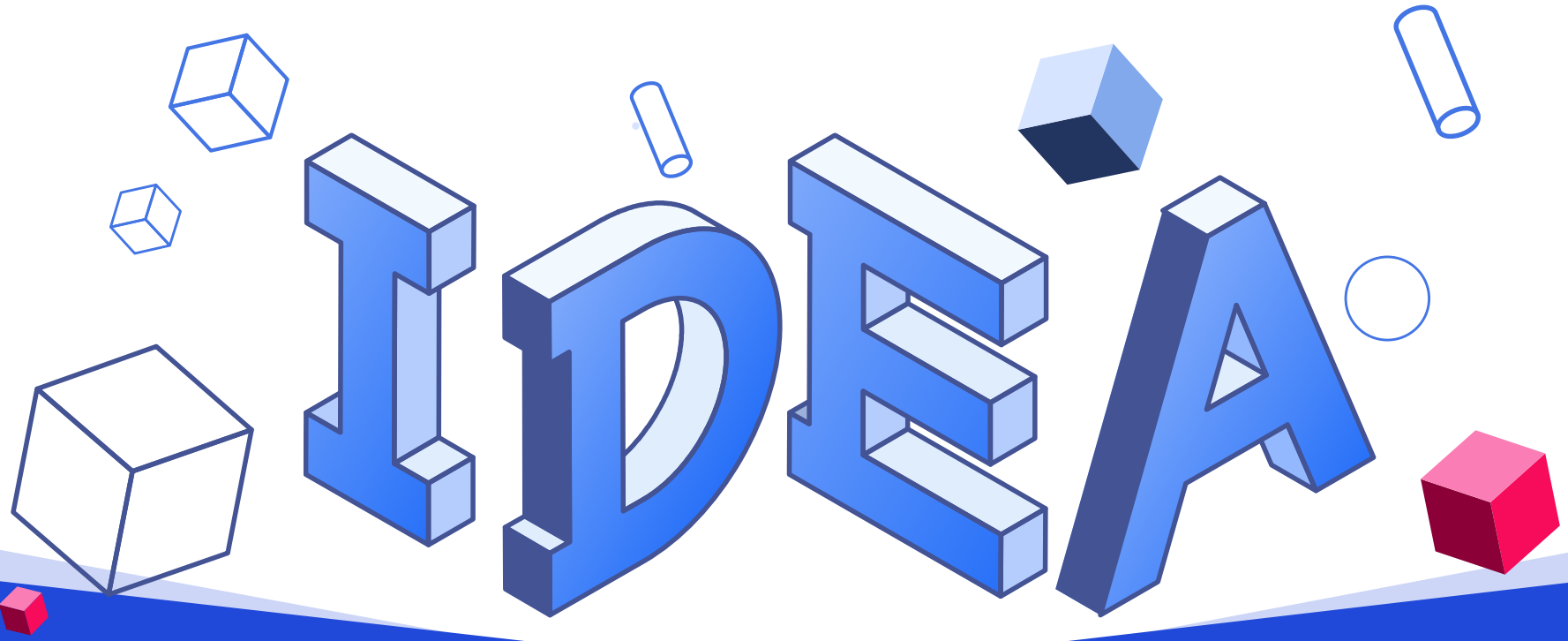


信息学

初识DFS题解

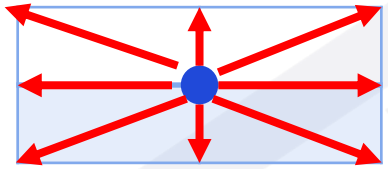
DFS解题的关键

遇到一个DFS的题目，你可能需要思考：

- 搜索可能的最大状态数
- 搜索时，上一层需要给下一层传递的信息(递归参数的传递)
- 搜索的边界条件

迷宫问题

题意：入口和出口分别在左上角和右上角,可以走八个方向，问不重复走每个点的路径数



设置增量数组：

```
int dx[8] = {  0, 1, 1, 1, 0, -1, -1, -1 };
int dy[8] = {  1, 1, 0, -1, -1, -1, 0, 1 };
```

Q：每个节点可能的搜索状态?

八个方向，8种

Q：每个节点状态需要转移什么信息?

坐标信息，(x,y)

Q：边界条件是什么?

