状压DP

- 状压 DP 是动态规划的一种，通过将状态压缩为整数来达到优化转移的目的.

引入:有一个长度为 n 的序列 A,将其重排为 B,最大化 $\sum_{i=1}^{n} i * B_i$ 并输出,$n \leq 20$.

- 贪心可做,从小到大排序即可,且 n 可以出得更大.
- 以下是状态压缩的思路，我们假设目前已经排好了前 i 个位置,这 i 个数字构成了一个集合,我们可以利用一个二进制数 s 表示这个集合.若 i 在 s 这个集合中,那么 s 对应的二进制下第 i 位为 1，否则为 0.
- 举个例子 $s = 00110010$,那么表示第 $2, 5, 6$ 个数字在一个集合里.
- 根据 dp 的思想,我们定义 $f_{i,s}$ 表示为由集合 s 构成前 i 个数字所得到的最大值,我们最终要求的答案是 $f_{n,S}$,其中 S 表示全集.
- 接下来考虑状态转移，从 $f_{i,s}$ 到 $f_{i+1,s'}$ 这个过程，即`f[i+1][s ^ (1<<k)] = max(f[i+1][s ^ (1<<(k-1))],f[i][s] + (i+1) * A[k])`.

```
1  n = read();
2  for(int i=1;i<=n;i++) A[i] = read();
3  for(int i=1;i<=n;i++){
4      for(int s=0;s<(1<<n);s++){
5          for(int k=1;k<=n;k++){
6              if(s&(1<<(k-1))) continue;
7              f[i][s ^ (1<<(k-1))] = max(f[i][s ^ (1<<(k-1))],f[i-1][s] + i *
   A[k]);
8          }
9      }
10 }
11 printf("%d\n",f[n][(1<<n)-1]);
12 return 0;
```

代码实现如上，注意这里第 6 行要保证 k 不在 s 这个区间里.

现在进入正题

P3694 邦邦的大合唱站队

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  const int maxn = 1e5+20;
4  const int maxm = 2e6+20;
5  int n,m;
6  int sum[maxn][30],dp[maxm],s[maxm];
7  int main()
8  {
9      scanf("%d%d",&n,&m);
10     for(int i=1;i<=n;i++)
11     {
12         int x;scanf("%d",&x);
13         for(int j=1;j<=m;j++)   sum[i][j] = sum[i-1][j];
14         sum[i][x]++;
15     }
```

```
    for(int i=0;i<(1<<m);i++)
        for(int j=0;j<m;j++)
            if((1<<j)&i) s[i] += sum[n][j+1];
    memset(dp,0x3f,sizeof(dp));
    dp[0] = 0;
    for(int i=0;i<(1<<m);i++)
    {
        for(int j=0;j<m;j++)
        {
            if((1<<j)&i)
            {
                int res = (1<<j)^i ;
                dp[i] = min(dp[i],dp[res] + sum[n][j+1] - (sum[s[i]][j+1] - sum[s[i]-sum[n][j+1]][j+1]));
            }
        }
    }
    printf("%d\n",dp[(1<<m)-1]);
    return 0;
}
```

[P1879 [USACO06NOV] Corn Fields G]

题意:给一个 n 行 m 列的 $0/1$ 矩阵,0 表示不可以涂色,1 表示可以,要求不能有相邻的两个色块都涂色,问最多能给多少个位置涂色,$m,n \leq 12$.

我们先把问题简化,不考虑行之间的相邻涂色,对于每行处理出来不相邻的所有情况,这种合法情况可以用 m 位二进制存储下来,1 表示对应位置涂色,0 表示不涂色.

之后便把问题轻松地化为一维，这就要求每相邻两行的涂色方案对应的二进制数不存在同一位都为 1,即 1001 和 0011 是不合法的.

设计状态为 $f_{i,s}$ 表示第 i 行涂色方案为 s 时的方案数,转移时枚举上一行的状态为 s',检查 $s\&s'$ 是否为 0 即可.

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 5020;
const int mod  = 100000000;
int n,m,ans,tot;
int s[maxn],dp[15][maxn],mp[15][15];

bool judge(int x,int t)
{
    for(int i=0;i<m;i++)
    {
        if(mp[x][i+1]==0&&(t&(1<<i)))
            return true;
    }
    return false;
}

int main()
{
```

```
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            scanf("%d",&mp[i][j]);
    for(int i=0;i<(1<<m);i++)
    {
        if((i&(i<<1)))  continue;
        s[++tot] = i;
        if(!judge(1,i)) dp[1][i] = 1;
    }
    for(int i=2;i<=n;i++)
    {
        for(int j=1;j<=tot;j++)
        {
            if(judge(i,s[j])) continue;
            for(int k=1;k<=tot;k++)
            {
                if(judge(i-1,s[k])) continue;
                if((s[j]&s[k])) continue;
                dp[i][s[j]] = (dp[i][s[j]] + dp[i-1][s[k]]) % mod;
            }
        }
    }
    for(int i=1;i<=tot;i++) ans = (ans + dp[n][s[i]]) % mod;
    printf("%d\n",ans);
    return 0;
}
```

相似的题目 P2704 [NOI2001] 炮兵阵地

处理思路基本相同，把一行的状态压缩成一个二进制数，需要记录上面两行的状态才能转移.

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int maxn = 120;
int n,m,tot,cnt;
int dp[maxn][maxn][maxn],vis[maxn],t[maxn];

int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        char temp[12];
        scanf("%s",temp+1);
        for(int j=1;j<=m;j++)
        {
            if(temp[j]=='H')
                vis[i] += (1<<(j-1));
        }
    }
    for(int i=0;i<(1<<m);i++)
    {
```

```
        if(((i>>2)&i)||((i>>1)&i))
            continue;
        t[++tot] = i;
    }
    for(int i=1;i<=n+2;i++)
    {
        for(int j=1;j<=tot;j++)
        {
            for(int k=1;k<=tot;k++)
            {
                for(int ii=1;ii<=tot;ii++)
                {
                    if((t[ii]&t[j])||(t[ii]&t[k])||(t[j]&t[k])||
(vis[i]&t[j]))
                        continue;
                    dp[i][j][k] = max(dp[i][j][k],dp[i-1][k][ii] +
__builtin_popcount(t[j]));
                }
            }
        }
    }
    printf("%d\n",dp[n+2][1][1]);
    return 0;
}
```