E7 - Solution

A 不要温和地走进那个良夜

难度	考点
1	二维数组、字符串

题目分析

建立二维字符数组，初始化每行为对应字符串，每组数据输出对应的字符串即可。

示例代码 1

```c
#include <stdio.h>

int main() {
    int n;
    char poem[][105] = {
        "Do not go gentle into that good night,",
        "Old age should burn and rave at close of day;",
        "Rage, rage against the dying of the light.",
        "Though wise men at their end know dark is right,",
        "Because their words had forked no lightning they",
        "Do not go gentle into that good night.",
        "Good men, the last wave by, crying how bright",
        "Their frail deeds might have danced in a green bay,",
        "Rage, rage against the dying of the light.",
        "Wild men who caught and sang the sun in flight,",
        "And learn, too late, they grieved it on its way,",
        "Do not go gentle into that good night.",
        "Grave men, near death, who see with blinding sight",
        "Blind eyes could blaze like meteors and be gay,",
        "Rage, rage against the dying of the light.",
        "And you, my father, there on the sad height,",
        "Curse, bless me now with your fierce tears, I pray.",
        "Do not go gentle into that good night.",
        "Rage, rage against the dying of the light."
    };
    while (~scanf("%d", &n))
        printf("%s\n", poem[n - 1]);

    return 0;
}
```

示例代码 2

利用指针数组实现。区别在于该方法指针指向的字符串为常量，不可修改。

```c
#include <stdio.h>

int main() {
```

```
    int n;
    char *poem[] = {
        "Do not go gentle into that good night,",
        "Old age should burn and rave at close of day;",
        "Rage, rage against the dying of the light.",
        "Though wise men at their end know dark is right,",
        "Because their words had forked no lightning they",
        "Do not go gentle into that good night.",
        "Good men, the last wave by, crying how bright",
        "Their frail deeds might have danced in a green bay,",
        "Rage, rage against the dying of the light.",
        "Wild men who caught and sang the sun in flight,",
        "And learn, too late, they grieved it on its way,",
        "Do not go gentle into that good night.",
        "Grave men, near death, who see with blinding sight",
        "Blind eyes could blaze like meteors and be gay,",
        "Rage, rage against the dying of the light.",
        "And you, my father, there on the sad height,",
        "Curse, bless me now with your fierce tears, I pray.",
        "Do not go gentle into that good night.",
        "Rage, rage against the dying of the light."
    };
    while (~scanf("%d", &n))
        printf("%s\n", poem[n - 1]);

    return 0;
}
```

B 懒獭排序

难度	考点
1	循环结构、分支结构

题目分析

理解好题意后，本题就是这样的一个任务：

- 如果某数比前面的数都大，则输出这个数
- 否则不输出这个数

可以用一个变量来维护前面所有数的最大值，显然某数比前面的数都大，等价于该数比前面所有数的最大值大。

示例代码

```
#include <stdio.h>
int a[100005];

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

```
    for(int i = 1; i <= n ;i++)
        scanf("%d", &a[i]);

    // 开始时一定要给该变量一个比输入最小值还小的值
    // 利用浮点数的科学计数法
    int tem = -2e9;
    for(int i = 1; i <= n ;i++) {
        if(a[i] >= tem) {
            printf("%d ", a[i]);
            tem = a[i];
        }
    }
    return 0;
}
```

C 爱吃糖葫芦2

难度	考点
2	字符串拼接 `strcat`

题目分析

本题输入为三个字符串，输出为字符串拼接后的后半部分。需要注意的有3点：

1. 用于存储顺序的数组 `order` 应该满足题目要求，长度不小于1000
2. 用于存储拼接字符串的 `char` 类型数组要足够大，在本题应不小于$1000 \times 1000 = 1000000$
3. `ans` 字符串的后半部分的范围应当是$[\frac{|ans|}{2}, |ans|)$

有很多同学 `TLE`，得到 `0.75` 分，具体原因可以参考下面代码的3种 `case`。

- 对于 `case1`，因为每次判断都会调用 `strlen()` 函数，由于每次调用 `strlen()` 会返回同一个值，会造成多余的资源(时空间)消耗。
- 在 `case2` 中，我们仅调用一次并保存至变量 `len` 中即可解决这个问题
- 当然，也可以使用 `case3` 指针调用的方式避免这种问题。

示例代码

