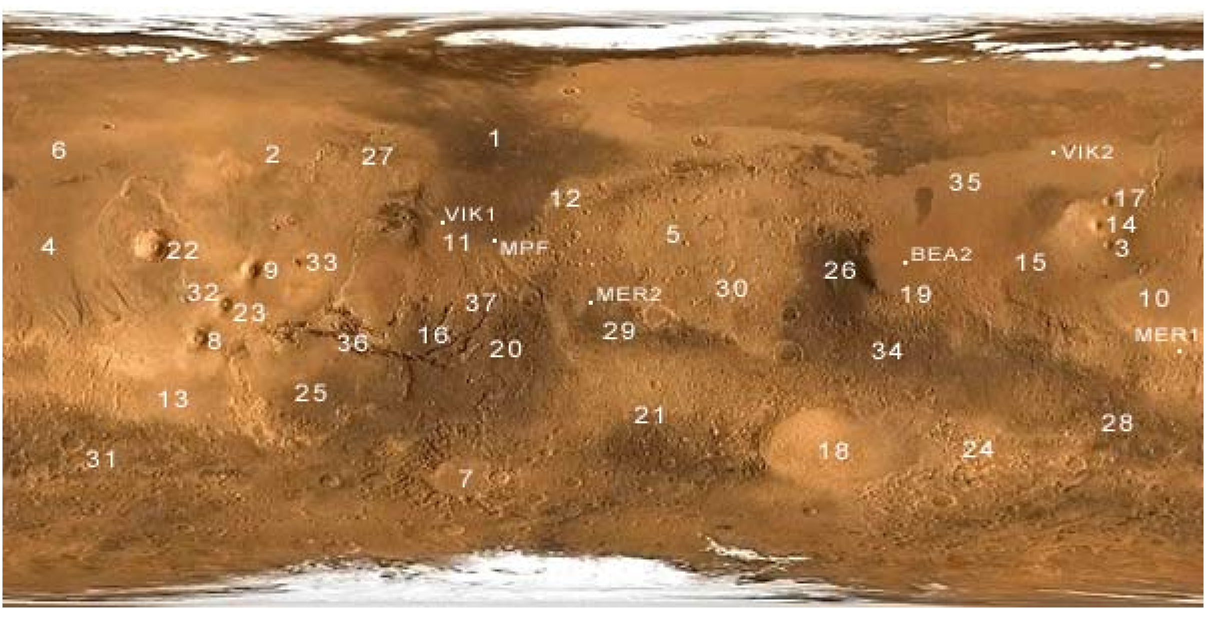
Mars is our neighbor in the solar system, about 1.5 times as far from the sun as the earth. In the past years we have seen several unmanned expeditions to the planet, some more succesful than others.



The surface of Mars (picture: http://www.marsbase.net/m/mars-map.php)

Imagine some time in the (near) future when Mars is colonised with several settlements scattered around the globe. Each settlement is equiped with a large array of solar panels and batteries to supply the energy needs of its inhabitants. Transport between the settlements is maintained by electric buggies that deliver people and goods around the planet. The range of these buggies is limited by the capacity of their battery and the payload they carry. To go from one place to the other, they may have to go from settlement to settlement, recharging the battery for every interval.

The Problem

You are hired by the Mars Public Transportation System (MPTS) to write their route planning software. Based on a list of settlements, the program should process transportation request for individual buggies and print a list of intermediate settlements where the buggy will recharge. Each interval along the trip should, of course, be shorter or equal to the range of the particular buggy, but within that restriction the total length of the trip should be as short as possible. In case it is impossible to make the trip with a buggy with the given range, the program should calculate the minimum range a buggy should have to make the trip.

The Facts

For this problem we'll consider Mars to be a perfectly flat sphere with a radius of 3390 kilometer. Places on Mars are localised by giving their latitude and longitude in radians. The northpole is at latitude $\pi/2$, the equator has latitude 0 and the southpole is at latitude $-\pi/2$. Longitude is given from east to west with a value from 0 to 2π, starting at an arbitrary meridian called the zero-meridian.

Between two places a buggy will always take the shortest possible route, that is along a great circle. MPTS identifies settlements with a unique string of upto 20 characters, the location code. Legal characters for a location code are: upper- and lowercase letters, digits and the underscore; space characters are not allowed. For every settlement the location of the MPTS transfer station is listed by giving the location code and its latitude and longitude in radians, as defined above. Loading, unloading and recharging takes place at these transfer stations, which can be considered dimensionless points on the surface of Mars.

