状态压缩动态规划

山西大学附属中学 杨培君

提 纲

一、状态压缩动态规划的概念

二、位运算符及常见的位运算操作

三、状压dfs——P1433 吃奶酪

四、状压bfs——P2622 关灯问题 II

五、基于连通性（棋盘类）状态压缩的动态规划模型

——P1896 [SCOI2005] 互不侵犯

——P1879 [USACO06NOV] Corn Fields G 玉米田

——P2704 [NOI2001] 炮兵阵地

六、集合类状态压缩的动态规划模型

——P5911 [POI 2004] PRZ

——P10447 最短 Hamilton 路径

——P3959 [NOIP 2017 提高组] 宝藏

七、总结、习题

一、概念

状态压缩是信息学竞赛中的一个常用技巧。

主要思想是利用01二进制来表示**状态**信息，当状态维数很高，但是实际可行总量不大时，可以将状态**压缩**到一个整数或者长整数中，同时利用位运算来高效处理。

常用于搜索和dp中。

状态压缩动态规划算法是一个小技巧，一般应用在集合问题中。当DP状态是集合时，把集合的组合或者排列用一个二进制数表示。

状态压缩动态规划是指将状态进行压缩存储的动态规划，例如在设计动态规划的状态时，可以用n位二进制数表示长度为n的int数组，并利用位运算实现状态的快速转移。

二、位运算符

&	0	1
0	0	0
1	0	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

~	0	1
	1	0

位运算符	位运算	目数	例	答案
&	与	双目	$(3)_{10}\&(2)_{10}$	$(2)_{10}$
|	或	双目	$(3)_{10}\|(2)_{10}$	$(3)_{10}$
^	异或	双目	$(9)_{10}\wedge(5)_{10}$	$(12)_{10}$
~	取反	单目	$\sim(1)_{10}$（int 类型）	$(-2)_{10}$

二、位运算符

位运算符	位运算	目数	例	答案
<<	左移	双目	$3 \ll 4$	48
>>	右移	双目	$-3 \gg 1$	−2

左移：高位丢掉，低位补0，不溢出等效于乘2

右移：低位丢掉，高位补符号位

1<<1，1左移1位，等于2

1<<3，1左移3位，等于8

17>>1，17右移1位，等于8

二、常见的位运算操作

1. 判断整数x的奇偶性	1. x & 1
2. 判断整数x的二进制第i位（从右向左）是1还是0	2. ((x >> (i-1)) & 1) == 1
3. 将整数x的二进制第i位改为1	3. x = x | (1 << (i-1))
4. 取整数x的二进制数的最低位1	4. x &(-x) (lowbit)
5. 去掉整数x的二进制最低位1	5. x = x & (x - 1)

三、状压dfs——P1433 吃奶酪

题面： 房间里有 n 块奶酪。一只老鼠要把它们都吃掉，问至少要跑多少距离？老鼠一开始在 (0,0) 点处。

输入： 第1行有一个整数，表示奶酪的数量 n。第 2 到第 (n+1) 行，每行两个实数，第 (i+1) 行的实数分别表示第 i 块奶酪的横纵坐标 xi, yi。

输出： 输出一行一个实数，表示要跑的最少距离，保留 2 位小数。

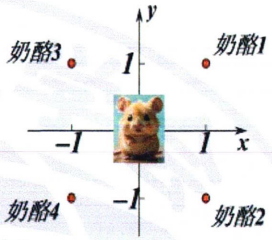
保证 1≤n≤15，｜xi｜,｜yi｜≤200，小数点后最多有 3 位数字

输入样例：

```
4
1 1
1 -1
-1 1
-1 -1
```

输出样例：

```
7.41
```



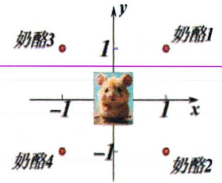
三、状压dfs——P1433 吃奶酪

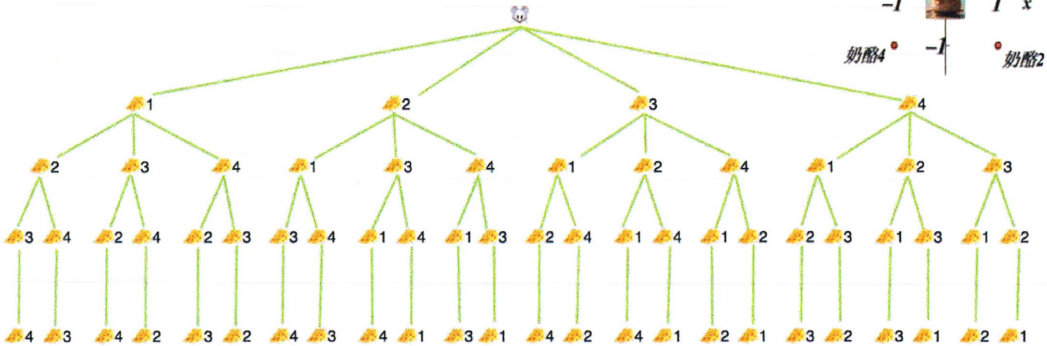
：老鼠

：奶酪

暴搜，有N!种方案

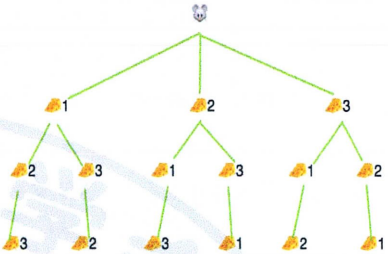
$15! \approx 1.3*10^{12}$





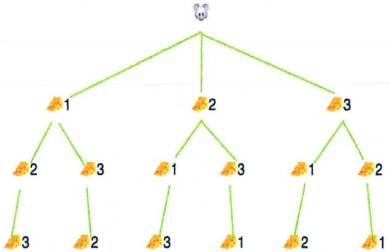
三、状压dfs——P1433 吃奶酪

```
#include <bits/stdc++.h>
using namespace std;
int n;
double x[16], y[16], ans=1e9;
bool vis[16];
//计算第 i 个奶酪和第 j 个奶酪之间的距离
double cal(int i, int j)
{
    return sqrt((x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]));
}
//现在在第几个奶酪位置，现在走过的距离为 curdis，现在吃了curnum 个奶酪
void dfs(int curid, double curdis, int curnum);
int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i){
        scanf("%lf %lf", &x[i], &y[i]);
    }
    //从 第 0 个奶酪 (0, 0) 出发，现在走了 0 的距离，已经吃了 0 块奶酪
    dfs(0, 0, 0);
    printf("%.2lf\n", ans);
    return 0;
}
```



三、状压dfs——P1433 吃奶酪

```
//现在在第几个奶酪位置，现在走过的距离为 curdis，现在吃了curnum 个奶酪
void dfs(int curid, double curdis, int curnum)
{
    //如果这一条分支吃完了所有的 n 块奶酪
    if(curnum==n){                                  //递归函数
        //打擂台求最小值，并结束分支
        ans=min(ans, curdis);
        return;
    }
    //如果没有吃完所有的奶酪，则遍历所有可以吃的奶酪，进行 dfs
    for(int i=1; i<=n; ++i){
        //如果第 i 块奶酪没有被吃过
        if(vis[i]==false){
            //则吃第 i 块奶酪，尝试向下去递归
            vis[i]=true;
            dfs(i, curdis+cal(curid, i), curnum+1);
            //回溯
            vis[i]=false;
        }
    }
}
```



