状态压缩动态规划

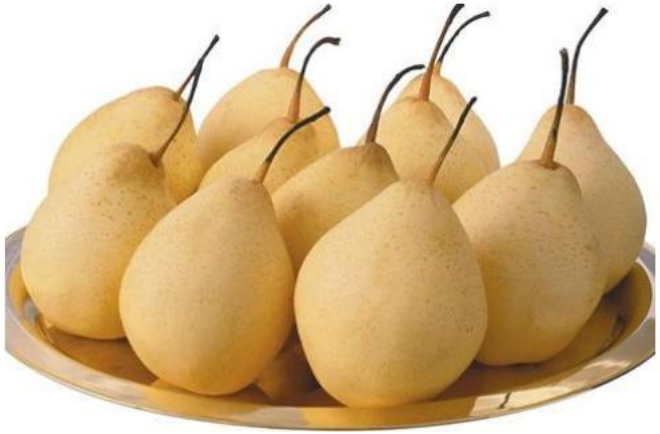
信智学院　陈琰宏

状态压缩

状态压缩就是把一个复杂的，难以直接描述的状态或事物，用一个简单的、计算机能够识别的、便于操作的数字，字符或者是结构体来表示的过程。

如桌上有三种水果，苹果，梨子和香蕉，分别有4,11,3个。

状态压缩


我们给三种水果一个标号，苹果标记为1,梨子标记为2，香蕉标记为3

那么我们就可以简单的用一个数组a来描述现在桌面上的状态，a[1]=4,a[2]=11,a[3]=3

这个过程就是状态压缩。

状态压缩

天花板上有三盏灯，都是开着的。我们先给三个灯标上1,2,3,然后再来用数字1表示灯是开着的，0表示关着的。那么a[1]=a[2]=a[3]=1就是现在三盏灯的状态。

我们可以用一个字符串来示来表示灯的状态,000即三盏灯都是关着的，111即三盏灯都是开着的。

而每盏灯只有两种状态，意味着一个位置上要么是0要么是1，那我们可以用一个二进制数字来代表这个状态，这样，三盏灯对应的全部状态可以用0到7这8个数字来表示。

状态压缩的使用特点

从状态压缩的特点来看，这个算法适用的题目符合以下的条件：

1.解法需要保存一定的状态数据（表示一种状态的一个数据值），每个状态数据通常情况下是可以通过2进制来表示。这就要求状态数据的每个单元只有两种状态，比如说棋盘上的格子，放棋子或者不放，或者是硬币的正反两面。这样用0或者1来表示状态数据的每个单元，而整个状态数据就是一个一串0和1组成的二进制数。

2.解法需要将状态数据实现为一个基本数据类型，比如int，long等等，即所谓的状态压缩。状态压缩的目的一方面是缩小了数据存储的空间，另一方面是在状态对比和状态整体处理时能够提高效率。这样就要求状态数据中的单元个数不能太大，比如用int来表示一个状态的时候，状态的单元个数不能超过32（32位的机器）。

从数学的角度考虑

难以操作的状态

方便操作的状态

我们定义两个函数 f 和 g f(A)=B ，g(B)=A 这就是状态压缩所要表达的意思。

而这两个函数是存在于我们头脑中的，由我们来定义这两个函数，将一个复杂的状态转换成一个 简单的便于操作的状态，然后在解决问题的时候就会比较轻松了。

讲个简单的问题

考虑01背包问题的小数据情况。

给出n<10个物品，每个物品有一个价值v，一个质量w，

而你有一个背包，最多可以装W质量的东西，体积无限大。

问怎么选择可以获得最大的价值

由于n很小，我们可以直接通过dfs来计算答案。
现在我们来考虑状态压缩要怎么处理，
首先，答案一定是在n个物品中选择了m<=n个，每个物品只有要和不要两种选择，那么同之前的电灯一样，我们可以用一个二进制数字来表示最终答案所代表的状态。
假设n是5 那么01100就可以表示选择了第2和第3件物品的状态，
然后我们去计算一下这样的价值和质量，看看质量是不是合法，然后更新一下答案即可。

[2819] 01背包

```
3  int w[105],v[105],n,W;
4  int main(){
5      scanf("%d%d", &W, &n);
6      for (int i = 0; i < n; i++)
7          scanf("%d%d", &w[i], &v[i]);
8      int ans = 0;
9      for (int i = 0; i < (1 << n); i++)
10     {
11         int a = 0, b = 0;
12         for (int j = 0; j < n; j++)
13         {
14             if (i&(1 << j)) a += v[j], b += w[j];
15         }
16         if (b <= W) ans = max(ans, a);
17
18     }
19     printf("%d\n", ans);
```

输入样例

10 4

2 1

3 3

4 5

7 9

输出样例

12

状态压缩——预备知识

位运算(*bitwise operation*)。其优先级:

~ > & > ^ >|

名称	C/C++样式	简记法则
按位与	&	全一则一，否则为零
按位或	|	有一则一，否则为零
按位取反	~ (!)	是零则一，是一则零
按位异或	^	不同则一，相同则零
左移位	<<	a<<k等价于$a*2^k$
右移位	>>	a>>k等价于$a/2^k$

位运算-左移 <<

假设有个变量x=1；

那么x=x<<1表示的意思如下：

x:0000000000000001（二进制表示）

x<<1:0000000000000010（也就是把x的每一位向左移动1位的结果，右边不够的用0添上）

此时x=x<<1 ==2，其实可以看做乘2

x<<i 表示的就是左移i位,可以看做乘 i 次 2

位运算-右移 >>

假设有个变量x=2;

那么x=x>>1表示的意思如下：

x:　　0000000000000010（二进制表示）

x>>1:0000000000000001（也就是把x的每一位向右移动1位的结果，左边不够的用0添上）

此时x=x>>1 ==1，其实可以看做除2

x>>i 表示的就是右移i位,可以看做除 i 次2

位运算-或运算|

设x=101， y=010(都是二进制表示)

那么x|y表示如下

x: 101

y: 010

x|y:111

其实也就是将x和y表示成二进制，然后按位做或运算

此时x|y==7(10进制表示)

位运算-与运算 &

设x=101 y=010(都是二进制表示)

那么x&y表示如下

x: 101

y: 010

x&y: 000

将x和y表示成二进制，然后按位做与运算

此时x&y==0(10进制表示)

位运算-非运算 ~

设x=101(都是二进制表示)

那么~x表示如下

x:　　101

~x:　　010

其实也就是将x表示成二进制，然后按位做取反运算

此时~x==2(10进制表示)

位运算-与运算^

设x=1010 y=1100（都是二进制表示）

那么x^y表示如下

x: 1010

y: 1100

x^y: 0110

其实也就是将x和y表示成二进制，然后按位做异或运算，也就是相同为0，不同为1

此时x^y==6(10进制表示)

位运算-获取一个或多个固定位的值

假设x=1010(10进制的10)

我们要获取从左边数第2为的值，那么我们可以这样来取

x&(1<<1)也就是

x:　　1010

1<<1:　　0010

x&(1<<1)　0010

这样我们就可以通过判断x&(1<<2)是否等于0来知道这一位是0还是1了

当然我们可以用x&(3<<2)来取得第3位和第4位

位运算-把一个或多个固定位的值置零

假设x=1010(10进制的10)

