DP第一讲

基础

概念

三要素："状态" "阶段" "决策"

三个基本条件：子问题重叠性，无后效性，最优子结构性质

把原问题视作若干个重叠子问题进行求解，每个子问题求解的过程就是一个"阶段"。

LIS(最长上升子序列)

$d[i] = max_{0 \le j < i, a[j] < a[i]} d[j] + 1$

LCS（最长公共子序列）

$$d[i,j] = max \begin{cases} d[i-1][j] \\ d[i][j-1] \\ d[i-1][j-1]+1 \quad \text{if a[i] = b[i]} \end{cases} \tag{1}$$

数字三角形

$$d[i][j] = a[i][j] + max \begin{cases} d[i-1][j] \\ d[i-1][j-1] \quad \text{if j>1} \end{cases} \tag{2}$$

背包

0/1背包

问题模型：给定N个物品，其中第 i 个物品的体积为V_i 价值为W_i 。有一容积为M的背包，要求选择其中一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

阶段：已经处理的物品数

状态：$d[i][j]$ 表示从前 i 个物品中选出了总体积为 j 的物品放入背包，物品的最大价值和

$$d[i][j] = max\begin{cases} d[i-1][j] \\ d[i-1][j-V_i]+W_i & \text{if j} \geq V_i \end{cases} \tag{3}$$

```
memset(d,0xcf,sizeof d); // -INF
d[0][0] = 0;
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++)
        d[i][j] = d[i-1][j];
    for(int j=v[i];j<=m;j++)
        d[i][j] = max(d[i][j], d[i-1][j-v[i]] + w[i]);
}
```

因为状态转移时, 第 i 层结果只受 i-1 层影响，所以只需要两层数组就可以解决

```
int d[2][MAX_VAL+1];
memset(d,0xcf,sizeof d);
d[0][0] = 0;
for(int i=1;i<=n;i++){
    for(int j=0;j<=m;j++)
        d[i & 1][j] = d[(i-1) & 1][j];
    for(int j=v[i];j<=m;j++)
        d[i & 1][j] = max(d[i & 1][j], d[(i-1) & 1][j-v[i]] +
w[i]);
}
```

用一维数组优化

```
int d[MAX_VAL+1];
memset(d,0xcf,sizeof d);
d[0] = 0;
for(int i=1;i<=n;i++)
    for(int j=m;j>=v[i];j--)
        f[j] = max(f[j],f[j-v[j]] + w[i]);
```

1. 团队分组（Uva-1627）

有n($n \leq 100$)个人，把他们分成非空的两组，使得每个人都被分到一组，且同组中的人相互认识。要求两组的成员人数尽量接近。多解时输出任意方案，无解时输出No Solution。

例如：1认识2，3，5；2认识1，3，4，5；3认识1，2，5；4认识1，2，3；5认识1，2，3，4

完全背包

问题模型：给定N种物品，其中第 i 种物品的体积为V_i 价值为W_i，每种都有无限个 。有一容积为M的背包，要求选择其中一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

```
memset(d,0xcf,sizeof d);
for(int i=1;i<=n;i++)
    for(int j=v[i];j<=m;j++)
        d[j] = max(d[j],d[j-v[i]] + w[i]);
```

多重背包

问题模型：给定N种物品，其中第 i 种物品的体积为V_i 价值为W_i，每种有C_i个 。有一容积为M的背包，要求选择其中一些物品放入背包，使得物品总体积不超过M的前提下，物品的价值总和最大。

- 直接拆分成0/1背包也能做
- 二进制拆分为0/1背包，每种物品为 $logC_i$个

```
#include <bits/stdc++.h>
using namespace std;
const int N = 2010;
int n,m;
int f[N];
struct Good{
    int v,w;
};
int main(){
    vector<Good> goods;
    cin>>n>>m;
    for(int i=0;i<n;i++){
        int v,w,s;
        cin>>v>>w>>s;
        for(int k=1;k<=s;k*=2){
            s -= k;
            goods.push_back({v*k,w*k});
        }
    }
    for(auto good:goods){
        for(int j=m;j>=good.v;j--){
            f[j] = max(f[j],f[j-good.v] + good.w);
        }
    }
    cout<<f[m]<<endl;
    return 0;
}
```

