Problem 1 – Maximal Path

We are given a tree of **N** nodes, each containing a distinct integer number (between 1 and 2147483640, inclusive) and optionally a set of descendent nodes. Write a program that finds a path from some leaf of the tree to another (different) leaf of the tree with maximal sum of its nodes and prints this sum.

Input

The input data should be read from the console.

The first input line contains **N** - the number of nodes in the tree.

At the next **N-1** lines there are pairs of numbers in format (p_1 <- p_2) each meaning that node $\mathbf{p_1}$ is parent of the node $\mathbf{p_2}$. See the example bellow.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

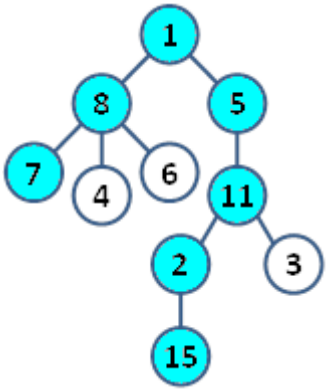
The output data should be printed on the console.

At the only output line you should print the maximal sum of nodes found.

Constraints

- **N** will be between 2 and 3000, inclusive.
- Allowed working time for your program: 0.80 seconds.
- Allowed memory: 16 MB.

Example

Input example	Output example	Explanation
10 (5 <- 11) (1 <- 8) (11 <- 3) (8 <- 7) (1 <- 5) (11 <- 2) (8 <- 6) (2 <- 15) (8 <- 4)	49	The maximal path is: 7 -> 8 -> 1 -> 5 -> 11 -> 2 -> 15 which is same as: 15 -> 2 -> 11 -> 5 -> 1 -> 8 -> 7  7+8+1+5+11+2+15=49

Problem 2 – Shopping Center

A **shopping center** keeps a set of products. Each product has name, price and producer. Your task is to model the shopping center and design a data structure holding the products. Write a program that executes **N** commands, given in the input (a single command at a line):

- **AddProduct name;price;producer** – adds a product by given name, price and producer. If a product with the same name / producer/ price already exists, the newly added product does not affect the existing ones (duplicates are allowed). As a result the command prints “**Product added**”.
- **DeleteProducts name;producer** – deletes a product (or several products) by given product name and producer. As a result the command prints “**X products deleted**” where **X** is the number of deleted products or “**No products found**” if no such products exist.
- **DeleteProducts producer** – deletes all products by given producer. As a result the command prints “**X products deleted**” where **X** is the number of deleted products or “**No products found**” if no such products exist.
- **FindProductsByName name** – finds all products by given product name. As a result the command prints a list of products in format **{name;producer;price}**. The list items should be ordered in alphabetical order. You should print each product on a single line**.** If no products exist with the specified name, the command prints “**No products found**”.
- **FindProductsByPriceRange fromPrice;toPrice** – finds all products whose price is greater or equal than **fromPrice** and less or equal than **toPrice**. As a result the command prints a list of products in format **{name;producer;price}**. The list items should be ordered in alphabetical order. You should print each product on a single line**.** If no products exist within the specified price range, the command prints “**No products found**”.
- **FindProductsByProducer producer** – finds all products by given producer. As a result the command prints a list of products in format **{name;producer;price}**. The list items should be ordered in alphabetical order. You should print each product on a single line**.** If no products exist by the specified producer, the command prints “**No products found**”.

See the examples bellow.

Input

The input data should be read from the console. On the first line you will be given the number **N** of the commands. On each of the next **N** lines you will be given a command in the format described above.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console. The output should contain the output from each command from the input.

Constraints

- **N** will be between 1 and 100 050, inclusive.
- All strings specified in the commands (e.g. product names and producers) consist of alphabetical characters, numbers and spaces.

- Prices are given as real numbers with up to 2 digits after the decimal point, (e.g. 133.58, or 320)
- The ‘.’ symbol is used as decimal separator.
- Prices should be printed with exactly 2 digits after the decimal point (e.g. 320.30 instead of 320.3).
- Allowed working time for your program: 2.50 seconds. Allowed memory: 256 MB.
- **Important: Please use StringBuilder to store your output and print it at the end of the input.**

Example

Input example	Output example
14 AddProduct IdeaPad Z560;1536.50;Lenovo AddProduct ThinkPad T410;3000;Lenovo AddProduct VAIO Z13;4099.99;Sony AddProduct CLS 63 AMG;200000;Mercedes FindProductsByName CLS 63 AMG FindProductsByName CLS 63 FindProductsByName cls 63 amg AddProduct 320i;10000;BMW FindProductsByName 320i AddProduct G560;999;Lenovo FindProductsByProducer Lenovo DeleteProducts Lenovo FindProductsByProducer Lenovo FindProductsByPriceRange 100000;200000	Product added Product added Product added Product added {CLS 63 AMG;Mercedes;200000.00} No products found No products found Product added {320i;BMW;10000.00} Product added {G560;Lenovo;999.00} {IdeaPad Z560;Lenovo;1536.50} {ThinkPad T410;Lenovo;3000.00} 3 products deleted No products found {CLS 63 AMG;Mercedes;200000.00}

Problem 3 – Friends in Need

Pesho hates ordinary walking. This is so not because he is lazy, but he much more likes jumping and running. One day in Pesho's village an unexpected snow started to rain and caught Pesho unprepared – he was not wearing the appropriate clothes and shoes. But Pesho hated ordinary walking so much, that he continued jumping, instead of walking. While jumping the way home, Pesho slipped, fell down and hurt his leg. Then Pesho's friends in need started wondering: "Which hospital is closest to our homes, so when we take Pesho there, we will walk the minimal distance to our homes?" Your task is to help Pesho's friends, before he falls down again!