我们要把从左边数第2为的值置零，那么我们可以这样来做

x&(~(1<<1))也就是

x:　1010

~(1<<1):　1101

x&(~(1<<1))：　1000

当然我们可以用x&(~(3<<2))来把第3位和第4位置零

位运算-一个或多个固定位的值取反

假设x=1010(10进制的10)

要把从左边数第2为的值取反，可以这样做

x^(~(1<<1))

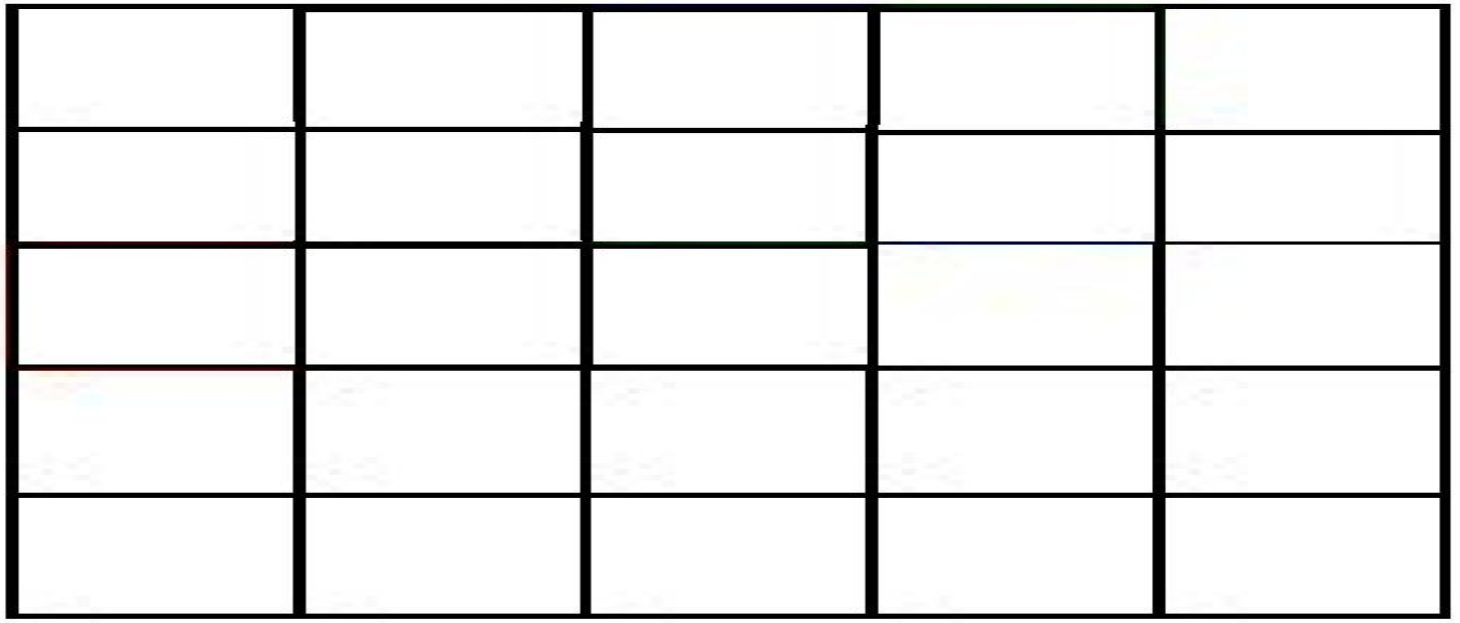
也就是如果x=1010

x:	1010
1<<1:	0010
~(1<<1):	1101
	1010
x^(1<<1) :	0111

当然我们可以用x^(3<<2)来把第3位和第4位取反

1 [2993] 棋盘放置

在*n***n*(*n*≤*10*)的方格棋盘上放置*n*个车(可以攻击所在行、列)，求使他们互相不能攻击的方案总数。



数学角度思考

我们一行一行放置，则第一行有n种选择，第二行n-1，……，最后一行只有1种选择，根据乘法原理，答案就是n!

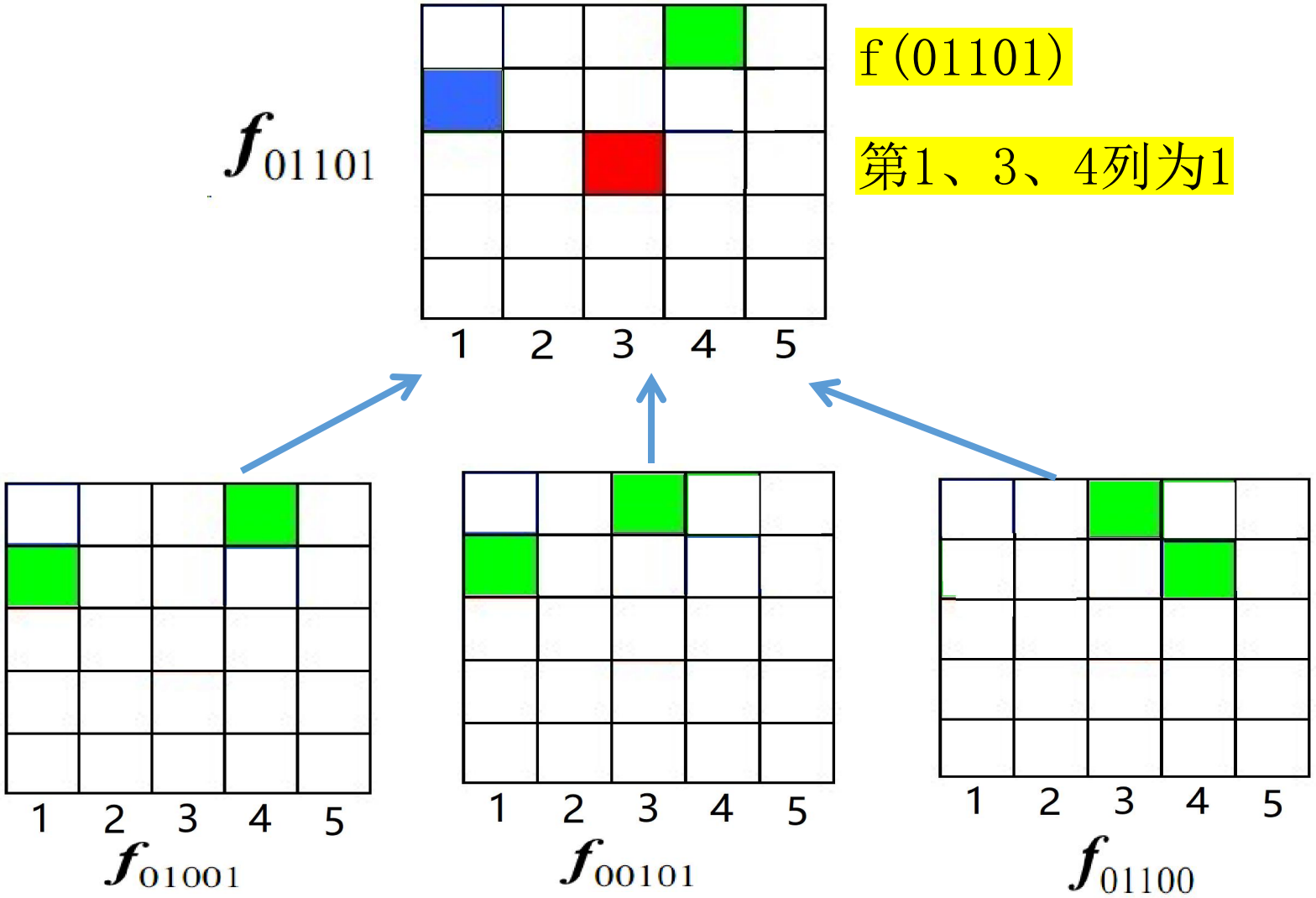
这里既然以它作为状态压缩的引例，当然不会是为了介绍组合数学。我们下面来看另外一种解法：

