DATA STRUCTURES AND ALGORITHMS

ASSIGNMENT-6
Language allowed: C

February 15, 2020

A. Strings and Games

Your friend gifted you a new game. This game is oddly based on strings. In this game, given a string, you have to find *minimum* length string you get after removing all immediate duplicates (An immediate duplicate is a substring consisting of successively occurring identical characters.) from the string. Note that the *characters are removed in the order they arrive*, i.e. left to right.

Input

The only line of input contains a string S ($1 \leq |S| \leq 3.10^6$) consisting of lowercase latin letters.

Output

Print the string after all immediate duplicates are removed. If after all operations the string length becomes zero, print "VANISHED".

---

input
abccc

output
ab

---

input
abbaccabba

output
VANISHED

---

input
ttypuuyykkjhkj

output
ypjhkj

---

input
ttyuuyykkjhkj

output
jhkj

B. Schedule it!

One of the most famous process scheduling algorithms is the *Round Robin*. In this algorithm, each process is given a *fixed time quantum* to execute on the processor. Each process is assigned a process ID and has an associated arrival time (denoting the time when it arrives in the system) and an execution time (denoting the time for which it has to execute on the processor). All arriving processes are *queued up* in the system according to their arrival time and executed serially. If 2 processes have the same arrival time, then the process having the lower ID is executed first. If a process does not complete within the given time quantum, it is put back at the tail of the ready queue (with whatever execution time is still remaining). When a process completes its execution, it is removed from the process queue, and immediately the next process is put on the processor. Note that a process can finish its execution before the time quantum duration is up. Hence, in this task, given a set of processes along with their arrival time and execution time, can you figure out the order of execution and the *total waiting time* for all the processes? (see sample case for clarity). The waiting time for a certain process is defined as follows: *the total amount of time a process has been waiting in the queue.*

Input

The first line of input contains two space-separated integers N ($1 \leq N \leq 10^4$), T ($1 \leq T \leq 15$) denoting the number of processes and the time quantum duration respectively. Each of the following N lines contains three space-separated integers P_i ($1 \leq P_i \leq 10^4$), A_i ($0 \leq A_i \leq 2.10^5$) and B_i ($1 \leq B_i \leq 10^3$) denoting the process ID, arrival time and execution time of the i^{th} process respectively. Assume that the processes are assigned IDs in ascending order of arrival time, with ties as per arrival time broken arbitrarily.

Output

In the first line, print a single integer W denoting the total waiting time of all the processes. In the second next line print a sequence of space-separated integers S, denoting the sequence in which the processes are scheduled on the processor (the j^{th} integer denotes the process ID of the process which is scheduled on the processor at the j^{th} instant).

```
input
7 2
56 0 3
60 0 4
61 1 1
65 1 1
72 1 5
75 6 2
81 7 4
output
47
56 60 61 65 72 56 60 75 81 72 81 72

explanation: The waiting times of processes 56, 60, 61, 65, 72, 75 and 81 are 6,
7, 3, 4, 14, 5 and 8 respectively.
```

C. Convert it!

In this task, you will be given an *infix expression* (consisting of integers and +, -, /, * , % operators) and you need to find its *postfix notation* adhereing to the precedence rules. Write a program to do the same. It is assured that the given sequence will be completely bracketed. Assume precedence in the following order (%, / *) > (+, -) implying that all of %, / and * have higher precedence over both + and -. %, / and * have the same precedence. Also, + and - have the same precedence. The order of evaluation for operators having the same precedence is decided using the brackets.

Input

The only line of input consists of a single string S ($1 \leq |S| \leq 5.10^6$) denoting the expression (in infix notation) to be parsed. Assume that all variables given in the input will be lowercase latin letters.

Output

Print a single string P, denoting the postfix notation of the given string S. The postfix notation should not contain any brackets.

---

input

```
(((a+b)-(c+d))%((x)%(y*z)))
```

output

```
ab+cd+-xyz*%%
```

D. Challenged!

You are challenged by your friend Prakhar to make a modification on the regular stack data structure. Given a stack, he asks you to *report the minimum element* of the stack in *O(1)* time. The stack will also have operations to *push*, *pop* and *peek*. So, can you take up this challenge? Note that, you should use dynamic memory allocation to solve this problem. Direct use of an array to implement the stack must not be made. Also, the size of the stack cannot be taken as input.

Input

The first line contains a single integer N ($1 \leq N \leq 3.10^6$), denoting the number of queries for the problem. Each query can be any one of the following:

- **push X:** ($-10^9 \leq X \leq 10^9$). This query denotes that integer X that needs to be pushed into the stack. The stack can keep on growing in size so that "STACK OVERFLOW" never occurs as long as the number of push operations is less than or equal to 3.10^6.
- **getMin**: This query denotes that the minimum value in the stack must be fetched (but not removed from the stack).
- **pop:** This query denotes that the top of the stack must be popped or removed.
- **peek:** This query denotes that the top of the stack must be fetched (but not removed from the stack.

