https://www.luogu.com.cn/problem/P4362

chenjiuri_tanchidejiutoulong

20mins

- 毫无感觉，毫无思路。
  - 于是探究解空间。
    - 一种比较有序的枚举顺序，慢慢的选一组。
    - 分析这些解空间，来找出重复结构。
    - 借此来指导完成状态设计优化等等问题。
- 如果问题弱化，变成单纯分成m组的最小花费。
  - 那么最小花费应该是0.
  - 但是现在有一个组上加了限制。
- 分类讨论—假设肯定可以构造出解。
  - 如果祖数大于3.那么就是一个简单选出一组，让相邻边尽量少，并且总权重尽量少的问题。
    - 因为此时另外一组，必然可以构造出一个0解。
    - 这里转化为与一个分组背包问题。
  - 如果组数等于2.
    - 分成两个子图，保证两个子图的边权和最小。
    - 这里也可以进行分组背包，但是迁移的方式会发生一些变化。
- 状态定义方式如下：
  - $f_{i,j,\ 0/1}$：第i个节点是选/不选情况之下。里面分出k为第一组分配k个果实的最小花费。
  - 然后去琢磨状态转移方程。

---

solve

- 大方向上（状态定义）正确的，但是小方向上存在着一定的问题。
- 二哥分类讨论，发现性质的角度正确。
- 但是对于不同情况之下的状态转移方程的处理没有做好。

```
#include <bits/stdc++.h>
using namespace std;

void MAIN();
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(nullptr), cout.tie(nullptr);
    MAIN();
}
typedef long long ll;
const int maxn = 300 + 10;
//------code-----٩(´ω`*)و ------靓仔代码-----٩(´ω`*)و ----talk is cheap , show
me the code--------
int n, m, K;
```

```
struct edge
{
    int x;
    int to;
    int w;
} e[maxn << 1];
int head[maxn];
int tot = 0;
void add(int x, int y, int w)
{
    e[++tot].to = head[x];
    e[tot].x = y;
    e[tot].w = w;
    head[x] = tot;
}

int f[maxn][maxn][2];
int tmp[maxn][2];

void dfs(int now, int fa)
{
    f[now][1][1] = f[now][0][0] = 0;
    for (int i = head[now]; i; i = e[i].to)
    {
        if (e[i].x == fa)
            continue;
        dfs(e[i].x, now); //先对其进行更新。
        // f[now][0][0] += f[e[i].x][0][0];
        // f[now][1][1] += f[e[i].x][0][0];
        // for (int j = K; j >= 0; j--)      //这里是一个什么样的背包问题？
        //     for (int k = 0; k <= j; k++) //选择多少个。
        //     {
        //         //之前的都是不选的。
        //         f[now][j][0] = min({f[now][j][0], f[now][j - k][0] +
f[e[i].x][k][0] + (m == 2) * e[i].w, f[now][j - k][0] + f[e[i].x][k][1]});
        //         f[now][j][1] = min({f[now][j][1], f[now][j - k][1] +
f[e[i].x][k][0], f[now][j - k][1] + f[e[i].x][k][1] + e[i].w});
        //     }

        //第二种尝试：

        memcpy(tmp, f[now], sizeof(tmp));
        memset(f[now], 0x3f, sizeof f[now]);
        int u = now;
        int v = e[i].x;
        for (int j = 0; j <= K; ++j)
        {
            for (int t = 0; t <= j; ++t)
            {
                f[u][j][0] = min(f[u][j][0], min(f[v][t][0] + tmp[j - t][0] +
(m == 2) * e[i].w, f[v][t][1] + tmp[j - t][0]));
                f[u][j][1] = min(f[u][j][1], min(f[v][t][1] + tmp[j - t][1] +
e[i].w, f[v][t][0] + tmp[j - t][1]));
```

```
            }
        }
    }
}

void MAIN()
{

    memset(f, 0x3f, sizeof f);
    //  cout << f[1][1][1];
    // 0的选择这下都为0；
    //
    //初始化为较大值。
    //初始化问题：
    //第一普遍上不存在，或者没有计算的状态都是0；
    // f[u][1][1]=f[u][0][0]=0;
    cin >> n >> m >> K;
    for (int i = 1; i < n; i++)
    {
        int x, y, w;
        cin >> x >> y >> w;
        add(x, y, w);
        add(y, x, w);
    }
    dfs(1, 0);
    if (K + m >= n)
        cout << -1 << '\n';
    else
        cout << f[1][K][1] << '\n';
}
```

生长思考：

- 关于树形dp，dp过程中要考虑多种不一样的点。
  - 这里其实就是一个多重背包加上滚动数组优化的问题。
  - 但是普通的这样写并不对。
- 可能是初始化上出了问题。
  - 手推一下基本的状态。
  - 上面个自己尝试两种方式（靠近多重背包滚动数组优化的方式都是不行的）。
  - 和第一版本得多重背包不同点是，多重背包求最大，而这里求得是最小。
- 初始化的解决方法
  - 所有叶子节点有$f_{s,0,0} = f_{s,1,1} = 1$
  - 非叶子节点，就是$f_{s,1,1} = f_{s,0,0} =$ 第一个$son->f_{son,0,0}$;
- 搞不懂上面出了什么问题。但是总之
  - 有时候，用了滚动数组，使用得是一个轮子式的滚动。发现过不了的时候，不妨考虑退一步
  - 比如两个指针，指向两个数组。然后用两个指针，简单swap.
  - 如上做法。