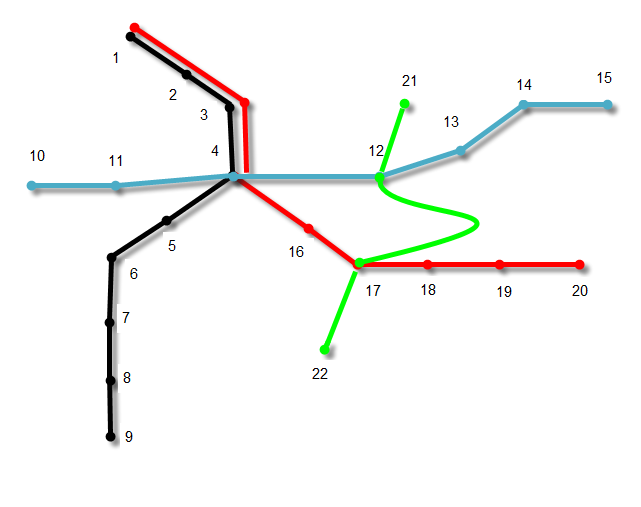
<https://www.techgig.com/challenge/contiquest>

Quickest Metro Route (100 Marks)

Given a metro map, find the quickest route between the given origin station to destination station.  
                  
  
**Points to remember:**1. All the routes are bi-directional.  
2. In case the train starts and ends at a same station, in case of a circular route, it ll halt after reaching that particular station for 120 seconds. Anyone who wants to catch the train in the opposite direction (according to the line definition in program input) will have to wait for 120 seconds after reaching the station.   
3. Each halt at a station adds 15 seconds to the total travel time.  
4. A route can include more than one lines.  
5. Transfer between two lines is possible whenever they share a station. For Eg.   
         a. Transfer between Green and Blue lines is possible at station 12  
         b. Transfer between Green and Red lines is possible at station 17  
6. Each transfer would add 30 seconds to the total travel time.  
7. The resulting route should not be broken.  
8. Each station should be visited at most one time in the route.  
9. When there are multiple paths between any two stations, the winner is chosen based on these rules:  
        a. The quickest path wins.  
        b. If more than one paths are taking the same time, the path with less transfers wins.  
        c. If the chosen paths have same number of transfers, the one with less halts wins.  
        d. If the chosen paths have same number of transfers and same number of halts, the one with lesser wait time wins. Wait time is the time a passenger has to spend waiting at the station rather than in the metro. 

**Input Format**

You will be given a function which contains two arguments - **Argument 1 -**An array of strings(Lines- Black, Blue, Red, Green) of the form -  
N  
<< Station\_1 >-< Station\_2 >-< Time\_in\_sec >#< Station\_2 >-< Station\_3 >-< Time\_in\_sec ># >  
 < < Station\_1 >-< Station\_2 >-< Time\_in\_sec >#< Station\_2 >-< Station\_3 >-< Time\_in\_sec ># >,  
< < Station\_1 >-< Station\_2 >-< Time\_in\_sec >#< Station\_2 >-< Station\_3 >-< Time\_in\_sec ># >  
 < < Station\_1 >-< Station\_2 >-< Time\_in\_sec >#< Station\_2 >-< Station\_3 >-< Time\_in\_sec ># >  
  
Here each string defines the stations transition in a line. All the station transition of each line are separated by #. Each station transition is a tuple of three entries (separated by '-'), i.e. the origin station, the destination station and the time it takes to travel from origin to destination. Note that the time is mentioned in seconds.  
For example:41-2-30#2-3-25#3-4-30#4-5-45#5-6-30#6-7-15#7-8-60#8-9-40  
10-11-45#11-4-60#4-12-60#12-13-45#13-14-30#14-15-35  
1-3-40#3-4-25#4-16-30#16-17-15#17-18-20#18-19-30#19-20-25  
21-12-30#12-17-180#17-22-45  
  
**Argument 2 -**  
A string of containing origin\_station and destination\_station separated by '#' < origin\_station >#< destination\_station >  
For example:  
12#18

**Constraints**

1 <= |S| <= 1000

**Output Format**

The output will be an array of strings of the same form as Input 1. Here each string defines the chosen transition of a line, which are coming in quickest route between the given origin station to destination station.  
  
Note:  
If no station transition chosen in route from a line, output array should contains "NC" for that line.For example - {NC, 12-4-60, 4-16-30#16-17-15#17-18-20, NC}  
Since all edges between stations are bi-directional, the program should reverse the direction of edges in the result so that the route form origin to destination is connected as well as following in the right direction.  
  
For example:  
For Input- {5-6-30#6-7-15#7-8-60#8-9-40} 9#5  
Output should be- {9-8-40#8-7-60#7-6-15#6-5-30}  
Note that all the edges in the result are in opposite direction as compared to the input string.

**Sample TestCase 1**

Input

4

1-2-30#2-3-25#3-4-30#4-5-45#5-6-30#6-7-15#7-8-60#8-9-40

10-11-45#11-4-60#4-12-60#12-13-45#13-14-30#14-15-35

1-3-40#3-4-25#4-16-30#16-17-15#17-18-20#18-19-30#19-20-25

21-12-30#12-17-180#17-22-45

12#18

Output

NC

12-4-60

4-16-30#16-17-15#17-18-20

NC

Explanation

In the example given above, there are two paths between stations 12 and 18, i.e.  
1. [Green- {12-17-180}, Red- {17-18-20}] Total time- 200 + 60 = 260 Seconds  
2. [Blue- {12-4-60}, Red- {4-16-30, 16-17-15, 17-18-20}]  
Total time- 125 + 30 + 60 = 215 Seconds.

**Time Limit(X):**

1.00 sec(s) for each input.

**Memory Limit:**

512 MB

**Source Limit:**

100 KB

**Allowed Languages:**

C, C++, C++11, C++14, C#, Java, Java 8, PHP, PHP 7, Python, Python 3, Perl, Ruby, Node Js, Scala, Clojure, Haskell, Lua, Erlang, Swift, VBnet, Js, Objc, Pascal, Go, F#, D, Groovy, Tcl, Ocaml, Smalltalk, Cobol, Racket, Bash, GNU Octave, Rust, Common LISP, R, Julia, Fortran, Ada, Prolog, Icon, Elixir, CoffeeScript, Brainfuck, Pypy, Lolcode, Nim, Picolisp, Pike, Whitespace

[Judge Environment](https://www.techgig.com/platform-faq)