单调队列优化背包

```
#include <bits/stdc++.h>
using namespace std;
const int N = 200010;
int n,m,f[N],g[N],q[N];
int main(){
    scanf("%d%d",&n,&m);
    for(int i=0;i<n;i++){
        int v,w,s;
        scanf("%d%d%d",&v,&w,&s);
        memcpy(g,f,sizeof f);
        for(int j = 0;j < v;j ++){
            int l = 0,r = -1;
            for(int k=j;k<=m;k+=v){
                f[k] = g[k];
                if(l <= r && k-s*v > q[l])l++;
                if(l <= r)f[k] = max(f[k],g[q[l]] + (k-
q[l])/v*w);
                while(l <= r && g[q[r]] - (q[r]-j)/v*w <=
g[k]-(k-j)/v*w) r--;
                q[++r] = k;
            }
        }
    }
    cout<<f[m]<<endl;
    return 0;
}
```

分组背包

有依赖的背包

背包求方案数

背包求具体方案

区间DP

石子合并

有N堆石子排成一排，其中第 i 堆石子的重量为 A_i 。每次可以选择其中相邻的两堆石子合并成一堆，形成的新石子堆的重量以及消耗的体力都是两堆石子的重量之和。求把全部N堆石子合成一堆最少需要消耗多少体力。 $1 \le N \le 300$

- $d[l][r]$ 表示把最初的第 l 堆到第 r 堆石子合并成一堆，需要消耗多少的体力
- $d[l][r] = min_{l \le k < r} d[l][k] + d[k+1][r] + \Sigma_{i=l}^{r} A_i$

[注] : 计算`d[l][r]`时，区间`[l][r]`中所有长度小于len = r-l+1 的子区间的答案都必须求出。否则无法向`d[l][r]`转移。这也就是告诉我们，可以用区间的长度作为阶段。

```
memset(d,0x3f,sizeof d);
for(int i=1;i<=n;i++){
    d[i][i] = 0;sum[i] = sum[i-1] + a[i];
}
for(int len = 2;len <= n;len ++){
    for(int l=1;l<=n-len+1;l++){
        int r = l+len-1;
        for(int k=l;k<r;k++)
            d[l][r] = min(d[l][r],d[l][k]+d[k+1][r]);
        d[l][r] += sum[r] - sum[l-1];
    }
}
```

金字塔

一棵树上面的每个结点都带有一个颜色(用字母表示)，然后给定这棵树dfs之后的颜色序列，求树的结构的可能。

例如 ABABABA 有五种可能。

- d[l,r]表示该区间的可能的树结构的数量。那么必有$s[l] = s[r]$ 。所以从左到右找 k ($s[l] = s[k] \&\& s[l+1] == s[k-1]$) 这样就有 $d[l+1][k-1]$ 为第一颗子树的可能结构数。至于$[k, r]$ 之间如何组合，递归子问题求解。
- 由此可知该题用记忆化搜索写会比较好写，当然递推也可以。

```
#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9;
char s[303];
int d[303][303];
int solve(int l,int r){
```

```
    if(l > r)return 0;
    if(l == r)return 1;
    if(d[l][r] != -1)return d[l][r];
    d[l][r] = 0;
    for(int k=l+2;k<=r;k++)
        if(s[l] == s[k] && s[l+1] == s[k-1]){
            d[l][r] = (d[l][r] + (long long)solve(l+1,k-1) * 
solve(k,r)%mod)%mod;
        }
    return d[l][r];
}
int main(){
    memset(d,-1,sizeof d);
    scanf("%s",s+1);
    printf("%d\n",solve(1,strlen(s+1)));
    return 0;
}
```

