1.线性、背包、区间、树形dp
虚树
2.仙人掌DP
1.最短路
2.小C的独立集
3.Winter Festival
3.概率dp
4.计数dp
4.1 区间覆盖计数问题
5.优化技巧
5.1 单调队列、单调栈
5.2 斜率优化
5.3 四边形不等式
5.4 决策单调性
5.5 轮廓线(插头)dp
5.6.数据结构优化(动态)dp
1.最大子段和
5.7 状压、倍增
5.8 数位统计
5.9 wqs二分
6.dp套dp
1.Hero meet devil
2.Editing Explosion
3.Square
4.Domino Colorings
8.特殊dp技巧
1.寻路(两人同时走)

1.线性、背包、区间、树形dp

1.当物体数量很多，体积较小时，可以贪心到背包剩余体积小于某个时间复杂度内的值，剩下来的背包，错误率较小。

2.n个物体，V空间，每种物体c_i个，v_i体积，w_i价值

$for(i,1,n)for(j,V,v_i)for(k,1,c_i)dp[j]=max\{dp[j-kv_i]+kw_i\}$

多重背包，二进制优化为把该种物体分为1,2,4,...,left份进行dp,$O(nVlogm)$;单调队列优化，$O(nV)$，令$s=j/v_i,d=j\%v_i$

$dp[j]=max\{dp[sv_i+d-kv_i]+kw_i\}=max\{dp[(s-k)v_i+d]-(s-k)w_i\}+sw_i,k\le c_i$

令$k'=s-k,dp[j]=max\{dp[k'v_i+d]-k'w_i\}+sw_i,k'\ge s-c_i$

```
#include <bits/stdc++.h>
using namespace std;
const int N=4e4+10;
int n,V;
struct node{
    int v,w,c;
}a[110];
int dp[N];
signed main(){
    cin>>n>>V;
```

```
    for (int i=1;i<=n;i++)scanf("%d%d%d",&a[i].w,&a[i].v,&a[i].c);
    int ans=0;
    for (int i=1;i<=n;i++){
        int v,w,c;
        v=a[i].v,w=a[i].w,c=a[i].c;
        if(v==0){
            ans+=w*c;continue;
        }
        c=min(c,V/v);
        for(int d=0;d<v;d++){
            deque<pair<int,int>>q;
            int s=(V-d)/v;
            for(int j=0;j<=s;j++){
                while(!q.empty()&&dp[j*v+d]-j*w>=q.back().first)q.pop_back();
                q.push_back({dp[j*v+d]-j*w,j});
                while(!q.empty()&&q.front().second<j-c)q.pop_front();
                dp[j*v+d]=max(dp[j*v+d],q.front().first+j*w);
            }
        }
    }
    cout<<ans+dp[V]<<endl;
    return 0;
}
```

虚树

CF613D Kingdom and its Cities：n个城市，n-1条路，形成树，有q格询问，每次给出k和k个数代表这k个城市是重要城市，问最少占领多少非重要城市能使得每两个重要城市之间不连通，如果不行，输出-1

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+10;
int n,k;
vector<int>v[N];
int st[21][N*2],father[N];
int dfn[N],tms=0,o[N*2],dep[N];
void dfs(int x,int fa) {
    father[x]=fa;
    dep[x]=dep[fa]+1;st[0][++tms]=x;dfn[x]=tms;
    for (int i:v[x]){
        if(i==fa)continue;
        dfs(i,x);st[0][++tms]=x;
    }
}
void build(){
    for (int i=1;i<=tms;i++)o[i]=log(i)/log(2)+1e-7;
    for (int i=1;i<=o[tms];i++){
        for (int j=1,u,v;j+(1<<i)-1<=tms;j++) {
            u=st[i-1][j];v=st[i-1][j+(1<<i-1)];
            st[i][j]=dep[u]<dep[v]?u:v;
        }
    }
}
int LCA(int u,int v) {
    if (!u||!v)return 0;
    u=dfn[u];v=dfn[v];
    if (u>v)std::swap(u,v);
```

