**Решение:**

Това решение ползва една структура от данни представяща разбиване на множество, която позволява лесно да се извършва операцията обединение на две групи от разбиването – наричат я “disjoint sets” или “union find”.

Всяка група се представя като дърво от елементи (пазим си само бащата на всеки възел), а коренът му се явява „официалния представител”, или „големия брат”, или „вожда” на всички под него. Когато обединяваме групи, просто закачаме вожда на едната за вожда на другата – така че да станат баща и син.



Можем да намерим вожда на произволен елемент като тръгнем нагоре по дървото до корена. Два елемента са в една и съща група от разбиването точно когато имат един и същ вожд.

Смисълът който влагаме е следния: разбиваме множеството от агенти на групи, в рамките на всяка от които сме сигурни че агентите са приятели. Това представяне е удачно, защото приятелството (~) е релация на еквивалентност:

* Аз съм приятел със себе си (a~a)
* За двама приятели чувствата са взаимни (a~b => b~a)
* Приятелите на моите приятели са и мои приятели (a~b & b~c => a~c)

Трябва ни и начин да представяме знанието си че две групи приятели са врагове помежду си. Враждата (\*) при този „двуполюсен модел” (има само две чужди държави) не е нищо друго освен антоним на приятелството:

* Ако не си ми приятел, значи си ми враг, и обратно (a~b <=> !(a\*b))

От изброените по-горе принципи, могат да се изведат интересни следствия:

* За двама врагове чувствата са взаимни (a\*b => b\*a)
* Враговете на моите приятелите са и мои врагове (a~b & b\*c => a\*c)
* Враговете на моите врагове са ми приятели (a\*b & b\*c => a~c)

Доказват се лесно с допускане на противното.

И така, оказва се достатъчно да си пазим най-много по един враг за група приятели. Няма смисъл да пазим по два врага, защото ще е ясно че те са приятели помежду си и веднага можем да ги обединим.

За конкретната реализация ползваме два масива:

friend[x] – Бащата на x в дървото (гората) на приятелството.

enemy[x] – Ще го ползваме само ако x е вожд – enemy[x] сочи към вожда на вражеската група приятели. Ако не знаем враг за групата на x, тогава enemy[x] ще бъде -1. Винаги ще поддържаме масива такъв че ако enemy[x] == y, то и enemy[y] == x.

Ще прилагаме последователно информацията от дадените телефонни разговори, т.е. ще прилагаме операциите: „направи x и y приятели”, и „направи x и y врагове”. Да означим с getLeader(x) функцията която ни връща вожда на x, а за всяка стъпка нека:

lx = getLeader(x);

ly = getLeader(y);

ex = enemy[lx];

ey = enemy[ly];

Разглеждаме следните случаи:

* „направи X и Y приятели”
	+ ако lx == ly, те вече са си приятели, не правим нищо.
	+ ако enemy[lx] == ly, нещо не е наред – стигнали сме до противоречие. Това не трябва да се случва според условието на задачата.
	+ иначе, сприятеляваме lx и ly, определяйки едното от тях за вожд и закачайки другото за вожда (нека за определеност закачим ly за lx), и след това:
		- ако ex != -1 и ey != -1, сприятеляваме ex и ey, а този от тях, който е нов вожд, го правим взаимен враг с lx.
		- ако точно едно от ex и ey е различно от -1, правим го взаимен враг с lx.
		- ако ex == -1 и ey == -1, всичко е ok, не правим нищо.
* „направи X и Y врагове”
	+ ако enemy[lx] == ly, те вече са си врагове, не правим нищо.
	+ ако lx == ly, имаме противоречие.
	+ иначе, правим lx и ly взаимни врагове, като преди това: ако ex != -1, сприятеляваме го с ly; и ако ey != -1, сприятеляваме го с lx.

И накрая, отговаряме на въпросите за отношенията произволни x и y:

* ако getLeader(x) == getLeader(y), те са приятели.
* ако enemy[getLeader(x)] == getLeader(y), те са врагове.
* иначе – не е ясно.