Considering accuracy, MPTS has a regulation that you should follow in your program:

Distances between settlements are to be treated as an integral number of kilometers when adding, comparing and reporting them. When you calculate the distance between two settlements, it is unavoidable to make use of floating point calculations. Once calculated, however, this distance should be rounded to the nearest integral number of kilometers. Surprizingly enough, distances between settlements on Mars are such that their fractional part is never so close to 0.5 kilometer that rounding would become ambiguous.

Input

The input will consist of several scenarios.

Each scenario starts with an integer between 2 and 100, the number of locations on Mars, on a line by itself. Then there is a line for each location stating its MPTS location code, its latitude and its longitude, in that order. Latitude and longitude are given in radians with 6 decimals following the decimal dot.

Then follows a list of requests. The number of requests (between 1 and 100) appears on a line by itself, followed by one line for each request stating the location code of the starting settlement, the location code of the destination settlement and the range of the buggy, in that order. The two location codes will always be different. The range will be given in kilometers.

Items on a line will be separated by one or more spaces. Lines will never be longer than 128 characters.

A scenario with 0 locations and 0 requests will terminate the input. This scenario should not be processed.

Output

For each scenario state the scenario number (starting from 1) on a line by itself in the format '`Scenario` X`:`', followed by a line containing 30 hyphens ('-').

Then for each request, state the request in the following format on a line by itself: '`From` X `to` Y `with range` Z `km:`'. If a route from start to destination is possible with that range, output all settlements, including start and destination, in the order visited together with their cummulative distance along the shortest possible route, in the format 'X `at` Y `km`'. If no route is possible for that range, output one line: '`No route for this range, minimum required range is` X `km.`'. Follow the output of each request with a line containing 30 hyphens (`-`).

Separate two scenarios by a blank line.

Distances should be printed in kilometers. X, Y and Z in the format strings should be replaced by their appropriate values and output should be printed without the surrounding quotes.

A special corrector will check your program's output, so if more than one possible answer exists, print any one. When printing a route it is OK to add extra spaces between items in a line of output as long as the line is never longer than 128 characters.

Epilogue (you might need it to solve the problem)

There are many formulas to calculate the distance between two points, given their latitude and longitude on a perfect sphere. One of the simplest is based on the 'Law of Cosines':

```
a = sin(lat1) * sin(lat2)
b = cos(lat1) * cos(lat2) * cos(lon2 - lon1)
c = arccos(a + b)
d = R * c
```

Here R is the radius of the sphere and d is the distance; a, b and c are intermediates.

This formula, although correct, suffers from accuracy errors when implemented on a computer (or calculated by hand using slide rule or lookup tables). To cope with the inaccuracy, navigators use a special set of trigonometric functions, one being the *versine* which is defined as: $versine(x) = 1-cos(x)$, and an other, *haversine*, being half that value. It is easy to see that

$$haversine(x) = (1 - cos(x))/2 = sin^2(x/2)$$

The so called 'Haversine Formula' uses this function to calculate the distance, without the loss of accuracy from the formula stated above:

```
a = haversine(lat2 - lat1)
b = cos(lat1) * cos(lat2) * haversine(lon2 - lon1)
c = 2 * atan2(sqrt(a + b), sqrt(1 - a - b))
d = R * c
```

Here $sqrt(x)$ is the square root function, and $atan2(num, den)$ is the arctangens function $atan(num/den)$ that also gives the correct answer when $den = 0$.

Sample Input

```
11
Lousberg         0.500000    1.000000
Rasschaert       0.000000    0.500000
Lubbers          0.000000    1.000000
van_den_Hoogen  -0.500000    1.000000
Bink             0.000000    1.500000
van_de_Kieft     0.200000    0.800000
Bronkhorst      -0.300000    1.100000
van_Dijk         0.001000    1.001000
Zijlstra         0.010000    1.020000
Duponselle      -0.250000    0.900000
Ramnath         -0.400000    0.600000
3
Lousberg        van_den_Hoogen      1200
Rasschaert      Ramnath             1000
Lubbers         van_Dijk              10
0
0
```

Sample Output

```
Scenario 1:
------------------------------
From Lousberg to van_den_Hoogen with range 1200 km:
Lousberg              at     0 km.
van_de_Kieft          at  1198 km.
Lubbers               at  2154 km.
Duponselle            at  3065 km.
van_den_Hoogen        at  3969 km.
------------------------------
From Rasschaert to Ramnath with range 1000 km:
No route for this range, minimum required range is 1217 km.
------------------------------
From Lubbers to van_Dijk with range 10 km:
Lubbers               at     0 km.
van_Dijk              at     5 km.
------------------------------
```