```
#include <stdio.h>
#include <string.h>

int n;
char s[10][1009];
char order[1009];
char ans[1000009];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", s[i]);
```

```
    }
    scanf("%s", order);
    for (int i = 0; i < strlen(order); i++) {
        strcat(ans, S[order[i] - '1']);
    }
//  case1:只能0.75,会卡一个数据点
//    for (int i = strlen(ans) / 2; i < strlen(ans); i++)
//        printf("%c", ans[i]);

//  case2:可以ac
//    for (int len = strlen(ans), i = len / 2; i < len; i++)
//        printf("%c", ans[i]);

//  case3:可以ac
    printf("%s", ans + strlen(ans) / 2);
}
```

D 魔法师水獭与标准咒语指南 2

难度	考点
3	strstr、字符串、数组与指针

题目分析

直接使用 `strstr` 函数在字符串 `str` 中查找子串的首地址值。

如果查找操作返回地址值为 `NULL` 说明目标子串不存在，如果返回地址值不为 `NULL`，说明目标子串存在，记录本次查找到的子串首地址，同时将计数变量自增 1，然后从这次查到的位置的**后一个位置**开始，继续循环执行查找、计数累加、记录首地址的操作，直到查找失败返回 `NULL`，此时记录下的最后一个地址就是 `str` 字符串中最后一个目标子串的首地址

因为一个数组在内存中是连续存储的，而数组首地址指向的是数组中下标为 `0` 的元素的位置，将数组中某个指定位置的地址值与数组首地址作差后，得到的结果就是两个位置的下标差，即指定位置在数组中的下标。所以将前面查到的目标子串最后一次出现位置的首地址减去 `str` 数组首地址，就得到了目标子串首字母在 `str` 数组中的位置下标。

课下部分同学询问为什么计数变量有时候 $+1$，有时候 $+\ \mathrm{strlen}(\mathrm{substr})$，其实前者相当于不同的子串之间允许重合，后者相当于不同子串之间不能重合（可以用第二个样例思考），要视具体情况选用正确的做法。

示例代码

```
#include <stdio.h>
#include <string.h>

char str[1005];
char substr[1005];

int main() {
    gets(str);
    int n;
```

```
    scanf("%d", &n);
    getchar();
    for(int i = 1; i <= n; i++) {
        gets(substr);
        char *r, *pos = str;
        // 表示有没有
        r = strstr(pos, substr);
        if(r != NULL) {
            while(r != NULL) {
                printf("%d ", r - str);
                // 自增一位
                pos = r + 1;
                r = strstr(pos, substr);
            }
        } else
            printf("Spell Not Found!");
        printf("\n");
    }
    return 0;
}
```

E 字符轮换

难度	考点
4	字符串

题目分析

可以用一个变量表示正在读到的序列是数字序列还是非数字序列，然后一直读，读到另一种的时候就改变这个变量的状态表示切换到对另一种序列的读入，同时将刚读完的序列输出（如果是数字序列就输出对应的字符，否则输出累加后的值，在读入的同时做累加）。示例代码如下。

示例代码

```
#include <stdio.h>
#include <string.h>

int main() {
    int T;
    char deal[105];
    while (gets(deal) != NULL) {
        int len = strlen(deal);
        int numing = 0, num = 0, otr = 0;
        for (int i = 0; i < len; ++i) {
            if (isdigit(deal[i])) {
                if (!numing && otr != 0) printf("%d", otr);
                numing = 1;
                otr = 0;
                num = 10 * num + deal[i] - '0';
            } else {
                if (numing) printf("%c", num);
```

