【PAT A1034】 Head of a Gang

Input Specification:

Output Specification:

Sample Input 1:

Sample Output 1:

Sample Input 2:

Sample Output 2:

翻译：

输入规格：

输出规格：

示例输入1：

示例输出1：

示例输入2：

样本输出2：

题意：

样例解释：

思路：

step1：

step2:

step3：

Step4:

注意点：

代码如下：

知识点补充：

【PAT A1034】 Head of a Gang

One way that the police finds the head of a gang is to check people's phone calls. If there is a phone call between A and B, we say that A and B is related. The weight of a relation is defined to be the total time length of all the phone calls made between the two persons. **A "Gang" is a cluster of more than 2 persons** who are related to each other w**ith total relation weight being greater than a given threshold *K***. In each gang, the one with maximum total weight is the head. Now given a list of phone calls, you are supposed to find the gangs and the heads.

Input Specification:

Each input file contains one test case. For each case, the first line contains two positive numbers N and K (both less than or equal to 1000), the number of phone calls and the weight threthold, respectively. Then N lines follow, each in the following format:

```
Name1 Name2 Time
```

where `Name1` and `Name2` are the names of people at the two ends of the call, and `Time` is the length of the call. A name is a string of three capital letters chosen from `A`-`Z`. A time length is a positive integer which is no more than 1000 minutes.

Output Specification:

For each test case, first print in a line the total number of gangs. Then for each gang, print in a line the name of the head and the total number of the members. It is guaranteed that the head is unique for each gang. The output must be sorted according to the alphabetical order of the names of the heads.

Sample Input 1:

```
8 59
AAA BBB 10
BBB AAA 20
AAA CCC 40
DDD EEE 5
EEE DDD 70
FFF GGG 30
GGG HHH 20
HHH FFF 10
```

Sample Output 1:

```
2
AAA 3
GGG 3
```

Sample Input 2:

```
8 70
AAA BBB 10
BBB AAA 20
AAA CCC 40
DDD EEE 5
EEE DDD 70
FFF GGG 30
GGG HHH 20
HHH FFF 10
```

Sample Output 2:

```
0
```

翻译：

警方找到帮派头目的一种方法是查看人们的电话。如果A和B之间有电话，我们说A和B是相关的。关系的权重定义为两人之间所有通话的总时间长度。“帮派”是指两个以上的人组成的一个群体，他们之间的关系总权重大于给定的阈值K。在每个帮派中，总权重最大的是头部。现在给你一份电话清单，你应该找到帮派和头目。

输入规格：

每个输入文件包含一个测试用例。对于每种情况，第一行分别包含两个正数N和K（均小于或等于1000）、电话号码和重量threthold。接下来是N行，每行的格式如下：

```
Name1 Name2 Time
```

其中Name1和Name2是通话两端的人的名字，Time是通话的长度。名称是从**A-Z**中选择的三个大写字母组成的字符串。时间长度是不超过1000分钟的正整数。

输出规格：

对于每个测试用例，首先在一行中打印组的总数。然后，将每个帮派的头目姓名和成员总数打印成一行。保证每个帮派的头像都是独一无二的。输出必须按照标题名称的字母顺序排序。

示例输入**1**：

```
// 包含两个正数N和K（均小于或等于1000）：电话号码和重量threthold
8 59
// Name1 Name2 Time
// Name1和Name2是通话两端的人的名字，Time是通话的长度
// 名称是从A-Z中选择的三个大写字母组成的字符串
AAA BBB 10

BBB AAA 20

AAA CCC 40
```

```
DDD EEE 5

EEE DDD 70

FFF GGG 30

GGG HHH 20

HHH FFF 10
```

示例输出**1**：

```
// 在一行中打印组的总数
2
// 将每个帮派的头目姓名和成员总数打印成一行
// 保证每个帮派的头像都是独一无二的。输出必须按照标题名称的字母顺序排序。
AAA 3

GGG 3
```

示例输入**2**：

```
8 70

AAA BBB 10

BBB AAA 20

AAA CCC 40

DDD EEE 5

EEE DDD 70

FFF GGG 30

GGG HHH 20
```

```
HHH FFF 10
```

样本输出**2**：

```
0
```

题意：

给出若干人之间的通话长度（视为无向边），这些通话将他们分为若干组。每个组的总边权设为该组内的所有通话的长度之和，而每个人的点权设为该人参与的通话长度之和。现在给定一个阈值k,且只要一个组的总边权超过K，并满足成员人数超过2，则将该组视为“犯罪团伙Gang”, 而该组内点权最大的人视为头目。要求输出“犯罪团伙"的个数，并按头目姓名字典序从小到大的顺序输出每个 " 犯罪团伙"的头目姓名和成员人数。