```
    int d=o[v-u+1];u=st[d][u];v=st[d][v-(1<<d)+1];
    return dep[u]<dep[v]?u:v;
}
vector<int>vv[N];
int a[N];
bool cmp(int x,int y){
    return dfn[x]<dfn[y];
}
void add(int x,int y){
    vv[x].push_back(y);vv[y].push_back(x);
}
int vis[N],g[N];
int sta[N],top;
int res;
void dfs1(int x,int fa){
    int nm=0;
    for (int i:vv[x]){
        if (i==fa)continue;
        dfs1(i,x);nm+=g[i];g[i]=0;
    }
    if(vis[x])res+=nm,g[x]=1;
    else if (nm>1)++res,g[x]=0;
    else g[x]=nm?1:0;
    vis[x]=0;vv[x].clear();
}
signed main(){
    int x,y,q;
    cin>>n;
    for (int i=1;i<n;i++){
        scanf("%d%d",&x,&y);v[x].push_back(y);v[y].push_back(x);
    }
    dfs(1,0);build();
    cin>>q;
    while (q--){
        scanf("%d",&k);
        for (int i=1;i<=k;i++)scanf("%d",&a[i]),vis[a[i]]=1;
        bool ff=0;
        for (int i=1;i<=k;i++)if (vis[father[a[i]]]){ff=1;break;}
        if (ff){for (int i=1;i<=k;i++)vis[a[i]]=0;puts("-1");continue;}
        sort(a+1,a+1+k,cmp);
        //建立虚树
        sta[top=1]=a[1];
        for (int i=2;i<=k;i++){
            int t=LCA(a[i],sta[top]);
            while (dep[t]<dep[sta[top-1]])add(sta[top],sta[top-1]),top--;
            if (t!=sta[top]){
                add(t,sta[top]);
                if(sta[top-1]==t)top--;
                else sta[top]=t;
            }
            sta[++top]=a[i];
        }
        while (top>1)add(sta[top],sta[top-1]),top--;
        res=0;dfs1(sta[1],0);
        printf("%d\n",res);
    }
    return 0;
}
```

2.仙人掌DP

仙人掌：没有重边和自环的无向连通图，每条边最多属于一个简单环

特性：n个点最多n-1个简单环，仙人掌最多2n-2条边

遍历方法：可以表示为一棵树加非树边，非树边必然是祖孙关系，每个非树边是一个简单环

判别方法：可以用非树边x-y的树边标记一下，如果已经被标记过了，则图不是仙人掌

对于一个点a的连向b的边：1.b是a的fa,continue;2.!dfn[b],继续搜索这棵树;3.dfn[b]<dfn[a],简单环,把b->a路径的边标记一下属于这个环,也就是点x到fa[x],只需标记x;4.dfn[b]>dfn[a],跳过

```
const int N=1e5+10;
vector<pair<int,int> >v;
int cnt=0,dfn[N];
int f[N];
int circleNum=0;//环数
int nodeCircle[N];
void dfs(int x,int fa){
    dfn[x]=++cnt;
    for (auto i:v[x]){
        int y=i.first;
        if (y==fa)continue;
        if (!dfn[y]){
            f[y]=x;dfs(y,x);
        }
        else if(dfn[y]<dfn[x]){
            circleNum++;
            int u=x;
            while (u!=y){
                nodeCircle[u]=circleNum;
                u=f[u];
            }
        }
    }
}
```

圆方树:点为圆,环为方,对于一个环,新建一个方点,环上点和方点都连一条边,不存在一条边两端点是方点

1.最短路

询问仙人掌图的两点最短路

圆圆边的边权为原边权

圆方边的边权为圆点到该环在圆方树上深度最小的圆点的最短路

对于z=lca(x,y)

若z为圆点,则树上距离

若z为方点,则x,y爬到离z差一步的距离,再求环内最短距离

仙人掌dp,正常做法,树点树dp,环点环dp

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

```
const int N=2e5+10;
int n,m,q,x,y,z;
vector<pair<int,int> >v[N],e[N];
int cnt=0,dfn[N];
int fa[N],dis1[N];
int circleNum=0;//环数
int nodeCircle[N],circleDis[N];
int dep[N],f[N][21];
void add(vector<pair<int,int> >*v,int x,int y,int z){
    v[x].push_back({y,z});v[y].push_back({x,z});
}
void dfs(int x,int ff){
    dfn[x]=++cnt;
    for (auto i:v[x]){
        int y=i.first;
        if (y==ff)continue;
        if (!dfn[y]){
            dis1[y]=dis1[x]+i.second;
            fa[y]=x;dfs(y,x);
            if (!nodeCircle[y]){
                add(e,x,y,i.second);
            }
        }
        else if(dfn[y]<dfn[x]){
            circleNum++;
            circleDis[circleNum]=dis1[x]-dis1[y]+i.second;
            int u=x;
            while (u!=y){
                nodeCircle[u]=circleNum;
                add(e,u,n+circleNum,min(dis1[u]-dis1[y],dis1[x]-
dis1[u]+i.second));
                u=fa[u];
            }
            add(e,u,n+circleNum,0);
        }
    }
}
int dis[N];//圆方树的dis
void dfs1(int u,int ff){
    dep[u]=dep[ff]+1;
    for(int i=0;i<=19;i++)//2^0 ~ 2^19
        f[u][i+1]=f[f[u][i]][i];//递推公式，上面讲过了。
    for(auto i:e[u]){
        int v=i.first;
        if(v==ff)continue;
        dis[v]=dis[u]+i.second;
        f[v][0]=u;
        dfs1(v,u);
    }
}
int xx,yy;
inline int LCA(int x,int y){
    if(dep[x]<dep[y]) swap(x,y);//让x深度较大
    //用“暴力”的思想：先让x,y跳到同一深度，然后一起往上跳
    for(int i=20;i>=0;i--){//倒着for，x能多跳尽量多跳 ，才能优化时间
        if(dep[f[x][i]]>=dep[y]) x=f[x][i];//先跳到同一层
        if(x==y) return y;
    }
```