三、状压dfs——P1433 吃奶酪

超时，考虑优化

思考：

当老鼠走蓝色路线（奶酪2——奶酪1——奶酪3）

到达奶酪3处，发现走过的路径长度大于

通过绿色路线（奶酪1——奶酪2——奶酪3）

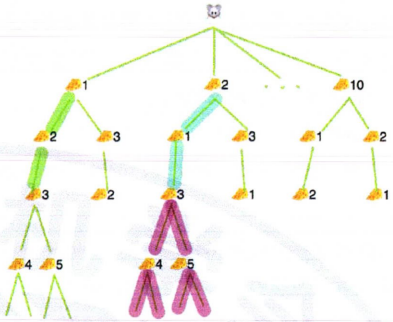
到达奶酪3处，

两种路线均会吃掉奶酪1、2、3

请问粉色的路线还有没有走的必要？

如果前面经过相同的奶酪（无关顺序）

且比现在路线更优，则剪枝。



三、状压dfs——P1433 吃奶酪

超时，考虑优化

如果前面经过相同的奶酪（无关顺序），且比现在路线更优，则剪枝

状态：当前吃掉哪些奶酪

如何表示状态？

表示的状态便于当做数组下标，来标记前面是否走过

使用01二进制来表示

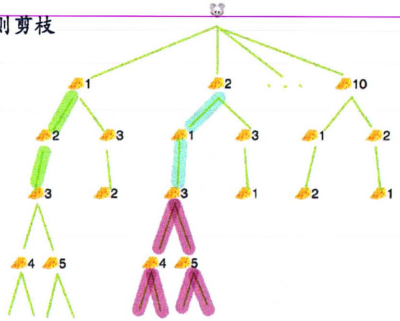
对于三个奶酪1、2、3，用3位二进制数表示奶酪的状态

从右至左每一位依次表示奶酪1、2、3是否被吃掉

（1表示被吃掉，0表示未被吃掉）

111(7)表示奶酪3、奶酪2、奶酪1被吃掉

110(6)表示奶酪3、奶酪2被吃掉、奶酪1未被吃掉



三、状压dfs——P1433 吃奶酪

如果前面经过相同的奶酪（无关顺序），且比现在路线更优，则剪枝

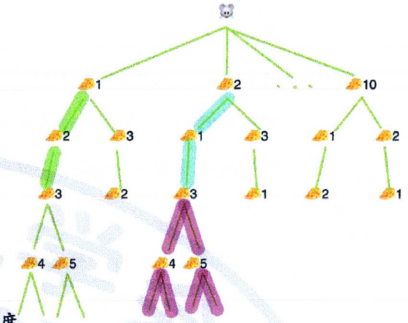
对于三个奶酪1、2、3，用3位二进制数表示奶酪的状态

从右至左每一位依次表示奶酪1、2、3是否被吃掉

（1表示被吃掉，0表示未被吃掉）

111(7)表示奶酪3、奶酪2、奶酪1被吃掉

110(6)表示奶酪3、奶酪2被吃掉、奶酪1未被吃掉



有了状态之后，如何记忆化？

斐波那契数列问题的记忆化，vis[i]=true表示计算过了

吃奶酪：站在相同的奶酪处，并且前面吃掉的奶酪相同，比较路线长度

需要二维，第一维：状态表示前面吃掉的奶酪，第二维：表示当前在哪个奶酪处，

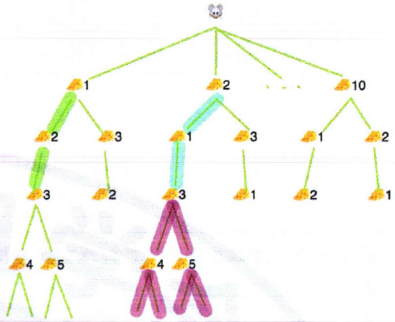
值表示走过的距离。

三、状压dfs——P1433 吃奶酪

```
#include <bits/stdc++.h>
using namespace std;
int n, cnt;
double x[16], y[16], dist[16][16], ans=1e9, disdp[1<<15][16];
bool vis[16];
double cal(int i, int j)
{
    return sqrt((x[i]-x[j])*(x[i]-x[j])+(y[i]-y[j])*(y[i]-y[j]));
}
//现在在第几个奶酪位置，现在走过的距离为 curdis，现在吃了curnum 个奶酪，现在的状态
void dfs(int curid, double curdis, int curnum, int curstate);
int main()
{
    for(int i=0; i<(1<<15); ++i){
        for(int j=0; j<16; ++j){
            disdp[i][j]=1e9;
        }
    }
    scanf("%d", &n);
    for(int i=1; i<=n; ++i){
        scanf("%lf %lf", &x[i], &y[i]);
    }
    for(int i=0; i<=n; ++i){
        for(int j=0; j<=n; ++j){
            dist[i][j]=cal(i, j);
        }
    }
    //从 第 0 个奶酪 (0, 0) 出发，现在走了 0 的距离，已经吃了 0 块奶酪，当前的状态
    dfs(0, 0, 0, 0);
    printf("%.2lf\n", ans);
    return 0;
}
```

三、状压dfs——P1433 吃奶酪

```
//现在在第几个奶酪位置，现在走过的距离为 curdis，现在吃了curnum 个奶酪，现在的状态
void dfs(int curid, double curdis, int curnum, int curstate)
{
    //如果这一条分支吃完了所有的 n 块奶酪
    if(curnum==n){
        //打擂台求最小值，并结束分支
        ans=min(ans, curdis);
        return;
    }
    //如果没有吃完所有的奶酪，则遍历所有可以吃的奶酪，进行 dfs
    for(int i=1; i<=n; ++i){
        //如果第 i 块奶酪没有被吃过
        if(vis[i]==false){
            //最优性剪枝，在做决策时就剪，比做了决策后再剪要快
            if(curdis+dist[curid][i]>ans){
                return;
            }
            //最优性剪枝，相当于记忆化搜索，如果前面有路径经过一系列相同的点到达过
            //点i，并且走的距离比 curdis+dist[curid][i] 更小，则不优，直接跳过
            if(disdp[curstate+(1<<(i-1))][i]<curdis+dist[curid][i]){
                continue;
            }
            //则吃第 i 块奶酪，尝试向下去递归
            vis[i]=true;
            //用状态压缩的思想记录从开始走了一些列点之后到达 i 的最优距离
            disdp[curstate+(1<<(i-1))][i]=curdis+dist[curid][i];
            //dfs 递归
            dfs(i,curdis+dist[curid][i],curnum+1,curstate+(1<<(i-1)));
            //回溯
            vis[i]=false;
        }
    }
}
```



四、状压bfs——P2622 关灯问题 II

题面： 现有n盏灯，以及m个按钮。每个按钮可以同时控制这n盏灯——按下了第i个按钮，对于所有的灯都有一个效果。按下按钮i对于第j盏灯，有三种可能效果：

如果a[i][j]为1，那么当这盏灯开了的时候，把它关上，否则不管；

如果为-1的话，如果这盏灯是关的，那么把它打开，否则也不管；

如果是0，无论这灯是否开，都不管。

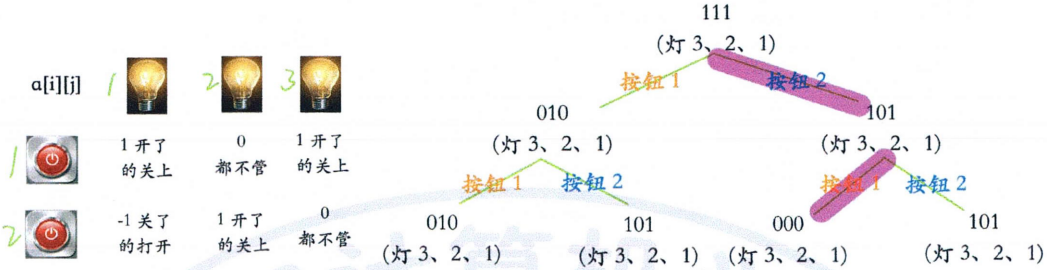
现在这些灯都是开的，给出所有开关对所有灯的控制效果，求问最少要按几下按钮才能全部关掉。n<=10,m<=100