树形DP

阶段：节点从深到浅（子树从小到大）的顺序作为DP的阶段

状态：第一维为节点编号。第二维根据题目会有所不同

决策：在递归子树，解决了子树问题之后，回溯时从子结点向本节点进行转移

树形DP即树上的动态规划问题

引入——树的最大独立集

> 对于一棵n个结点的无根树，选出尽量多的结点，使得任何两个结点均不相邻（称为最大独立集），然后输入n-1条无向边，输出一个最大独立集（如果有多解，输出一解）

解决

用d_i 来表示以 i 为根节点的子树的最大独立集大小

考虑当前结点 i，如果选择 i，则 i 的儿子全部不能选，问题转化为求出 i 的孙子d值之和。如果不选 i，则转换为了求 i 的儿子的d值之和

所以现在有两种做法

- 计算 i 的 d值时，需要知道它的儿子和孙子的 d值和，
- 计算 i 的儿子或者孙子的 d 值时，用刷表法将贡献计算到结点 i 中

其实二维数组就解决了这个问题。

1. 关于树——遗留的几个问题

1> 树的重心

> 对于一棵 n 个结点的无根树，找到一个点，使得把树变成以该节点为根的有根树时，最大子树的结点树最小。换句话说，删除这个点后最大联通块（一定是树）的结点树最小。

- d[i] 表示以 i 为根的子树的结点个数。

2> 树的直径

> 对于一棵 n 个结点的无根树，找到一条最长路径，换句话说，要找到两个点，使得他们的距离最远

- d[i]表示 i 的子树中，距离结点 i 到叶子的最大距离。
- 转移：d[i] = max(d[j]) + 1;
- 将子结点的d[j]都求出后，排个序选前两大，更新答案。
- 有没有不排序的做法?

1.没有上司的舞会

给定一棵树，每个结点都有一个权值。要求从中选取一些点，这些点两两之间没有连边，并且使得权值和最大。

- `d[x][0]`表示以x为根的子树中不选x的最大权值和
- `d[x][1]`表示以x为根的子树中选择x的最大权值和

转移时我们考虑 当前结点x

- 不选择x，即更新`d[x][0]`
  - $d[x][0] = \Sigma_{s \in Son(x)} max(d[s][0], d[s][1])$
- 选择x，即更新`d[x][1]`
  - $d[x][1] = a[x] + \Sigma_{s \in Son(x)} d[s][0]$

【注】: 对于树，我们一般用vector来存。

```
vector<int> son[10010];
int d[10010][2],v[10010],h[10010],n;
void dp(int x)
{
    d[x][0] = 0;
    d[x][1] = a[x];
    for(int i=0;i<son[x].size();i++)
    {
        int y = son[x][i];
        dp(y);
        d[x][0] += max(d[y][0],d[y][1]);
        d[x][1] += d[y][0];
    }
}
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)scanf("%d",&a[i]);
    for(int i=1;i<n;i++)
```

```
    {
        int x,y;
        scanf("%d%d",&x,&y);
        v[x] = 1;
        son[y].push_back(x);
    }
    int root;
    for(int i=1;i<=n;i++)
    {
        if(!v[i]){
            root = i;
            break;
        }
    }
    dp(root);
    cout<<max(d[root][0],d[root][1]);
    return 0;
}
```

加一点点难度的例题：Hali-Bula的晚会（uva1220）

好了，看了简单题，那我们稍微加一点点难度

2.校赛C题LaTale

题目：

Legend goes that in the heart of ocean, exists a gorgeous island called LaTale, which has n cities. Specially, there is only one way between every two cities.

In other words, n cities construct a tree connected by n-1 edges. Each of the edges has a weight w.