样例解释：

样例 1

样例 1 示意图如图 10-15 所示。

设三个相同的字母用一个字母表示，如 A 表示 AAA，总共分为三个组，总边权从左至右分别为 70、75、60，均超过了阈值 59，但第二组只有两个成员（D 和 E），因此只有第一组和第三组被视为 Gang。其中，第一组的头目是 A（点权为 70），第三组的头目是 G（点权为 50）。

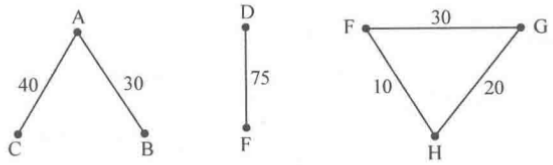

图 10-15　样例 1 示意图

样例 2

阈值变为 70，因此三组都不是 Gang。

思路：

step1：

首先解决的问题是姓名与编号的对应关系。方法有二：

1、使用map<string,int>直接建立字符串与整型的映射关系；

2、使用字符串hash的方法将字符串转换为整型，编号与姓名的对应关系则可以直接用string数组进行定义，或者使用map<int,string>也是可以的。

step2:

获取每个人的点权，即与之相关通话记录的时长之和，而显然可以在读入时就进行处理（假设A与B的通话时长为T，那么A和B相连的点权分别增加T）。事实上，该步是在求与某个点相连的边的边权之和。

step3：

进行图的遍历。使用DFS遍历每个连通块，目的是获取每个连通块的头目（即连通块内点权最大的结点）、成员个数、总边权。其中DFS对单个连通块的遍历逻辑如下：

```
// DFS 函数访问单个连通块，nowVisit为当前访问的编号
//head 为头目， numMember为成员编号，totalValue为连通块的总边权，均为引用
void DFS(int nowVisit, int& head, int& numMember, int&
totalValue){
    numMember++; //成员人数+1
    vis[nowVisit] = true;  //标记nowVisit已访问
    if(weight[nowVisit] > weight[head]){  // weight是点权
        head = nowVisit; // 当前访问结点的点权大于头目的点权，则更新头目
    }

    // G[][] :邻接矩阵G
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(G[nowVisit][i] > 0){  // 如果从nowVisit能到达i
            totalValue += G[nowVisit][i]; //连通块的总边权增加该边权
```

```
            G[nowVisit][i] = G[i][nowVisit] = 0; //删除这边，防止回
头
            if(vis[i] == false){ // 如果i未被访问，则递归访问i
                DFS(i,head,numMember,totalValue);
            }
        }
    }
}
```

Step4:

步骤 4：通过步骤 3 可以获得连通块的总边权 totalValue。如果 totalValue 大于给定的阈值 K，且成员人数大于 2，则说明该连通块是一个团伙，将该团伙的信息存储下来。

注：可以定义 map<string, int>，来建立团伙头目的姓名与成员人数的映射关系。由于 map 中元素自动按键从小到大排序，因此自动满足了题目要求的“姓名字典序从小到大输出”的规定。

```
//DFSTrave函数遍历整个图，获取每个连通块的信息
void DFSTrave(){
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(vis[i] == false){ // 如果i未被访问
            int head = i,numMember = 0, totalValue = 0; //头目、成
员数、总边权
            DFS(i,head,numMember,totalValue);
            if(numMember > 2 && totalValue > K){ //成员数大于2,且总
边权大于K
                //head人数为numMember
                //map<string,int> Gang; //head--->人数
                // 定义map<string,int>，来建立团伙头目的姓名与成员人数的
映射
                //map<int,string> intToString; // 编号⟶ 姓名
                 Gang[intToString[head]] = numMember;
            }
        }
    }
}
```

注意点：

注意点

① 由于通话记录的条数最多有 1000 条，这意味着不同的人可能有 2000 人，因此数组大小必须在 2000 以上。

② map<type1, type2>是自动按键 type1 从小到大进行排序的，因此使用 map<string, int>建立头目姓名与成员人数的关系便于输出结果。当然，也可以使用结构体来存放头目姓名与成员人数，但会增加一定的代码量。代码如下：

```
struct Gang {
    string head;        //团伙头目
    int numMember;      //成员数量
}arrayGang[maxn];
int numGang = 0;        //团伙个数
bool cmp(Gang a, Gang b) {
    return a.head < b.head;     //按头目姓名的字典序从小到大排序
}
```