状态压缩递推(States Compressing Recursion,SCR)

状态压缩角度思考

取棋子的放置情况作为状态，某一列如果已经放置棋子则为1，否则为0。这样，一个状态就可以用一个最多20位的二进制数表示。

例如 $n = 5$, 第1、3、4列已经放置，则这个状态可以表示为01101(从右到左)。设f_s 为达到状态s的方案数，即f(01101)表示前3行，在第1、3、4列已经放置了棋子的方案数。 尝试建立f 的递推关系。


f_{01101}
1 2 3 4 5
f(01101)
第1、3、4列为1
1 2 3 4 5
f_{01001}
1 2 3 4 5
f_{00101}
1 2 3 4 5
f_{01100}

这三种情况互不相交，且只可能有这三种情况，根据加法原理，应该等于这三种情况的和，写成递推式就是：

$$f_{01101} = f_{01100} + f_{01001} + f_{00101}$$

这个式子相当于01101分别从右到左把有1的位置变为零，然后相加。

推广状态S，那么

$$f[S] = \Sigma f[S \hat{} (1<<(i-1))],$$

其中i是枚举的状态S中每一个1的位置（从低位到高位）。然后依次去掉一个1。

边界条件：f[0] = 1。

位运算-与运算^

设x=1010 y=1100（都是二进制表示）

那么x^y表示如下

x: 1010

y: 1100

x^y: 0110

其实也就是将x和y表示成二进制，然后按位做异或运算，也就是相同为0，不同为1

此时x^y==6(10进制表示)

$$f[S] = \Sigma f[S \hat{} (1<<(i-1))],$$

$$\begin{array}{r} 01101 \\ \wedge\ 00001 \\ \hline 01100 \end{array} \qquad \begin{array}{r} 01101 \\ \wedge\ 00100 \\ \hline 01001 \end{array} \qquad \begin{array}{r} 01101 \\ \wedge\ 01000 \\ \hline 00101 \end{array}$$

通过异或操作，出去每次都少一个1的状态

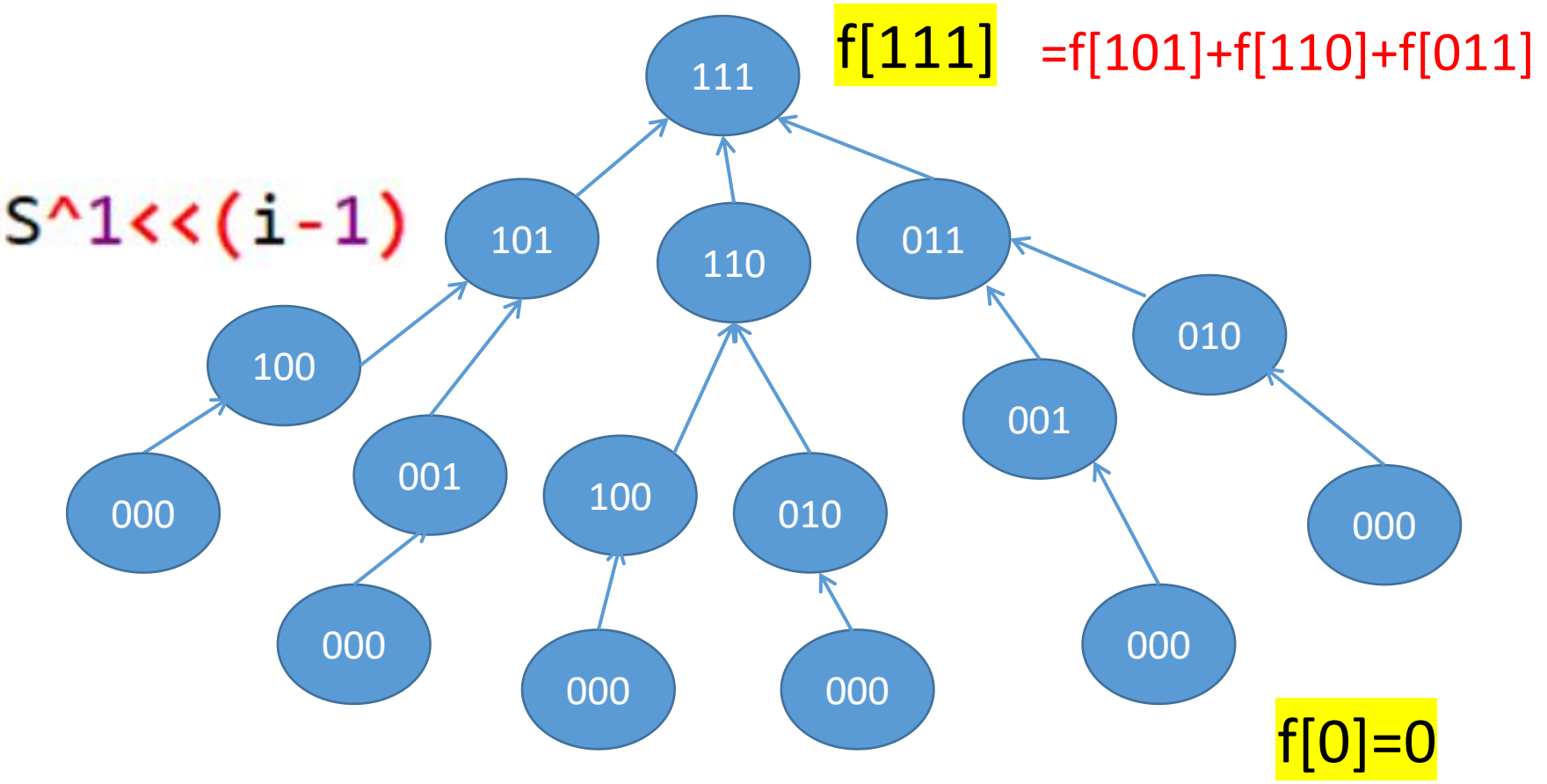
[2993] 棋盘放置

在 n*n(n≤10)的方格棋盘上放置 n 个车(可以攻击所在行、列)，求使它们不能互相攻击的方案总数。

输入 n
输出 使它们不能互相攻击的方案总数

输入样例
5
输出样例
120

n=3


f[111]
=f[101]+f[110]+f[011]
S^1<<(i-1)
111
101
110
011
100
001
100
010
001
010
000
000
000
000
000
000
f[0]=0