Define d(u, v) the length between city u and city v. Under the condition of u differing from v, please answer how many pairs(u, v) in which d(u, v) can be divisible by 3.   Pair(u,v) and pair(v,u) are considered the same.

> 简化题意：
>
> 给定一棵树，每条边上面都有一个权值。定义$d[u][v]$为从u到v的距离，求满足$d[u][v]\%3=0$ 的 pair(u,v)的数量

按照上面的题意，可以想到

- `d[x][0]`表示x为根的子树中，距x长度模3等于0 的结点的数量。以此类推$d[x][1],d[x][2]$

考虑如何转移?

转移很简单

$d[x][0]=\Sigma_{s\in Son(x)}(d[s][(3-edge[x][s]\%3)]+1)$,其他相同

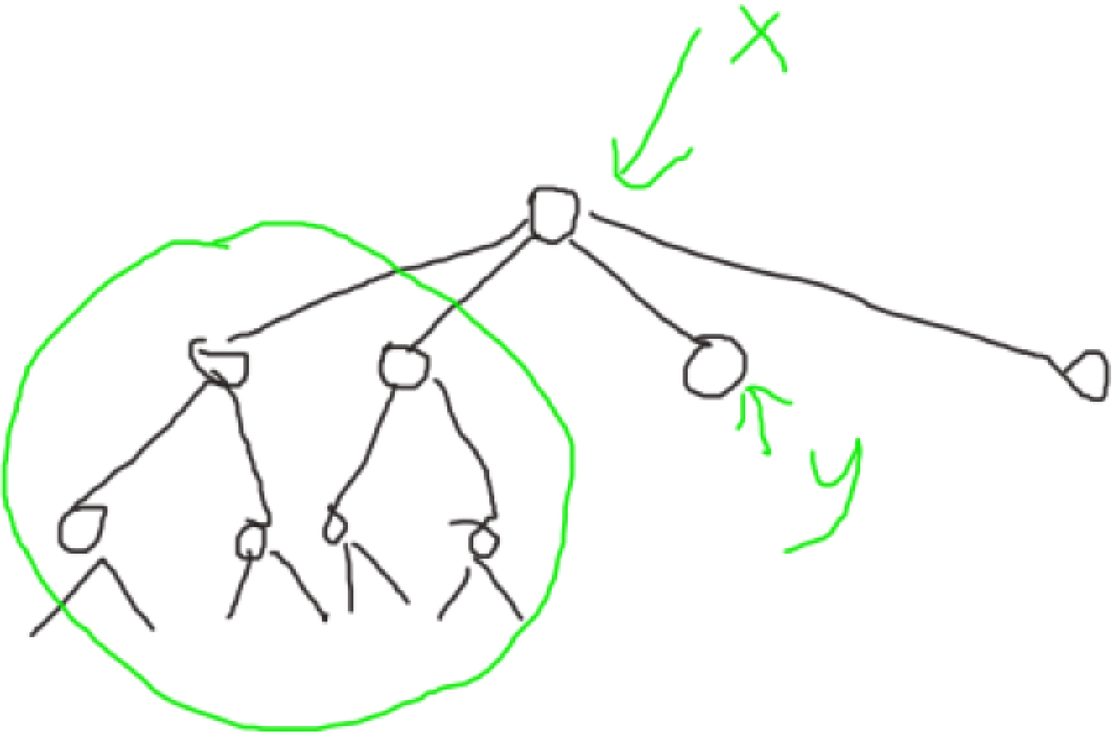
但是该怎么计算答案呢?