抵达出口，x == 1 && y == n

注意：本题的迷宫中有障碍点，
也有要求不能重复走过之前的点

a[ ][ ]:存迷宫信息
f[ ][ ]:存每个点的是否走过

参考代码


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

int cnt, n;
int dx[9] = {  0, 1, 1, 1, 0, -1, -1, -1 };  //增量数组
int dy[9] = {  1, 1, 0, -1, -1, -1, 0, 1 };
bool a[100][100];
bool f[100][100];  //标记数组
void dfs(int x, int y)
{
    if (x == 1 && y == n) {  //边界条件
        cnt++;
        return;
    }
    for (int i = 0; i < 8; i++) {
        int xx = x + dx[i];
        int yy = y + dy[i];
        if (xx <= n && xx >= 1 && yy >= 1 && yy <= n && a[xx][yy] == 0 && f[xx][yy] == 0) {
            f[xx][yy] = 1;  //标记点已经走过
            dfs(xx, yy);
            f[xx][xy] = 0;  //回溯
            xx -= dx[i];
            yy -= dy[i];
        }
    }
}
```

```
int main()
{
    cin >> n;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            cin >> a[i][j];
        }
    }
    f[1][1] = 1;  //注意起点一定要先标记
    dfs(1, 1);
    cout << cnt << endl;
    return 0;
}
```

奇怪的电梯

题意：电梯只有四个按钮：开，关，上，下
最开始在1楼，给出每个楼层的能够移动的楼层数，问能否从A到B

Q：每个节点可能的搜索状态?

上下2个方向，2种

Q：每个节点状态需要转移什么信息?

当前所在的楼层k

Q：边界条件是什么?

抵达终点楼层，k==n

参考代码


```
#include <bits/stdc++.h>
using namespace std;
int n, s, t, a[209], cnt = 0, f [209] , ans = 10000000;
int x, y;
void dfs(int x){
    if (x == t) {  //边界条件
        ans = min(cnt, ans);
        return;
    }
    if (f[1) {
        returnx] == ;
    }
   if (x > n || x < 1) {
        return;
    }
    f [x] = 1;
    cnt++;
    dfs(x + a[x]);  //往上
    dfs(x - a[x]);  //往下
    cnt--;
    f [x] = 0;
}
```

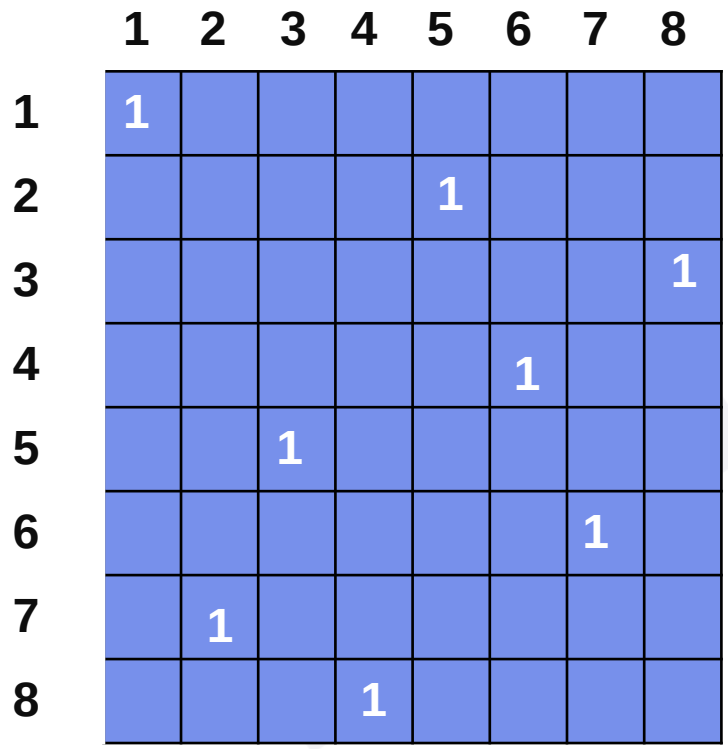
```
int main()
{
    cin >> n >> s >> t;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    dfs(s);
    if (ans == 10000000) {
        cout << "-1";
    } else {
        cout << ans;
    }
    return 0;
}
```

看起来更优雅的写法