```
                otr += deal[i];
                numing = 0;
                num = 0;
            }
        }
        if (numing) printf("%c\n", num);
        else printf("%d\n", otr);
    }
    return 0;
}
```

F 旋转变化!

难度	考点
5	二维数组

题目分析

首先观察每个点会转多少次，可以发现，从外往内的第 i 圈上的点会转 i 次。

之后观察旋转方式有没有什么特征。对于(x,y) 位置的点，第一次会到达 $(y,N+1-x)$, 第二次会到达 $(N+1-x,N+1-y)$，第三次会到达 $(N+1-y,x)$，第四次是 (x,y)。

也就是说，四次一循环。

结合以上两个性质，就可以很快的解决本题目了。

示例代码

```
#include<stdio.h>
#include<string.h>
int n;
char s[3005][3005];
char b[3005][3005];
void deal(int i,int j){
    int k=100000;
    if(i<k) k=i;
    if(j<k) k=j;
    if(n+1-i<k) k=n+1-i;
    if(n+1-j<k) k=n+1-j;
    k%=4;
    if(k==0){
        b[i][j]=s[i][j];
    }
    if(k==1){
        b[j][n+1-i]=s[i][j];
    }
    if(k==2){
        b[n+1-i][n+1-j]=s[i][j];
    }
    if(k==3){
        b[n+1-j][i]=s[i][j];
```

```
    }
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%s",s[i]+1);
    }
    for(int i=1;i<=n;++i){
        for(int j=1;j<=n;++j)
            deal(i,j);
    }
    for(int i=1;i<=n;++i){
        puts(b[i]+1);
    }
    return 0;
}
```

G IP 地址合法性判断

难度	考点
5	字符串

题目分析

本题主要运用 **模拟** 方法对字符串进行处理，结合题目规定的合法格式逐步判断即可。

示例代码 1

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>

char ip[10086];
// 判断是否为合法的 IPv4 地址
int isIPv4() {
    int n = strlen(ip), count = 0;
    for (int i = 0; i < n && count <= 3;) {
        int j = i, x = 0;
        // 找到一段连续的数字
        while (j < n && isdigit(ip[j]) && x <= 255) {
            x = x * 10 + (ip[j++] - '0');
        }
        // 未找到任何数字，或包含前导零，或数字超过 255
        if (i == j || (j - i > 1 && ip[i] == '0') || x > 255)
            return 0;
        i = j + 1;
        if (j == n)
            break;
        // 分隔符不是 '.'
        if (ip[j] != '.')
            return 0;
```

```
            count++;
        }
        // 有 3 个 '.' 且不在两端
        return count == 3 && ip[0] != '.' && ip[n - 1] != '.';
}

// 判断是否为合法的 IPv6 地址
int isIPv6() {
        int n = strlen(ip), count = 0;
        for (int i = 0; i < n && count <= 7;) {
            int j = i;
            // 找到一段连续的十六进制字符
            while (j < n && ((ip[j] >= 'a' && ip[j] <= 'f') || (ip[j] >= 'A' && ip[j]
<= 'F') || isdigit(ip[j]))) {
                j++;
            }
            // 长度超过 4，或未找到合法的十六进制字符
            if (i == j || j - i > 4)
                return 0;
            i = j + 1;
            if (j == n)
                continue;
            // 分隔符不是 ':'
            if (ip[j] != ':')
                return 0;
            count++;
        }
        // 有 7 个 ':' 且不在两端
        return count == 7 && ip[0] != ':' && ip[n - 1] != ':';
}

int main() {
        int n;
        scanf("%d", &n);
        while (n--) {
            scanf("%s", ip);
            if (isIPv4()) {
                printf("IPv4\n");
            }
            else if (isIPv6()) {
                printf("IPv6\n");
            }
            else {
                printf("64vPI\n");
            }
        }
        return 0;
}
```

