DP

区间dp

```
#include <bits/stdc++.h>
#define LOCAL
//#define int long long
//每次合并相邻两堆
using namespace std;
const int N=1e3+5;
int T,n,a[N],sum[N],dp[N][N];
signed main(){
#ifdef LOCAL
    freopen("data.in","r",stdin);
    freopen("data.out","w",stdout);
#endif
cin>>n;
for(int i=1;i<=n;i++){
    cin>>a[i];
}
sum[1]=a[1];
for(int i=2;i<=n;i++){//前缀和求区间和 （i,j）区间  这堆石子的数量
    sum[i]=sum[i-1]+a[i];
}
memset(dp,0,sizeof dp);//  dp[i][i]  长度为1区间默认0  （）不用花费
//  区间dp  dp[i][j]  表示(i,j)区间 最小值 枚举k  dp[i][j]=min( dp[i][k], (dp[k+1]
[j])  +w)//长度较短的区间已生成过
for(int len=2;len<=n;len++){//长度》=2
    for(int i=1;i+len-1<=n;i++){//枚举左端点
        int j=i+len-1;      //  右端点
        dp[i][j]=0x3f3f3f3f;
        for(int k=i;k<j;k++){// 枚举分界点k
            dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+sum[j]-sum[i-1]);
        }

    }
}
cout<<dp[1][n]<<endl;


    return 0;
}
```

树形dp

```
#include <bits/stdc++.h>
#define LOCAL
using namespace std;
//没有上司的舞会
int t,n,a[6005],in[6005];//in入度  a开心值
vector<int>G[6005];
int dp[6005][2];//dp[i][1/0]第i个点去或不去  子树的快乐总值
```

```
void dfs(int u){//dp     DAG图无需标记
  for(int j=0;j<G[u].size();j++){
      int v=G[u][j];
     dfs(v);//递归到叶子结点
     }
  //*****回溯时dp***核心     （从叶子开始）
for(int j=0;j<G[u].size();j++){
  int v=G[u][j];
  dp[u][0]+=max(dp[v][0],dp[v][1]);
  dp[u][1]+=dp[v][0];
}
}
int main(){
while(cin>>n,n){
memset(a,0,sizeof a);
memset(dp,0,sizeof dp);
memset(in,0,sizeof in);
for(int i=1;i<=n;i++){
  cin>>a[i];
  dp[i][1]=a[i];//初值直接装
   G[i].clear();
}
 int u,v;
for(int i=1;i<n;i++){//肯定n-1
  cin>>u>>v;
  G[v].push_back(u);//v->u 注意
  in[u]++;
}
cin>>u>>v;//两个0
int root=0;
for(int i=1;i<=n;i++){
  if(in[i]==0){//入度为0   根结点
    dfs(i);
    root=i;
    break;
  }
}
int ans=max(dp[root][1],dp[root][0]);
cout<<ans<<endl;
}
    return 0;
}
```

数位dp

原理

利用记忆化搜索，从高位到低位，满足条件跳出返回1，否则返回0

用limit参数控制上界 只有没有限制的状态才转移

dp[pos] [state] 记录， 转移 pos 为搜到的位数（从大到小）， state 储存状态 可多维

模板

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int b[N];                                          // 每一位
int dp[N][state];                                  // dp 数组根据实际情况决定  state  可以对应dfs参数
int dfs(int pos, int state, bool limit) {  //位置(从高位到低位)    状态state
                                                   //可多个   limit:是否达到上界_有限制 : 前面每一位都达到上界  （与n的前                                                                相同）
这样后面每一位都限制上界  不然可以取遍1~9
  if (pos == 0)
     if()  return 1;  //   满足条件为1  否则9
     else  return 0;
    //  是上界则则下一位只能到 b[pos]  否则能到9
  if (!limit && ~dp[pos][state]) return dp[pos][state];
  int end = limit ? b[pos] : 9;  //该位最大值
  int ans = 0;
  for (int i = 0; i <= end; i++) {
    if ()  // 满足某种条件
      ans += dfs(pos - 1, state, limit && i == end);//   is_max每位都为最高位
  }
  if (!limit) dp[pos][state] = ans;  //没有限制才转移（1~9）  不然无法准确判断
  return ans;