```
#include <bits/stdc++.h>
using namespace std;
int n, a, b, ans = 0x7ffffff;
int to[205];
bool vis[205];
void dfs(int now, int sum)
{
    if (now == b)
        {ans = min(ans, sum);return;}
    if (sum > ans)
        return;
    vis[now] = 1;
    if (now + to[now] <= n && !vis[now + to[now]])  //向上
        dfs(now + to[now], sum + 1);
    if (now - to[now] >= 1 && !vis[now - to[now]])  //向下
        dfs(now - to[now], sum + 1);
    vis[now] = 0;
}
int main()
{
    cin >> n >> a >> b;
    for (int i = 1; i <= n; i++)
        cin >> to[i];
    vis[a] = 1;
    dfs(a, 0);
    if (ans != 0x7ffffff)
        cout << ans;
    else
        cout << -1;
    return 0;
}
```

N皇后问题

来源于8皇后问题

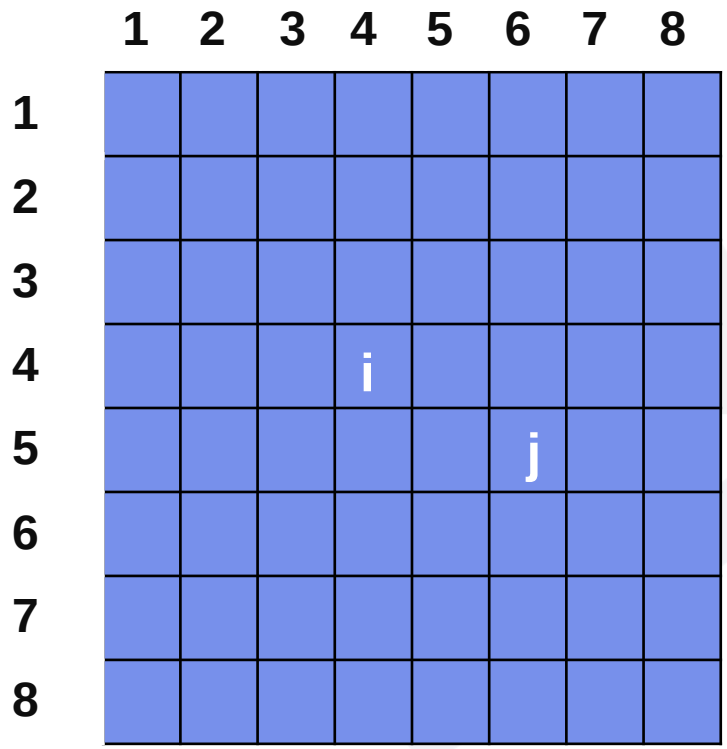


如何找出所有的八皇后的摆法?

最容易想到的方法：**枚举**

穷举8个皇后的所有可能位置组合，逐一判断是否可以互相被吃掉，得到最终解。

分析



如何判断各个皇后之间位置是否冲突?

1.是否在同一列? `a[i]!=a[j]`

2.是否在同一斜线?

```
//标记所处的列和两个斜线被占用
lie[j]=1;
zx[i+j]=1;
yx[i-j+7]=1;
```

也有数学规律判断：abs(i-j)!=abs(a[i]-a[j])

分析

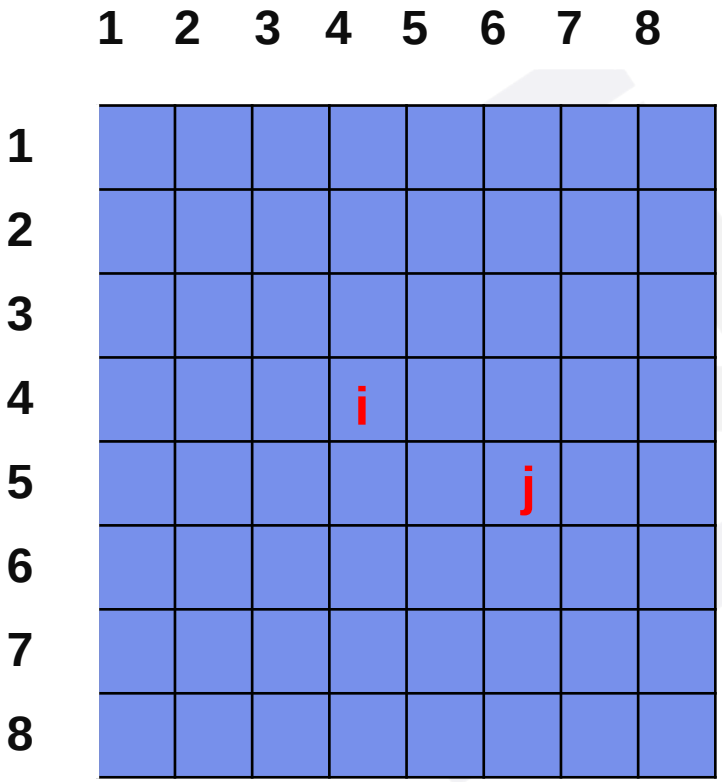

