

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

```
1  #include <iostream>
2  using namespace std;
3  int calc(int x,int k){
4      int sum=0;
5      while(x!=0){
6          sum=sum+x%k;
7          x/=k;
8      }
9      return sum;
10 }
11 int main()
12 {
13     int ans=0;
14     for(int i=1;i<=2024;i++){
15         if(calc(i,2)==calc(i,4)){
16             ans++;
17         }
18     }
19     printf("%d",ans);
20     return 0;
21 }
```

```
1  public class Main {
2      public static int calc(int x, int k) {
3          int sum = 0;
4          while (x != 0) {
5              sum += x % k;
6              x /= k;
7          }
8          return sum;
9      }
10
11     public static void main(String[] args) {
12         int ans = 0;
13         for (int i = 1; i <= 2024; i++) {
14             if (calc(i, 2) == calc(i, 4)) {
15                 ans++;
16             }
17         }
18         System.out.println(ans);
19     }
20 }
21
```

```

1 def calc(x, k):
2     total = 0
3     while x != 0:
4         total += x % k
5         x //= k
6     return total
7
8 ans = sum(1 for i in range(1, 2025) if calc(i, 2) == calc(i, 4))
9
10 print(ans)
11

```

## 数字串个数

```

1 #include <iostream>
2 using namespace std;
3 const int MOD = 1e9 + 7;
4
5 // 逐步累乘并取模
6 long long brute_force_pow(long long base, int exp) {
7     long long result = 1;
8     for (int i = 0; i < exp; i++) {
9         result = (result * base) % MOD;
10    }
11    return result;
12 }
13
14 int main() {
15     long long term1 = brute_force_pow(9, 10000);
16     long long term2 = (2 * brute_force_pow(8, 10000)) % MOD;
17     long long term3 = brute_force_pow(7, 10000);
18
19     long long ans = (term1 - term2 + term3 + MOD) % MOD;
20     cout << ans << endl;
21
22     return 0;
23 }
24

```

```

1 public class Main {
2     static final int MOD = (int) 1e9 + 7;
3
4     // 逐步累乘并取模
5     public static long bruteForcePow(long base, int exp) {
6         long result = 1;
7         for (int i = 0; i < exp; i++) {
8             result = (result * base) % MOD;
9         }
10        return result;
11    }
12
13    public static void main(String[] args) {
14        long term1 = bruteForcePow(9, 10000);
15        long term2 = (2 * bruteForcePow(8, 10000)) % MOD;
16        long term3 = bruteForcePow(7, 10000);
17    }
18 }
19

```

```

17
18         long ans = (term1 - term2 + term3 + MOD) % MOD;
19         System.out.println(ans);
20     }
21 }
22

```

```

1  MOD = int(1e9 + 7)
2
3  # 逐步累乘并取模
4  def brute_force_pow(base, exp):
5      result = 1
6      for _ in range(exp):
7          result = (result * base) % MOD
8      return result
9
10 term1 = brute_force_pow(9, 10000)
11 term2 = (2 * brute_force_pow(8, 10000)) % MOD
12 term3 = brute_force_pow(7, 10000)
13
14 ans = (term1 - term2 + term3 + MOD) % MOD
15 print(ans)
16

```

## 连连看

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  int ri[2010][2010];
5  int le[2010][2010];
6  int main() {
7      ios::sync_with_stdio(false);
8      cin.tie(nullptr);
9
10     int n, m;
11     cin >> n >> m;
12     vector<vector<int>> g(n, vector<int>(m));
13     for (int i = 0; i < n; i++) {
14         for (int j = 0; j < m; j++) {
15             cin >> g[i][j];
16         }
17     }
18
19     ll ans = 0;
20
21
22     for (int i = 0; i < n; i++) {
23         for (int j = 0; j < m; j++) {
24             int x = g[i][j];
25             // 累加右对角线和左对角线的计数
26             ans += ri[i + j][x] + le[j - i + 1000][x];
27             // 更新右对角线和左对角线的计数
28             ri[i + j][x]++;
29             le[j - i + 1000][x]++;
30         }
31     }
32
33

```