```
    for(int i=20;i>=0;i--){//此时x,y已跳到同一层
        if(f[x][i]!=f[y][i]){//如果 f[x][i]和f[y][i]不同才跳
            x=f[x][i];
            y=f[y][i];
        }
    }
    xx=x;yy=y;
    return f[x][0];//跳完上述步骤后，两点离LCA仅一步之遥，让x(或y)再向上跳一步就是LCA。
}

signed main(){
    cin>>n>>m>>q;
    for (int i=1;i<=m;i++){
        scanf("%lld%lld%lld",&x,&y,&z);
        add(v,x,y,z);
    }
    dfs(1,0);
    dfs1(1,0);
    while (q--){
        scanf("%lld%lld",&x,&y);
        int t=LCA(x,y),ans=0;
        if (t<=n){
            ans=dis[x]+dis[y]-2*dis[t];
        }
        else{
            int r=abs(dis1[xx]-dis1[yy]);
            ans=dis[x]+dis[y]-dis[xx]-dis[yy]+min(r,circleDis[nodeCircle[xx]]-
r);
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

2.小C的独立集

求仙人掌最大独立集大小，dp[x][i][j],x代表x这个点，i代表x这个点选不选，j代表x这个点所在环底部那个点选不选

```
#include<bits/stdc++.h>
using namespace std;
const int N=2e5+10;
int n,m,q,x,y,z;
vector<int >v[N],e[N];
int cnt=0,dfn[N];
int fa[N];
int circleNum=0,nodeCircle[N];//环数
int dp[N][2][2],top[N],bot[N];
void add(vector<int >*v,int x,int y){
    v[x].push_back(y);v[y].push_back(x);
}
void dfs(int x,int ff){
    dfn[x]=++cnt;
    dp[x][1][0]=dp[x][1][1]=1;dp[x][0][0]=dp[0][1]=0;
    for (int i:v[x]){
        int y=i;
        if (y==ff)continue;
```

```
        if (!dfn[y]){
            fa[y]=x;dfs(y,x);
            #define rep(a) for(int a=0;a<2;a++)
            rep(xi)rep(xj){
                int mx=0;
                rep(yi)rep(yj){
                    if (xi&&yi)continue;
                    if (bot[y]&&yi!=yj)continue;
                    if (top[x]&&top[x]==nodeCircle[y]&&xi&&yj)continue;
                    if
(nodeCircle[x]&&nodeCircle[x]==nodeCircle[y]&&xj!=yj)continue;
                    mx=max(mx,dp[y][yi][yj]);
                }
                dp[x][xi][xj]+=mx;
            }
        }
        else if(dfn[y]<dfn[x]){
            circleNum++;
            int u=x;
            while (u!=y){
                nodeCircle[u]=circleNum;
                u=fa[u];
            }
            top[y]=circleNum;
            bot[x]=circleNum;
        }
    }
}
int findMax(int x){
    int mx=0;
    for (int i=0;i<2;i++){
        for (int j=0;j<2;j++){
            mx=max(mx,dp[x][i][j]);
        }
    }
    return mx;
}
signed main(){
    cin>>n>>m;
    for (int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        add(v,x,y);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            dfs(i,0);
            ans+=findMax(i);
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

3.Winter Festival

一组方案合法当且仅当满足以下两个条件：

1. 对于任意两条不同的边(x, y), (x, z),(*x*,y),(x,z)，它们的边权和mod 3不等于1。
2. 对于任意一个简单环，里面的所有边的边权和必须是奇数。

求所有边的边权之和最小值