```
for (a[1] = 1; a[1] <= 8; ++a[1])  //第一行的八个位置
        for (a[2] = 1; a[2] <= 8; ++a[2]) //第二行的八个位置
            for (a[3] = 1; a[3] <= 8; ++a[3]) //第三行的八个位置
                for (a[4] = 1; a[4] <= 8; ++a[4]) //第四行的八个位置
                    for (a[5] = 1; a[5] <= 8; ++a[5]) //第五行的八个位置
                        for (a[6] = 1; a[6] <= 8; ++a[6]) //第六行的八个位置
                            for (a[7] = 1; a[7] <= 8; ++a[7]) //第七行的八个位置
                                for (a[8] = 1; a[8] <= 8; ++a[8]) //第八行的八个位置
                                {
                                    if (!Chongtu())//如果冲突，则继续枚举
                                        continue;
                                    else
                                    {
                                        统计方案数或者打印当前的结果;
                                    }
                                }
```

枚举的方法会产生许多无用的状态，导致程序超时

如果能够排除那些没有前途的状态，会节约时间(递归回溯)

分析

f(i)表示放第i个(行)皇后



算法框架：

```
void f(int n)
{
        枚举当前皇后所在行的每一列
          如果不冲突
          {
                1.放入皇后
                2.标记这一列
                3.标记两条对角线
                4.如果放完了所有皇后，输出结果
                5.如果没有放完，继续放下一个皇后
                6.如果当前这一步已经不可行，一步步回退并恢复到
                  之前的状态(回溯)
          }
}
```

代码一

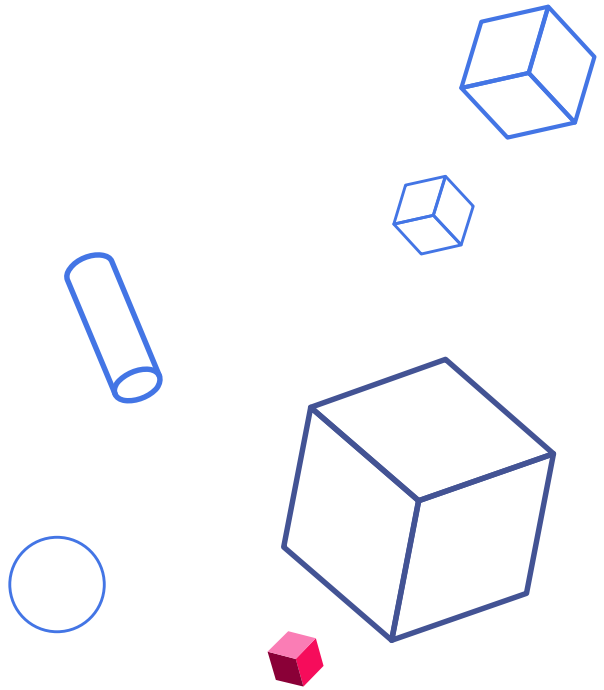

```
int dfs(int i){
        int j;
        if(i>8) {print();return;} //边界条件
        for(j=1;j<=8;j++)
          if(!lie[j]&&!zx[i+j]&&!yx[i-j+7]) { //判断是否在同一列或同一行
                a[i]=j;  //在第i行j列放置皇后
                lie[j]=1;  //标记所处的列和两个斜线被占用
                zx[i+j]=1;
                yx[i-j+7]=1;
                dfs(i+1);  //继续放下一个(行)皇后
                lie[j]=0;       //回溯时，把占用的行列恢复为未占用状态
                zx[i+j]=0;
                yx[i-j+7]=0;
        }
}
```

代码二