示例代码 2

库函数 `strtok` （需要头文件 `string.h` ）可以按给定字符将字符串分割；
库函数 `atoi` （需要头文件 `stdlib.h` ）可以将字面为整数的字符串转换为 `int` 类型；
库函数 `isxdigit` （需要头文件 `ctype.h` ）可以判断给定字符是否为十六进制字符。

```c
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 判断是否为有效的IPv4地址
int isIpV4(char *IP) {
    // 以 '.' 结尾
    if (IP[strlen(IP) - 1] == '.') {
        return 0;
    }
    char *token = strtok(IP, ".");
    int cnt = 0;
    while (token != NULL) {
        cnt++;
        // 每部分的长度不在1到3之间
        if (strlen(token) > 3 || strlen(token) == 0) {
            return 0;
        }
        // 包含前导零且长度不为1
        if (token[0] == '0' && strlen(token) != 1) {
            return 0;
        }
        // 计算数字
        for (int i = 0; i < strlen(token); i++) {
            // 不是数字
            if (!isdigit(token[i])) {
                return 0;
            }
        }
        // 将字符串转换为整数
        int num = atoi(token);
        // 数字超过255
        if (num < 0 || num > 255) {
            return 0;
        }
        token = strtok(NULL, ".");
    }
    // IPv4地址有4个部分
    return cnt == 4;
}

// 判断是否为有效的IPv6地址
int isIpV6(char *IP) {
    // 以 ':' 结尾
    if (IP[strlen(IP) - 1] == ':') {
        return 0;
    }
    char *token = strtok(IP, ":");
    int cnt = 0;
    while (token != NULL) {
        cnt++;
        // 每部分的长度不在1到4之间
        if (strlen(token) > 4 || strlen(token) == 0) {
            return 0;
```

```
        }
        // 不是合法的十六进制字符
        for (int i = 0; i < strlen(token); i++) {
            if (!isxdigit(token[i])) {
                return 0;
            }
        }
        token = strtok(NULL, ":");
    }
    // IPv6地址有8个部分
    return cnt == 8;
}

const char *isValid(char *IP) {
    // 若包含'.'，则检查是否是IPv4地址
    if (strchr(IP, '.') != NULL) {
        return isIpV4(IP) ? "IPv4" : "64vPI";
    }
    // 若包含':'，则检查是否是IPv6地址
    else if (strchr(IP, ':') != NULL) {
        return isIpV6(IP) ? "IPv6" : "64vPI";
    } else {
        return "64vPI";
    }
}

int main() {
    int n;
    scanf("%d", &n);
    while (n--) {
        char IP[100];
        scanf("%s", IP);
        printf("%s\n", isValid(IP));
    }
    return 0;
}
```