```
#include<bits/stdc++.h>
using namespace std;
#define rep(a,b) for(int a=0;a<b;a++)
const int N=2e5+10;
int n,m,q,x,y,z;
vector<int>v[N],e[N];
int cnt=0,dfn[N];
int fa[N];
int circleNum=0,nodeCircle[N];//环数
int dp[N][3][2][3],top[N],bot[N];
int buf[N][2][8][3][2][3];
//dp[x][i][j][k]为i点以下子树消耗的最小值 x为当前子树 i为i到父亲的边取值 j为i到i父亲所在环边权和%2 k为i到i父亲所在环非树边长度
void add(vector<int >*v,int x,int y){
    v[x].push_back(y);v[y].push_back(x);
}
void update(int &x,int y){
    if (x==-1||x>y)x=y;
}
int addIsOne(int x){
    if (x==0)return 1;
    if (x==1)return 0;
    if (x==2)return 2;
}
bool isOK(int S,int x){//x放入S是否可以，也就是是否组成1的两条直线
    return S&(1<<addIsOne(x));
}
bool dfs(int x,int ff){
    dfn[x]=++cnt;
    //int h[1<<3][3][2][3],g[1<<3][3][2][3];
    auto h=buf[x][0],g=buf[x][1];
    memset(h,-1,sizeof (int)*144);memset(g,-1,sizeof (int)*144);
    //rep(i,3)rep(j,3)h[1<<i][i][0][j]=0;
    rep(j,3)rep(y,3)h[1<<j][j][0][y]=0;
    for (int i:v[x]){
        int y=i;
        if (y==ff)continue;
        if (!dfn[y]){
            fa[y]=x;
            if (!dfs(y,x))return false;
            memcpy(g,h,sizeof (int)*144);
            memset(h,-1,sizeof (int)*144);
            rep(s,8)
            rep(y1,3){
                if (isOK(s,y1))continue;
                rep(y2,2)rep(y3,3)rep(x1,3)rep(x2,2)rep(x3,3){
                    if (g[s][x1][x2][x3]==-1||dp[y][y1][y2][y3]==-1)continue;
                    if (nodeCircle[y]){
                        if (nodeCircle[x]==nodeCircle[y]){
```

```
                            if (x3!=y3)continue;//同环内非树边要相同
                            update(h[s|(1<<y1)][x1][(x2+y2+y1)%2][x3],g[s][x1]
[x2][x3]+dp[y][y1][y2][y3]+y1);
                        }
                        else{//x是top
                            if ((y2+y1)%2==0)continue;//环不为奇
                            if (isOK(s|(1<<y1),y3))continue;//非树边连上是否组成1的
相交线段
                            update(h[s|(1<<y1)|(1<<y3)][x1][x2][x3],g[s][x1][x2]
[x3]+dp[y][y1][y2][y3]+y1);
                        }
                    }
                    else{
                        update(h[s|(1<<y1)][x1][x2][x3],g[s][x1][x2][x3]+dp[y]
[y1][y2][y3]+y1);
                    }
                }
            }
        }
        else if(dfn[y]<dfn[x]){
            circleNum++;
            int u=x;
            while (u!=y){
                if (nodeCircle[u])return false;
                nodeCircle[u]=circleNum;
                u=fa[u];
            }
            top[y]=circleNum;
            bot[x]=circleNum;
        }
    }
    rep(s,8)
    rep(i,3)rep(j,2)rep(k,3){
        if (h[s][i][j][k]==-1)continue;
        if (bot[x]){
            if (isOK(s,k))continue;
            update(dp[x][i][(j+k)%2][k],h[s][i][j][k]+k);
        }
        else{
            update(dp[x][i][j][k],h[s][i][j][k]);
        }
    }
    return true;
}
signed main(){
    memset(dp,-1,sizeof dp);
    cin>>n>>m;
    for (int i=1;i<=m;i++){
        scanf("%d%d",&x,&y);
        add(v,x,y);
    }
    int ans=0;
    for(int i=1;i<=n;i++){
        if(!dfn[i]){
            if (!dfs(i,0))return puts("-1"),0;
            int tmp=-1;
            rep(j,3)rep(x,2)rep(y,3)if (dp[i][j][x][y]!=-1)update(tmp,dp[i][j][x]
[y]);
```

```
            ans+=tmp;
        }
    }
    cout<<ans<<endl;
    return 0;
}
```

3.概率dp

1-n每个数有个概率pi，当生成数>=所有已生成数时，继续生成数，直到有数<max(已生成数)，计分为生成的数的个数的平方，问得分期望

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=1e5+10,M=1e4+10,inf=0x3f3f3f3f,mod=998244353;
int n,m,k;
int p[N],sp[N];
int dp[N],g[N],f[N];
int qpow(int x,int y){
    int ans=1;
    while (y){
        if (y&1)ans=ans*x%mod;
        y>>=1;x=x*x%mod;
    }return ans;
}
signed main(){
    cin>>n;
    int sum=0;
    for (int i=1;i<=n;i++){
        scanf("%lld",&p[i]);sum+=p[i];
    }
    sum=qpow(sum,mod-2);
    for (int i=1;i<=n;i++)p[i]=p[i]*sum%mod;
    for (int i=n;i>=0;i--){
        int sum=0;
        for (int j=1;j<=n;j++)(sum+=p[j]*(g[j]+1))%=mod;
        g[i]=sum*qpow(1-p[i]+mod,mod-2)%mod;
//上三行等价于g[i]=p[i]g[i]+p[i+1]g[i+1]+...+p[n]g[n]+1,记录后续数字生成个数的期望值
        sum=0;
        for (int j=1;j<=n;j++)(sum+=p[j]*(dp[j]+2*g[j]+1))%=mod;
        dp[i]=sum*qpow(1-p[i]+mod,mod-2)%mod;
//上三行等价于dp[i]=p[i](dp[i]+2*g[i])+p[i+1](dp[i+1]+2*g[i+1])+...+p[n]
(dp[n]+2*g[n])+1,记录后续数字积分的期望值(i+1)^2-i^2=2i+1
    }
    cout<<dp[0]<<endl;
    return 0;
}
```

