民的陈述

        // 检查每个村民的陈述
        for (int i = 0; i < n; i++) {
            int n1 = (i + 1) % n; // i+1（环形）
            int n2 = (i + 2) % n; // i+2（环形）

            // 获取 i, i+1, i+2 的身份
            int c = (mask >> i) & 1;
            int a = (mask >> n1) & 1;
            int b = (mask >> n2) & 1;

            // 检查陈述是否成立
            if (c == 1) { // 当前村民是诚实者
                if (!((a == 1 && b == 0) || (a == 0 && b == 1))) {
                    ok = false;
                    break;
                }
            } else { // 当前村民是说谎者
                if ((a == 1 && b == 0) || (a == 0 && b == 1)) {
                    ok = false;
                    break;
                }
            }
        }

        // 如果当前组合满足所有陈述
        if (ok) {
            cnt++;
            // 统计当前组合中的说谎者数量
            for (int i = 0; i < n; i++) {
                if (((mask >> i) & 1) == 0) {
                    total++;
                }
            }
        }
    }

    // 输出结果
    cout << total << endl;

}
int main() {
    for(int i=1;i<=20;i++) solve(i);
    return 0;
}
```

```
import java.util.*;

public class Main {
    public static void solve(int n) {
        int total = 0; // 所有满足条件的组合中，说谎者的总数
        int cnt = 0;   // 满足条件的组合数量

        // 遍历所有可能的身份组合（2^n 种）
        for (int mask = 0; mask < (1 << n); mask++) {
            boolean ok = true; // 当前组合是否满足所有村民的陈述

            // 检查每个村民的陈述
            for (int i = 0; i < n; i++) {
                int n1 = (i + 1) % n; // i+1（环形）
                int n2 = (i + 2) % n; // i+2（环形）

                // 获取 i, i+1, i+2 的身份
                int c = (mask >> i) & 1;
                int a = (mask >> n1) & 1;
                int b = (mask >> n2) & 1;

                // 检查陈述是否成立
                if (c == 1) { // 当前村民是诚实者
                    if (!((a == 1 && b == 0) || (a == 0 && b == 1))) {
                        ok = false;
                        break;
                    }
                } else { // 当前村民是说谎者
```

```java
                    if ((a == 1 && b == 0) || (a == 0 && b == 1)) {
                        ok = false;
                        break;
                    }
                }
            }

            // 如果当前组合满足所有陈述
            if (ok) {
                cnt++;
                // 统计当前组合中的说谎者数量
                for (int i = 0; i < n; i++) {
                    if (((mask >> i) & 1) == 0) {
                        total++;
                    }
                }
            }
        }

        // 输出结果
        System.out.println(total);
    }

    public static void main(String[] args) {
        for (int i = 1; i <= 20; i++) solve(i);
    }
}

```

```python
def solve(n):
    total = 0  # 所有满足条件的组合中，说谎者的总数
    cnt = 0    # 满足条件的组合数量

    # 遍历所有可能的身份组合（2^n 种）
    for mask in range(1 << n):
        ok = True  # 当前组合是否满足所有村民的陈述

        # 检查每个村民的陈述
        for i in range(n):
            n1 = (i + 1) % n  # i+1（环形）
            n2 = (i + 2) % n  # i+2（环形）

            # 获取 i, i+1, i+2 的身份
            c = (mask >> i) & 1
            a = (mask >> n1) & 1
            b = (mask >> n2) & 1

            # 检查陈述是否成立
            if c == 1:  # 当前村民是诚实者
                if not ((a == 1 and b == 0) or (a == 0 and b == 1)):
                    ok = False
                    break
            else:  # 当前村民是说谎者
                if (a == 1 and b == 0) or (a == 0 and b == 1):
                    ok = False
                    break

        # 如果当前组合满足所有陈述
        if ok:
            cnt += 1
            # 统计当前组合中的说谎者数量
            total += sum(1 for i in range(n) if ((mask >> i) & 1) == 0)

    # 输出结果
    print(total)


if __name__ == "__main__":
    for i in range(1, 21):
        solve(i)
```