```
bool Chongtu(int t)  //判断与之前放置的皇后是否冲突
{
        for (int j = 1; j <t; j++)
                if (a[j] == a[t] || (abs(j - t)) ==(abs(a[j] - a[t])))
                        return 1;
        return 0;
}
void dfs(int step)
{
        if (step > 8){   //边界条件
                cnt++;   //可以统计答案总数
                print(); //也可以输出本次八皇后的放置情况
                return;
        }
        else{
           for (int i = 1; i <= 8; i++){
                a[step] = i;  //记录皇后放置的列
                if (!Chongtu(step))  //不冲突能放
                  dfs(step + 1); //放置下一个
           }
        }
}
```

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

西南大学附属中学
High School Affiliated to Southwest University



题目分界线

四色问题

题意：用4种颜色给n个球染色，相邻的球颜色不能重复

题目给的二维数组表示两个点之间的相邻关系

8

0 0 0 1 0 0 1 0 → 1,4相邻

0 0 0 0 0 1 0 1

0 0 0 0 0 0 1 0

1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

1 0 1 0 0 0 0 0

0 1 0 0 0 0 0 0

Q：每个节点可能的搜索状态？

4种颜色

Q：每个节点状态需要转移什么信息？

当前涂的第几个球k

Q：边界条件是什么？

所有球填完，k>n

参考代码


```
#include <bits/stdc++.h>
using namespace std;
bool map[9][9];
int c[9];
int ans = 0, N;

void dfs(int n)
{
    int i, j;
    if (n > N) {
        ans++;
        return;
    }
    for (j = 1; j <= 4; j++) //枚举4种颜色
    {
        for (i = 1; i < n; i++)
            if (map[i][n] && c[i] == j) //判断与前面各点是否相邻但颜色相同
                break;
        if (i == n) {
            c[n] = j;
            dfs(n + 1);
            c[n] = 0;
        }
    }
}
```

```
int main()
{
    cin >> N;
    int i, j;
    for (i = 1; i <= N; i++)
        for (j = 1; j <= N; j++)
            cin >> map[i][j];
    dfs(1);
    cout << ans;
    return 0;
}
```

由于是个环，各个位置都是一样的，所以可以先固定一个人的位置，然后进行排列

素数环与这题相似

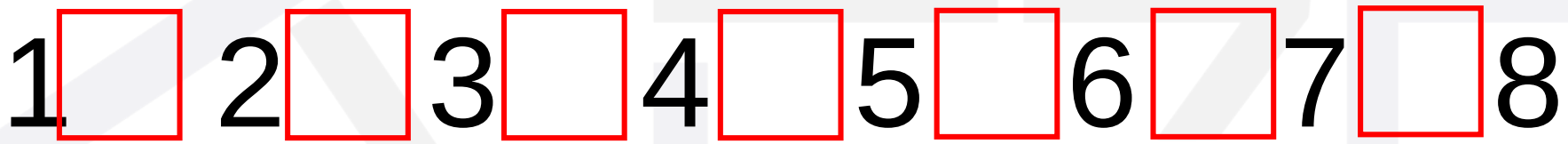
参考代码


```
int f[100], a[1000], b[100], N, cnt, k;
void dfs(int n, int num) //当填的位置n和前一个入座的人的编号;
{
    if (n == N) {
        if (abs(a[1] - a[num]) <= k) {  //因为是个环，首尾判断
            cnt++;
            return;
        }
    }
    else {
        for (int i = 1; i <= N; i++) {
            if (!f[i] && abs(a[num] - a[i]) <= k) {  //满足条件就填进去
                f[i] = 1;
                dfs(n + 1, i);
                f[i] = 0;
            }
        }
    }
}
```