4.计数dp

4.1 区间覆盖计数问题

给出一些区间$[L_i, R_i]$，请你找到这些区间的一个最小的子集，使得子集里的区间完全覆盖$[0, M]$

方法：将所给覆盖区间从小到大排序，先比较起点，若一致则比较终点。被覆盖区间的起点是0，那么找出所有区间起点小于0中的最合适的区间。要求用尽量少的区间覆盖，所以选择右端点更大的区间，它包含区间长度更长，更易覆盖。如果在所有覆盖区间中找到了解，但右端点小于M，则把找到的覆盖区间的右端点定为新的覆盖区间的起点。

覆盖的是区间，而不是整点

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=2e5+10;
int n,l,r;
struct node{
    int l,r;
}a[N];
bool cmp(node a,node b){
    if (a.l==b.l)return a.r>b.r;
    return a.l<b.l;
}
void solve(){
    cin>>n>>l>>r;
    for (int i=1;i<=n;i++)scanf("%d%d",&a[i].l,&a[i].r);
    sort(a+1,a+1+n,cmp);
    int now=1,mx=0,p;
    vector<int>v;
    while (l<r){
        if (now>n){v.clear();break;}
        while(now<=n&&a[now].l<=l){
            if (mx<a[now].r){
                mx=a[now].r;p=now;
            }
            now++;
        }
        if (mx==l){v.clear();break;}
        v.push_back(p);
        l=mx;
    }
    cout<<v.size()<<endl;
    for (int i:v){
        cout<<a[i].l<<" "<<a[i].r<<endl;
    }
}
signed main(){
    int T;
    cin>>T;
    while (T--){
        solve();
    }
    return 0;
}
```

5.优化技巧

5.1 单调队列、单调栈

滑动窗口最大值、多重背包优化

如果一个选手比你小，还比你强，你就可以退役了

1.n天，每天有一个[l,r]温度范围，问最大连续升温天数为多少

```
#include <bits/stdc++.h>
using namespace std;
const int N=1e6+10;
int n,q,k;
struct node{
    int l,r;
}a[N];

signed main(){
    cin>>n;
    for(int i=1;i<=n;i++)scanf("%d%d",&a[i].l,&a[i].r);
    deque<pair<int,pair<int,int>>>q;//{i,{l,r}}
    int ans=1;
    for (int i=1;i<=n;i++){
        while (!q.empty()&&q.front().second.first>a[i].r)q.pop_front();//如果l的最
大值>a[i].r pop
        if (!q.empty())ans=max(ans,i-q.front().first+1);
        int t=i;
        while(!q.empty()&&q.back().second.first<a[i].l){
            t=q.back().first;
            q.pop_back();//关于l的单调下降队列，维护l的最大值
        }
        q.push_back({t,{a[i].l,a[i].r}});
    }
    cout<<ans<<endl;
    return 0;
}
```

5.2 斜率优化

5.3 四边形不等式

四边形不等式常见于区间dp的优化：$f[i][j]=min\{f[i][k]+f[k+1][j]+w(i,j),i\le k<j\}$

$w(i,j)$表示将闭区间$[i,j]$合并产生的费用，状态有$O(n^2)$个，每个状态的决策有$O(n)$个，总复杂度为$O(n^3)$

若函数$w(i,j)$满足$w(a,c)+w(b,d)\le w(a,d)+w(b,c)$，$a\le b<c\le d$，则称w满足四边形不等式。

若函数$w(i,j)$满足$w(a,d)\ge w(b,c)$，$a\le b\le c\le d$，则称w关于区间包含关系单调。可以看做长区间的值大于等于它包含的短区间的值。

设使$f[i][j]$取到最小值的k为$p[i][j]$

那么有如下定理：

1.若w满足四边形不等式，且关于区间包含关系单调，则f也满足四边形不等式。

2.若f满足四边形不等式，则$p[i][j-1] \le p[i][j] \le p[i+1][j]$，即如果把p看做一个矩阵的话，p在每一行上单调非降，在每一列上单调非降。

3.w满足四边形不等式，当且仅当$w(i,j)+w(i+1,j+1) \le w(i+1,j)+w(i,j+1)$。

用第二个定理就可以优化决策，k取$p[i][j-1] \to p[i+1][j]$

判断是否满足四边形不等式：

1.直接判断

2.打表出p数组，观察是否单调

5.4 决策单调性

5.5 轮廓线(插头)dp

5.6.数据结构优化(动态)dp

1.最大子段和

支持单点修改，求最大子段和

5.7 状压、倍增

5.8 数位统计

5.9 wqs二分

6.dp套dp

dp问题A，用dpB计算有多少个输入可以使得A结果是x

1.Hero meet devil

字符集大小为4，ACGT，给定长度为n的字符串S，对于每一个$0 \le k \le n$，问有多少个长度为m的字符串T，使得LCS(S,T)=k(最长公共子序列)

$n \le 15, m \le 1000$

$if(S[i] == T[j]) dp[i][j] = dp[i-1][j-1]+1, else\ dp[i][j] = min(dp[i-1][j], dp[i][j-1])$

压缩j的维度，进行差分