    LL solve(LL n)
{
    int cnt=0;
    while(n)
    {
        b[cnt++]=n%10;
        n/=10;
    }
    return dfs(cnt-1,-1,false,true);
}
```

例题

1.模板题：1~n 中含49 的数的个数

```
#include <bits/stdc++.h>
//# pragma GCC optimize(3)
#define int long long
#define endl "\n"
using namespace std;
//1~n 所有数中含有49的数字有多少个
//  到着做  算不含的
const int N = 2e5 + 5;
int T, n, dp[30][2];  //dp[i][0/1]  表示1~i位 的满足不含49个数（无限制（1~i=10^i-1））第二维表示该位是否位为4
int b[30];// 每一位数字
int dfs(int pos, bool is_4, bool limit) {  //位置(从高位到低位)    状态state
is_4上一位是4(也可直接传数)
```

```
                                                    //可多个    limit:是否达到上界_有限制 ：前面每一位都达到上界  （与n的前几位相同）    这样后面每一位都限制上界  不然可以取遍1~9
  if (pos == 0)
    return 1;  //把0视为一种情况（基数） 最后减去  能到0位的都算一种
  if (!limit && ~dp[pos][is_4]) return dp[pos][is_4]; //  没有限制才转移（1~9）
  int end = limit ? b[pos] : 9;  //该位最大值
  int ans = 0;
  for (int i = 0; i <= end; i++) {//枚举pos-1位(下一位)
    if (!(is_4&&i==9))  // 满足某种条件 （不含49）
      ans += dfs(pos - 1, i==4, limit && i == end);//   is_max每位都为最高位
//如果是*49****  则 算到9****后再
  }
  if (!limit) dp[pos][is_4] = ans;  //没有限制才转移（1~9）  不然无法准确判断
  return ans;
}
void solve(){
    cin>>n;
    memset(dp,-1,sizeof dp);
    int k=1;
    int temp=n;
    while(temp)
    {
        b[k++]=temp%10;
        temp/=10;
    }
    k--;
    int ans=n-dfs(k,false,true)+1;// //减去的时候多减了一个0，要把它加上
    cout<<ans<<endl;
}
signed main() {
  std::ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

  cin >> T;
  while (T--) {
      solve();
  }
  return 0;
}

//正着做 需要多一维state ：因为只要前面有49出现就会向后传递


//   dp[pos][pre][state]为第pos位上，前一个数为pre，目前状态为state（所枚举的这个数字是否含有相邻的49）时的数字数量。
//需要多一维state ：因为只要前面有49出现就会向后传递
// 要额外考虑之前是否含49
LL dp[25][10][2];
int b[25];
LL dfs(int pos,int pre,bool state,bool limit)
{
    if(pos==-1)
        return state==1;  //满足条件返回1
    if(!limit&&dp[pos][pre][state]!=-1)
        return dp[pos][pre][state];
    int up=limit?b[pos]:9;
    LL ans=0;
```

```
    for(int i=0;i<=up;i++)
    {
        ans+=dfs(pos-1,i,state||i==9&&pre==4,limit&&i==b[pos]); //或    当前是或之前是都要算上
    }
    if(!limit)
        dp[pos][pre][state]=ans;
    return ans;
}

LL solve(LL n)
{
    int cnt=0;
    while(n)
    {
        b[cnt++]=n%10;
        n/=10;
    }
    return dfs(cnt-1,-1,false,true);
}
```

2.模板题2： 两个条件

hdu3652

题意：求<=n，满足包含13且能被13整除的数的个数

```
#include <bits/stdc++.h>
//# pragma GCC optimize(3)
#define int long long
#define endl "\n"
using namespace std;
//题意：求<=n，满足包含13且能被13整除的数的个数//
const int N = 2e5 + 5;
int T, n, dp[30][15][10][2];
int b[30];// 每一位数字
int dfs(int pos, int mod,int pre, bool state,bool limit) {  //位置(从高位到低位)  从0到该位模13结果   上一位   到该位是否含13   限制
                                                            //可多个    limit:是否达到上界_有限制 ： 前面每一位都达到上界  （与n的前几位相同）   这样后面每一位都限制上界  不然可以取遍1~9
  if (pos == 0){//最后一位跳出
        if(state&&mod==0)  return 1;  //满足条件
      //同时满足两个跳出条件
        else return 0;
  }
  if (!limit && ~dp[pos][mod][pre][state]) return dp[pos][mod][pre][state]; //没有限制才转移（1~9）
  int end = limit ? b[pos] : 9;  //该位最大值
  int ans = 0;
  for (int i = 0; i <= end; i++) {//枚举pos-1位(下一位)
      ans += dfs(pos - 1, (mod*10+i)%13, i,state||(pre==1&&i==3),limit && i == end);//   is_max每位都为最高位
//如果是*49****  则 算到9****后再
```