输入： 前两行两个整数，分别是 n 和 m。接下来 m 行，每行 n 个整数，第 (i+2) 行的第 j 个整数为 a[i][j]，表示第 i 个开关对第 j 个灯的效果。

输出： 最少的按按钮的次数。如果没有任何办法使其全部关闭，输出 −1。

输入样例：	输出样例：
3 2 1 0 1 -1 1 0	2

四、状压bfs——P2622 关灯问题 II



最开始所有灯都是开的，目标所有灯都是关的，求最少按按钮次数，可以把所有灯的状态进行压缩，1表示灯是开的，0表示灯是关的。那么10盏灯的状态就可以用10位二进制数进行表示，可以将每一种状态看作一个点，将按一次按钮看作边，连接按该按钮前后灯的状态，形成一个图论问题。我们可以从起点（全亮）开始搜索（全亮状态入队列），每次取出队首，把所有开关按一遍，如果走到的新状态的最少次数dis[s]有更新，就把它入队列；如果走到的新状态是全灭，输出结果即可。如果队列空了，输出-1。

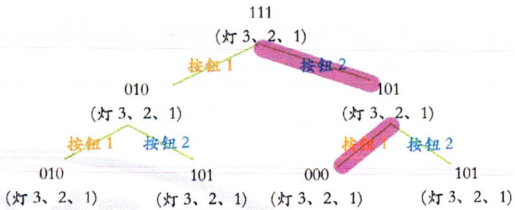
四、状压bfs——P2622 关灯问题 II

```
#include <bits/stdc++.h>
using namespace std;
int n, m, a[110][11], state, dis[1<<10], vis[1<<10], light[11], curdis;
queue<int> q;
void bfs();
int main()
{
    cin >> n >> m;
    for(int i=1; i<=m; ++i){     //m个按钮
        for(int j=1; j<=n; ++j){     //n盏灯
            cin >> a[i][j];
        }
    }
    bfs();
    cout << "-1" << endl;
    return 0;
}
```

a[i][j]	1	2	3
1	1 开了的关上	0 都不管	1 开了的关上
2	-1 关了的打开	1 开了的关上	0 都不管

四、状压bfs——P2622 关灯问题 II

```
void bfs()
{
    q.push((1<<n)-1);
    vis[(1<<n)-1]=true;
    while(!q.empty()){
        state=q.front();
        q.pop();
        curdis=dis[state];
        //尝试按每一个按钮，查看灯的状态
        for(int i=1; i<=m; ++i){
            int curstate=state;
            for(int j=1; j<=n; ++j){    //枚举每一盏灯
                light[j]=curstate&1;    //获取第 j 盏灯的状态
                curstate>>=1;
                if(a[i][j]==1){
                    light[j]=0;
                }
                else if(a[i][j]==-1){
                    light[j]=1;
                }
            }
            for(int j=n; j>=1; --j){
                curstate<<=1;
                curstate+=light[j];
            }
            if(!vis[curstate]){
                if(curstate==0){
                    cout << curdis+1 << endl;
                    exit(0);
                }
                vis[curstate]=true;
                dis[curstate]=curdis+1;
                q.push(curstate);
            }
        }
    }
}
```



使用位运算实现三种操作：

1. $a[i][j]=1$关灯：将第j位置为0　　if(a[i][j]==1) s&=(1<<n)-1-(1<<(j-1));

2. $a[i][j]=-1$开灯：将第j位置为1　　if(a[i][j]==-1) s|=1<<(j-1);

3. $a[i][j]=0$ 不处理

五、基于连通性（棋盘类）状压dp

基于连通性状态压缩的动态规划模型（棋盘类状压dp）

基于连通性状态压缩的动态规划是指在状态中记录若干个元素的连通情况的动态规划，

又称轮廓线 DP 或插头 DP，此类动态规划常见于棋盘模型中。

五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

题面： 在N×N（1≤N≤ 9）的棋盘里面放K（1≤K≤N*N）个国王，使他们互不攻击，共有多少种摆放方案？国王能攻击到它上下左右，以及左上、左下、右上、右下八个方向上附近的各一个格子，共8个格子。

输入： 只有一行，包含两个数N,K。

输出： 所得的方案数

输入样例：	输出样例：
3 2	16

五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

在N×N（1≤N≤ 9）的棋盘里面放K（1≤K≤N*N）个国王，使他们互不攻击，共有多少种摆放方案？国王能攻击到它上下左右，以及左上、左下、右上、右下八个方向上附近的各一个格子，共8个格子。

能不能摆国王，只与上一行有关系

状态表示dp[i][j][s]

含义：前i行已经放完了，当前放了j个国王，

第i行国王的摆放状态为s的方案数。

s是一个N位二进制数

		K				
K				K		K

五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

状态表示dp[i][j][s]
含义：前i行已经放完了，当前放了j个国王，第i行国王的摆放状态为s的方案数。
s是一个N位二进制数

集合：所有只摆在前i行，已经摆了j个国王，并且第i行国王的摆放状态为s的所有方案的集合

属性：方案数

		K				
K				K		K

互不侵犯

状态计算，能不能摆国王，只与上一行有关系，考虑第i行由第i-1行转移过来
dp[i][j][a]：前i行已经放完了，并且第i行的状态是a，第i-1行的状态是b，当前总共摆了j个国王的方案数。
dp[i-1][j-count(a)][b]：前i-1行已经放完了，并且第i-1行的状态是b，当前总共摆了j-count(a)个国王的方案数。count(a)表示第i行状态a中1（国王）的数量。
dp[i][j][a]+=dp[i-1][j-count(a)][b];
状态转移限制条件：
(j>=count(a)、且状态a和状态b不互相攻击)
第i行状态为a，第i-1行状态为b，需要满足两个条件：
1. 第i-1行和第i行都不能有相邻的国王（a、b的二进制中不能有相邻的1）
2. 第i-1行的国王和第i行的国王不能相互攻击 (a&b)==0, (a|b)不能有相邻的1

初始状态：
dp[0][0][0]=1
目标状态：dp[n+1][k][0]

五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

dp[i][j][k]表示前i行放j个国王，并且第i行的状态为k的放法数

状态转移方程：

$$\mathrm{dp[i][j][k]} = \sum_{x=0}^{(1\ll n)-1} dp[i-1][j-count(k)][x]$$

其中count(k)表示状态k的二进制表示有多少个1（即第i行国王个数）

这个做法的状态数是n*n*n*(2^n)，转移需要枚举上一行的所有状态，

最终总复杂度是O(n^3*2^2n)，大约是2e8

		K				
K				K		K

五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

进一步优化

每一行的可行状态数有限，可以预处理出所有可行状态（左边右边没有国王）并存在vector state中，状态合法的条件为((i<<1) & i) == 0

这样就能大幅度减少状态数，N为9时，有89种合法状态。

第i行（状态为a）与第i−1行(状态为b)不冲突判断条件为：

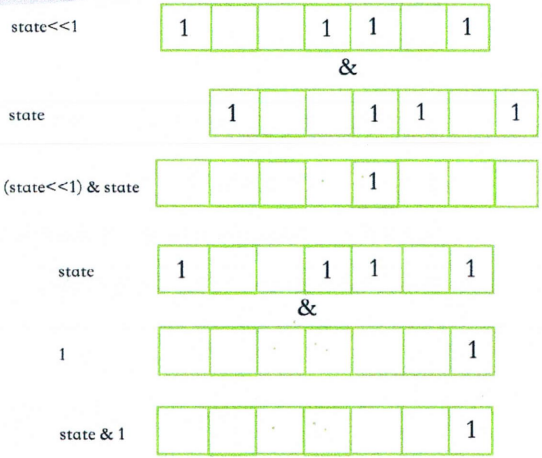
!(a&b)&&!((a<<1)&b)

