2025暑期集训第7场

T1.填数

- 从$(1,2)$到(n,n)逐步填写新的数字，新填写的数字与其左方和上方的数的和不能为质数。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 15;
int a[N][N];
bool vis[N * N];
int n;
bool isp[255];

bool prime(int n)
{
    if(n == 1) return false;
    for(int i = 2; i * i <= n; ++i) if (n % i == 0) return false;
    return true;
}

bool dfs(int x, int y)
{
    if(x == n + 1) return true;
    for(int i = 1; i <= n * n; ++i)
    {
        if(vis[i]) continue;
        if(x != 1 && !isp[i + a[x - 1][y]]) continue;
        if(y != 1 && !isp[i + a[x][y - 1]]) continue;
        a[x][y] = i, vis[i] = 1;
        if(y < n) if(dfs(x, y + 1)) return true;
```

```
        if(y == n) if(dfs(x + 1, 1)) return true;
        a[x][y] = 0, vis[i] = 0;
    }
    return false;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= 2 * n * n - 1; ++i)
        isp[i] = prime(i);
    a[1][1] = 1, vis[1] = 1;
    bool res = dfs(1, 2);
    if(!res) puts("NO");
    else
    {
        for(int i = 1; i <= n; ++i)
        {
            for(int j = 1; j <= n; ++j)
                printf("%d ", a[i][j]);
            puts("");
        }
    }
    return 0;
}
```

- 优化：要求相邻之和为质数，那么直接枚举质数，判断能不能拆出新填的数字。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 15;
int a[N][N];
bool vis[N * N];
int n;
bool isp[255];
int p[255], m;

bool prime(int n)
{
    if(n == 1) return false;
    for(int i = 2; i * i <= n; ++i) if (n % i == 0) return false;
    return true;
}

bool dfs(int x, int y)
{
    if(x == n + 1) return true;
    int t = max(a[x - 1][y], a[x][y - 1]);
    // 枚举两数之和(质数)
    for(int j = 1; j <= m; ++j)
    {
        if(p[j] <= t) continue;
        int i = p[j] - t;
        if(i > n * n) break;
        if(vis[i]) continue;
        if(x != 1 && !isp[i + a[x - 1][y]]) continue;
        if(y != 1 && !isp[i + a[x][y - 1]]) continue;
        a[x][y] = i, vis[i] = 1;
        if(y < n) if(dfs(x, y + 1)) return true;
        if(y == n) if(dfs(x + 1, 1)) return true;
        a[x][y] = 0, vis[i] = 0;
    }
```

```
    return false;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i <= 2 * n * n - 1; ++i)
        if(isp[i] = prime(i)) p[++m] = i;
    a[1][1] = 1, vis[1] = 1;
    bool res = dfs(1, 2);
    if(!res) puts("NO");
    else
    {
        for(int i = 1; i <= n; ++i)
        {
            for(int j = 1; j <= n; ++j)
                printf("%d ", a[i][j]);
            puts("");
        }
    }
    return 0;
}
```

T2.倒水

- BFS
- 队列中记录三个水壶的水量、倒水步数即可。
- 可以定义标记数组`vis[110][110]`来标记A, B壶中水的状态，因为三个壶的水总量是固定的。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int C = 3;
const int N = 110;
struct Node
{
    int v[C], step;
};
int c[C], d;
bool vis[N][N];

int bfs()
{
    queue<Node> q; q.push({{c[0], 0, 0}, 0});
    vis[c[0]][c[1]] = true;
    while(!q.empty())
    {
        Node x = q.front(); q.pop();
        if(x.v[0] == d || x.v[1] == d || x.v[2] == d) return x.step;
        for(int i = 0; i < C; ++i)
            for(int j = 0; j < C; ++j)
            {
                if(i == j) continue;
                // i杯没有水 j杯水满
                if(x.v[i] == 0 || x.v[j] == c[j]) continue;
                Node y = {{x.v[0], x.v[1], x.v[2]}, x.step + 1};
                int t = min(x.v[i], c[j] - x.v[j]);
```

