动态规划

dp 主要的流程为 设计状态-->推导递推表达式-->数据结构/其他技巧优化计算 。

注意设计状态后检验是否覆盖所有情况，是否无后效性。

构造一个可做的状态尤为重要，这部分需要多练。

I.期望DP

期望的计算

- 定义：$E(X)=\sum i\cdot P(X=i)$ ....(1)
- 递推：$E(X)=\sum P(Y\to X)\cdot(E(Y)+w(Y\to X))$ ....(2)

  或：$E(X)=\sum P(Y\to X)E(Y)+w$ ，（转移代价相同）。
- 二项分布期望：$x\sim B(n,p),\ E(x)=np$ ....(3)
- 性质：$E(X+Y)=E(X)+E(Y), E(nX)=nE(X)$

概率的计算

- 古典概型公式：$P(X)=\frac{N(X)}{N_{总}}$ ....(4)
- 递推：$P(X)=\sum P(Y\to X)\cdot P(Y)$ ....(5)

基础例题：

一般设计状态时，尽量利用等价性，设置一些具有概括性的状态。

如使用 “考虑了 i 个 xx 的情况”，而不是“考虑了具体某 i 个的情况”。

正推与逆推：初状态已知则正推，末状态已知则逆推。

- 入门：甲，乙每题正确率为 α 和 β ，答对一题得一分，一人领先 4 分时获胜，求甲胜率。

  考虑分差，只有 9 种情况。设 f_i 表示分差为 i 时甲的胜率，则 $f_i=af_{i-1}+bf_i+cf_{i+1}$ 。

  其中 $a=(1-\alpha)\beta$ ，$b=\alpha\beta+(1-\alpha)(1-\beta)$ ，$c=\alpha(1-\beta)$ ，$f_{-4}=0$，$f_4=1$。

  解方程即可。
- n 个格子，每次随机选取一个涂黑，求涂色 m 次后，被涂黑格子的期望个数。[递推法]
  - 设计状态：设第 i 次涂色后期望为 f_i 。

    递推式：由（2）式，$f_i=(\frac{f_{i-1}}{n})f_{i-1}+(1-\frac{f_{i-1}}{n})(f_{i-1}+1)=(1-\frac{1}{n})f_{i-1}+1$.

    优化计算：一阶含常数递推，可使用矩阵优化，或者数学计算通项公式。
  - 由（4）式，最终某个格子被涂黑的概率 $P=\frac{N_{涂黑}}{N_{总}}$ ， 则最终涂黑格子数 $\xi\sim B(n,P)$

    其中 $N_{总}=n^m$，$N_{涂黑}=N_{总}-N_{未涂黑}=n^m-(n-1)^m$

    即 $P=1-(1-\frac{1}{n})^m$ ，由（3）式，$E=nP=n-n(1-\frac{1}{n})^m$

    快速幂计算即可。
- n 个格子，每次随机选取一个涂黑，求涂满的期望涂色次数。[逆推法]
  - 设计状态：已经涂满 i 个格子，期望还需 f_i 次才能涂满。

递推式：由（2）式，$f_i=(1-\frac{i}{n})f_{i+1}+\frac{i}{n}f_i+1$，即 $f_i=f_{i+1}+\frac{n}{n-i}$

- 构造状态：涂满 i 个格子期望涂色次数为 f_i 是否可做?

- 从数轴原点开始每次随机向左或向右走一格（原点处只向右走），求走到 n 点的期望步数。[无限累加法]
  - 设计状态：第一次走到 i 点的期望步数为 f_i

    递推式：$f_{i+1}=\sum\limits_{k=0}^{+\infty}k\cdot(\frac{1}{2})^{k+1}\cdot g_i+f_i+1$。

    注意到我们构造了辅助状态 g_i 表示从 i 点向左走一步再走回来的总期望步数。

    递推式：$g_{i+1}=2+\sum\limits_{k=0}^{+\infty}k\cdot(\frac{1}{2})^{k+1}g_i$

    由数学知识，$S_n=\sum\limits_{i=0}^{n}\frac{i}{2^{i+1}}=1-\frac{n+2}{2^{n+1}}$，$n\to+\infty$ 时 $S_n\to 1$。