源代码

```
#include <iostream>
using namespace std;
const int INF=(1<<20)-1;
int f[INF];
int main()
{
    int n;
    scanf("%d",&n);
    int i,S,j;
    f[0]=1;
    for(S=1;S<=(1<<n)-1;S++)
    {
        for(i=1;i<=n;++i)
        {//依次判断那些位置有棋子
            if(( S & 1<<(i-1))>0)//状态S的第i列有棋子,
            {//如果有棋子
                int s=S^1<<(i-1);//去掉第i列的1
                //int s=S-(1<<(i-1));
                f[S]+=f[s];

            }
        }
    }
    printf("%d",f[(1<<n)-1]);
    return 0;
}
```

源代码

```
3   const int INF=(1<<20)-1;
4   int f[INF];
5   int main()
6   {
7       int n;
8       scanf("%d",&n);
9       int i,S,j;
10      f[0]=1;
11      for(S=1;S<=___1_____;S++)
12      {
13          for(i=1;i<=n;++i)
14          {//依次判断那些位置有棋子
15              if(( ___2_____)//状态S的第i列有棋子,
16              {//如果有棋子
17                  int s=___3_____;//去掉第i列的1
18
19                  f[S]+=f[s];
20
21              }
22          }
23      }
24      printf("%d",___4_____);
25      return 0;
26  }
```

2 [7286] 国王放置

在n*m的棋盘上放置k个国王，要求k个国王互相不攻击，有多少种不同的放置方法。假设国王放置在第(x,y)格，国王的攻击的区域是:(x-1,y-1)，(x-1,y),(x-1,y+1),(x,y-1),(x,y+1),(x+1,y-1),(x+1,y),(x+1,y+1)。

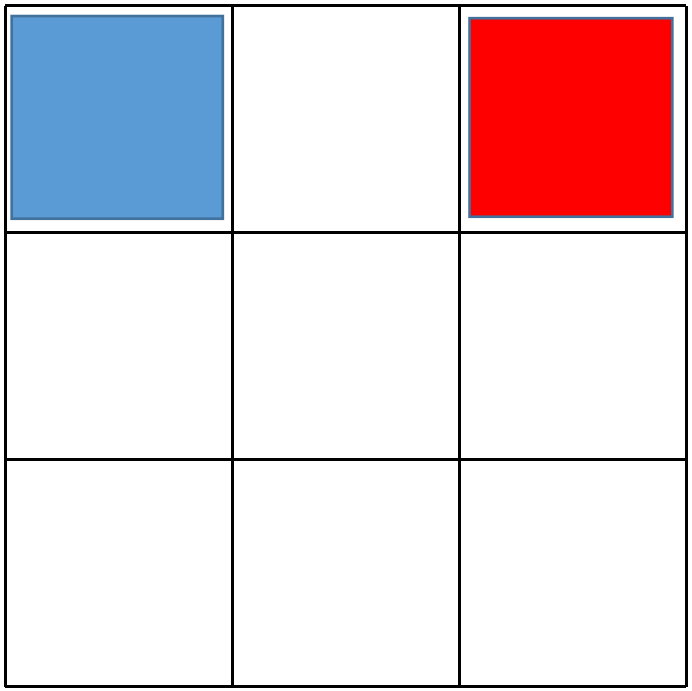
输入包括三个数 n,m,k，(0<n,m<=8,0<k<=n* m)n,m,k，(0<n,m<=5,0<k<=n* m) 含义如题面。

输出不同的放置的方案数，每个结果占一行。

输入样例 4 4 4

输出样例 79

[7286] 国王放置



5种

3种

4种

2种

0种

1种

1种

共计16种

分析

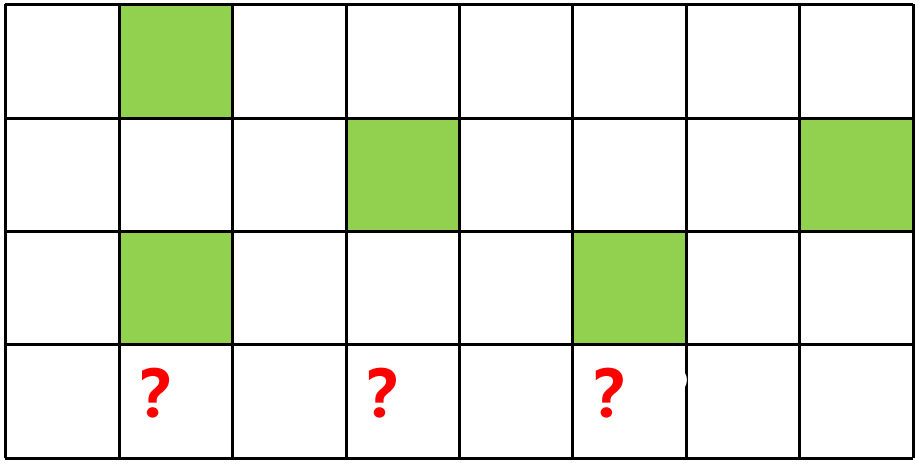
这种 棋盘放置类 问题，在没有事先知道一些特定性质的情况下来做，都会想到爆搜。本题的数据规模，也是向着爆搜去设置的。如果我们直接爆搜，则时间复杂度为 2^{n^2} ，会超时。

分析

考虑一下如何进行动态规划。由于在第 i 层放置国王的行为，受到i-1 层和 i+1 层以及 i 层的状态影响。那么我们就可以规定从上往下枚举的顺序，这样考虑第 i 层状态时，只需考虑 i-1 层的状态即可。于是我们可以考虑把层数 i 作为动态规划的 阶段 进行 线性DP。

而第 i 阶段需要记录的就是前i-1层放置了的国王数量 j，以及在第 i 层的 棋盘状态 k。

状态表示



i

01000000

00010001

01000100

state ?

f[i][j][s] 表示前i行共摆了j个国王，第i行的状态为s的所有方案数

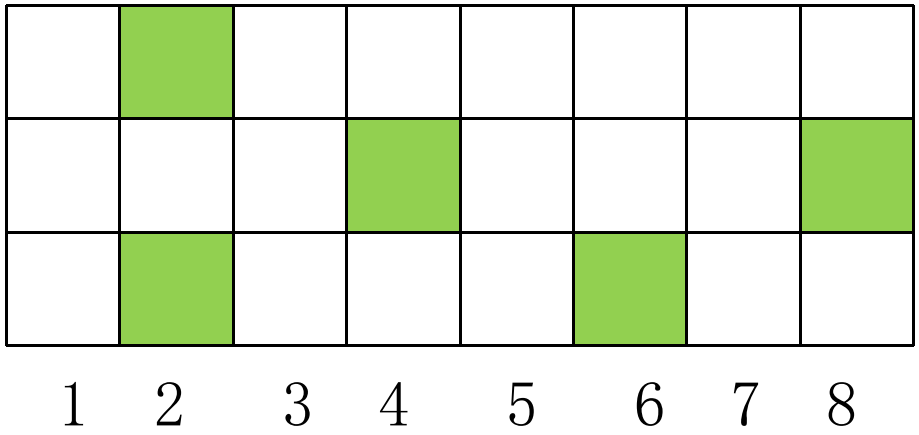
如f[3][5][68]表示?