根据`d[x][0]` ,我们只能知道x的子树中，到x距离模三为0的节点数，但是子结点与子结点并没有考虑完整。

> x如果有大于等于两颗子树，那么第一颗子树与第二颗子树的点对就完全没有考虑

最暴力的做法就是从x的子树中枚举两个子树，然后算答案。

但是仔细想想，两个子树要想有联系，必须经过x才可以



假设现在计算到了 x 的子结点 y，递归完y之后，我们得到了`d[y][0],d[y][1],d[y][2]`

然后`d[x][0],d[x][1],d[x][2]`都还保存了左侧绿圈内的点的答案。所以现在是计算 y 子树跟绿圈内的点的大好时机。设z为`edge[x][y]`，那么`(a+b+z)%3==0`时，`d[x][a] * d[y][b]`就是一组对答案的贡献。

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5+10;
int head[N],ver[N*2],edge[N*2],nxt[N*2],tot,n;
long long d[N][3];
long long res;
void add(int x,int y,int z){
    ver[++tot] = y;
    edge[tot] = z;
    nxt[tot] = head[x];
    head[x] = tot;
}
```

```
void dfs(int x,int fa){
    d[x][0]++;
    for(int i=head[x];i;i=nxt[i]){
        int y = ver[i];
        if(y == fa)continue;
        dfs(y,x);
        int z = edge[i];
        for(int a=0;a<3;a++){
            int b = (3-z-a+3)%3;
            res += d[x][a] * d[y][b];
        }
        for(int a=0;a<3;a++)d[x][(a+z)%3] += d[y][a];
    }
}
int main(){
    int T;scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        res = 0;
        tot = 0;
        for(int i=1;i<=n;i++){
            d[i][0] = d[i][1] = d[i][2] = 0;
            head[i] = 0;
        }
        for(int i=1;i<n;i++){
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            z%=3;
            add(x,y,z);
            add(y,x,z);
        }
        dfs(1,0);
        printf("%lld\n",res);
    }
```

```
    return 0;
}
```

3.完美的服务(uva-1218)

题意：有n($n \le 10000$)台机器形成树状结构。要求在其中一些机器上安装服务器，使得每台不是服务器的计算机恰好和一台服务器计算机相邻。求服务器的最少数量。

- $d(u,0)$表示u是服务器，则每个子结点可以是服务器也可以不是
- $d(u,1)$表示u不是服务器，但u的父亲是服务器，这意味着u的所有子结点都不是服务器
- $d(u,2)$表示u和u的父亲都不是服务器，这意味着u恰好有一个儿子是服务器

转移：

- $d(u,0) = sum\{min(d(v,0), d(v,1))\} + 1$
- $d(u,1) = sum(d(v,2))$
- $d(u,2) = min(d(u,1) - d(v,2) + d(v,0))$

4.保卫Zonk(uva-12093)

题意：给定一个有n($n \le 10000$)个结点的无根树，有两种装置A和B，每种都有无限多个。

- 在某个节点X使用A装置需要花费C1($C1 \le 1000$)的花费，并且此时与结点X相连的边都被覆盖
- 在某个结点X使用B装置需要花费C2($C2 \le 1000$)的花费，并且此时与结点X相连的边以及与结点X相连的点相连的边都被覆盖

首先想到

- d[0] 表示x不装
- d[1] 表示x装A
- d[2] 表示x装B

怎么转移？d[0] x不装，x与子结点所连的边该有x的fa处理还是该由子结点处理？所以还需要细化状态

- d[0] 表示x不装，与子结点的边由子结点处理（x的fa不装B）
- d[1] 表示x装A，（好像等同于子结点或fa装B，总之有一个B就完事了）
- d[2] 表示x装B
- d[3] 表示x不装，与子结点的边由fa装B处理(那么d[1]里面只需要处理一个子结点装B就好了)

转移

- $d[x][0]+=min(d[v][1],d[v][2])$
- $d[x][1]+=min(d[v][0],d[v][1],d[v][2])$ 最后还需考虑是给x装A好还是给一个子结点装B好
- $d[x][2]=C2+\Sigma_{v\in son(x)}min(d[v][0],d[v][1],d[v][2],d[v][3])$
- $d[x][3]=\Sigma_{v\in son(x)}min(d[v][0],d[v][1],d[v][2])$