    故 $g_{i+1}=g_i+2, g_1=2$, 即 $g_n=2n, f_{i+1}=f_i+2i+1$

    易得 $f_n=n^2$

- [UVA11021 Tribles]
  - 题目描述：一种生物寿命一天，死后产生若干后代，其中产生 i 个概率为 p_i，$(0\le i<n)$。初始时有 k 个生物，求第 m 天该种生物全部死亡的概率。
  - 状态：初始时有 1 个生物时，第 i 天没有生物存活概率为 f_i

    初始时有 k 个生物时，第 i 天没有生物存活概率为 f_i^k。

    那么第一个生物死后产生 s 个生物，它们需要在 $i-1$ 天内全部死亡。

    即：$f_i=\sum\limits_{s=0}^{n-1}f_{i-1}^{s}p_s$

- [P1654 OSU!]
  - 题目描述：一个 01 串，第 i 位为 1 的概率为 p_i，定义该串的权值为所有连续的 1 的长度的三次方求和。求权值的期望。
  - 举例：0111011010：连续的 1 有 111,11,1 三段，权值为 $3^3+2^3+1^3=36$
  - 状态：设第 $1-i$ 位权值和期望为 f_i，结尾有连续 len_i 个 1.

    则 $f_{i+1}=f_i+E((len_i+1)^3-len_i^3)p_{i+1}=f_i+(3E(len_i^2)+3E(len_i)+1)p_{i+1}$

    $E(len_i)=(E(len_{i-1})+1)p_i$

    $E(len_i^2)=(E(len_{i-1}^2)+E((len_{i-1}+1)^2-len_{i-1}))p_i=(E(len_{i-1}^2)+2E(len_{i-1})+1)p_i$。

DAG 上期望

对于一张图，某点到终点的期望步数 $E_u=\sum\limits_{u\to v}(E(v)+w(u\to v))p(u\to v)$

- 典中典：[P4316 绿豆蛙的归宿]
  - 描述：给出一张DAG，每次等可能向一条出边移动，求走到终点的期望步数。
  - 方法：直接代公式，反向建图后按拓扑序计算即可。

    ```
    void calc(){
        queue<int> q;
        q.push(n);
    ```

```
        dp[n]=0;
        while(!q.empty()){
            int u=q.front();
            q.pop();
            for(int i=head[u];i;i=nxt[i]){
                int v=to[i];
                dp[v]+=(dp[u]+w[i])/deg[v];
                if(!(--cnt[v]))q.push(v);
                //cnt 与 deg 均为度数数组。
            }
        }
    }
```

无向图上期望

如果在无向图上（或有向有环图），我们可能绕一圈回到某个点，就不能使用 dp 解决问题了。

但是公式还是能代的，只是最后写出的不是递推式而是方程组，高斯消元硬解即可。

- P3211 [HNOI2011]XOR和路径

  题目大意：给出一张无向图，每次等可能向一条边移动，求走到终点时，经过边权异或和的期望。

  异或的期望好像不好求，可以把每一位分开考虑，最后拼回来。

  代公式：$E_u=\frac{1}{deg_u}(\sum\limits_{w_{u,v}=0}E_v+\sum\limits_{w_{u,v}=1}(1-E_v))$

  移项：$deg_uE_u+\sum\limits_{w_{u,v}=1}E_v-\sum\limits_{w_{u,v}=0}E_v=\sum\limits_{w_{u,v}=1}1$

  设起点点第 i 位期望为 f_i ，则答案为 $\sum 2^if_i$ 。

  由于是方程组，其实已经不一定需要逆推了，不过正着推可能有些细节问题。

II.状压DP

常用操作的实现

元素均从 0 编号，注意运算符优先级。

- 只包含第 i 个元素的集合：`1<<i`
- 全集：`(1<<n)-1`
- 向集合中添加元素 $S|i$ ：`S= S|(1<<i)`
- 从集合中去除元素：`S=S&(~(1<<i))`
- 判断集合是否包含元素：`S&(1<<i)`
- 单个数位反转：`S^(1<<i)`
- 交集：`S&T`
- 判断是否有相邻元素：`(S&(S<<1))||(S&(S>>1))`