```
int main()
{
    int i;
    scanf("%d%d", &N, &k);
    for (i = 1; i <= N; i++) {
        scanf("%d", &a[i]);
    }
    f[1] = 1; //固定让第一个人在第一个位置
    dfs(1, 1);
    printf("%d\n", cnt);
    return 0;
}
```

Zero Sum

给出1~n的数字，可以在里面填“+”，“-”，“ ”，问有多少种可以使得式子为0的方案并输出



Q：每个节点可能的搜索状态？

3种

Q：每个节点状态需要转移什么信息？

当前选择的第几个位置k

Q：边界条件是什么？

全部空隙都填完了

这道题如果大家排列问题弄明白了的话，其实搜索比较简单

题目的难点在于，如何计算出不同填法所得到的值

“披着dfs外壳的模拟题”

参考代码


```
#include <bits/stdc++.h>
using namespace std;
typedef long long int LL;
int N;char signs[10];
void dfs(int k){
    if (k == N - 1) {  //边界条件
        LL sum = 0, cur = 1;  //sum是式子的和，cur是当前位置的值
        int s = 1; // 符号，+为1，-为-1
        for (int i = 0; i < k; i++) {
            if (signs[i] == '')  //如果是空格，就按照题意计算
                cur = cur * 10 + (i + 2);
            else if (signs[i] == '+') {
                sum += cur * s;
                s = 1;
                cur = i + 2;
            }
              else {
                sum += cur * s;
                s = -1;
                cur = i + 2;
            }
        }
        sum += cur * s;  //计算目前整个式子的值
        if (sum == 0) {  //为0输出
            cout << 1;
            for (int i = 0; i < k; i++)
                cout << signs[i] << (i + 2);
            cout << endl;
        }
        return;
    }
    signs[k] = ' ';  //三种选择
    dfs(k + 1);
    signs[k] = '+';
    dfs(k + 1);
    signs[k] = '-';
    dfs(k + 1);
}
```

```
int main()
{
    cin >> N;
    dfs(0);
    return 0;
}
```

另一种风格的代码


```
#include <bits/stdc++.h>
using namespace std;
int n, a[11];
char s[3] = { ' ', '+', '-' };
bool pd()
{
    int ans = 0, t;
    for (int i = 1; i <= n; i++) {
        if (a[i] == 0)
            continue;
        t = i;
        for (int j = i + 1; j <= n; j++) {
            if (a[j] != 0)
                break;
            t = t * 10 + j;
        }
        if (a[i] == 1)
            ans += t;
        else
            ans -= t;
    }
    if (ans == 0)
        return 1;
    return 0;
}
```