③ 由于每个结点在访问后不应再次被访问，但是图中可能有环，即遍历过程中发生一条边连接已访问结点的情况。此时为了边权不被漏加，需要先累加边权，再去考虑结点递归访问的问题（样例中 FFF、GGG、HHH 的环在处理时就会碰到这种情况）。而这样做又可能导致一条边的边权被重复计算（例如累加完边 FFF→GGG 的边权后，当访问 GGG 时又会累加边 GGG→FFF 的边权），故需要在累加某条边的边权后将这条边删除（即将反向边的边权设为 0），以避免走回头路、重复计算边权。

④ 本题也可以使用并查集解决。在使用并查集时，只要注意合并函数中需要总是保持点权更大的结点为集合的根结点（原先的合并函数是随意指定其中一个根结点为合并后集合的根结点），就能符合题目的要求。而为了达到题目对总边权与成员人数的要求，需要定义两个数组：一个数组用来存放以当前结点为根结点的集合的总边权；另一个数组用来存放以当前结点为根结点的集合中的成员人数。这样当所有通话记录合并处理完毕后，这两个数组就自动存放了每个集合的总边权和成员人数，再根据题意进行筛选即可，这里不再给出相关代码。

代码如下：

```cpp
#include <iostream>
#include <string>
#include <map>
using namespace std;

// 总人数，由于通话记录的条数最多有1000条，这意味着不同的人可能有2000人

const int maxn = 2010; //总人数
const int INF=1000000000; // 无穷大

map<int,string> intToString; // 编号⟶ 姓名
map<string,int> stringToInt; //姓名--->编号
map<string,int> Gang; //head--->人数
int G[maxn][maxn] = {0},weight[maxn]={0}; // 邻接矩阵G、点权weight
```

```
int n,k,numPerson = 0; //边数n、下限k、总人数numPerson
bool vis[maxn] = {false}; //标记是否被访问

// DFS 函数访问单个连通块，nowVisit为当前访问的编号
//head 为头目， numMember为成员编号，totalValue为连通块的总边权，均为引用
void DFS(int nowVisit, int& head, int& numMember, int&
totalValue){
    numMember++; //成员人数+1
    vis[nowVisit] = true;  //标记nowVisit已访问
    if(weight[nowVisit] > weight[head]){  // weight是点权
        head = nowVisit; // 当前访问结点的点权大于头目的点权，则更新头目
    }

    // G[][] :邻接矩阵G
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(G[nowVisit][i] > 0){  // 如果从nowVisit能到达i
            totalValue += G[nowVisit][i]; //连通块的总边权增加该边权
            G[nowVisit][i] = G[i][nowVisit] = 0; //删除这边，防止回
头
            if(vis[i] == false){ // 如果i未被访问，则递归访问i
                DFS(i,head,numMember,totalValue);
            }
        }
    }
}


//DFSTrave函数遍历整个图，获取每个连通块的信息
void DFSTrave(){
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(vis[i] == false){ // 如果i未被访问
            int head = i,numMember = 0, totalValue = 0; //头目、成
员数、总边权
            DFS(i,head,numMember,totalValue);
            if(numMember > 2 && totalValue > k){ //成员数大于2,且总
边权大于K
                //head人数为numMember
                //map<string,int> Gang; //head--->人数
                // 定义map<string,int>，来建立团伙头目的姓名与成员人数的
映射
                //map<int,string> intToString; // 编号⟶ 姓名
                 Gang[intToString[head]] = numMember;
```

```
            }
        }
    }
}

//change函数返回姓名str对应的编号
int change(string str){
    if(stringToInt.find(str) != stringToInt.end()){
        return stringToInt[str];  //返回编号
    }else{
        stringToInt[str] = numPerson; //str的编号为numPerson
        intToString[numPerson] = str; //numPerson对应str
        return numPerson++;  //总人数+1
    }
}

int main(){
    int w;
    string str1,str2;
    cin>>n>>k;
    for(int i=0;i<n;i++){
        cin>>str1>>str2>>w; //输入的两个端点和点权
        int id1 = change(str1); // 将str1 转换为编号Id1
        int id2 = change(str2);
        weight[id1] += w; //Id1的点权增加w
        weight[id2] += w;
        G[id1][id2] += w; //边id1——>id2的边权增加w
        G[id2][id1] += w;
    }

    DFSTrave(); //遍历整个图的所有连通块，获取Gang的信息

    cout<<Gang.size() <<endl; //Gang的个数
    map<string,int>::iterator it;
    for(it = Gang.begin();it!=Gang.end();it++){ //遍历所有Gang
        cout<<it->first<<" "<<it->second<<endl; //输出信息
    }
    return 0;
}
```