```
#include <bits/stdc++.h>
using namespace std;
const int N=1000+10;
const int mod=1e9+7;
int n,k;
string s,ss="ACGT";
long long f[2][1<<15],ans[N];
int g[2][N];
void decode(int S) {
    for (int i=1;i<=k;++i) g[0][i]=(S>>(i-1))&1;
    for (int i=1;i<=k;++i) g[0][i]+=g[0][i-1];
}
int encode() {
    int S=0;
    for (int i=1;i<=k;++i)S|=(g[1][i]-g[1][i-1])<<(i-1);
```

```
    return S;
}
int tranS[1<<15][4];
int sumone(int x){
    int cnt=0;
    for (;x;x-=x&(-x))++cnt;
    return cnt;
}
void solve(){
    cin>>s;k=s.size();s=" "+s;
    cin>>n;
    for (int S=0;S<(1<<k);S++){
        for (int j=0;j<4;j++){
            decode(S);
            for (int i=1;i<=k;i++){
                if (ss[j]==s[i])g[1][i]=g[0][i-1]+1;
                else g[1][i]=g[1][i-1]>g[0][i]?g[1][i-1]:g[0][i];
            }
            int T=encode();
            tranS[S][j]=T;
        }
    }
    memset(f[0],0,sizeof(f[0]));
    f[0][0]=1;
    for (int i=0;i<n;i++) {
        int now=i&1,nxt=now^1;
        memset(f[nxt],0,sizeof(f[nxt]));
        for (int S=0;S<(1<<k);S++){
            if(f[now][S]){
                for (int j=0;j<4;j++){
                    f[nxt][tranS[S][j]]=(f[nxt][tranS[S][j]]+f[now][S])%mod;
                }
            }
        }
    }
    for (int i=0;i<=k;i++)ans[i]=0;
    for (int i=0;i<(1<<k);++i)ans[sumone(i)]=(ans[sumone(i)]+f[n&1][i])%mod;
    for (int i=0;i<=k;++i)printf("%d\n",ans[i]);
}
signed main(){
    int T;
    cin>>T;
    while (T--){
        solve();
    }
    return 0;
}
```

2.Editing Explosion

对于两个字符串A和B，定义它们之间的编辑距离为最少的对A的编辑次数使得A和B相等，每次编辑可以是插入/删除/修改一位字符。

给定一个仅由26个大写字母构成的字符串S，统计有多少个仅由26个大写字母构成字符串与S的编辑距离恰好为d。

```
#include<bits/stdc++.h>
```

```
using namespace std;
#define int long long
const int N=1e5+10,mod=998244353;
string s;
int d,k;
array<int,11>g;
map<array<int,11>,int>f[2];

signed main(){
    cin>>s>>d;k=s.size();s=" "+s;
    for (int i=0;i<=k;i++)g[i]=i;
    f[0][g]=1;
    int ans=0;
    if (k==d)ans++;
    for (int r=1;r<=20;r++){
        int nxt=r&1,now=1-nxt;
        g[0]=r;
        for (auto o:f[now]){
            array<int,11> e=o.first;
            int w=o.second;
            for (int i=0;i<26;i++){
                for (int j=1;j<=k;j++){
                    if ('A'+i==s[j])g[j]=e[j-1];
                    else g[j]=min(min(e[j],g[j-1]),e[j-1])+1;
                }
                (f[nxt][g]+=w)%=mod;
                if (g[k]==d)(ans+=w)%=mod;
            }
        }
        f[now].clear();
    }
    cout<<ans<<endl;
    return 0;
}
```

3.Square

给你一个n * n(n <= 8)的棋盘，上面有一些格子必须是黑色，其它可以染黑或者染白，对于一个棋盘，定义它的优美度为它上面最大的连续白色子正方形的边长。

对于每个0 <= i <= n，问有多少种染色方案使得棋盘的优美度为i?

轮廓线dp