```

31     }
32
33     cout << ans * 2 << "\n"; // 输出结果
34     return 0;
35 }

```

```

1  import java.util.Scanner;
2
3  public class Main {
4      static final int OFFSET = 1000; // 用于平移负索引
5      static int[][] ri = new int[2010][2010]; // 右对角线计数
6      static int[][] le = new int[2010][2010]; // 左对角线计数
7
8      public static void main(String[] args) {
9          Scanner scanner = new Scanner(System.in);
10         int n = scanner.nextInt(), m = scanner.nextInt();
11         int[][] g = new int[n][m];
12
13         for (int i = 0; i < n; i++) {
14             for (int j = 0; j < m; j++) {
15                 g[i][j] = scanner.nextInt();
16             }
17         }
18
19         long ans = 0;
20
21         for (int i = 0; i < n; i++) {
22             for (int j = 0; j < m; j++) {
23                 int x = g[i][j];
24                 ans += ri[i + j][x] + le[j - i + OFFSET][x];
25                 ri[i + j][x]++;
26                 le[j - i + OFFSET][x]++;
27             }
28         }
29
30         System.out.println(ans * 2);
31         scanner.close();
32     }
33 }
34

```

```

1  import sys
2
3  OFFSET = 1000 # 用于平移负索引
4  MAX_N = 2010 # 确保索引范围足够
5
6  def main():
7      # 读取输入
8      n, m = map(int, sys.stdin.readline().split())
9      g = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]
10
11     # 右对角线计数和左对角线计数
12     ri = [{ } for _ in range(MAX_N)]
13     le = [{ } for _ in range(MAX_N)]
14
15     ans = 0

```

```

16
17     # 遍历矩阵
18     for i in range(n):
19         for j in range(m):
20             x = g[i][j]
21             ans += ri[i + j].get(x, 0) + le[j - i + OFFSET].get(x, 0)
22             ri[i + j][x] = ri[i + j].get(x, 0) + 1
23             le[j - i + OFFSET][x] = le[j - i + OFFSET].get(x, 0) + 1
24
25     print(ans * 2)
26
27 if __name__ == "__main__":
28     main()
29

```

## 神奇闹钟

```

1  #include <stdio>
2
3  // 判断闰年
4  bool isLeap(int y) {
5      return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
6  }
7
8  // 计算从1970年到目标时间的秒数
9  long long toSec(int y, int m, int d, int h, int min, int s) {
10     int md[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
11     if (isLeap(y)) md[1] = 29;
12
13     long long days = 0;
14     for (int i = 1970; i < y; i++) days += isLeap(i) ? 366 : 365;
15     for (int i = 1; i < m; i++) days += md[i - 1];
16     days += d - 1;
17
18     return days * 86400LL + h * 3600LL + min * 60LL + s;
19 }
20
21 // 将秒数转换为日期时间
22 void toDate(long long sec, int &y, int &m, int &d, int &h, int &min, int
&s) {
23     long long days = sec / 86400LL;
24     sec %= 86400LL;
25
26     h = sec / 3600LL;
27     sec %= 3600LL;
28     min = sec / 60LL;
29     s = sec % 60LL;
30
31     y = 1970;
32     while (true) {
33         int cnt = isLeap(y) ? 366 : 365;
34         if (days < cnt) break;
35         days -= cnt;
36         y++;
37     }
38
39     int md[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

```

