**DP:**

**旅行商问题：**

状压dp:

先弗洛伊德，然后dp

状态dp[st][i]表示经过了st中标记的点，现在在i点的最小经过距离

**注意n=1的情况！！**

**斯坦纳树问题:**

思路跟旅行商问题相似

当然还是先floyd

状态dp[st][k]表示需要连通的点的状态st下，包括点i的时候需要的最短距离。

转移：

枚举st的所有子状态  
dp[st][k]=min{dp[subst][k]+dp[st-subst][k]}  
枚举k和i

dp[st][k]=min{dp[st][i]+dis[i][k]}

这个有循环，则从dp[st][k]小的开始算（可以从小的开始算,也可以从小的开始更新）

最后求答案是dp:

先枚举1-2^8-1的所有状态

这样每2位就是0-3的一个数，代表4个要求所在的组

把每组的值加起来就是这一个状态的答案

（即枚举状态就是枚举分组情况）

比如00011111就是第一个和第二个要求为一组，第三个要求为一组，第四个要求为一组，答案为dp[00001111]+dp[00110000]+dp[11000000]

然后求最小值就是答案了

floyd();

int mx,mi;

//初始状态

for(int i=0;i<8;i++)

for(int j=0;j<N;j++)

dp[1<<i][j]=dis[goal[i]][j];

for(int i=1;i<(1<<8);i++)

{

if(!(i&(i-1)))//初始状态，略过

continue;

for(int k=0;k<N;k++)

{

vis[k]=0;

//枚举子状态，都由k点连接

**/\***

for(int j=0;j<i;j++)

{

if((i|j)==i) dp[i][k]=min(dp[i][k],dp[i-j][k]+dp[j][k]);

}

**\*/**

for(int j=i;j;j=(j-1)&i)

dp[i][k]=min(dp[i][k],dp[i-j][k]+dp[j][k]);

}

//更新每个k的值（这个状态可能不是由k点连接而来的情况最小，而是通过其他点连接）

for(int j=0;j<N;j++)

{

mx=inf;

for(int k=0;k<N;k++)

{

if(dp[i][k]<mx&&!vis[k])

{

mx=dp[i][k];

mi=k;

}

}

vis[mi]=1;

for(int k=0;k<N;k++)

dp[i][k]=min(dp[i][k],dp[i][mi]+dis[mi][k]);

}

}

//枚举每种可能的结果

int res=inf,nst,rt,nres;

for(int i=0;i<(1<<8);i++)

{

nres=0;

for(int j=0;j<4;j++) //把状态相同的归为一类

{

nst=0;

for(int k=0;k<8;k+=2)

{

if((i>>k&3)==j)

{

nst+=3<<k;

rt=goal[k];

}

}

if(nst)

nres+=dp[nst][rt];

}

res=min(res,nres);

}

printf("%d\n",res);

**树形dp:**

实际就是dfs的记录（跟一般dp类似），一般一个点dfs结果需要一个数组来表示，由此点往下的点的数组推出这个数组（为了消除后效性可以加维数）。注意想好状态转移再写…

**概率dp**

（如果有自环）（如果没自环可以正推）:

1. 将要求状态的概率设为1
2. 如果i状态能转移到j,k,l…状态，则dp[i]=dp[j]+dp[k]+dp[l]+..
3. 初始状态的概率即为所求概率
4. 若有环则解方程组
5. 如果答案很小，注意精度问题（如果可以直接判断为0则直接输出0之类的）

6.状态不好递推可以考虑记忆化搜索之类的

\*如果正算不好算的话，考虑算反的概率p’，再用1-p’得到p

**斜率优化dp：**

当dp状态只有1维，复杂度为O(n^2)，并且对于状态i，有

若看看k<j<i,且g(j,k)=(f1(j)-f1(k))/(f2(j)-f2(k))<=h(i)，则由j状态转移至i状态会比k状态转移至i状态要优，则满足斜率优化条件

模板：

while(tail>head&&getUp(que[head+1],que[head])<=i\*getdown(que[head+1],que[head]))

head++;

DOdp(i,que[head]);

while(tail>head&&judge(now,que[tail],que[tail-1]))

tail--;

que[++tail]=i;

bool judge(int i,int j,int k)

{

if(getdown(j,k)==0) return true;//如果需要特判

if(getUp(i,j)\*getdown(j,k)<getUp(j,k)\*getdown(i,j)) //=

return true;

return false;

}

**插头dp：**

解决棋盘类哈密顿回路问题，

，其中j<>i，i，j in S．

复杂度

1. 状态用M+1\*M/2位表示，每一位表示这个点上或者下的插头所在连通块，如果没插头则是0（连通块用最小表示法）
2. 每个状态在每个点最多只能更新出2个状态（因为上，左两个插头已经定下）
3. 每个点的状态用size较小的hash表存起来，用于下一个点
4. 各种细节注意