H 水獭哲学家就餐问题

难度	考点
6	模拟

题目分析

本题是一道较为复杂的模拟问题，核心思想在于观察当前就餐的水獭哲学家什么时候结束就餐，将时间快进到有水獭哲学家停止就餐的时候。

请结合下面带有详细注释的代码理解模拟过程。

示例代码

```
# include <stdio.h>
# include <limits.h>

int n, m; // n 表示水獭哲学家的数量，m 表示水獭哲学家就餐的总次数
int a[100000]; // a[k] 表示第 k 个水獭哲学家每次就餐的耗时
int b[100000]; // b[k] 表示第 k 个水獭哲学家需要就餐的次数
int c[100000]; // 如果第 k 个水獭哲学家正在就餐，那么 c[k] = 1，否则 c[k] = 0
long long int d[100000]; // d[k] 表示第 k 个水獭哲学家最新一次就餐结束的时间
long long int t; // t 表示当前时间

// 令当前可以就餐的水獭哲学家中最饿的那一个水獭哲学家开始就餐，然后返回这个水獭哲学家的编号
// 如果当前没有水獭哲学家可以开始就餐，那么返回 0
int f() {
    int x = 0;
    long long int y = LLONG_MAX;
    for (int i = 1; i <= n; i += 1) {
        if (b[i] > 0 && c[i] == 0) {
            if (c[i == 1 ? n : i - 1] == 0 && c[i == n ? 1 : i + 1] == 0) {
                if (d[i] < y) {
                    x = i;
                    y = d[i];
                }
            }
        }
    }
    if (x > 0) {
        b[x] -= 1;
        c[x] = 1;
        d[x] = t + a[x];
        printf("time %lld : otter %d pick-up\n", t, x);
    }
    return x;
}

// 观察当前就餐的水獭哲学家什么时候结束就餐，将时间快进到有水獭哲学家停止就餐的时候，然后让它们停止就餐
void g() {
    long long int y = LLONG_MAX;
    for (int i = 1; i <= n; i += 1) {
        if (c[i] == 1) {
            if (d[i] < y) {
                y = d[i];
            }
        }
    }
    t = y;
    for (int i = 1; i <= n; i += 1) {
        if (c[i] == 1 && d[i] == t) {
            c[i] = 0;
            m -= 1;
            printf("time %lld : otter %d put-down\n", t, i);
        }
    }
```

```
}
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i += 1) {
        scanf("%d", &a[i]);
    }
    for (int i = 1; i <= n; i += 1) {
        scanf("%d", &b[i]);
        m += b[i];
    }
    while (m > 0) {
        while (f()) {
            ;
        }
        g();
    }
}
```

I ddz 与 vya 分苹果

难度	考点
6	循环、数学

题目分析

由于 $s = |a_1 - a_2| + |a_2 - a_3| = |a_2 - a_1| + |a_2 - a_3|$ ，a_1 与 a_3 对称， a_2 是最特殊的，我们侧重考虑他。

假设 ddz 从袋子中拿出的苹果的下标分别是 i_1, i_2, i_3 ，他们只可能满足以下四种情况中的一种：

1. $i_1 < i_2 < i_3$
2. $i_3 < i_2 < i_1$
3. $i_2 < min(i_1, i_3)$
4. $max(i_1, i_3) < i_2$

事实上， vya 可以强制 ddz 走到上述情况中的任意一种，对于任意一种情况 $i_{j_1} \le i_{j_2} \le i_{j_3}$ ，将 1 到 i_{j_1} 放入 j_1 号袋子，将 $i_{j_1} + 1$ 到 $i_{j_3} - 1$ 放入 j_2 号袋子，将 i_{j_3} 到 n 放入 j_3 号袋子即可。

显然，对于前两种情况，交换 i_1 与 i_2 或者交换 i_3 与 i_2 变成第三或第四种情况会更优，所以我们只考虑后面两种情况。

对于第三种情况，假设 $i_2 + 1 < min(i_1, i_3)$ ，我们看 $i_2 + 1$ 的几种情况：

1. 将 $i_2 + 1$ 放入第一个袋子，那 ddz 在第一个袋子中肯定会选择 $i_2 + 1$ 而不是 i_1 ，因为 $w_{i_1} - w_{i_2} \ge w_{i_2+1} - w_{i_2}$
2. 将 $i_2 + 1$ 放入第二个袋子，那 ddz 在第二个袋子中肯定会选择 $i_2 + 1$ 而不是 i_2 ，因为 $(w_{i_1} - w_{i_2}) + (w_{i_3} - w_{i_2}) \ge (w_{i_1} - w_{i_2+1}) + (w_{i_3} - w_{i_2+1})$
3. 将 $i_2 + 1$ 放入第三个袋子，那 ddz 在第三个袋子中肯定会选择 $i_2 + 1$ 而不是 i_3 ，因为 $w_{i_3} - w_{i_2} \ge w_{i_2+1} - w_{i_2}$

所以 $i_2+1<min(i_1,i_3)$ 一定更劣，所以 $i_2+1=min(i_1,i_3)$，同理对于第四种情况我们有 $i_2-1=max(i_1,i_3)$。