知识点补充：

1. map.end() 指向map的最后一个元素 之后 的地址，无论执行map.erase(iter)还是map.add(key, value)， map.end()所返回的值永远不会发生变化，都是指向同一块内存。
1. map.begin() 指向map的第一个元素， map.begin() 可能随着map.erase(iter)或是map.add(key, value)操作而发生改变。例如当第一个元素被删除后，map.begin() 就发生了改变，指向原来第一个元素之后的那个元素了。或是如果新插入一个键值对，该键值对的key放到btree（我们假设map内部是由btree实现的，实际上也可能有别的实现方式）中会排在map.begin()->first的前面，那么 map.begin() 也会指向新插入的这个键值对了。
1. map.erase(iter) 执行后，当前iter就失去意义了，再执行++iter就会出问题。

```
#include <iostream>
#include <string>
#include <map>
using namespace std;

// 总人数，由于通话记录的条数最多有1000条，这意味着不同的人可能有2000人

const int maxn = 2010; //总人数
const int INF=1000000000; // 无穷大

map<int,string> intToString; // 编号—> 姓名
map<string,int> stringToInt; //姓名--->编号
map<string,int> Gang; //head--->人数
int G[maxn][maxn] = {0},weight[maxn]={0}; // 邻接矩阵G、点权weight
int n,k,numPerson = 0; //边数n、下限k、总人数numPerson
bool vis[maxn] = {false}; //标记是否被访问

// DFS 函数访问单个连通块，nowVisit为当前访问的编号
//head 为头目， numMember为成员编号，totalValue为连通块的总边权，均为引用
void DFS(int nowVisit, int& head, int& numMember, int&
totalValue){

    numMember++; //成员人数+1
```

```
    vis[nowVisit] = true;  //标记nowVisit已访问

    if(weight[nowVisit] > weight[head]){  // weight是点权
        head = nowVisit; // 当前访问结点的点权大于头目的点权，则更新头目
    }

    // G[][] :邻接矩阵G
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(G[nowVisit][i] > 0){  // 如果从nowVisit能到达i
            totalValue += G[nowVisit][i]; //连通块的总边权增加该边权
//          //  ! ! !
//          G[nowVisit][i] = G[i][nowVisit] = 0; //删除这边，防止回头

            if(vis[i] == false){ // 如果i未被访问，则递归访问i
                DFS(i,head,numMember,totalValue);
            }
        }
    }
}


//DFSTrave函数遍历整个图，获取每个连通块的信息
void DFSTrave(){
    for(int i=0;i<numPerson;i++){ //枚举所有人
        if(vis[i] == false){ // 如果i未被访问
            int head = i,numMember = 0, totalValue = 0; //头目、成员数、总边权
            DFS(i,head,numMember,totalValue);
            if(numMember > 2 && totalValue/2 > k){ //成员数大于2,且总边权大于K
                //head人数为numMember
                //map<string,int> Gang; //head--->人数
                // 定义map<string,int>，来建立团伙头目的姓名与成员人数的映射
                //map<int,string> intToString; // 编号⟶ 姓名
                 Gang[intToString[head]] = numMember;
            }
        }
    }
}
```

```
//change函数返回姓名str对应的编号
int change(string str){
    //  已经处理过，转换过了编号
    if(stringToInt.find(str) != stringToInt.end()){
        return stringToInt[str];  //返回编号
    }else{
        // 转换成编号
        stringToInt[str] = numPerson; //str的编号为numPerson
        intToString[numPerson] = str; //numPerson对应str
        return numPerson++;  //总人数+1
    }
}

int main(){
    int w;
    string str1,str2;
    cin>>n>>k;
    for(int i=0;i<n;i++){
        cin>>str1>>str2>>w; //输入的两个端点和点权
        int id1 = change(str1); // 将str1 转换为编号Id1
        int id2 = change(str2);
        weight[id1] += w; //Id1的点权增加w
        weight[id2] += w;
        G[id1][id2] += w; //边id1——>id2的边权增加w
        G[id2][id1] += w;
    }

    DFSTrave(); //遍历整个图的所有连通块，获取Gang的信息

    cout<<Gang.size() <<endl; //Gang的个数

    map<string,int>::iterator it;
    for(it = Gang.begin();it!=Gang.end();it++){ //遍历所有Gang
        cout<<it->first<<" "<<it->second<<endl; //输出信息
    }
    return 0;
}
```