```
                y.v[i] -= t, y.v[j] += t;
                if(vis[y.v[i]][y.v[j]]) continue; // 水的状态出现过
                q.push(y), vis[y.v[i]][y.v[j]] = 1;
            }
    }
    return -1;
}

int main()
{
    for(int i = 0; i < C; ++i) scanf("%d", &c[i]);
    scanf("%d", &d);
    int ans = bfs();
    if(ans != -1) printf("%d\n", ans);
    else puts("false");
    return 0;
}
```

T3.Power Calculus

- 从x^1开始，每次可以通过乘以/除以历史上生成的指数，可以生成新的指数，直到生成x^n为止。
- 因为要找到最短的操作序列，所以理论上需要遍历所有的方案，那么生成树的深度可能非常大。
- 故而可以利用迭代加深，从0开始枚举允许的操作序列长度，判断能否得出答案。
- 同时$n \leq 1000$，可以用数组进行记忆化。

```cpp
#include <bits/stdc++.h>
using namespace std;

int ans[1010];
int n, deep;
int a[10010];

bool dfs(int x, int step)
{
    // (a^x)不断自己乘自己 < a^n
    if((x << (deep - step)) < n) return false;
    if(step > deep) return false;
    if(x == n) return true;
    a[step] = x;
    for(int i = 0; i <= step; ++i)
    {
        if(dfs(x + a[i], step + 1)) return true;
        if(dfs(abs(x - a[i]), step + 1)) return true;
    }
    return false;
}

int main()
{
    while(scanf("%d", &n) != EOF && n)
    {
        if(ans[n]) { printf("%d\n", ans[n]); continue; }
```

```
        for(deep = 0; ;++deep)
            if(dfs(1, 0)) {printf("%d\n", ans[n] = deep); break;}
    }
    return 0;
}
```

T4.天气预报

- 搜索枚举每一天降雨的云的位置：
  - 判断下雨的位置会不会覆盖赶集或过节的日子。
  - 是否存在有地方连续7天不下雨。
- 优化1：判断连续7天不下雨，不需要判断每一个位置，只需要判断四个角$(1, 4, 13, 16)$是否连续7天不下雨。
- 优化2：用记忆化记录第day天，云在(x, y)，四个角连续几天不下雨的是否会失败，在遇到同样状态时直接判断失败。

```cpp
// 只需要记录四个角(1, 4, 13, 16)的最近降雨情况即可
// 四个角是否合法等价于整个图是否合法
#include <bits/stdc++.h>
using namespace std;
const int N = 370;
int d[9][2] = { {0, 0}, {-2, 0}, {-1, 0}, {1, 0}, {2, 0}, {0, -2}, {0, -1}, {0, 1}, {0, 2} };
int tb[N][5][5];
bool vis[5][5][N][8][8][8][8];  // 记忆化
int n;

// 四个角最近下雨是什么时候
struct Node
{
    int a, b, c, d;
};

// 第day天  云在(x, y)
bool dfs(int x, int y, int day, Node s)
{
    if(vis[x][y][day][s.a][s.b][s.c][s.d]) return false;    // 状态只要访问过 一定是失败的
    if(day - s.a >= 7 || day - s.b >= 7 || day - s.c >= 7 || day - s.d >= 7)
    {
        vis[x][y][day][s.a][s.b][s.c][s.d] = true;
        return false;
```

```
    }
    if(day == n) return true;
    for(int i = 0; i <= 8; ++i)
    {
        int nx = x + d[i][0], ny = y + d[i][1];
        if(nx < 1 || nx > 3 || ny < 1 || ny > 3) continue;
        if(tb[day + 1][nx][ny] || tb[day + 1][nx + 1][ny] || tb[day + 1][nx][ny + 1] || tb[day + 1][nx
+ 1][ny + 1]) continue;
        Node ts = s;
        if(nx == 1 && ny == 1) ts.a = day + 1;
        if(nx == 1 && ny == 3) ts.b = day + 1;
        if(nx == 3 && ny == 1) ts.c = day + 1;
        if(nx == 3 && ny == 3) ts.d = day + 1;
        if(dfs(nx, ny, day + 1, ts)) return true;
    }
    vis[x][y][day][s.a][s.b][s.c][s.d] = true;
    return false;
}

int main()
{
    while(scanf("%d", &n) != EOF && n)
    {
        memset(tb, 0, sizeof(tb));
        memset(vis, 0, sizeof(vis));
        for(int i = 1; i <= n; ++i)
            for(int j = 1; j <= 4; ++j)
                for(int k = 1; k <= 4; ++k)
                    scanf("%d", &tb[i][j][k]);
        if(tb[1][2][2] || tb[1][2][3] || tb[1][3][2] || tb[1][3][3]) { puts("0"); continue; }
        if(dfs(2, 2, 1, {0, 0, 0, 0})) puts("1");
        else puts("0");
    }
    return 0;
}
```