例题

- 有 n 个格子，每次选一个涂色，选择第 i 个格子概率为 p_i ，保证 $\sum p_i=1$ ，求全部涂色的期望次数。

  和第一部分的题设类似，但是每个格子不等价了，我们只能用状压表示每个格子的状态。

  设 $f(S)$ 表示集合 S 已被涂满，还需再涂色的期望次数。

  则：$f(S)=\sum f(S|i)p_i+1$ 。复杂度 $O(n2^n)$

```
int lim=(1<<n);
for(int S=lim-1;~S;--S){
    double in=1,out=1;
    for(int i=1;i<n;++i)
        if(S&(1<<i))in-=p[i];
        else out+=f[S|(1<<i)]*p[i];
    f[S]=out/in;
}
```

- P1896 [SCOI2005] 互不侵犯

  在 $N*N$ 的棋盘上摆放 k 个棋子，要求棋子两两不相邻（八个方向上），求方案数（$1\le N\le 9$）。

  当前行的方案只与上一行有关，设 $f(i,S,k)$ 表示考虑前 i 行，上一行棋子摆放状态为 S，共摆了 k 个。

  则 $f(i,S,k)=\sum\limits_{T\to S}f(i-1,T,k-|S|)$，复杂度 $O(n^3 2^{2n})$。

```
bool linecheck(int x){
    return !((x&(x<<1))or(x&(x>>1)));
}

bool check(int las,int cur){
    return !((las&cur)or(las&(cur<<1))or(las&(cur>>1)));
}

int main(){
    cin>>n>>k;
    lim=1<<n;
    for(int S=0;S<lim;++S)
        if(linecheck(S)){
            st.push_back(S);
            num.push_back(__builtin_popcount(S));
        }
    cnt=st.size();
    for(int i=0;i<cnt;++i)
        for(int j=0;j<cnt;++j)
            if(check(st[i],st[j]))
                nxt[i].push_back(j);

    for(int i=0;i<cnt;++i)
        dp[1][i][num[i]]=1;
    for(int i=2;i<=n;++i)
      for(int j=0;j<cnt;++j){
        for(auto t:nxt[j])
            for(int x=num[j];x<=i*n;++x)
                dp[i][j][x]+=dp[i-1][t][x-num[j]];
      }
    long long ans=0;
    for(int i=0;i<cnt;++i)
        ans+=dp[n][i][k];
    cout<<ans<<endl;
}
```

- TSP 问题

  状压最经典的应用：在图上找到一个经过所有点的环，使环长度最小。

  设未经过点集为 S，目前在 v 点的最短路径为 $f(S,v)$

  则 $f(S,v)=\min\limits_{u\notin S}\{f(S|u,u)+w_{u,v}\}$

三进制状压

当状态不能用 0/1 表示时，也可以使用状压。如果有三种情况，就是三进制状压，更高进制同理。

我们用两个二进制位表示一个三进制位。

- TSP 问题

  经过图上每个点各 **两次** ，求最短路径。

  设状态集合 S 表示每个点被经过 0/1/2 次，方法一样。

数位DP

统计 $[1,n]$ 范围内符合某些条件的数的个数，应当逐位考虑。

状态设计：$f(info,i,limit)$ ，$info$ 为判断需要的信息，i 为当前位数，$limit$ 表示是否达到上界，通常还要注意前导 0 的处理。

一般使用记忆化搜索求解 f 。

要求解 $[l,r]$ 内的答案，拆解为 $[1,r]-[1,l-1]$ 即可。

- 求 $[1,n]$ 内各位之和是位数的两倍的数的个数。

  $f(sum,len,i,limit)$ 表示当前是第 i 位，位数为 len ，当前枚举到第 i 位且是/否达到上界时的答案数。

  有 $f(sum,len,i,limit)=\sum\limits_{0\le k\le U}f(sum+k,len+1-[sum+k=0],i-1,limit\&[k=U])$ 。

  其中 $limit$ 取 1 时 $U=n_i$ ，否则 $U=9$ 。$f(2i,i,-1,0/1)=1,f(0,0,-1,0)=0$ 。

```
int calc(int dep,int sum,int len,int lim){
    if(dep<0)
        return len&&(sum==len*2);
    if(dp[dep][sum][len][lim]!=-1)return dp[dep][sum][len][lim];
    int mx=lim?dig[dep]:9;
    int ans=0;
    for(int i=0;i<=mx;++i)
         ans+=calc(dep-1,sum+i,len+(len||i),lim&&(i==mn));
    return dp[dep][sum][len][lim]=ans;
}
```