```
void dfs(int k)
{
    if (k > n) {
        if (pd()) {
            printf("1");
            for (int i = 2; i <= n; i++)
                printf("%c%d", s[a[i]], i);
            printf("\n");
        }
    } else {
        for (int i = 0; i <= 2; i++) {
            a[k] = i;
            dfs(k + 1);
            a[k] = 0;
        }
    }
}
int main()
{
    scanf("%d", &n);
    a[1] = 1;
    dfs(2);
}
```

工作分配问题

这道题目比较简单，求一个最小的代价组合

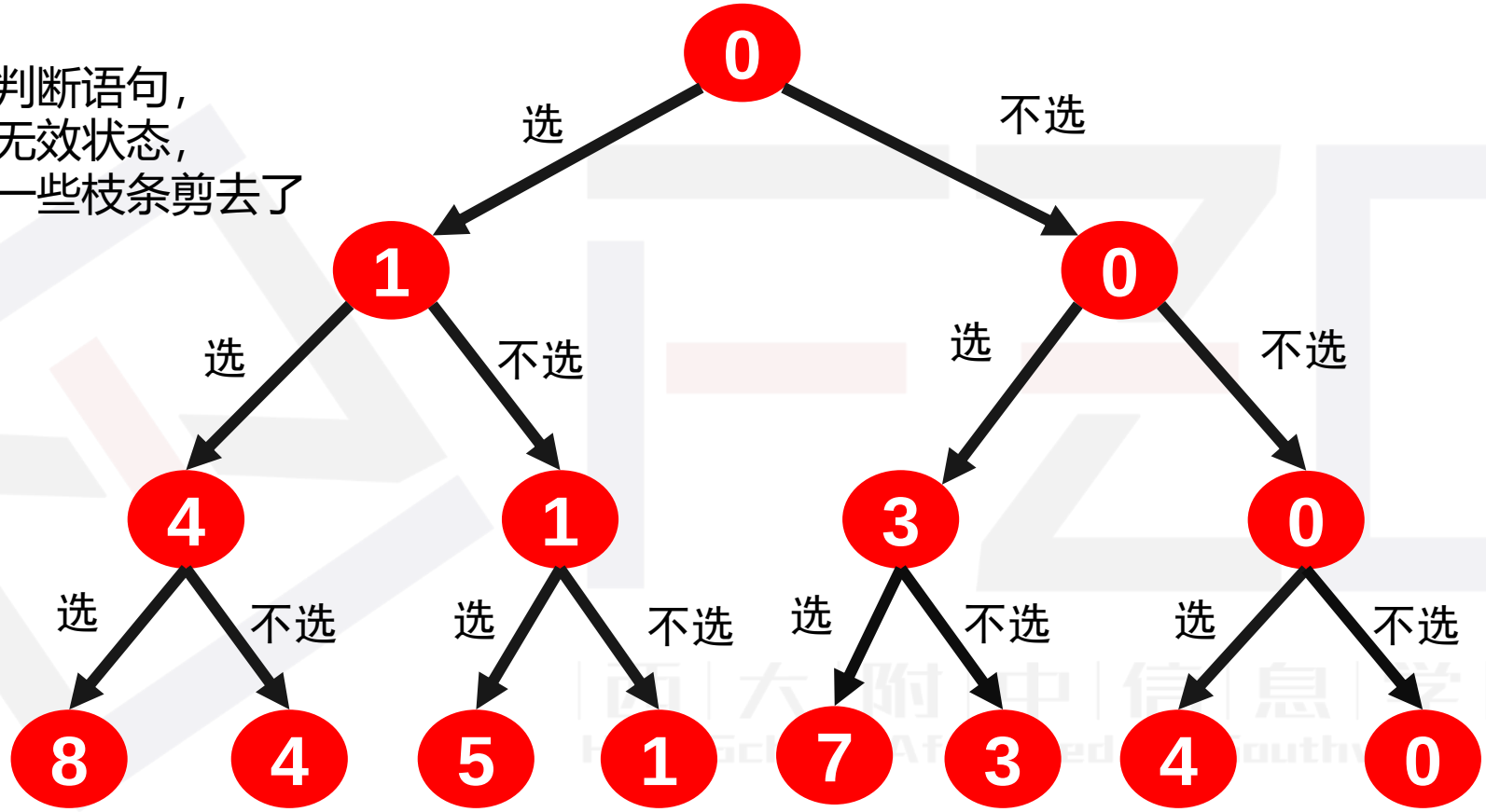
```
#include <bits/stdc++.h>
using namespace std;
int n, v[21][21], Min = 2147483647, a[4] = { 0 };
void dfs(int k, int sum)
{
    if (k == n + 1) {
        Min = min(Min, sum);
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (a[i] == 0) {
            a[i]++;
            dfs(k + 1, sum + v[k][i]);
            a[i]--;
        }
    }
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &v[i][j]);
        }
    }
    dfs(1, 0);
    printf("%d", Min);
    return 0;
}
```

交上去，只有81分，怎么解决?

剪枝

分析

剪枝：
通过一些判断语句，
去除一些无效状态，
类似于把一些枝条剪去了



工作分配问题

这道题目比较简单，求一个最小的代价组合

```
#include <bits/stdc++.h>
using namespace std;
int n, v[21][21], Min = 2147483647, a[4] = { 0 };
void dfs(int k, int sum)
{
    if (k == n + 1) {
        Min = min(Min, sum);
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (a[i] == 0) {
            a[i]++;
            dfs(k + 1, sum + v[k][i]);
            a[i]--;
        }
    }
}
int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            scanf("%d", &v[i][j]);
        }
    }
    dfs(1, 0);
    printf("%d", Min);
    return 0;
}
```

```
if(sum>Min) return ;
```

如果某次搜索的sum已经超过了当前的Min，就没必要再搜下去了

小结

DFS的常规题目都可以按照三个方向去分析
想要学好DFS没有捷径，不光需要思考，也需要动手敲码积累经验

Thanks

For Your Watching