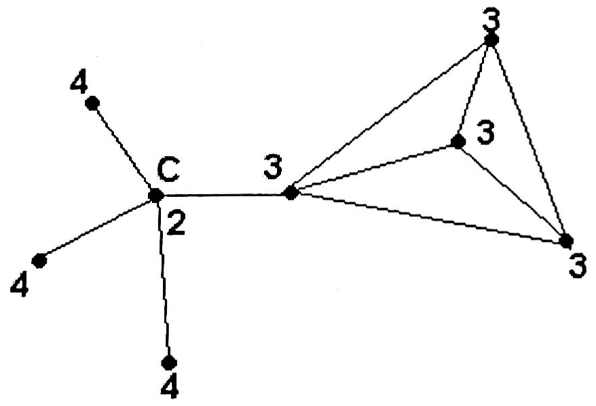
See notes page 2

B: The Heart of the Country

The nation of Graphia is at war. The neighboring nations have for long watched in jealousy as Graphia erected prosperous cities and connected them with a network of highways. Now they want a piece of the pie.

Graphia consists of several cities, connected by highways. Graphian terrain is rough, so the only way to move between the cities is along the highways. Each city has a certain number of troops quartered there. Graphia's military command knows that it will require a certain number of troops, `K`, to defend any city. They can defend a city with the troops stationed there, supported by the troops in any other city which is directly connected with a highway, with no cities in between. Any troops further away than that simply cannot get there in time. They also know that their enemies will onlyattack one city at a time – so the troops in a city can be used to defend that city, as well as any of its neighbors. However, if a city can't be defended, then the military command must assume that the troops quartered in that city will be captured, and cannot aid in the defense of Graphia. In the case below, suppose *K*=10. City C might seem well defended, but it will eventually fall.



Graphia's leadership wants to identify the Heart of their country – the largest possible group of cities that can mutually defend each other, even if all of the other cities fall.

More formally, a city is defensible if it can draw a total of at least `K` troops from itself, and from cities *directly adjacent* to it. A set of cities is defensible if every city in it is defensible, using *only* troops from itself and adjacent cities *in that set*. The Heart of the country is the largest possible defensible set of cities - that is, no other defensible set of cities has more cities in it.

Input

There will be several data sets. Each set begins with two integers, *N* and *K*, where *N* is the number of cities (`3 <= N <= 1000`), and *K* is the number of troops required to defend a city. The cities are numbered `0` through `N-1`.

On the next *N* lines are descriptions of the cities, starting with city `0`. Each of the city description lines begins with an integer *T*, indicating the number of troops quartered in that city (`0 <= T <= 10000`). This is followed by an integer *M*, indicating the number of highways going out of that city, and then *M* integers, indicating the cities those highways go to. No two highways will go from and to the same cities, so every city in each list will be unique. No highway will loop from a city back to the same city. The highways go both ways, so that if city *I* is in city *J*'s list, then it's guaranteed that city *J* will be in city *I*'s list in the input. The input will end with a line with two space-separated `0`'s.

Output

For each data set, print two integers on a single line: The number of cities in the heart of the country, and the number of troops in the heart of the country. Print a space between the integers. There should be no blank lines between outputs.

Sample Input

```
4 900
100 2 1 2
200 2 0 3
500 2 0 3
1000 2 1 2
4 900
100 3 1 2 3
200 3 0 3 2
500 3 1 3 0
1000 3 2 1 0
0 0
```

Sample Output

```
3 1700
4 1800
```

Heart of the Country (ACM SE Regional 2008)

(Notes by Kip Irvine)

This problem gives you a list of connected cities with individual troop levels, and asks you to determine which cities are defensible. A city is defensible if the total of its troops and its immediate neighbors are at least a certain required level. What makes it iteresting is that when a city falls because it cannot be defended, you have to update the numbers of available troops for all of its immediate neighbors.

The general strategy is to build a connected graph and then iterate through the cities one by one. For each city, evaluate its number of available troops (its own troops plus its neighbors). If this total is too small, mark the city as indefensible, and add it to a queue. After doing this, begin removing the cities from the front of the queue one by one; for each city in the queue, subtract its number of troops from the total available troops of its neighbors. If this causes any neighboring city to become indefensible, add that city to the back of the queue. Keep doing this until the queue is empty. Finally, iterate through the list of cities and count the ones who have not been marked indefensible, add their number of troops to a total. After this loop has finished, print the count and the total troops.

```
                                              queue Q
for each dataset:
    read n
    read k
    for i = 1 to n
        read citytroops[i]
        read m
        for j = 1 to m                                    adjacent
            add (adj[i], read c)                          c = city ID
    next

    for i = 1 to n
        totaltroops[i] = citytroops[i]
        for each k in adj[i]
            totaltroops[i] += citytroops[k]
        if totaltroops[i] < k
            indefensible[i] = true
            Q.add(i)
        else
            indefensible[i] = false
    next
    while (Q.size > 0)
        c = Q.removeFirst()
        for each ngh in adj[c]                            // ngh = neighbors
            totaltroops[ngh] -= citytroops[c]
    if  totaltroops[ngh] < k  and not indefensible[ngh]
        indefensible[ngh] = true
        Q.addlast(ngh)

cities = 0 , troops = 0
for i = 1 to N
    if not indefensible[i]
        cities ++
        troops += citytroops[i]

print cities, troops
```

C - Knight Moves

A friend is doing research on the *Traveling Knight Problem (TKP)*, in which you find the shortest closed tour of knight moves that visits each square of a given set of *n* squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that the opposite is true. So you offer him to write a program that solves the "difficult" part. Your job is to write a program that takes two squares *a* and *b* as input and then determines the number of knight moves on a shortest route from *a* to *b*.

Input

The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (`a-h`) representing the column and a digit (`1-8`) representing the row on the chessboard.

Output

For each test case, print one line saying "`To get from xx to yy takes n knight moves.`".

Sample Input

```
e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6
```

Sample Output

```
To get from e2 to e4 takes 2 knight moves.
To get from a1 to b2 takes 4 knight moves.
To get from b2 to c3 takes 2 knight moves.
To get from a1 to h8 takes 6 knight moves.
To get from a1 to h7 takes 5 knight moves.
To get from h8 to a1 takes 6 knight moves.
To get from b1 to c3 takes 1 knight moves.
To get from f6 to f6 takes 0 knight moves.
```