树形dp

树形dp就是在树上处理一系列的dp问题，一般是通过自下而上或者自上而下递归形式dp处理

树的直径

定义：树上两点之间最远的距离

我们考虑用树形dp求解，$dp[u]$代表以u点可以到达他子树中的最远距离，在dp中，我们先递归获取了u点中以v_i的子树信息，对于经过点u和v的最大距离，肯定是$dp[u]+dp[v]+edge[u,v]$，对全部的这个取max就是直径了，同时，我们也需要向上合并dp信息，对于$dp[u]$来说，他的最大值一定是在本身$dp[u]$和$dp[v]+edge[u,v]$中取max所以只要这样维护即可

```
int ans=0;
vector<pair<int,int>>G[N];
void dfs(int u,int fa){
    for(auto &[v,w]:G[u])if(fa!=v){
        dfs(v,u);
        ans=max(ans,dp[u]+dp[v]+w);
        dp[u]=max(dp[u],dp[v]+w);
    }
}
```

树的重心

定义：在树中找到一个点，以他为根时，他的各个子树大小中的最大值最小

```
int ans=0;
vector<int>G[N];
int sz[N],maxx[N],ans=2e9,ansu;
void dfs(int u,int fa){
    sz[u]=1;
    maxx[u]=0;
    for(auto &v:G[u])if(fa!=v){
        dfs(v,u);
        sz[u]+=sz[v];
        maxx[u]=max(maxx[u],sz[v]);
    }
    maxx[u]=max(maxx[u],n-sz[u]);
    if(maxx[u]<ans)ans=maxx[u],ansu=u;
}
```

树的中心

定义：在树中找到一个点，他到其他所有点中的最远距离最近

我们可以吧距离分成两部分，u点子树内到他的最远距离，和子树外的最远距离，两者取max

对于子树内的最远距离很容易dp得到$dp[u]=max\{dp[v]+edge[u,v]\}$

对于子树外的的点来说，我们可以从上向下dp得到，对于v点，他子树外的最大距离，要么是父节点u加上u向下最长的距离，或者向上的最长距离，但是如果v刚好在u向下的最长路径上，那么就会发生问题，所以要额外记录次长距离，在v在最长路径上时，次长距离可能大于往上的最长距离。所以总的思路就是做两次dfs，第一次先把每个先到子树的最长距离和次长距离记录。在第二次的时候计算子树外的最长距离

```
vector<pair<int,int>>G[N];
int dp[N][3];
void dfs(int u,int fa){
    dp[u][0]=0;dp[u][1]=0;
    for(auto &[v,w]:G[u])if(fa!=v){
        dfs(v,u);
        if(dp[v][0]+w>=dp[u][0]){
            dp[u][1]=dp[u][0];
            dp[u][0]=dp[v][0]+w;
        }else if(dp[v][0]+w>=dp[u][1]){
            dp[u][1]=dp[v][0]+w;
        }
    }
}

void dfs2(int u,int fa){
    for(auto &[v,w]:G[u])if(fa!=v){
        if(w+dp[v][0]==dp[u][0])dp[v][2]=max(dp[u][2]+w,dp[u][1]+w);
        else dp[v][2]=max(dp[u][2]+w,dp[u][0]+w);
        dfs2(v,u);
    }
}
```

树上背包

树上背包一般是树上的dp类似背包一样的效果，如题目[P2014 CTSC1997] 选课 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)主要是复杂度的证明分析，代码比较简单

首先设置dp，$dp[i][j]$代表在i点的子树内选了j门课的最大贡献，显然选课需要有前置，所以在i点子树内选一门课只能选自己，$dp[i][1]$的分数就是本身分数，然后选0门课我们需要给他负无穷的贡献，因为需要达成依赖的关系，这样在下方不选课的时候，该处贡献永远无法产生。

```
vector<int>G[N];
int dp[N][N];
int temp_dp[N];
void dfs(int u,int fa){
    dp[u][0]=-1e9;
    dp[u][1]=a[u];
    for(auto &v:G[u]){
        dfs(v,u);
        for(int i=0;i<=m+1;i++)temp_dp[i]=dp[u][i];
        for(int i=0;i<=m+1;i++){
            for(int j=0;j<=m+1;j++){
                temp_dp[i+j]=max(temp_dp[i+j],dp[u][i]+dp[v][j]);
            }
        }
        for(int i=0;i<=m+1;i++)dp[u][i]=temp_dp[i];
    }
```

```
}
```

然后这个复杂度显然是$O(n*m^2)$的

然后树上背包主要的是对这部分代码优化

首先一个比较简单的优化,我们发现第一重循环不需要从$m+1$开始，因为子树大小不一定有这么大,然后第二重循环是枚举v这个子树大小的，我们也可以对其进行一定的限制

```
vector<int>G[N];
int dp[N][N],sz[N];
int temp_dp[N];
void dfs(int u,int fa){
    dp[u][0]=-1e9;
    dp[u][1]=a[u];
    sz[u]=1;
    for(auto &v:G[u]){
        dfs(v,u);
        for(int i=0;i<=min(m+1,sz[u]);i++)temp_dp[i]=dp[u][i];
        for(int i=min(m+1,sz[u])+1;i<=min(m+1,sz[u]+sz[v]);i++)temp_dp[i]=0;
        for(int i=0;i<=min(m+1,sz[u]);i++){
            for(int j=0;j<=min(m+1-i,sz[v]);j++){
                temp_dp[i+j]=max(temp_dp[i+j],dp[u][i]+dp[v][j]);
            }
        }
        sz[u]+=sz[v];
        for(int i=0;i<=min(m+1,sz[u]);i++)dp[u][i]=temp_dp[i];
    }
}
```

接下来我们证明优化后这分代码的时间复杂度会达到$O(n*m)$

我们观察dp部分，两重for循环可以理解成枚举两个子树内的点对，然后将这些点合并为一个子树，所以这个本质过程实际上是枚举了所有点对的复杂度，然后点对数量只有n^2个然后再循环中我们将范围对m取min了所以复杂度为$O(n*m)$。