此外还要注意国王的总数量j要大于等于第i行状态a中的国王数量和。

		K				
K				K		K

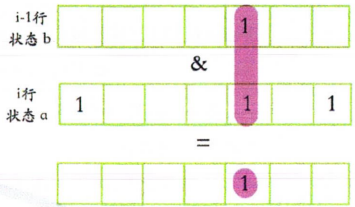
五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

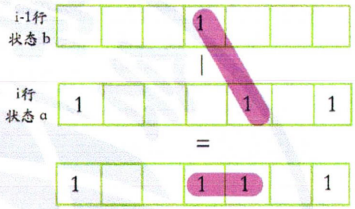
```cpp
#include <iostream>
#include <vector>
using namespace std;
int n, k;
//第一维表示行数，第二维表示当前放的国王总数，第三维
//表示当前行的国王状态数，最多9个国王，全为1时状态为511
long long dp[11][82][512];
//state 记录当行不冲突的状态，N 为 9 时，有 89 种
//asd[s] 记录与当前行 s 不冲突的状态，便于转移
vector<int> state, asd[513];
//检查状态 a 是否有相邻的 1
bool check(int state)
{
    return !((state<<1) & state);
}
//统计某个状态里面，1 的数量
int count(int state)
{
    int num=0;
    while(state){
        num+=state&1;
        state>>=1;
    }
    return num;
}
```



五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

```
int main()
{
    cin >> n >> k;
    //枚举一行所有的状态，0 表示不放国王，1 表示放国王
    //拿 n 等于 10 为例，
    //总共有 0000000000 ~ 1111111111，共 1<<10，也就是 1024 种方案
    //第一种方案，1<<n是 1后面 n 个 0
    for(int i=0; i<(1<<n); ++i){
        //从这么多状态中，找出没有相邻1 的状态，因为同一行相邻位置是 1 的话会攻击
        if(check(i)){
            state.push_back(i);
        }
    }
    cout << state.size() << endl;
    //预处理出，从某个状态可以到达的下一状态，也就是可以作为相邻两行的状态
    //上下相同位置不攻击，斜着不攻击
    for(auto a:state){
        for(auto b:state){
            //相邻两行状态中上下挨着的格子不能都为 1，并且 斜着也不能都为 1
            //如果斜着位置都是1，a | b就会出现相邻位置是1的情况，check就会 false
            if((a&b)==0 && check(a|b)){
                //放到vector里，表示状态a、b可作为上下相邻的两行，即可以互相转移
                asd[a].push_back(b);
            }
        }
    }
    ... ...
}
```





五、棋盘类状压——P1896 [SCOI2005] 互不侵犯

```
int main()
{
    ... ...
    //前 0 行，一个国王不放，状态为 n 个 0 的方案数是 1
    dp[0][0][0]=1;
    //枚举 第 1 行到第 n+1 行
    for(int i=1; i<=n+1; ++i){
        //对于前 i 行，枚举已经放的国王数
        for(int j=0; j<=k; ++j){
            //枚举第 i 行当前选择的状态 a
            for( auto a:state){
                //枚举第 i-1 行可能的状态b，并从 前i-1行的情况转移到前 i 行
                for(auto b:asd[a]){
                    //如果当前总的国王数大于等于第 i 行选择状态 a 的国王数
                    //则 f[i][j][a]可以由前i-1 行，放 j-count(a) 个国王转移
                    //过来，转移过来的状态 b 不能和状态a 冲突（已经预处理过）
                    if(j>=count(a)){
                        dp[i][j][a]+=dp[i-1][j-count(a)][b];
                    }
                }
            }
        }
    }
    cout << dp[n+1][k][0] << endl;
    return 0;
}
```

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

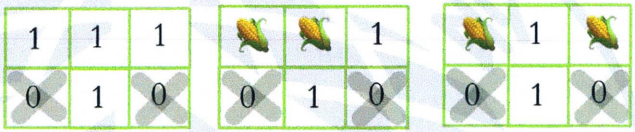
题面： 农场主 John 新买了一块长方形的新牧场，这块牧场被划分成 M 行 N 列 (1≤M≤12,1≤N≤12)，每一格都是一块正方形的土地。John 打算种上美味的草，有些土地贫瘠，不能用来种草。奶牛们喜欢独占一块草地，于是 John 不会选择两块相邻的土地，即没有哪两块草地有公共边。John 想知道，如果不考虑草地的总块数，那么，一共有多少种种植方案可供他选择？（当然，把新牧场完全荒废也是一种方案）

输入： 第一行：两个整数 M 和 N，第 2 到第 M+1 行：每行包含 N 个用空格隔开的整数，描述了每块土地的状态。第 i+1 行描述了第 i 行的土地，所有整数均为 0 或 1，是 1 的话，表示这块土地足够肥沃，0 则表示这块土地不适合种草。

输出： 一个整数，即牧场分配总方案数除以 10^8的余数。

输入样例：
2 3
1 1 1
0 1 0

输出样例：
9



灰色 X 区域贫瘠，不能种　　不合法，玉米有邻边　　合法，玉米无邻边

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

玉米田与上一题（互不侵犯）的联系：

相似点：能不能种玉米，只与上一行有关系

区别：

互不侵犯：周围8个位置不能摆

玉米田：周围四个不能摆

状态表示（s是一个N位二进制数）

互不侵犯：dp[i][j][s]

含义：前i行已经放完了，当前放了j个国王，第i行国王的摆放状态为s的方案数。

玉米田：dp[i][s]，因为不限制数量，所以不需要互不侵犯中间的第二维

含义：前i行已经种完玉米了，第i行玉米的种植状态为s的方案数。

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

玉米田

状态表示dp[i][s]
含义：前i行已经种完了，第i行玉米的种植状态为s的方案数。
s是一个N位二进制数

集合：所有只种在前i行，并且第i行玉米的种植状态为s的所有方案的集合

属性：方案数

状态计算，能不能种玉米，只与上一行有关系，考虑第i行由第i-1行转移过来
dp[i][a]：前i行已经种完了，并且第i行的状态是a，第i-1行的状态是b 的方案数。
dp[i-1][b]：前i-1行已经种完了，并且第i-1行的状态是b的方案数。
dp[i][a]+=dp[i-1][b];
状态转移限制条件：
（状态a和状态b不互相攻击）
第i行状态为a，第i-1行状态为b，需要满足3个条件：
1. 第i-1行和第i行不能有相邻的玉米（a、b的二进制中不能有相邻的1）
2. 第i-1行的玉米和第i行的玉米不能有邻边 (a&b)==0
3. 每一行的玉米不能种在贫瘠的土地上（ok[i]|a)<=ok[i]）

初始状态：dp[0][0]=1
目标状态：dp[m+1][0]

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

dp[i][k]表示前i行已经种完玉米了，并且第i行的状态为k的种植方案数

状态转移方程：

$$dp[i][k] = \sum_{x=0}^{(1\ll n)-1} dp[i-1][x]$$

时间复杂度=状态数*每次转移的时间复杂度

=12*O(状态数)*O(状态数)