```
//2s
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=8+1,mod=1e9+7;
char s[N][N];
int n;
map<vector<int>,int>f[2];
int c[N];
void solve(){
    f[0].clear();f[1].clear();
    int nxt=0,now;
    vector<int>h;for (int i=0;i<=n+1;i++)h.push_back(0);//第0位记录当前边长最大值
    f[nxt][h]=1;
```

```
    for (int i=1;i<=n;i++){
        for (int j=1;j<=n;j++){
            now=nxt;nxt^=1;f[nxt].clear();
            for (auto k:f[now]){
                auto p=k.first;
                if (j==1){for (int i=n+1;i>1;i--)p[i]=p[i-1];}
                p[j]=0;
                (f[nxt][p]+=k.second)%=mod;
                if (s[i][j]=='o'){
                    auto p=k.first;
                    if (j==1){for (int i=n+1;i>1;i--)p[i]=p[i-1];}
                    if (j>1)p[j]=min(p[j],min(p[j-1],p[j+1]))+1;
                    else p[j]=1;
                    p[0]=max(p[0],p[j]);
                    (f[nxt][p]+=k.second)%=mod;
                }
            }
        }
    }
    memset(c,0,sizeof c);
    for (auto k:f[nxt]){
        (c[k.first[0]]+=k.second)%=mod;
    }
    for (int i=0;i<=n;i++)cout<<c[i]<<endl;
}
signed main(){
    int T;
    cin>>T;
    while (T--){
        cin>>n;
        for (int i=1;i<=n;i++){
            scanf("%s",s[i]+1);
        }
        solve();
    }
    return 0;
}
```

```
//不知为何53ms
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=10,mod=1e9+7;
int n,k;
string a[N];
int dp[N][1<<(N+1)];
int ans[N+10];
int b[N],bit[N];
signed main(){
    int T;
    cin>>T;
    while (T--){
        cin>>n;
        for (int i=1;i<=n;i++){
            cin>>a[i];a[i]=" "+a[i];
        }
        int sum=1;
```

```
        for (int i=1;i<=n;i++){
            b[i]=0;
            for (int j=1;j<=n;j++){
                if (a[i][j]=='o')sum=sum*2%mod;//记录所有染色方案数
                else b[i]|=(1<<(j-1));
            }
        }
        ans[1]=1;ans[n+1]=sum;
        for(int sz=2;sz<=n;++sz){//正方形大小,sz=2时，计算的时<2正方形的所有方案数
            memset(dp,0,sizeof(dp));
            dp[0][0]=1;
            bit[0]=1;for(int i=1;i<n;++i)bit[i]=bit[i-1]*sz;//bit[i]=qpow(sz,i)
            int tot=bit[n-sz+1];
            for(int i=1;i<=n;++i){//每行
                for(int st=0;st<tot;++st){//所有的状态，每个状态记录i到i+sz-1下方白格子
的最小值，也就是正方形高最高能取多少
                    if(!dp[i-1][st])continue;
                    for(int j=0;j<1<<n;++j){//枚举当前行的染色方案
                        if (b[i]&j)continue;
                        int nxt=0;
                        for(int tmp=j,l=1;l+sz-1<=n;++l,tmp>>=1){
                            int now=(tmp&((1<<sz)-1))==((1<<sz)-1)?st/bit[l-
1]%sz+1:0;//新st的l个变量处是否全为1
                            if(now>=sz){nxt=-1;break;}
                            nxt+=now*bit[l-1];
                        }
                        if(nxt!=-1)(dp[i][nxt]+=dp[i-1][st])%=mod;
                    }
                }
            }
            ans[sz]=0;for(int st=0;st<tot;++st)(ans[sz]+=dp[n][st])%=mod;
        }
        for(int i=0;i<=n;++i)cout<<(ans[i+1]-ans[i]+mod)%mod<<endl;
    }
    return 0;
}
```

4.Domino Colorings

先考虑1*2和2*1的骨牌，填入n*m的方格中，有多少可能，对于i,j点的填写，记录前一个状态的轮廓线n个点是否填，上方是空的，则填上方和i，j点;否则，左边空填左边和i,j点或者不填

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=12;
int n,m,k;
int dp[N*N][1<<N];
bool check(int x,int y){
    return x&(1<<y);
}
signed main(){
    while (cin>>n>>m){
        if (n==0&&m==0)break;
        memset(dp,0,sizeof dp);
        dp[0][(1<<n)-1]=1;
        for (int i=1;i<=m;i++){
```

```
            for (int j=1;j<=n;j++){
                int t=(i-1)*n+j;
                for (int k=0;k<(1<<n);k++){
                    if (!check(k,j-1))dp[t][k|(1<<(j-1))]+=dp[t-1][k];
                    else{
                        if (j>1&&!check(k,j-2))dp[t][k|(1<<(j-1))|(1<<(j-
2))]+=dp[t-1][k];
                        dp[t][(k|(1<<(j-1)))^(1<<(j-1))]+=dp[t-1][k];
                    }
                }
            }
        }
        int t=n*m;
        cout<<dp[t][(1<<n)-1]<<endl;
    }
    return 0;
}
```