```

40     if (isLeap(y)) md[1] = 29;
41
42     m = 1;
43     while (days >= md[m - 1]) {
44         days -= md[m - 1];
45         m++;
46     }
47     d = days + 1;
48 }
49
50 int main() {
51     int T;
52     scanf("%d", &T);
53
54     while (T--) {
55         int y, m, d, h, min, s, x;
56         scanf("%d-%d-%d %d:%d:%d", &y, &m, &d, &h, &min, &s, &x);
57
58         long long target = toSec(y, m, d, h, min, s);
59         long long interval = x * 60LL;
60         long long last = (target / interval) * interval;
61
62         int aY, aM, aD, aH, aMin, aS;
63         toDate(last, aY, aM, aD, aH, aMin, aS);
64
65         printf("%04d-%02d-%02d %02d:%02d:%02d\n", aY, aM, aD, aH, aMin,
aS);
66     }
67
68     return 0;
69 }

```

```

1  import java.util.Scanner;
2
3  public class Main {
4      // 判断闰年
5      static boolean isLeap(int y) {
6          return (y % 4 == 0 && y % 100 != 0) || (y % 400 == 0);
7      }
8
9      // 计算从1970年到目标时间的秒数
10     static long toSec(int y, int m, int d, int h, int min, int s) {
11         int[] md = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
12         if (isLeap(y)) md[1] = 29;
13
14         long days = 0;
15         for (int i = 1970; i < y; i++) days += isLeap(i) ? 366 : 365;
16         for (int i = 1; i < m; i++) days += md[i - 1];
17         days += d - 1;
18
19         return days * 86400L + h * 3600L + min * 60L + s;
20     }
21
22     // 将秒数转换为日期时间
23     static void toDate(long sec, int[] date) {
24         long days = sec / 86400L;
25         sec %= 86400L;

```

```

26
27     date[3] = (int) (sec / 3600L);
28     sec %= 3600L;
29     date[4] = (int) (sec / 60L);
30     date[5] = (int) sec;
31
32     int y = 1970;
33     while (true) {
34         int cnt = isLeap(y) ? 366 : 365;
35         if (days < cnt) break;
36         days -= cnt;
37         y++;
38     }
39
40     int[] md = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
41     if (isLeap(y)) md[1] = 29;
42
43     int m = 1;
44     while (days >= md[m - 1]) {
45         days -= md[m - 1];
46         m++;
47     }
48     date[0] = y;
49     date[1] = m;
50     date[2] = (int) (days + 1);
51 }
52
53 public static void main(String[] args) {
54     Scanner scanner = new Scanner(System.in);
55     int T = scanner.nextInt();
56     scanner.nextLine(); // 读取换行符
57
58     while (T-- > 0) {
59         String[] input = scanner.nextLine().split("[:-]");
60         int y = Integer.parseInt(input[0]);
61         int m = Integer.parseInt(input[1]);
62         int d = Integer.parseInt(input[2]);
63         int h = Integer.parseInt(input[3]);
64         int min = Integer.parseInt(input[4]);
65         int s = Integer.parseInt(input[5]);
66         int x = Integer.parseInt(input[6]);
67
68         long target = toSec(y, m, d, h, min, s);
69         long interval = x * 60L;
70         long last = (target / interval) * interval;
71
72         int[] result = new int[6];
73         toDate(last, result);
74
75         System.out.printf("%04d-%02d-%02d %02d:%02d:%02d\n",
76             result[0], result[1], result[2], result[3], result[4],
77             result[5]);
78     }
79     scanner.close();
80 }
81 }
82

```

```

1  # 判断闰年
2  def is_leap(y):
3      return (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0)
4
5  # 计算从1970年到目标时间的秒数
6  def to_sec(y, m, d, h, min, s):
7      md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
8      if is_leap(y):
9          md[1] = 29
10
11     days = 0
12     for i in range(1970, y):
13         days += 366 if is_leap(i) else 365
14     for i in range(1, m):
15         days += md[i - 1]
16     days += d - 1
17
18     return days * 86400 + h * 3600 + min * 60 + s
19
20 # 将秒数转换为日期时间
21 def to_date(sec):
22     days = sec // 86400
23     sec %= 86400
24
25     h = sec // 3600
26     sec %= 3600
27     min = sec // 60
28     s = sec % 60
29
30     y = 1970
31     while True:
32         cnt = 366 if is_leap(y) else 365
33         if days < cnt:
34             break
35         days -= cnt
36         y += 1
37
38     md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
39     if is_leap(y):
40         md[1] = 29
41
42     m = 1
43     while days >= md[m - 1]:
44         days -= md[m - 1]
45         m += 1
46     d = days + 1
47
48     return y, m, d, h, min, s
49
50 def main():
51     T = int(input())
52
53     for _ in range(T):
54         date_time = input().strip()
55         x = int(input())
56

```