=12*2^12*2^12

≈2e8

最终总复杂度可过。

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

进一步优化

每一行的可行状态数有限，可以预处理出所有可行状态（没有相邻的玉米）并存在vector state中，状态合法的条件为((i<<1) & i) == 0

这样就能大幅度减少状态数，N为12时，有377种合法状态。

第i行（状态为a）与第i-1行(状态为b)不冲突判断条件为：

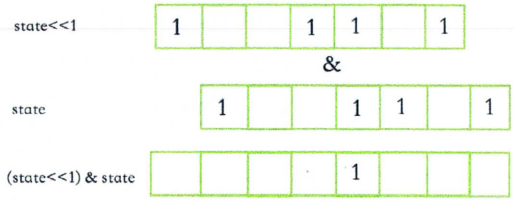
(a&b)==0，也可以根据a预处理出b

此外还要注意玉米不能种植在贫瘠的土地上。

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

dp[i][j]：前i行已经种完了，第i行玉米的种植状态为s的方案数。

```
#include <iostream>
#include <vector>
using namespace std;
int m, n, ok[13], x, mod=1e8, dp[13][1<<12];
vector<int> state, asd[1<<12];
//判断是否有相邻的 1，没有相邻的 1 返回 true
bool check(int state)
{
    return !((state<<1) & state);
}
```

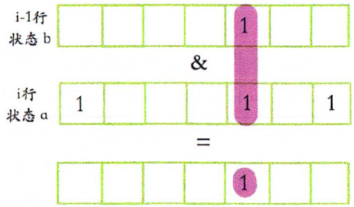


五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

```
int main()
{
    cin >> m >> n;
    ok[0]=(1<<n)-1;
    //处理土地贫瘠状况，例如第 i 行的土地情况为 1 1 0，则ok[i]=3(二进制011)
    for(int i=1; i<=m; ++i){
        for(int j=1; j<=n; ++j){
            cin >> x;
            //ok[i]记录第 i 行土地的情况，1 表示适合种植
            ok[i]+=x<<(j-1);
        }
    }
    ... ...
}
```

五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

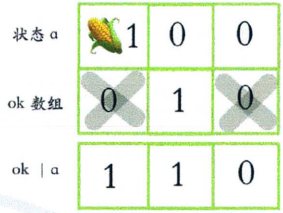
```
int main()
{
    ... ...
    //预处理所有没公共边的状态，也就是没有相邻 1 的状态
    for(int i=0; i<(1<<n); ++i){
        if(check(i)){
            state.push_back(i);
        }
    }
    //预处理所有可以转移的状态，也就是上下两行没有相邻 1 的状态
    for(auto a:state){
        for(auto b:state){
            if((a&b)==0){
                //a状态可以由 b 状态转移过来
                asd[a].push_back(b);
            }
        }
    }
}
```



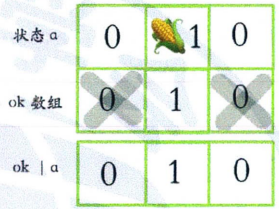
五、棋盘类状压——P1879 [USACO06NOV] Corn Fields G

```
int main()
{
    ... ...
    //dp 求解
    dp[0][0]=1;
    //枚举每一行
    for(int i=1; i<=m+1; ++i){
        //枚举第 i 行的状态
        for(auto a:state){
            //枚举第 i-1 行可行（与第 i 行不冲突）的状态
            for(auto b:asd[a]){
                //如果玉米没有种在贫瘠的土地上
                if((ok[i]|a)<=ok[i]){
                    dp[i][a]+=dp[i-1][b];
                    dp[i][a]%=mod;
                }
            }
        }
    }
    //输出答案
    cout << dp[m+1][0] << endl;
    return 0;
}
```

dp[i][a]：前i行已经种完了，第i行玉米的种植状态为a的方案数。
dp[i-1][b]：前i-1行已经种完了，第i-1行玉米的种植状态为b的方案数。



(ok | a)>ok 不合法



(ok | a)<=ok 合法

五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

一个 N×M 的地图由 N 行 M 列组成，地图的每一格可能是山地（用 H 表示），也可能是平原（用 P 表示），如右图。

在每一格平原地形上最多可以布置一支炮兵部队（山地上不能够部署炮兵部队）；如果在地图中的灰色所标识的平原上部署一支炮兵部队，则图中的黑色的网格表示它能够攻击到的区域：沿横向左右各两格，沿纵向上下各两格。图上其它白色网格均攻击不到。从图上可见炮兵的攻击范围不受地形的影响。

现在，将军们规划如何部署炮兵部队，在防止误伤的前提下（保证任何两支炮兵部队之间不能互相攻击，即任何一支炮兵部队都不在其他支炮兵部队的攻击范围内），在整个地图区域内最多能够摆放多少我军的炮兵部队。

$N \leqslant 100, M \leqslant 10$

P	P	H	P	H	H	P	P
P	H	P	H	P	H	P	P
P	P	P	H	H	H	P	H
H	P	H	P	P	P	P	H
H	P	P	P	P	H	P	H
H	P	P	H	P	H	H	P
H	H	H	P	P	P	P	H

五、棋盘类状压——P2704 [NOI2001] 炮兵阵地

炮兵阵地

状态表示dp[i][j][k]
含义：已经摆完前i行，第i行的状态是state[j],第i-1行的状态是state[k],并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量

集合：所有只摆在前i行，并且第i行摆放的状态为state[j]，第i-1行摆放的状态为state[k]的所有方案的集合

属性：最多摆放炮兵阵地数量

状态计算，能不能摆炮兵，与上两行有关系，考虑第i行由第i-1行转移过来
dp[i][a][b]：前i行已经摆完了，并且第i行的状态是a，第i-1行的状态是b 的最大炮兵数。
dp[i-1][b][c]：前i-1行已经种完了，并且第i-1行的状态是b，第i-2行的状态是c的最大炮兵数。
dp[i][a][b]=max(dp[i][a][b], dp[i-1][b][c]+cnt[a]);

目标状态：dp[n+2][0][0]
n+2行没有放，n+1行没有放

状态转移限制条件：
第i行状态为a，第i-1行状态为b，第i-2行状态为c，需要满足3个条件：
1. 状态a、b、c两两之间不能互相攻击
2. 每一行的炮兵行内不能互相攻击（不能有相邻的1，也不能有间隔1个0相邻的1，比如101）
3. 每一行的炮兵不能摆在山地

五、棋盘类状压——P2704 [NOI2001] 炮兵阵地

dp[i][j][k]表示已经摆完前i行，第i行的状态是state[j],第i-1行的状态是state[k],并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量

状态转移方程：

$$\mathrm{dp[i][j][k]} = max(\mathrm{dp[i][j][k]}, \mathrm{dp[i-1]}[k][l] + cnt[\mathrm{j}])$$

其中state[l]为第i-2行的摆放状态，cnt[j]为第i行炮兵阵地的摆放数量

时间复杂度=状态数*每次转移的时间复杂度

=100*O(状态数)*O(状态数)*O(状态数)

=100*2^10*2^10*2^10

≈1e11

直接暴力会超时

五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

考虑优化

每一行的可行状态数有限，可以预处理出所有可行状态（没有相邻的炮兵阵地，也没有间隔1个格子的炮兵阵地）并存在vector state中，

状态合法的条件为((i<<1) & i) == 0 && ((i<<2) & i) == 0

这样就能大幅度减少状态数，N为10时，有60种合法状态。

实际复杂度为100*60^3=2e7

第i行（状态为a）、第i−1行(状态为b)、第i-2行(状态为c)两两不冲突判断条件为：

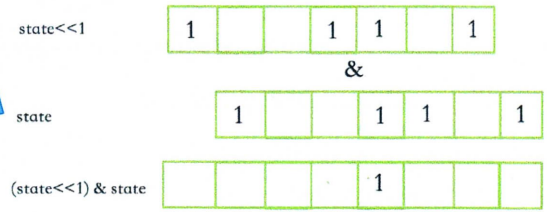
(a&b)==0 && (a&c)==0 && (b&c)==0

此外还要注意炮兵阵地不能摆放在山地上。

五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

```cpp
#include <iostream>
#include <vector>
using namespace std;
int n, m, mymap[110], dp[110][70][70], cnt[70], statenum;
char c;
vector<int> state;
//状态 state 对应的二进制数中不能含有相邻的2个 1，也不能出现 101
//这种情况，如果有的话就返回 false
bool check(int state)
{
    if(((state<<1) & state) || ((state<<2) & state)){
        return false;
    }
    return true;
}
int count(int state)
{
    int num=0;
    while(state){
        num+=state&1;
        state>>=1;
    }
    return num;
}
```