用二进制表示该状态，就是 $(01000100)_2 = (68)_{10}$

状态表示

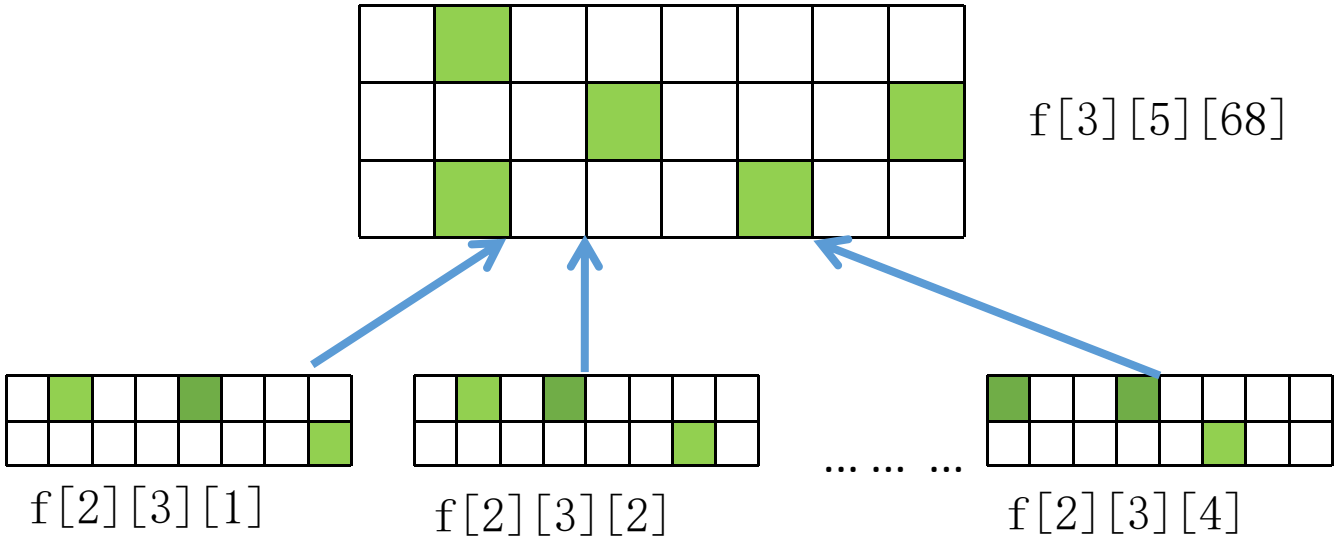
设f[i][j][s] 表示前i行共摆了j个国王，第i行的状态为s的所有方案数。

f[3][5][34]表示前3行共放了5个国王，第三的状态为（00100010）如下图所示为：



状态转移

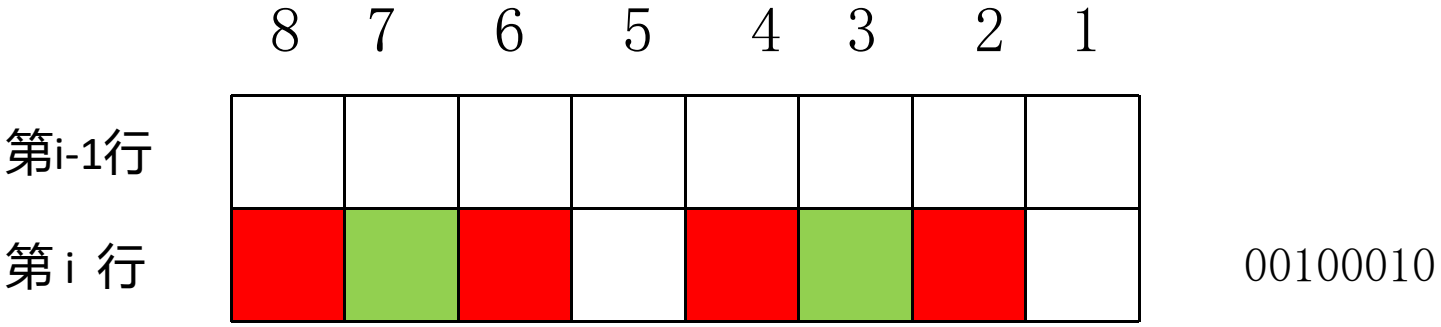
因为第i行的状态已经确定，所以第i行的国王数和国王的位置已经确定：



$$f[i][j][s]=\sum f[i-1][j-count(s)][b]$$

f[i-1][j-count[s][b] 已经摆完了前i-1排，并且第i-1排的状态是b,共摆了了j-count(s)个国王的所有方案。

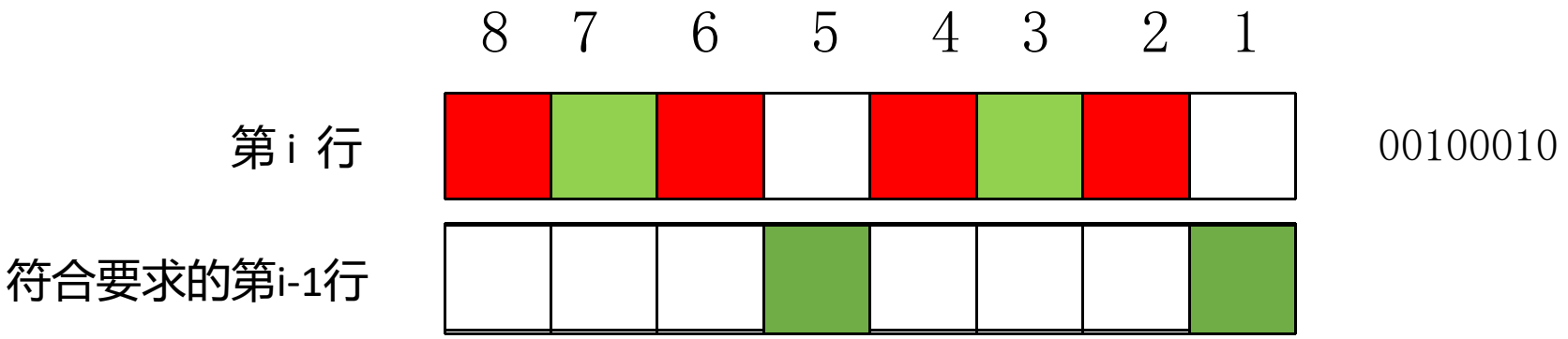
状态转移



第i-1层的状态b可选择状态空间为2^8种状态，但这些状态与第i层不能相互攻击到，同时第i-1层自身也要符合要求，即要满足：

1. 第i-1层内不能有相邻的两个1
2. 第i-1层不能与第i层相互攻击到

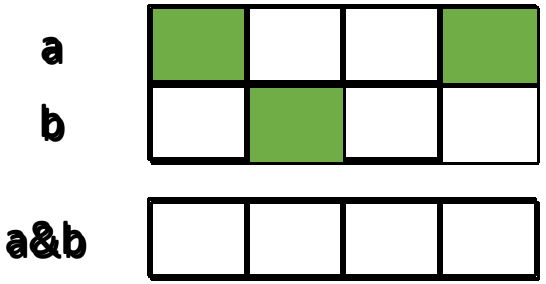
状态转移

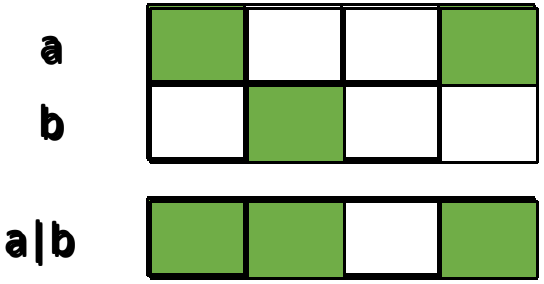


a表示第i-1行状态，b表示第i行状态，则

(a&b)==0　表示上下行国王不再同一列

(a|b)　　表示上下两行哪些位置相邻的国王(a|b不能相邻的1)