```

57     # 解析输入的时间
58     y, m, d = map(int, date_time.split(' ')[0].split('-'))
59     h, min, s = map(int, date_time.split(' ')[1].split(':'))
60
61     target = to_sec(y, m, d, h, min, s)
62     interval = x * 60
63     last = (target // interval) * interval
64
65     aY, aM, aD, aH, aMin, aS = to_date(last)
66
67     print(f"{aY:04d}-{aM:02d}-{aD:02d} {aH:02d}:{aMin:02d}:{aS:02d}")
68
69 if __name__ == "__main__":
70     main()

```

```

1  # 判断闰年
2  def is_leap(y):
3      return (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0)
4
5  # 计算从1970年到目标时间的秒数
6  def to_sec(y, m, d, h, min, s):
7      md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
8      if is_leap(y):
9          md[1] = 29
10
11     days = 0
12     for i in range(1970, y):
13         days += 366 if is_leap(i) else 365
14     for i in range(1, m):
15         days += md[i - 1]
16     days += d - 1
17
18     return days * 86400 + h * 3600 + min * 60 + s
19
20 # 将秒数转换为日期时间
21 def to_date(sec):
22     days = sec // 86400
23     sec %= 86400
24
25     h = sec // 3600
26     sec %= 3600
27     min = sec // 60
28     s = sec % 60
29
30     y = 1970
31     while True:
32         cnt = 366 if is_leap(y) else 365
33         if days < cnt:
34             break
35         days -= cnt
36         y += 1
37
38     md = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
39     if is_leap(y):
40         md[1] = 29
41
42     m = 1

```

```

43     while days >= md[m - 1]:
44         days -= md[m - 1]
45         m += 1
46     d = days + 1
47
48     return y, m, d, h, min, s
49
50 def main():
51     T = int(input())
52
53     for _ in range(T):
54         date_time = input()
55
56         # 解析输入的时间
57         y, m, d = map(int, date_time.split(' ')[0].split('-'))
58         h, min, s = map(int, date_time.split(' ')[1].split(':'))
59         x = (int)(date_time.split(' ')[2])
60         target = to_sec(y, m, d, h, min, s)
61         interval = x * 60
62         last = (target // interval) * interval
63
64         aY, aM, aD, aH, aMin, aS = to_date(last)
65
66         print(f"{aY:04d}-{aM:02d}-{aD:02d} {aH:02d}:{aMin:02d}:{aS:02d}")
67
68 if __name__ == "__main__":
69     main()

```

## 蓝桥村的真相

```

1  #include <iostream>
2  using namespace std;
3  void solve(int n){
4
5      int total = 0; // 所有满足条件的组合中，说谎者的总数
6      int cnt = 0;   // 满足条件的组合数量
7
8      // 遍历所有可能的身份组合 (2^n 种)
9      for (int mask = 0; mask < (1 << n); mask++) {
10         bool ok = true; // 当前组合是否满足所有村民的陈述
11
12         // 检查每个村民的陈述
13         for (int i = 0; i < n; i++) {
14             int n1 = (i + 1) % n; // i+1 (环形)
15             int n2 = (i + 2) % n; // i+2 (环形)
16
17             // 获取 i, i+1, i+2 的身份
18             int c = (mask >> i) & 1;
19             int a = (mask >> n1) & 1;
20             int b = (mask >> n2) & 1;
21
22             // 检查陈述是否成立
23             if (c == 1) { // 当前村民是诚实者
24                 if (!(a == 1 && b == 0) || (a == 0 && b == 1)) {
25                     ok = false;
26                     break;
27                 }

```