- windy数：求 $[a,b]$ 内不含前导 0 ，且相邻数字至少差 2 的数。

  $f(x,i,lead,limit)$ 表示上一位数字是 x ，当前在第 i 位，当前是否属于前导0，是否达到上界的方案数。

$$f(x,i,lead,limit)=\sum_{|k-x|\ge 2} f(k,i-1,lead\&[k=0],limit\&k=U)$$

```
int calc(int dep,int x,int lead,int lim){
    if(dep<0)return !lead;
    if(dp[dep][x][lead][lim]!=-1)return dp[dep][x][lead][lim];
    int mx=lim?dig[dep]:9;
    int ans=0;
    for(int i=0;i<=mx;++i)
      if(lead||i>=x+2||i+2<=x)
         ans+=calc(dep-1,i,lead&(i==0),lim&&(i==mx));
    return dp[dep][x][lead][lim]=ans;
}
```

- 萌数：求 $[l,r]$ 内含长度至少为 2 的回文串的数的个数。
- 花神的数论题：求 $[1,N]$ 内所有数字二进制下 1 的个数之积。($1\le N\le 10^{15}$)

树形&换根DP

- 没有上司的舞会：在树上选择任意个点，要求选择的点不相邻，求选择的点权和最大的方案。
    - $f_{i,0/1}$ 表示是否选择 i 号点时的最大方案。
    - $f_{u,0}=\sum_{v\to u}\max(f_{v,0},f_{v,1})$，$f_{u,1}=val_u+\sum_{v\to u} f_{v,0}$

换根DP

又称二次扫描法，用于求每个点作为根时的答案。

可以先指定一个根，然后考虑根变化时答案的变化，核心思想在于去除一个子树的贡献。

- STA-station：依次以 $1-n$ 中的所有点为根，分别求出所有点的深度和。
    - 考虑以 1 号点为根，f_u 表示 u 的子树内的点到 u 的距离和。$f_u=\sum_{v\to u}(f_v+siz_v)$，得到 1 号点的答案 g_1
    - 将根变为儿子节点 k，则 $g_k=f_k+(g_{fa}-f_k-siz_k+n-siz_k)=g_{fa}+n-2siz_k$.

```
void dfs1(int rt,int fa){
    siz[rt]=1;
    for(int i=head[rt];i;i=nxt[i])
        if(to[i]!=fa){
            dfs1(to[i],rt);
            siz[rt]+=siz[to[i]];
            f[rt]+=f[to[i]]+siz[to[i]];
        }
}
void dfs2(int rt,int fa){
    for(int i=head[rt];i;i=nxt[i])
        if(to[i]!=fa){
            int& v=to[i];
            g[v]=g[rt]+n-2*siz[v];
            dfs2(v,rt);
        }
```

```
}
```

- CF1324F：给出一棵树，上面有黑（权值为 -1）白（权值为 1 ）两种节点，从根节点出发，向相邻节点移动，求经过节点权值和的最大值，重复经过只记一次权值。分别求出每个点为根时的答案。
    - 考虑根固定的情况：设 f_u 为从 u 出发向更深的节点走，能得到的最大值，则 $f_u = \sum\limits_{v \to u} \max(f_v, 0) + val_u$
    - 将根变为儿子 k ：设 g_u 表示以 u 为根的答案，则 $g_u = f_u + \max(g_{fa} - \max(f_u, 0), 0)$
- 一道膜你赛的题：棋子初始时在某一点，可以向相邻的点移动（不重复），也可以不动，你的得分是所有经过的点权值之和。每次移动前某人会封锁一个相邻点，这个点你将永远不能经过，并且他会采取最优策略使你得分尽可能小。依次求出棋子初始时在每一点时的答案。
    - f_i 表示你已到达 i 点时一定还能获得的最大得分。则 $f_u = \sec\limits_{u \to v} f_v + val_u$ 。$\sec$ 表示第二大值。
    - g_i 表示棋子初始时在 i 点的答案。