复杂度分析

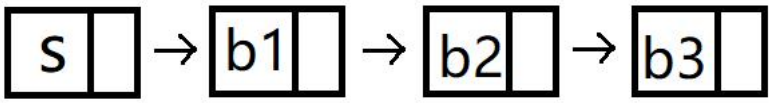
状态数量*每一个状态转移需要的计算量

行数 i<=10
国王数量 j<=100
状态数<=2^{10}=1000（实际符合要求的状态不多）
状态转移的计算量：1000

f[i][j][s]

源代码1

```
2  #include <vector>
3  #include <cstring>
4  using namespace std;
5
6  typedef long long LL;
7  const int N = 12, S = (1 << N);
8
9  LL f[N][N * N][S];// f[i][j][s] 第i行已经，共摆j个国王，i状态是s
10 int n, m;
11 int cnt[S];  // 每个状态里面1的个数
12 vector<int> state;  // 状态集
13 vector<int> head[S]; // 每个状态可以转移到的其他状态，相当于链表
14
```



源代码2

```
17  //判断状态是否合法，检查该状态是否存在连续的两个1
18  bool check(int state) {
19      for (int i = 0; i < n; i++)
20          //两位都是1
21          if ((state >> i & 1) && (state >> i + 1 & 1))
22              return false;
23      return true;
24  }
25  // 存在连续一返回false
26  bool check1(int x){
27      return !(x & (x >> 1));
28      // 如果存在连续1，那么x & x >> 1必定等于全1（所有位）
29      //取反的话必定等于0
30      // 如果不存在连续1的话，移一位刚好错开，
31      //x & x >> 1必定为0,取反为1
32  }
```

源代码3

```
34  // 统计状态x的国王数
35  int count(int x){
36      int ans = 0;
37      for(int i = 0; i < n; i ++)
38          ans += (x >> i & 1);
39      return ans;
40  }
41
42  int main(){
43      cin >> n >> m;
44
45      // 预处理阶段1：同行合法化
46      for(int i = 0; i <(1 << n); i ++){
47          if(check(i)){  // 是否存在连续1，如果不存在的话
48              state.push_back(i);  // i合法
49              cnt[i] = count(i);
50              // 统计一下这个状态每一行有多少国王
51          }
52      }
```

源代码4

```
54      // 预处理阶段2：相邻行合法化
55      // 枚举所有合法状态，预处理处可以转移的状态
56      for(int i = 0; i < state.size(); i ++){
57          for(int j = 0; j < state.size(); j ++){
58              int a = state[i];
59              int b = state[j];
60              // 相邻行不能在同一个位置有国王，并且不能在攻击范围有
61              if((a & b) == 0 && check(a | b))
62                  head[i].push_back(j);
63                  // 如果这两个状态可以满足条件即可以做邻居
64          }
65      }
```

源代码5

```
67      f[0][0][0] = 1;  // 方案数为1
68      // dp过程
69      for(int i = 1; i <= n + 1; i ++)  // 枚举n行
70          for(int j = 0; j <= m; j ++)  // 枚举国王数
71              for(int a = 0; a < state.size(); a ++)  // 枚举所有行合法状态
72                  for(int b = 0; b < head[a].size(); b ++){
73                  // 枚举所有行相邻合法状态,只有行相邻合法才能从前一行转移过来
74                      int c = cnt[state[a]];  // 第i行的所有行合法状态国王数
75                      if(j >= c)  // 可以继续放国王
76                          f[i][j][a] += f[i - 1][j - c][head[a][b]];
77                          // 从i-1行转移过来
78                  }
79      // n + 1行什么都不放，相当于只在1~n行放国王
80      //当处理n+1行是，表明n行已经处理完
81      cout << f[n + 1][m][0] << endl;
82      return 0;
```

数组写法

```
#include <iostream>
using namespace std;

typedef long long LL;
const int N = 12, M = 1 << 10, K = 110;

int n, m;
LL f[N][K][M]; // f[i][j][s] 第i行已经，共摆j个国王，i状态是s
bool st[M];// 存储所有合法状态集
bool mv[M][M];// // 每个状态可以转移到的其他状态
int cnt[M];// 每个状态里面1的个数

//判断状态是否合法，检查该状态是否存在连续的两个1
bool check(int state) {
    for (int i = 0; i < n; i++)
        //两位都是1
        if ((state >> i & 1) && (state >> i + 1 & 1))
            return false;
    return true;
}
```

```
22  int count(int state) {
23      int res = 0;//存储该状态1的个数
24      for (int i = 0; i < n; i++)
25          res += (state >> i & 1);
26      return res;
27  }
28
29  int main() {
30      cin >> n >> m;
31
32      for (int i = 0; i < 1 << n; i++) {
33          if (check(i)) st[i] = true;//如(110)2 不合法
34          if (st[i]) {
35              cnt[i] = count(i);// 统计一下这个状态每一行有多少国王
36              for (int j = 0; j < 1 << n; j++) {
37                  // 状态合法   同一列没有1
38                  if (check(j) && (i & j) == 0 && check(i | j))
39                      mv[i][j] = mv[j][i] = true;//状态i,j之间可以相互转换
40              }
41          }
42      }
43
```

```
44      f[0][0][0] = 1;
45      for (int i = 1; i <= n + 1; i++)
46          for (int j = 0; j <= m; j++)//国王数量
47              for (int k = 0; k < 1 << n; k++) {//枚举所有状态
48                  if (st[k]) {//如果状态合法
49                      for (int s = 0; s < 1 << n; s++) {
50                       //状态s合法，且s->k是可转移的，并且 国王数量不越界
51                          if (st[s] && mv[s][k] && j - cnt[k] >= cnt[s])
52                              f[i][j][k] += f[i - 1][j - cnt[k]][s];
53                      }
54                  }
55              }
56
57      cout << f[n + 1][m][0] << endl;
58      return 0;
59  }
```

3 [7284] 蒙德里安的梦想

求把N*M的棋盘分割成若干个1*2的的长方形，有多少种方案。例如当N=2，M=4时，共有5种方案。当N=2，M=3时，共有3种方案。

输入包含多组测试用例。每组测试用例占一行，包含两个整数N和M。当输入用例N=0，M=0时，表示输入终止，且该用例无需处理。

输出每个测试用例输出一个结果，每个结果占一行。

数据范围1≤N,M≤11

分析

核心：
先放横着的，再放竖着的。
总方案数，等于只放横着的小方块的合法方案数。
如何判断，当前方案是否合法?所有剩余位置，能否填充满竖着的小方块。
可以按列来看，每一列内部所有连续的空着的小方块，需要是偶数个。