而对于第三种情况来说 $max(i_1,i_3)$ 越大越好，所以 $max(i_1,i_3)=n$，同理对于第四种情况 $min(i_1,i_3)=1$。

所以答案为：

$$\max(\max_{3\le i\le n}((w_i-w_{i-1})+(w_i-w_1)),\max_{1\le i\le n-2}((w_{i+1}-w_i)+(w_n-w_i)))$$

示例代码

```c
#include <stdio.h>
#define max(a, b) ((a) > (b) ? (a) : (b))

long long w[1000006];

void work() {
    long long n, res = 0;
    scanf("%lld", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lld", &w[i]);
    }
    for (long long i = 2; i < n; ++i) {
        res = max(res, w[i] - w[i - 1] + w[i] - w[0]);
    }
    for (long long i = 0; i < n - 2; ++i) {
        res = max(res, w[i + 1] - w[i] + w[n - 1] - w[i]);
    }
    printf("%lld\n", res);
}

int main() {
    long long t; scanf("%lld", &t); while(t--) work();
    return 0;
}
```

J 摩卡与助教团建

难度	考点
6	搜索、剪枝

题目分析

本题的数据规模下搜索 + 简单的剪枝（即及时停止对不可能正确的分支的继续搜索）就能通过。

我们可以用一个数组 `b` 来记录每个位置的翻转情况，0 代表不翻转，1 代表翻转，然后依次枚举每个格子是否翻转的可能性，再通过合适的剪枝，减少搜索次数。在下面的示例代码中，我们使用了如下的剪枝策略：

- **可行性剪枝：** 如果枚举完了某一行 `b` 的可能，并且上一行中有灯在这种可能下是关着的，则直接停止继续搜索这个分支

- **最优性剪枝：** 如果某次翻转灯的数量已经比当前算得的最优解的数量还多，则直接停止继续搜索这个分支（因为这个分支一定不会带来最优解）

示例代码

```
# include <stdio.h>

# define N 5

int n = N;
int a[N + 2][N + 2];
int b[N + 2][N + 2];

int answer;
int number;

// 按行优先顺序，从左到右，从上到下枚举
void f(int x, int y) {
    // 如果枚举完了某一行
    if (y > n) {
        if (x > 1) {
            // 检查是不是上一行中所有灯都是亮的
            for (int i = 1; i <= n; i++) {
                if (1 ^ a[x - 1][i] ^ b[x - 2][i] ^ b[x - 1][i - 1] ^ b[x - 1][i]
^ b[x - 1][i + 1] ^ b[x][i]) {
                    return;
                }
            }
        }
        x = x + 1;
        y = 1;
    }

    // 相当于枚举完了整个数组
    if (x > n) {
        for (int i = 1; i <= n; i++) {
            if (1 ^ a[x - 1][i] ^ b[x - 2][i] ^ b[x - 1][i - 1] ^ b[x - 1][i] ^
b[x - 1][i + 1] ^ b[x][i]) {
                return;
            }
        }
        // 如果答案合法则更新最优解，因为比最优解大的都及时地停止了，所以该分支一定比最优解好
        answer = number;
        return;
    }

    // 如果不翻转这个位置的灯
    f(x, y + 1);
    // 如果翻转这个位置的灯 + 最优性剪枝
    if (number + 1 < answer) {
        b[x][y] = 1;
        number += 1;
        f(x, y + 1);
        number -= 1;
        b[x][y] = 0;
```

```
    }
}

int main() {
    int q;
    scanf("%d", &q);
    while (q--) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                scanf("%d", &a[i][j]);
            }
        }
        answer = n * n + 1;
        number = 0;
        f(1, 1);
        if (answer <= n * n) {
            printf("%d\n", answer);
        } else {
            printf("-1\n");
        }
    }
    return 0;
}
```