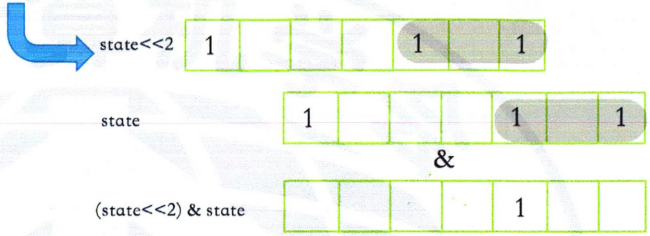
dp[i][j][k]表示已经摆完前i行，第i行的状态是state[j]，第i-1行的状态是state[k]，并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量



五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

```
#include <iostream>
#include <vector>
using namespace std;
int n, m, mymap[110], dp[110][70][70], cnt[70], statenum;
char c;
vector<int> state;
//状态 state 对应的二进制数中不能含有相邻的2个 1，也不能出现 101
//这种情况，如果有的话就返回 false
bool check(int state)
{
    if(((state<<1) & state) || ((state<<2) & state)){
        return false;
    }
    return true;
}
int count(int state)
{
    int num=0;
    while(state){
        num+=state&1;
        state>>=1;
    }
    return num;
}
```

dp[i][j][k]表示已经摆完前i行，第i行的状态是state[j]，第i-1行的状态是state[k]，并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量

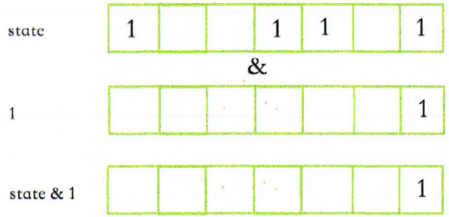


五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

```
#include <iostream>
#include <vector>
using namespace std;
int n, m, mymap[110], dp[110][70][70], cnt[70], statenum;
char c;
vector<int> state;
//状态 state 对应的二进制数中不能含有相邻的2个 1，也不能出现 101
//这种情况，如果有的话就返回 false
bool check(int state)
{
    if(((state<<1) & state) || ((state<<2) & state)){
        return false;
    }
    return true;
}
int count(int state)
{
    int num=0;
    while(state){
        num+=state&1;
        state>>=1;
    }
    return num;
}
```

状态3维，直接100*1024*1024会超空间

dp[i][j][k]表示已经摆完前i行，第i行的状态是state[j],第i-1行的状态是state[k],并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量



五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

P	P	H	P	H	H	P	P
P	H	P	H	P	H	P	P
P	P	P	H	H	H	P	H
H	P	H	P	P	P	P	H
H	P	P	P	P	H	P	H
H	P	P	H	P	H	H	P
H	H	H	P	P	P	P	H

```
int main()
{
    cin >> n >> m;
    for(int i=1; i<=n; ++i){
        for(int j=1; j<=m; ++j){
            cin >> c;
            if(c=='P'){
                mymap[i]=(mymap[i]<<1)+0;
            }
            else{
                mymap[i]=(mymap[i]<<1)+1;
            }
        }
    }
    //最多 60 个状态
    for(int i=0; i<(1<<m); ++i){
        if(check(i)){
            state.push_back(i);
            cnt[statenum]=count(i);
            statenum++;
        }
    }
    ... ...
}
```

五、棋盘类状压—— P2704 [NOI2001] 炮兵阵地

```
int main()
{
    ... ...
    for(int i=1; i<=n+2; ++i){
        //第 i 行的状态为 a=state[j]
        for(int j=0; j<state.size(); ++j){
            //第 i-1 行的状态为 b=state[k]
            for(int k=0; k<state.size(); ++k){
                //第 i-2 行的状态为 c=state[l]
                for(int l=0; l<state.size(); ++l){
                    int a=state[j];
                    int b=state[k];
                    int c=state[l];
                    //如果这三行冲突，不能相邻，跳过
                    if((a&b)!=0 || (a&c)!=0 || (b&c)!=0){
                        continue;
                    }
                    //如果第 i 行的状态和地图没有冲突，则转移
                    //考虑为什么不需要判断 b&map[i-1]==0 和 c&map[i-2]==0?
                    if((a&mymap[i])==0)
                        //第 i 行放 a
                        dp[i][j][k]=max(dp[i][j][k],dp[i-1][k][l]+cnt[j]);
                }
            }
        }
    }
    cout << dp[n+2][0][0] << endl;
    return 0;
}
```

dp[i][j][k]表示已经摆完前i行，第i行的状态是state[j],第i-1行的状态是state[k],并且所有摆放的炮兵相互之间攻击不到，且没有炮兵摆放在山地上的最多摆放数量

目标状态：dp[n+2][0][0]
前n+2行摆放完，并且第n+2行没有放，第n+1行
也没有放，相当于放完前n行

六、集合类状态压缩的动态规划模型

子集枚举，使用二进制数字来表示一种状态。对于“状态压缩动态规划”中的状态，通常与集合相关联。集合本身具有三个性质：确定性，互异性和无序性。这也就决定了只关心每个元素的存在状态，而这通常可以使用 0或者1 表示存在或者不存在。例如，一顿饭有8道菜，小明同学最终吃了某几道菜，必然是这8道菜的子集。

令1 表示吃了，0表示没吃，那么以下几个状态：10000001 表示只吃了1号菜和8号菜；01010101 表示吃了1,2,5,7 四道菜。

六、集合类状态压缩的动态规划模型

由此可见，可以通过一串 01码来清晰地表示一个集合的状态。同时，在确定了最低位的前提下，一串0-1码与一个二进制数一一对应。这种表示状态的方式被称为状态压缩，简称状压。

所谓“压缩”，即是用一个二进制数来保存一组状态，将一个集合的状态压缩进了一个二进制数中，而通常这个二进制数在十进制下的大小可以作为其编号。状态压缩本质上是进行了两步操作

1. 给这个集合的每个状态一个编号；
2. 通过该编号，轻易地访问该状态。

六、集合类状态压缩的动态规划模型——求子集

集合S={1, 2, 3}

S的子集有：

用3位二进制数来表示集合中是否包含1，2，3，1表示包含，0表示不包含，最低位表示数字1，最高位表示数字3，对应的二进制为：

{}	000
{1}	001
{2}	010
{3}	100
{1, 2}	011
{1, 3}	101
{2, 3}	110
{1, 2, 3}	111

六、集合类状态压缩的动态规划模型——求子集

集合S={1, 2, 3}

S的子集有：

{}	000
{1}	001
{2}	010
{3}	100
{1, 2}	011
{1, 3}	101
{2, 3}	110
{1, 2, 3}	111

用3位二进制数来表示集合中是否包含1，2，3，1表示包含，0表示不包含，最低位表示数字1，最高位表示数字3，对应的二进制为：

```
//枚举状态 s 的所有子集 j
for(int j=s; j; j=s&(j-1)){
    cout << j << endl;
}
```

S=111(二进制)，用S按位与确保子集不会出现其他数字

j=111

j=111&110=110

j=111&101=101

j=111&100=100

j=111&011=011

j=111&010=010

j=111&001=001

j=111&000=000

六、集合类状态压缩的动态规划模型——求子集

```
//枚举状态 s 的所有子集 j
for(int j=s; j; j=s&(j-1)){
   cout << j << endl;
}
```