```

28         } else { // 当前村民是说谎者
29             if ((a == 1 && b == 0) || (a == 0 && b == 1)) {
30                 ok = false;
31                 break;
32             }
33         }
34     }
35
36     // 如果当前组合满足所有陈述
37     if (ok) {
38         cnt++;
39         // 统计当前组合中的说谎者数量
40         for (int i = 0; i < n; i++) {
41             if (((mask >> i) & 1) == 0) {
42                 total++;
43             }
44         }
45     }
46 }
47
48 // 输出结果
49 cout << total << endl;
50
51 }
52 int main() {
53     for(int i=1;i<=20;i++) solve(i);
54     return 0;
55 }

```

```

1  import java.util.*;
2
3  public class Main {
4      public static void solve(int n) {
5          int total = 0; // 所有满足条件的组合中，说谎者的总数
6          int cnt = 0;   // 满足条件的组合数量
7
8          // 遍历所有可能的身份组合 (2^n 种)
9          for (int mask = 0; mask < (1 << n); mask++) {
10             boolean ok = true; // 当前组合是否满足所有村民的陈述
11
12             // 检查每个村民的陈述
13             for (int i = 0; i < n; i++) {
14                 int n1 = (i + 1) % n; // i+1 (环形)
15                 int n2 = (i + 2) % n; // i+2 (环形)
16
17                 // 获取 i, i+1, i+2 的身份
18                 int c = (mask >> i) & 1;
19                 int a = (mask >> n1) & 1;
20                 int b = (mask >> n2) & 1;
21
22                 // 检查陈述是否成立
23                 if (c == 1) { // 当前村民是诚实者
24                     if (!(a == 1 && b == 0) || (a == 0 && b == 1)) {
25                         ok = false;
26                         break;
27                     }
28                 } else { // 当前村民是说谎者

```

```

29         if ((a == 1 && b == 0) || (a == 0 && b == 1)) {
30             ok = false;
31             break;
32         }
33     }
34 }
35
36 // 如果当前组合满足所有陈述
37 if (ok) {
38     cnt++;
39     // 统计当前组合中的说谎者数量
40     for (int i = 0; i < n; i++) {
41         if ((mask >> i) & 1 == 0) {
42             total++;
43         }
44     }
45 }
46 }
47
48 // 输出结果
49 System.out.println(total);
50 }
51
52 public static void main(String[] args) {
53     for (int i = 1; i <= 20; i++) solve(i);
54 }
55 }
56

```

```

1  def solve(n):
2      total = 0 # 所有满足条件的组合中，说谎者的总数
3      cnt = 0   # 满足条件的组合数量
4
5      # 遍历所有可能的身份组合 (2^n 种)
6      for mask in range(1 << n):
7          ok = True # 当前组合是否满足所有村民的陈述
8
9          # 检查每个村民的陈述
10         for i in range(n):
11             n1 = (i + 1) % n # i+1 (环形)
12             n2 = (i + 2) % n # i+2 (环形)
13
14             # 获取 i, i+1, i+2 的身份
15             c = (mask >> i) & 1
16             a = (mask >> n1) & 1
17             b = (mask >> n2) & 1
18
19             # 检查陈述是否成立
20             if c == 1: # 当前村民是诚实者
21                 if not ((a == 1 and b == 0) or (a == 0 and b == 1)):
22                     ok = False
23                     break
24             else: # 当前村民是说谎者
25                 if (a == 1 and b == 0) or (a == 0 and b == 1):
26                     ok = False
27                     break
28

```

```
29         # 如果当前组合满足所有陈述
30         if ok:
31             cnt += 1
32             # 统计当前组合中的说谎者数量
33             total += sum(1 for i in range(n) if ((mask >> i) & 1) == 0)
34
35         # 输出结果
36         print(total)
37
38
39 if __name__ == "__main__":
40     for i in range(1, 21):
41         solve(i)
42
```