状态表示：f[i,j]表示已经将前i-1列摆好，且从第i-1列，伸出到第i列的的状态为j所有方案。

传统集合动态规划

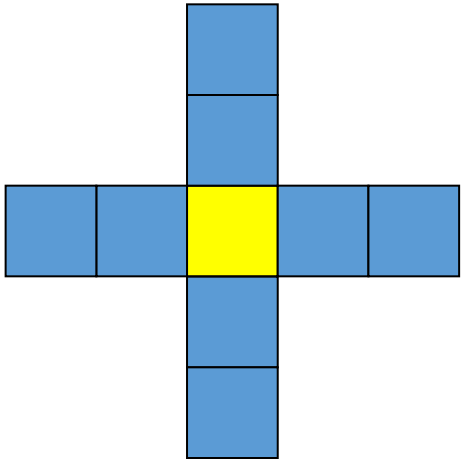
- 例题一：
- 给定n个点的有向带权图，求一条经过每个点一次的回路，并要求权和最小。
- 范围n<=15。

传统集合动态规划

- 显然对于某一个中间状态，影响它的最后结果的仅仅是当前所在点以及之前已经经过的点。而之前的路径行走情况与之后的解无关。
- 状态F[i,opt]，i表示当前所在点，opt是用2进制记录每个点是否已经经过。

传统集合动态规划

- 例题二：炮兵阵地 （NOI2001）
- 在N*M网格地图上部署炮兵部队。每个炮兵可以控制横纵2格范围。任意一对炮兵互相不能处于控制范围。
- 地图上有些点不能部署部队。
- N <= 100； M <= 10。



传统集合动态规划

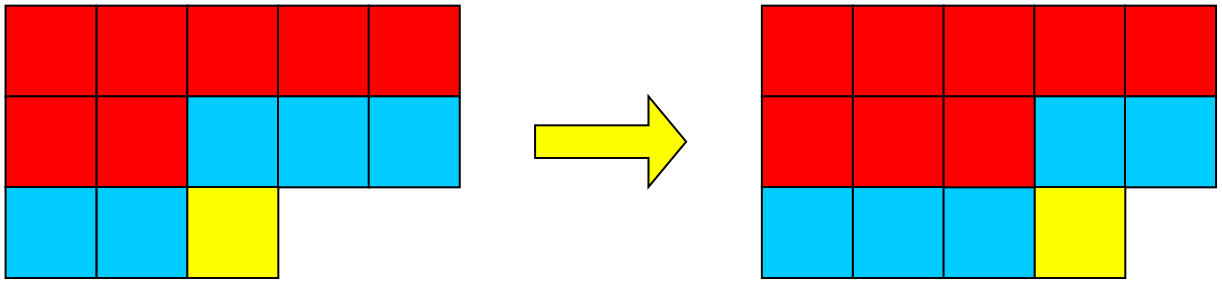
- 例题三：K-排列问题
- 考虑一个1~n的排列a[1],a[2],a[3]…a[n]，若max(abs(a[i]-i))=K，那么这个排列就称为K-排列。
- 求n个数的K-排列的个数。
- 范围： n<=100, K<=5

传统集合动态规划

- 例题四：生成树计数（NOI2007）
- 环状图，任意两个点距离不超过k则连边，求生成树个数。
- K<=5

实现

- 插头法



- 转移的复杂度降低
- 时间复杂度降低

传统集合动态规划

- 例题 Another Chocolate Maniac (Sgu132)
- 给定一个M*N的网格，网格中存在一些障碍物。在网格空地处放置最少的1*2的矩形块，使得网格中无法再放入1*2的矩形块。
- 1<=M<=70
- 1<=N<=7

基于连通性的状态压缩动态规划

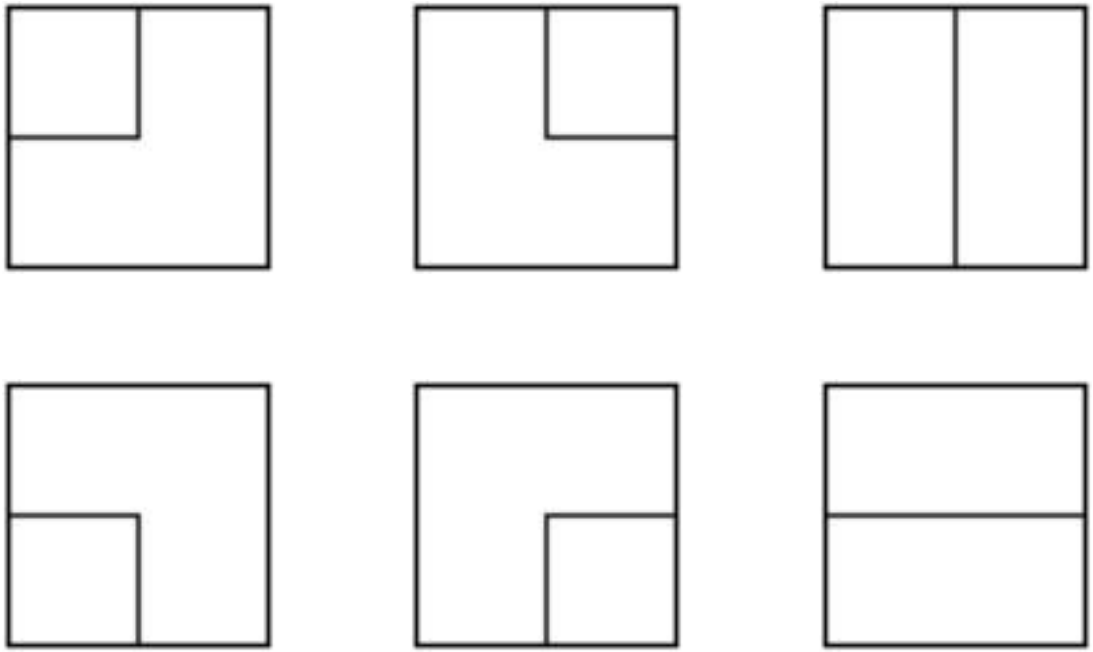
- 在网格中寻找一条或多条路径（回路）满足一定的条件，求方案数或路径总长度最短。
- 状态除了记录路径“出口”，还要记录其连通性。

基于连通性的状态压缩动态规划

- 例题一：Formula 1 （Ural 1519）
- 给你一个m * n的棋盘，有的格子是障碍，问共有多少条回路使得经过每个非障碍格子恰好一次.
- m, n ≤ 12.