设集合S有n个元素，对应的二进制为有n个1，求S所有子集的时间复杂度$O(2^n)$，因为每个元素都是可选可不选。

六、集合类状压dp——P5911 [POI 2004] PRZ

一只队伍在爬山时碰到了雪崩，他们在逃跑时遇到了一座桥，他们要尽快的过桥。任何时候队伍在桥上的人都不能超过桥的承重。所以过桥时只能分批过，当一组全部过去时，下一组才能接着过。队伍里每个人过桥都需要一定的时间，当一批队员过桥时时间应该算走得最慢的那一个，每个人也有特定的重量，求过桥的最少总时间。

$100 \leqslant W \leqslant 400, 1 \leqslant n \leqslant 16, 1 \leqslant t \leqslant 50, 10 \leqslant w \leqslant 100$

输入样例：	输出样例：
100 3 //承重能力W 队员人数n	42
24 60 //所需时间t 队员重量w	
10 40	
18 50	

六、集合类状压dp——P5911 [POI 2004] PRZ

dp[s]表示状态集合s中所有人全部过桥所需的最短时间

预处理出状态s过桥所需最短时间（要求状态s不超重）

转移的时候需要枚举s的子集i，将s拆分成子集i和子集j=s^i，子集j不超重的情况下转移方程为：

dp[s]=min(dp[s], T[j]+dp[i]) (W[j]≤w)

T[i]表示集合i中，过桥耗时最长那个人的过桥时间

W[i]表示集合i中，所有人的总重量

求补集：异或操作

```
//枚举状态 s 的所有子集 j
for(int j=s; j; j=s&(j-1)){
    cout << j << endl;
}
```

六、集合类状压dp——P5911 [POI 2004] PRZ

预处理

```
#include <bits/stdc++.h>
using namespace std;
int mx, n, dp[1<<16], t[17], w[17], T[1<<16], W[1<<16];
int main()
{
    cin >> mx >> n;
    for(int i=1; i<=n; ++i){
        cin >> t[i] >> w[i];
    }
    memset(dp, 0x3f, sizeof(dp));
    dp[0]=0;
    //一个状态（子集）中从最低位（记为 1 号人）开始至最高位，
    //数值为 1 的位置表示该号人一起过桥
    //预处理处每个子集中，花费的时间以及总共的重量
    for(int i=0; i<1<<n; ++i){  //枚举所有的状态
        for(int j=1; j<=n; ++j){    //枚举所有的人
            //如果状态（集合） i 包含 第 j 个人
            if((i>>(j-1))&1){
                //T[i]表示集合 i 中，过桥耗时最长那个人的过桥时间
                T[i]=max(T[i], t[j]);
                //W[i]表示集合 i 中，所有人的总重量
                W[i]+=w[j];
            }
        }
    }
    ... ...
}
```

预处理时间复杂度

2^n*n=2^16*16=1e6

六、集合类状压dp——P5911 [POI 2004] PRZ

状态转移

```
int main()
{
    ... ...
    //枚举所有的状态
    for(int i=0; i<1<<n; ++i){
        //枚举状态 i 的所有子集 j
        for(int j=i; j; j=i&(j-1)){
            if(W[j]<=mx){
                //从子集(状态)j转移过来，所以需确保
                //集合j中的人可以一起通过桥
                dp[i]=min(dp[i], dp[i^j]+T[j]);
            }
        }
    }
    cout << dp[(1<<n)-1] << endl;
    return 0;
}
```

设集合S有n个元素，对应的二进制为有n个1，求S每一个子集的所有子集，时间复杂度是$O(3^n)$

推导：

求子集时，对于有着 k 个 1 的二进制数字，枚举子集需要的时间复杂度为 2^k，

拥有 k 个 1 的数字的集合数量，用组合数学表示为：$\binom{n}{k}$

那么总的时间复杂度为

$\sum_{k=0}^{n}\binom{n}{k}2^k=\sum_{k=0}^{n}\binom{n}{k}2^k1^{n-k}=(1+2)^n=3^n$

状态转移的时间复杂度

3^n=4e7

六、集合类状压dp——P10447 最短 Hamilton 路径

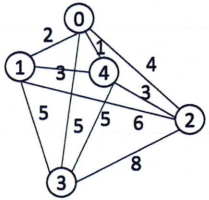
给定一张 n 个点的带权无向图，点从 0～n−1 标号，求起点 0 到终点 n−1 的最短 Hamilton 路径。Hamilton 路径的定义是从 0 到 n−1 不重不漏地经过每个点恰好一次。1 ≤ n ≤ 20。

输入格式：第一行输入整数n。接下来的n行每行n个整数，其中第i行第j个整数表示点i到j的距离（记为a[i, j]）。0 ≤ a[i, j] ≤ 1e7

对于任意的x, y, z，数据保证 a[x, x]=0，a[x, y]=a[y, x] 并且 a[x, y]+a[y, z]>=a[x, z]。

输出格式：输出一个整数，表示最短哈密顿路径的长度。

输入样例：	输出样例：
5	18
0 2 4 5 1	
2 0 6 5 3	
4 6 0 8 3	
5 5 8 0 5	
1 3 3 5 0	



暴力的方案，(n-2)!*n　　0-3-1-2-4

六、集合类状压dp——P10447 最短 Hamilton 路径

最短 Hamilton 路径

状态表示dp[i][j]
含义：所有从0点走到j点，经过集合i中所有点的路径的最小值，
从点0到点 j 所走过的点都存在状态i中，i是一个二进制压缩数例
如i=00110101,表示集合包含0、2、4、5这几个点

集合：所有从点0出发，走过点集i，停留在点j的路径长度的集合

属性：求最小值

状态计算，点0经过一系列点到点k，然后点k再到点j
dp[i][j]：从点0走的点j，经过集合i种所有点的路径最小值
dp[i-{j}][k]：从点0走到点k，经过集合i去掉点j之后所有点，的路径最小值。
根据倒数第二个点k来分类枚举
dp[i][j]=min(dp[i][j], dp[i-{j}][k]+w[k][j]); k=0,1,2...n-1
状态转移限制条件：
1. 状态集合i要包含点j
2. 状态集合i去掉点j之后，还需要包含点k

初始状态：dp[1][0]=0
目标状态：dp[(1<<n)-1][n-1]

六、集合类状压dp——P10447 最短 Hamilton 路径

```
#include <bits/stdc++.h>
using namespace std;
int n, dp[1<<20][20], w[20][20];
int main()
{
    cin >> n;
    for(int i=0; i<n; ++i){
        for(int j=0; j<n; ++j){
            cin >> w[i][j];
        }
    }
    memset(dp, 0x3f, sizeof(dp));
    //从 0 号点出发（对应的状态为0001）
    //没有到达任何点，走的路径为 0
    dp[1][0]=0;
    ... ...
}
```

六、集合类状压dp——P10447 最短 Hamilton 路径

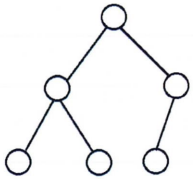
```
int main()
{
    ... ...
    //枚举所有经过点的状态，小于号的优先级低于<<
    for(int i=1; i<1<<n; i+=2){
        //枚举当前状态 i，停留在了 j 号点
        for(int j=0; j<n; ++j){
            //要想停在j点，那么状态i里面从最低位(第0位)数第j位需要是1
            if((i>>j)&1){
                //枚举所有可能的中转点k，经过状态i停留在k点，然后再从k点到j点
                for(int k=0; k<n; ++k){
                    //要想停留在k点，那么集合i去掉j点之后，还需要包含点 k
                    if((i-(1<<j))>>k & 1){
                        //状态i里去掉j号点，状态i-(1<<j)相当于状态i的子集
                        dp[i][j]=min(dp[i][j], dp[i-(1<<j)][k]+w[k][j]);
                    }
                }
            }
        }
    }
    //减号的优先级要高于<<
    cout << dp[(1<<n)-1][n-1] << endl;
    return 0;
}
```