As input data you will receive the homes of Pesho's friends, the hospitals and the distances between them. Your task is to find the hospital that is closest to the friends' homes. A distance from hospital to homes is the sum of the distances from the hospital to each of the friends' homes. You should find the smallest distance.

Input

The input data is read from the standard input (the console).

On the first line will be read three integer numbers N, M and H (separated by a single white space), that represent corresponding the number of points on the map (both hospitals and homes), the number of streets between points and the number of hospitals.

On the second line will be H integer numbers, the points on the map that are hospitals. All the rest of the points are homes.

On the next M lines will be the streets. Each line contains three positive integer numbers F, S and D. That means that there is a street between points F and S, and the distance between them is D. All the streets are directional, i.e. if a street from A to B exists, there is a street from B to A.

The input data will be always valid in the format described.

Output

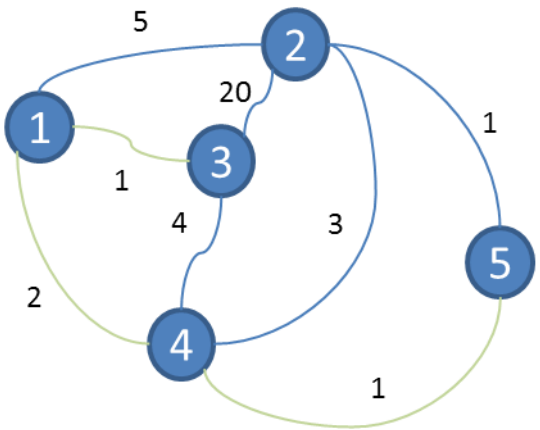
The output data should be written on the standard output (the console).

As output data you should write a single positive integer number – the minimal possible distance from one hospital to each of the Pesho's friends' homes.

Constraints

- 0 < N < 10000
- N - 1 <= M < 5 * N
- 0 < H < 100
- A way between a point and all other points will always exists
- A single street is no longer than 20.
- The allowed execution time is 2.5 seconds. Allowed memory used is 16 MB.

Examples

Sample Input	Sample output	Explanation
3 2 1 1 1 2 1 3 2 2	4	1 is the only hospital. The minimal distance from 1 to 2 is 1, and the distance from 1 to 3 is 3. The total distance to the hospital is 4.
5 8 2 1 2 1 2 5 4 1 2 1 3 1 3 4 4 4 5 1 2 4 3 5 2 1 2 3 20	6	We have two hospitals: 1 and 2 Distances from 1: to 3: 1, to 4: 2, to 5: 3, total 6. Distances from 2: to 3: 5, to 4: 2, to 5: 1, total 8. The winner is hospital 2 and the distance is 6 

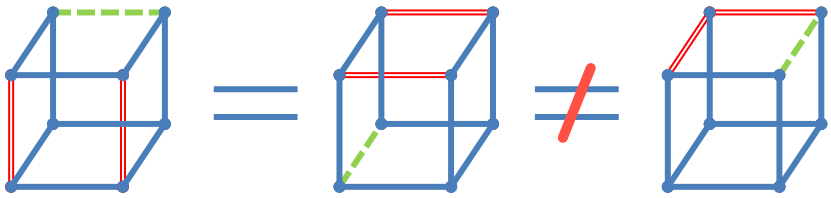
Problem 4 – Cubes

You are given 12 sticks of the same length, each colored with a color in range [1..4].

Write a program to find the number of different cubes that can be built using these sticks.

Note that two cubes are equal if a sequence of rotations exists that transforms the first cube to the second.

For example the first two cubes below are equal (after two rotations) but are different than the third cube:



Input

The input data should be read from the console.

The input contains 12 numbers in range [1..4] in a single line separated by spaces.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output data should be printed on the console.

The output should consist of single integer number – the number of different cubes that can be built using the given sticks.

Constraints

- The number of the sticks will be always 12 = the number of the edges of a cube.
- Each stick will be colored with one of the following colors: 1, 2, 3 or 4.
- Allowed working time for your program: 0.75 seconds.
- Allowed memory: 16 MB.

Examples

Input example	Output example
1 2 2 2 2 2 2 2 2 2 2 2	1

Input example	Output example
1 1 2 2 2 3 3 3 3 3 3 3	340

Problem 5 – Horse Matrix

We are given a square matrix of **N** x **N** cells. Some of the cells are empty ('-') and the rest are impassable ('**x**'). We need to move a horse from one position to another position in the matrix in the way horses jump in chess. At the figure it is shown where a horse located at given cell (indicated with "**o**") can go with a single jump – the 8 cells indicated with "*****".

	*		*	
*				*
		o		
*				*
	*		*	

Your task is to write a program that finds the shortest path of the horse (shortest sequence of jumps) to move from one position to another position in given square matrix by passing only through empty cells.

Note that rows are numbered from 0 from top to bottom and columns are numbered from 0 from left to right.

Input

The input data should be read from the console.

At the first input line there will be a single integer number **N**.

At the next **N** lines there are sequences of **N** characters ‘**-**‘, ‘**x**’, ‘**s**’, ‘**e**’ separated by a space (see the example below).

The cells denoted by ‘**s**’ and ‘**e**’ are respectively the starting cell and the end cell. There will be exactly one starting and end cell.

The input data will always be valid and in the described format. There is no need to check it explicitly.

Output

The output data should be printed on the console.

At the only output line the minimal number of horse jumps should be written or “**No**” if a path does not exist.

Constraints

- **N** will be between 1 and 500, inclusive.
- Allowed working time for your program: 0.10 seconds. Allowed memory: 16 MB.

Example

Input example	Output example	Explanation
4 - s e - x - x - x x - - - x - x	5	The path is: (0, 1) -> (2, 2) -> (0, 3) -> (1, 1) -> (2, 3) -> (0, 2)