K 超级摩卡与助教团建

难度	考点
7	贪心、状态压缩、搜索

题目分析

这题的数据规模相较于上一题实在是太大了，即使我们做了足够的剪枝优化，也还是会超时。

但实际上，不难发现，灯的处理顺序与最后答案无关，所以我们不妨假设先按下第一行的若干盏灯，之后就不管第一行了。不难发现，按完一些灯后，第一行有一些灯还是暗的，显然我们要处理他们把他们变亮，就只能通过按下第二行对应位置的灯，且不能按下第二行其它的灯（第一行所有灯都亮着是一种特殊的情况，第二行必须什么灯也不能按）。也就是说，假如我第一行按了哪些灯，我第二行必须按下哪些灯，才能使最后有可能所有灯都亮着；同理第二行能确定第三行，第三行能确定第四行……。所以根据第一行按下哪些灯，我们就能判断 2 到 19 行必须按下哪些灯，然后再看看按下这些灯后第十九行是不是都亮着，就能知道整个网格的灯都亮着（因为经过了上面的操作，前 18 行一定都亮着，所以只检查第 19 行就行）。

我们可以仿照之前的 `让我把记忆煎成饼` 来枚举第一行点亮了某些灯，也就是状态压缩的思想，把点亮了哪些灯用一个整数表示（比如点亮了第一盏和第三盏灯可以用整数 $5 = 101_{(2)}$ 来表示）。

示例代码

```
# include <stdio.h>

# define N 19
```

```
int n = N;
int f[1 << N];
int g[1 << N];
int a[N + 1];

int main() {
    f[1] = 1;
    // f[i] 表示按下 i 表示的若干盏灯（即 i 是这些灯状态压缩得到的整数），对这一行造成的变化
    // 显然 f[i] 等价于，如果上一行没亮的灯的状态表示是 i，那么这一行为了让上一行所有灯变亮发生
的变化
    for (int i = 0; i < n - 1; i++) {
        for (int j = (1 << i) - 1; j >= 0; j--) {
            f[(j << 2) ^ 3] = (f[(j << 1) ^ 1] << 1) ^ 2;
            f[(j << 2) ^ 2] = (f[(j << 1) ^ 1] << 1) ^ 1;
            f[(j << 2) ^ 1] = (f[(j << 1) ^ 0] << 1) ^ 3;
            f[(j << 2) ^ 0] = (f[(j << 1) ^ 0] << 1) ^ 0;
        }
    }

    // g[i] 表示 i 的二进制表示中有多少个 1
    for (int i = 0; i < n; i++) {
        for (int j = (1 << i) - 1; j >= 0; j--) {
            g[(j << 1) ^ 0] = g[j] + 0;
            g[(j << 1) ^ 1] = g[j] + 1;
        }
    }

    int q;
    scanf("%d", &q);
    while (q--) {
        // 读入 19 x 19 的矩阵
        for (int i = 1; i <= n; i++) {
            int s = 0;
            for (int j = 1; j <= n; j++) {
                int x;
                scanf("%d", &x);
                s = (s << 1) ^ (x ^ 1);
            }
            // 每一行用一个整数表示（状压思想）
            a[i] = s;
        }
        int answer = n * n + 1;

        // 开始枚举第一行的所有可能
        for (int i = (1 << n) - 1; i >= 0; i--) {
            // x 每一行按下灯后产生的变化，y 记录每一行翻转了多少盏灯，z 记录每一行要按下哪些
灯
            int x = f[i];
            int y = g[i];
            int z = i;
            for (int j = 1; j < n; j++) {
                int t = x;
                x  = f[t ^ a[j]] ^ z;
                y += g[t ^ a[j]];
                z  =   t ^ a[j];
```

```
            }
            if (x == a[n] && y < answer) {
                answer = y;
            }
        }
        if (answer <= n * n) {
            printf("%d\n", answer);
        } else {
            printf("-1\n");
        }
    }
    return 0;
}
```