时间复杂度
2^n*n^2=4e7

六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

给出一副无向图，n个点，m条边，每条边都有一个权值v且均未开通，先可以随便取一个起点，要开通一些边，使它成为一个连通图，开通一条边的代价为这条边的权值*起点到它的点的个数(起点也算)，求最小代价。

$1<=n<=12$　$0<=m<=10^3$　$v<=5*10^5$



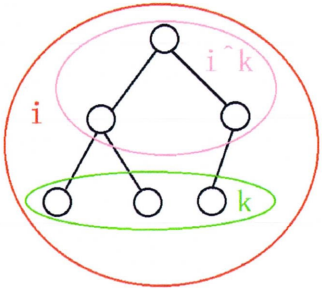
本质上是构成一棵树

考虑当前选了哪些点，形成点集。

每条路的代价与高度有关，考虑当前的高度

考虑状态dp[i][j]

含义：当前所选的点集是i，当前树的高度是j的最小权值



六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

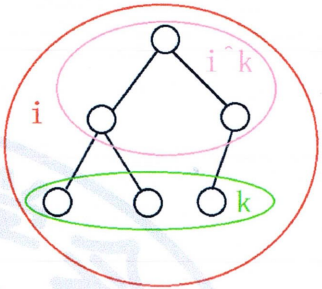
宝藏

状态表示dp[i][j]
含义：当前所选的点集是i，当前树的高度是j的最小权值

集合：所有选择的点集是i，且当前树的高度是j的所有方案

属性：权值最小值

状态计算
想求dp[i][j]，需要枚举一下第j层有哪些点，第j层的点相当于求i的子集，对应i的某个子集k一旦确定，前面点集i^k也确定了，第j层的点集k到前面点集i^k的最小值也是确定的，
不确定的是前面j-1层，i^k这些点集的形态，也就是dp[i^k][j-1]，如果可以使得dp[i^k][j-1]最小，那么就可以使得dp[i][j]最小
枚举集合 i，遍历i的每一个子集k(最后一层的点集)，枚举树高度j
dp[i][j]=min(dp[i][j], dp[i^k][j-1]+第j层的代价)
预处理：g[i][j]表示点i到点集j的最短距离



初始状态:dp[每个点做根][0]=0
目标状态:dp[(1<<n)-1][所有高度]

六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

```
#include <bits/stdc++.h>
using namespace std;
int n, m, d[13][13], g[13][1<<12], dp[1<<12][13], u, v, w,
ans=0x3f3f3f3f;
int main()
{
    // n, m 代表宝藏屋的个数和道路数。
    cin >> n >> m;
    memset(d, 0x3f, sizeof d);
    for(int i=1; i<=n; ++i){
        d[i][i]=0;
    }
    while(m--){
        cin >> u >> v >> w;
        d[u][v]=d[v][u]=min(d[u][v], w);
    }
    ... ...
}
```

六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

```
int main()
{
    ... ...
    memset(g, 0x3f, sizeof g);
    //枚举每一个点
    for(int i=1; i<=n; ++i){
        //枚举每一个状态(点集)
        for(int j=0; j< 1<<n; ++j){
            //枚举集合中的每一个点
            for(int k=1; k<=n; ++k){
                //如果集合 j 里面包含点 k 的话
                if(j>>(k-1) & 1){
                    g[i][j]=min(g[i][j], d[i][k]);
                }
            }
        }
    }
    ... ...
}
```

时间复杂度：$O(2^n*n^2)$

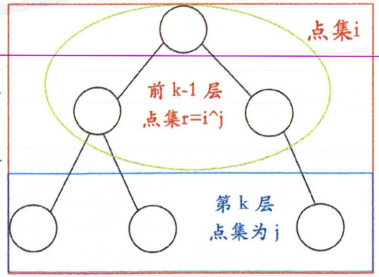
六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

```
int main()
{
    ... ...
    memset(dp, 0x3f, sizeof dp);
    //开发商免费打通根节点
    for(int i=0; i<n; ++i){
        dp[1<<i][0]=0;
    }
    //枚举所有的状态
    for(int i=1; i< 1<<n; ++i){
        //枚举集合 i 的所有的子集j，充当最后一层的点集
        for(int j=(i-1)&i; j; j=(j-1)&i){
            //计算除去最后一层点以后，前面层的点集，也就是求集合 i 中 j 的补集
            int r=i^j; //r=i-j也行
            //代价初始化为 0
            int cost=0;
            //枚举所有的点
            for(int k=1; k<=n; ++k){
                //如果 k 在集合 j 中，也就是说点 k 在当前的最后一层
                if(j>>(k-1) & 1){
                    cost+=g[k][r];  //加上点 k 到前面层构成的点集的最小边权
                    if(cost>=0x3f3f3f3f){
                        break;
                    }
                }
            }
            if(cost>=0x3f3f3f3f){
                continue;
            }
            //枚举层次，树最多 n-1层
            for(int k=1; k<=n-1; ++k){
                dp[i][k]=min(dp[i][k], dp[r][k-1]+cost*k);
            }
        }
    }
    ... ...
}
```

初始状态：dp[每个点做根][0]=0

时间复杂度：$O(2^n*n^2+3^n*n)\approx 6e6$

当前高度为k



状态计算:想求dp[i][k]，需要枚举一下第k层有哪些点，第k层的点相当于求i的子集，对应i的某个子集j一旦确定，前面点集r=i^j也确定了，第k层的点集k到前面点集r=i^k的最小值也是确定的，不确定的是前面k-1层，i^j这些点集的形态，也就是dp[i^j][k-1]，如果可以使得dp[i^j][k-1]最小，那么就可以使得dp[i][k]最小枚举集合i，遍历i的每一个子集j(最后一层的点集)，枚举树高度k

dp[i][k]=min(dp[i][k], dp[i^j][k-1]+第k层的代价)

六、集合类状压dp——P3959 [NOIP 2017 提高组] 宝藏

目标状态:dp[(1<<n)-1][所有高度]

```
int main()
{
    ... ...
    //枚举层数
    for(int i=0; i<n; ++i){
        ans=min(ans, dp[(1<<n)-1][i]);
    }
    cout << ans << endl;
    return 0;
}
```

七、总结、习题

- ✓ 状态压缩动态规划的概念
- ✓ 位运算符及常见的位运算操作
- ✓ 状压dfs——P1433 吃奶酪
- ✓ 状压bfs——P2622 关灯问题 II
- ✓ 基于连通性（棋盘类）状态压缩的动态规划模型
  - ——P1896 [SCOI2005] 互不侵犯
  - ——P1879 [USACO06NOV] 玉米田
  - ——P2704 [NOI2001] 炮兵阵地
- ✓ 集合类状态压缩的动态规划模型
  - ——P5911 [POI 2004] PRZ
  - ——P10447 最短 Hamilton 路径
  - ——P3959 [NOIP 2017 提高组] 宝藏

P1441 砝码称重

P3869 [TJOI2009] 宝藏

P10975 Mondriaan's Dream POJ2411

P1450 [HAOI2008] 硬币购物

P2831 [NOIP 2016 提高组] 愤怒的小鸟

P2150 [NOI2015] 寿司晚宴

P1357 花园

CF11D A Simple Task

ARC171D D - Rolling Hash

P8292 [省选联考 2022] 卡牌

P5972 [PA 2019] Desant 黑题