基于连通性的状态压缩动态规划

- 思想：状态压缩动态规划。
- 一个单元格中可能出现的路径情况：

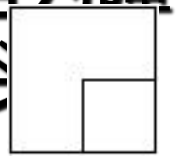


实现细节

- 总体实现：插头法
- 实现方法：记忆化搜索
- F[i,j,opt]表示当前是i行j列，最后扫描的总共m个格子的状态为opt的方案数。
- Opt的记录：m个格子向下伸出插头的情况，以及最后一个格子向右伸出插头的情况。
- 插头记录：0表示无插头，具体数字表示插头的属性（染色法记录属于第几个连通块，最小表示）。对于本题最多同时存在6个连通块的插头。

实现细节

- 转移：分类讨论插头方向。
- 1、当前格上方左方均有插头：只能将这两个连通块连接。（1种）
- 2、当前格只有上方有插头：将这个插头向下向右延伸。（2种）
- 3、当前格只有左方有插头：将这个插头向下向右延伸。（2种）
- 4、当前格周围无插头：若当前格为障碍物，则无插头，否则插入一个折线形插头



实现细节

- 合并连通块：
- 对于第一种情况，需要合并连通块。若不加限制，则会计算出包含多条回路的情况。
- 限制：和并连通块时，若两个插头属于同一个连通块，则当且仅当在最后一个有效格子中可以将这两个插头连接。
- 最后统计：计算到最后一个有效格子时，需要统计答案。此时，要保证当前状态没有剩余的插头。

实现细节

- 一些可能存在的问题：
- 直接开数组用序列记录插头好还是把状态压缩后记录好？
- 最小表示的如何实现？
- 如何减小常数？

实现细节

- 一个优化：
- 若当前格子连出的插头指向一个障碍物格子，可以直接剪枝。
- 对有障碍的情况，能减少很多无效状态。

基于连通性的状态压缩动态规划

- 例题二：Manhattan Writing（Japan2006）
- n*m网格有一些障碍，要求把两个2和两个3分别用折线连起来，总长度尽量小。
- n,m<=9

基于连通性的状态压缩动态规划

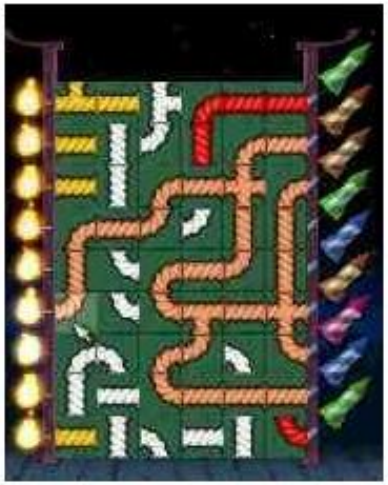
- 主体思想与之前相同。
- 不同点：
- 需要专门两种属性记录与数2和数3的连通插头。
- 无需考虑多余回路情况。（解肯定劣）

状态压缩动态规划中的剪枝

- 状态数是指数级别。
- 最小表示以减少状态数。
- 根据题目尽可能早地删去不可能成为最优的状态或不可行的状态。

状态压缩动态规划中的剪枝

- 例题：中国烟花 （zju2125）
- 给你一个9 * 6的棋盘，棋盘的左边有9根火柴，右边有9个火箭．棋盘中的每一个格子可能是一个空格子也可能是一段管道，管道的类型有4种：L型，一型，T型，十型。
- 给定棋盘的初始状态以及X，你的目标是旋转每个格子内的管道0，90，180或270度，使得当点燃左边第X根火柴后，被发射的火箭个数尽可能多。



状态压缩动态规划中的剪枝

- 状态：按照从左到右，从上到下的顺序依次考虑每一个格子，记录每个插头是否已经点燃以及它们之间的连通情况。
- 状态为：F[i,j,opt,fired]表示转移完(i, j)，最后扫描的总共10个插头的连通性为opt(把每个插头是否存在记录在opt中)，10个插头是否被点燃的2进制数fired的状态能否达到。

状态压缩动态规划中的剪枝

- 转移：依次枚举每一个格子的旋转方式(最多4种)，根据当前格子是否可以与上面的格子和左边的格子通过插头连接起来分情况讨论。
- 状态数太多，直接做TLE。
- 怎么办？

状态压缩动态规划中的剪枝

- 剪枝一：如果当前状态所有的插头全部都未被点燃，那么最后所有的火箭都不可能发射，所以这个状态可以舍去。一个显然的剪枝，却能剪掉近乎一半的状态。
- 剪枝二：如果轮廓线上有一个插头p，它没有被火柴点燃且没有其它的插头与它连通，那么这个插头可以认为是“无效”插头。这个状态可以剪枝。（必然存在不存在这个“无效”插头的状态）

状态压缩动态规划中的剪枝

- 剪枝三：对于一个格子(i, j)的两个状态(opt1,fired1)(opt2,fired2)，如果第一个状态的每一个存在的插头在第二个状态中不仅存在而且都被点燃，那么第一个状态可以剪枝。（一个状态必然不比第二个状态优）
- 剪枝四：边界状态，判断无效状态并剪枝。

练习

- Betsy的旅行（USACO）
- 给定一个N*N的网格（N<=9），要求从左上角的格子走到左下角的格子，并且经过所有格子恰好一次。
- 求Betsy能采用的路径方案数

练习

- Youth Hostel Dorm（Nwerc2007）
- 在一个n*m的网格旅馆中，边界上有唯一的门。除了门之外，其余的格子要么是空地，要么是床。一张床可访问到，当且仅当他相邻格子中有一个空地格子所在连通块是门所在的连通块，
- 问最多能访问到多少张床。
- 范围： n,m<=8

练习

- The Floor Bricks （Pku2285）
- 给定一个5行n列的空间，要求填入给定的m种方块（大小均不超过3*3）。
- 求最少需要多少个方块。

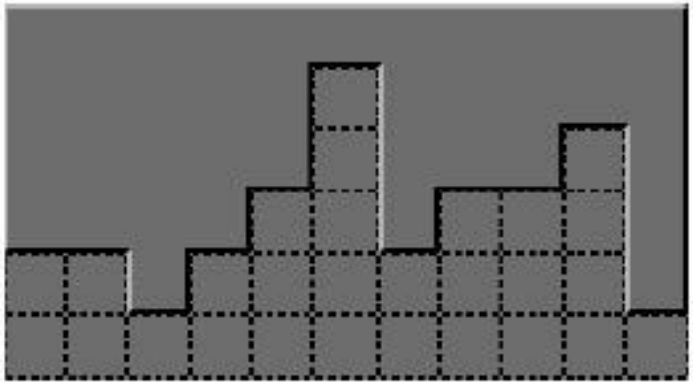


Fig.2

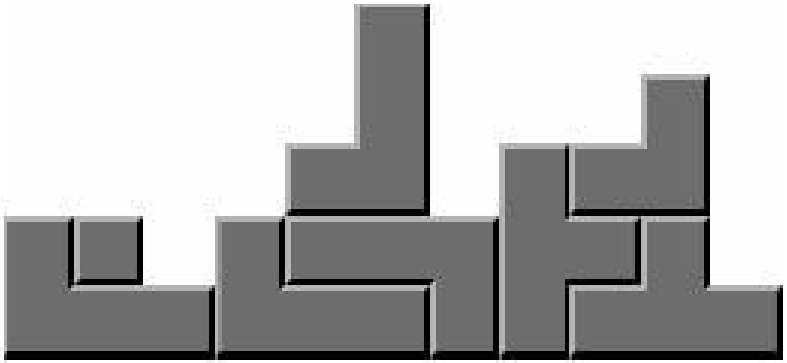


Fig.3