```
  }
  if (!limit)  dp[pos][mod][pre][state] = ans;  //没有限制才转移（1~9）  不然无法准确判断
  return ans;
}

signed main() {
  std::ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  while (cin>>n) {
    memset(dp,-1,sizeof dp);
    int k=1;
    int temp=n;
    //太大可以读字符串
    while(temp)
    {
        b[k++]=temp%10;
        temp/=10;
    }
    k--;
    int ans=dfs(k,0,0,false,true);// //减去的时候多减了一个0，要把它加上
    cout<<ans<<endl;  }
  return 0;
}
```

3.三个条件

题意：[l,r]

```
//    如果一个整数符合下面3个条件之一，那么我们就说这个整数和7有关——
//     1、整数中某一位是7；
//     2、整数的每一位加起来的和是7的整数倍；
//     3、这个整数是7的整数倍；
//     现在问题来了：吉哥想知道在一定区间内和7无关的数字的平方和
```

```
#include <bits/stdc++.h>
//# pragma GCC optimize(3)
#define int long long
#define endl "\n"
using namespace std;
//倒着  算满足的
const int N = 2e5 + 5;
int T, n, dp[30][10][10][2]; // dp[pos][mod1][mod2][state][2]  位置(从高位到低位)到该位每一位求和对7取模   之前组成的数模  到该位是否含7
int b[30];// 每一位数字
int dfs(int pos, int mod1,int mod2, bool state,bool limit) {  //位置(从高位到低位)到该位每一位求和对7取模   之前组成的数模  到该位是否含7  限制

  if (pos == 0){//最后一位跳出
        if(state||mod1==0||mod2==0)  return 1;  //满足条件
        else return 0;
```

```
  }
  if (!limit && ~dp[pos][mod1][mod2][state]) return dp[pos][mod1][mod2][state];
//  没有限制才转移（1~9）
  int end = limit ? b[pos] : 9;  //该位最大值
  int ans = 0;
  for (int i = 0; i <= end; i++) {//枚举pos-1位(下一位)
      ans += dfs(pos - 1,(mod1+i)%7, (mod2*10+i)%7,state||(i==7),limit && (i == end));//   is_max每位都为最高位
//如果是*49****  则 算到9****后再
  }
  if (!limit)  dp[pos][mod1][mod2][state] = ans;  //没有限制才转移（1~9）  不然无法准确判断
  return ans;
}
int cal(int n){
    memset(dp,-1,sizeof dp);
    int k=1;
    int temp=n;
    //太大可以读字符串
    while(temp)
    {
        b[k++]=temp%10;
        temp/=10;
    }
    k--;
    return dfs(k,0,0,false,true);// //减去的时候多减了一个0，要把它加上
}



signed main() {
  std::ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);
  cin>>T;
  int l,r;
  while (T--) {
    cin>>l>>r;
    int ans=cal(r)-cal(l-1);
    //ans=(r-l+1)-ans;
    cout<<ans<<endl;
    }
  return 0;
}
```

状压dp

实质：将有限状态数用二进制数表示

1.常用的二进制计算方法

一、取出x的第i位：y = (x>> (i-1)) & 1

二、将x的第i位取反：x = x ^ (1<< (i-1))

三、将x的第i位变为1： x = x | (1<< (i-1))

四、将x的第i位变成0： x = x & ~ （ 1<< (i-1))

五、将x最靠右的1变成0： x = x & (x-1)

六、取出最靠右的1 ： y=x&(-x)

七、把最靠右的0变成1: x | = (x-1)

2.例题

例1、关灯问题

分析:

考虑状态压缩，可以把灯的开和关视作1和0，则用一串01串（二进制）表示这一串灯的一个总的状态。

那么这题就可以直接广搜暴利解决，利用之前的计算方法，开灯（1）就是把对应的那一位0变成1： x|=1<<(i-1),如果本身是1的话当然没有任何影响。
同理，关灯（-1）的话就是把对应的那一位1变成0： x=x&~(1<<(i-1)),当然如果本身是0 ，也没有影响啦。

代码

