树形dp

树形DP准确的说是一种DP的思想，将DP建立在树状结构的基础上。

树形DP的主要实现方式是dfs。

一般可分为2种基本类型。

选择节点类

一般都是有选择性质的题目

```
一般状态设定为dp[maxn][2]，一般表示第i个点是否选择

存图部分：
struct node{
    ll next,to;    //next表示与第i条边同起点的下一条边的序号,to表示第i条边的终点,权值有没有看情况
}e[maxn];
ll head[maxn],tot;   //head数组记录了以i为起点的第一条边的序号。tot表示边数

inline void add(ll u,ll v){     //链式前向星存图
    tot++;
    e[tot]={head[u],v};
    head[u]=tot;
}

dp部分：
void dfs(ll u){    //树形dp一般都是从上到下的一个过程，所以一开始的选点一般是根节点
    for(int i=head[u];i;i=e[i].next){
        ll v=e[i].to;      //每次找到他的儿子节点编号
        dfs(v);         //每次找下去
        dp[u][0]+=max(dp[v][0],dp[v][1]);     //如果u没去则有两种选择
        dp[u][1]+=dp[v][0];     //如果u去了那么只有一种选择
    }
}
```

树形背包类

一般都是限定了个数边界的题目

```
dp[maxn][maxn]   //dp[i][j]表示以i节点为根节点，选j条边或者j个点的最优解
void dfs(ll u){
    for(int i=head[u];i;i=e[i].next){
        ll v=e[i].to;
        dfs(v);
        for(int j=m;j;j--){      //枚举该节点的可能
            for(int k=0;k<j;k++){      //枚举该节点子树的可能
                dp[u][j]=max(dp[u][j],dp[u][j-k]+dp[v][k]);    //枚举所有可能
            }
        }
    }
}
```

换根dp

换根dp，是针对无根树上 dp 问题的一个解题技巧。

无根树上 dp 时一般会先强制一个节点当整棵树的根，然后在考虑求解

但是，如果要求出所有点做根时的 dp 值该怎么办呢 ，暴力枚举n次显然是不行的（大概率收到个 tle ）

换根 dp 通常要通过两次 dfs 来实现，第一次 dfs 时随便指定一个点当根（一般为1）

举个例子

给定一个 n 个点的树，请求出一个结点，使得以这个结点为根时，所有结点的深度之和最大。

```
dp[maxn],ans[maxn]     //dp[i]表示以i为根节点的最大值,ans[i]表示i节点的子树个数
但是因为我们发现影响答案的不一定只在子树内部，在另一部分也是可以的，所以我们需要第二次dfs统计不在子树中的答案情况，但是这一部分的答案我们发现其实是可以利用他到根之间的子树的答案推出来

void dfs1(ll u,ll fa){
    ans[u]=0;
    for(int i=head[u];i;i=e[i].next){
        ll v=e[i].to;
        if(v==fa) continue;     //双向边的判断，所以如果是他的父节点直接跳过，不然会重复计算
        dfs1(v,u);
        ans[u]+=ans[v];
        dp[u]+=dp[v];
    }
    ans[u]++;
    dp[u]+=ans[u];
}
当根从u换到v时，我们发现原来所有的在v这棵子树的点的深度都-1，相应的，其他的不在v这棵子树的点的深度都会+1

void dfs2(ll u,ll fa){
    for(int i=head[u];i;i=e[i].next){
        ll v=e[i].to;
        if(v==fa) continue;
        dp[v]=dp[u]+n-2*ans[v];     //化简前:dp[u]-ans[v]+(n-ans[v])
        s=max(s,dp[v]);
        dfs2(v,u);
    }
}
```