1*2和2*1的黑白相间的骨牌，填入n*m的方格中，最终形成的填满状态黑白颜色有多少种可能。

$n \leq 6, m \leq 300$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=7,M=300+10,mod=1e9+7;
int n,m,k;
map<pair<int,unsigned long long>,int>f[2];//f[2][{i,j}] i为当前轮廓线的颜色状压 f[S]
为n个点匹配状态为S是否可行 j为f[S]这个bool数组取值为j

signed main(){
    cin>>n>>m;
    int now,nxt=0;
    f[nxt][{0,1ull<<((1<<n)-1)}]=1;
    for (int i=1;i<=m;i++){
        for (int j=1;j<=n;j++){
            now=nxt,nxt^=1;f[nxt].clear();
            for (auto it:f[now]){
                int t=it.first.first;
                unsigned long long S=it.first.second;
                int w=it.second;
                int uc=(t>>(j-1))&1,lc=(t>>(j-2))&1;//当前的黑白,左边的黑白
                for (int k=0;k<=1;k++){//当前取黑还是白
                    unsigned long long next_S=0;int next_t=(t-(uc<<(j-1)))|
(k<<(j-1));
                    for (int x=0;x<(1<<n);x++){//枚举所有匹配状态
                        if (S&(1ull<<x)){//该状态可行
                            if (!(x&(1<<(j-1)))){//上边
                                if (k!=uc){
                                    next_S|=1ull<<(x|(1<<(j-1)));
                                }
                            }
                            else{
                                if (j>1&&!(x&(1<<(j-2)))){//左边
                                    if (k!=lc){
                                        next_S|=1ull<<(x|(1<<(j-2)));//上边已经为1
不用修改本身位置
```

```
                                }
                            }
                            next_S|=1ull<<(x^(1<<(j-1)));//空
                        }
                    }
                }
                if (next_S)(f[nxt][{next_t,next_S}]+=w)%=mod;//存在可行状态
            }
        }
    }
    int ans=0;
    for (auto i:f[nxt]){
        if (i.first.second&(1ull<<((1<<n)-1)))(ans+=i.second)%=mod;
    }
    cout<<ans<<endl;
    return 0;
}
```

8.特殊dp技巧

1.寻路(两人同时走)

n*n矩阵，每个方格有一个数字，从左上角走到右下角，你选择的路线不同，会得到不同的Q，请对于每种可能的Q，计算有多少条路线对应这个Q，假设满足条件的路线数为f(Q)，你需要输出$\sum(f(Q)^2)$。

考虑两个人同时从起点开始走，每一步走相同的路，$O(n^4)$，设两人所在区域为(a,b),(c,d)，则a+b=c+d，最后一维可由其他维得出，所以可以除去一维，$O(n^3)$

```
#include<bits/stdc++.h>
using namespace std;
#define int long long
const int N=3e2+10,mod=1e9+7;
int n,k;
int a[N][N];
int dp[N][N][N];

signed main(){
    cin>>n;
    for (int i=1;i<=n;i++){
        for (int j=1;j<=n;j++){
            scanf("%lld",&a[i][j]);
        }
    }
    dp[1][1][1]=1;
    for (int i=1;i<=n;i++){
        for (int j=1;j<=n;j++){
            for (int k=1;k<=n;k++){
                if (i==1&&j==1&&k==1)continue;
                if (a[i][j]==a[k][i+j-k]){
                    dp[i][j][k]+=dp[i-1][j][k-1];
                    dp[i][j][k]+=dp[i][j-1][k-1];
                    dp[i][j][k]+=dp[i-1][j][k];
                    dp[i][j][k]+=dp[i][j-1][k];
                    dp[i][j][k]%=mod;
                }
```

```
            }
        }
    }
    cout<<dp[n][n][n]<<endl;
    return 0;
}
```