```
#include <bits/stdc++.h>
//# pragma GCC optimize(3)
#define int long long
#define endl "\n"
using namespace std;
//10维数组  用一个长度为10 的二进制数表示10个开关 （一位dp数组）
// bfs求最短路即可
int dp[2000];// 存答案
const int N = 2e3 + 5;
int T, n,m, a[N][N]; // a[i][j]表示第i个开关对第j个灯的效果
int vis[N];
void solve(){

cin>>n>>m;
for(int i=1;i<=m;i++){
    for(int j=1;j<=n;j++){
        cin>>a[i][j];
    }
}
memset(dp,-1,sizeof dp);dp[(1<<n)-1]=0; //初始化
queue<int>que;
int flag=0; //是否找到
que.push((1<<n)-1); //开始全亮的状态
vis[(1<<n)-1]=1;
while (!que.empty()){ //广搜
    int u=que.front();
```

```
    que.pop();
    for(int i=1;i<=m;i++){ //每个开关尝试
        int state=u;
        for(int j=1;j<=n;j++){
            if(a[i][j]==0)continue;//不变
            else if(a[i][j]==1)  state=state&~(1<<(j-1)); //第j位为1  关
            else if(a[i][j]==-1)  state|=(1<<(j-1)); //开
        }

        if(!vis[state]){
            vis[state]=1;
            que.push(state);
            dp[state]=dp[u]+1;
            if(state==0){
                flag=1;
                break;
            }
        }
    }
    if(flag)break;
}
    if(flag) cout<<dp[0]<<endl;
    else cout<<-1<<endl;
}
signed main() {
  std::ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

      solve();

  return 0;
}
```

例4. acw 1064小国王，（棋盘问题）

题意：

在 n×n 的棋盘上放 k 个国王，国王可攻击相邻的 8 个格子，求使它们无法互相攻击的方案总数。

思路：

按行枚举每列放与不放01，下一行的方案取决于上一行 dp[i] [j] [s] 表示摆了前i行，已放j个国王，第i行状态为s的方案数

```
#include <bits/stdc++.h>
//# pragma GCC optimize(3)
#define int long long
#define endl "\n"
using namespace std;

const int N=12;//棋盘
const int M=1<<11,K=110; //状态数和国王数
int T, n,k, a[N];
```

```
vector<int>state; //所有合法状态***
vector<int>tran[M]; //状态s  能转移到的所有合法状态  9预处理）
int  dp[N][K][M];   //表示摆了前i行，已放j个国王，第i行状态为s的方案数
int cnt[M]; //每个状态1的个数

bool check(int state) //判断没有相邻1
{
    return  !(state & state << 1);
}


void solve(){
    cin>>n>>k;
    //先预处理
    for(int i=0;i<=(1<<n)-1;i++){
        if(check(i)){
                state.push_back(i);
                cnt[i]=__builtin_popcount(i);  //顺便求二进制1的个数
            }
    }
    //  处理tran数组
    for(int i=0;i<state.size();i++){  //其实空间够 没必要用下标处理  见下一题
        int a=state[i];
        for(int j=0;j<state.size();j++){
            int b=state[j];
            if((a&b)==0&&check(a|b))
                tran[i].push_back(j); //注意是下标
        }
    }
//dp
dp[0][0][0]=1; // 初始化  啥也不摆算一种

for(int i=1;i<=n+1;i++){
    for(int j=0;j<=k;j++){//放的国王数
        for(int s=0;s<state.size();s++){//合法状态下标
            for(int b:tran[s]){//s  能到达的状态

                int count=cnt[state[s]];//a中一的个数
                if(count<=j){//满足数量
                    dp[i][j][s]+=dp[i-1][j-count][b]; //转移
                }
            }
        }
    }
}

// 技巧  ：答案总数转化为  摆到第n+1 行 改行一个没摆的方法数  减少计算
int ans=dp[n+1][k][0];
cout<<ans<<endl;
}
signed main() {
  std::ios::sync_with_stdio(false);
  cin.tie(0);
  cout.tie(0);

      solve();
```

```
  return 0;
}
```