Output

Print N lines (where the i^{th} line denotes the output of the i^{th} query) in the following way:

- For every *push* operation, print "OK pushed X", where X denotes the element pushed into the stack.
- For a successful *pop* operation, print "OK popped X" where X denotes the element popped.
- For a successful *getMin* operation, print the minimum element in the stack.
- For a successful *peek* operation, print the top of the stack.
- In case of an underflow (popping, peeking or fetching minimum element from an empty stack), print "STACK_UNDERFLOW".

```
input
13
push 13
push 14
push 34
push 10
push 11
peek
```

```
getMin
pop
pop
peek
getMin
push -65
getMin
```

output

```
OK pushed 13
OK pushed 14
OK pushed 34
OK pushed 10
OK pushed 11
11
10
OK popped 11
OK popped 10
34
13
OK pushed -65
-65
```

E. Lost Words

Aman has recently entered a weird cross-word competition. In this competition, each team is given a grid of letters and is asked to find if a certain word is present in the grid or not. To make the competition stiffer, the contestants are expected to report the indices of the *minimum grid point* starting from where the word could be found. Each letter in the grid can be used *exactly once* to form a word. If a letter at position *(i, j)* is chosen, then letters at either *(i+1, j)* or *(i-1, j)* or *(i, j+1)* or *(i, j-1)* can be chosen to continue the word. As Aman's teammate, help him win the first prize. You must not use recursion to solve this problem.

Input

The first line of input contains two space separated integers N ($1 \leq N \leq 500$) and M ($1 \leq M \leq 500$) denoting the dimensions of the grid. Each on the following N lines contain M lowercase latin letters denoting the letter grid. The next line contains a single integer Q ($1 \leq Q \leq 10$), denoting the number of *independent* queries for the problem. Each of the following Q lines contain a single string S_i ($1 \leq |S_i| \leq NM$) consisting of lowercase latin letters denoting the word that has to be found in the given grid.

Output

Print Q lines, (where the i^{th} line denotes the output of the i^{th} query) according to the following format: If the word is found in the grid, print the minimum coordinates (0-indexed) starting from which the word can be formed in the above said manner. If there are multiple such points, print the coordinates of the point having the minimum x-value. If 2 points have the same x-value, but different y-values, then print the coordinates of the point with the minimum y-value. If the word cannot be formed, print -1 for that query.

```
input
4 5
abhku
abgrs
bhyrr
mkytf
2
abbhk
yrrty

output
0 0
-1
```

F. Balance it!

You are given a sequence of brackets (square, curly and parantheses) and it is your task to check if the brackets given are a proper sequence of not, i.e., every opening bracket is properly closed. In other words, the brackets should satisfy the following requirements

- No bracket should be closed without opening it.
- For every opening bracket, the most recently opened bracket should be closed before all of its preceding brackets.

Input

The only line of input contains a string S ($1 \leq |S| \leq 5.10^6$) constiting of {, (, [,],), }.

Output

Print "YES" if the bracket sequence is well-formed and "NO" if the sequence is not well-formed.

---

input
```
[(){(){}()]{}()
```

output
```
YES
```

---

input
```
[(){(){(})]{}()
```

output
```
NO
```

G. Cat Walk

Muzaffar takes his cat for a walk every Sunday to the parks nearby. The city he lives in is shaped like a tree and has parks on all *leaf nodes*. Streets in the city are home to a pack of dogs. As Muzaffar and his cat are quite scared of dogs they wish to avoid them as much as they can. If a certain path (concatenation of streets) has more than K dogs in total, they will not take that path. Today is Sunday, and Muzaffar plans to take his cat for a walk again, but is busy brushing its fur and asks you to find the parks he can visit. Help him find the parks he can visit starting from his area (node) and encountering *atmost* K dogs on his way.
Assume the streets to be bidirectional and the city to be rooted at area (node) 0. Note that, recursion must be not used to solve this problem.

Input

The first line of input contains a single integer N ($1 \leq N \leq 5.10^3$) denoting the number of areas (nodes) in the city Muzaffar lives in. Each of the following N-1 lines contains three space-separated integers U_i, V_i and D_i denoting a street between areas (nodes) U_i and V_i ($0 \leq U_i, V_i \leq$ N-1) and has D_i ($0 \leq D_i \leq 10^3$) dogs living in it. The last line of input contains two space-separated integers P ($0 \leq P \leq$ N-1) and K ($0 \leq K \leq 10^3$) denoting the area (node) Muzaffar lives in and the integer K (as mentioned above).

Output

Print M integers, denoting the area (node) numbers of all the parks Muzaffar can visit in ascending order. If he cannot visit any park, print "MEOW".

---

input

```
11
0 1 10
0 2 2
0 4 2
3 1 1
7 3 7
7 10 8
8 9 0
8 6 2
4 6 2
5 4 3
4 7
```

output

```
2 5 9
```

H. Grading

You just finished grading the students after their compre and are verifying the grades. After seeing the histogram you wish to know the largest rectangular area that completely fits within the histogram. Write a simple program that will do the same for you.

Input

The first line contains a single integer N ($1 \leq N \leq 3.10^6$) denoting the number of columns in the histogram. The next line contains N space-separated integers A ($0 \leq A_i \leq 10^9$) denoting the heights of the columns (the i^{th} integer denotes the height of the i^{th} column). Assume the width of each column is is *one unit*.

Output

Print a single integer X, denoting the area of the largest rectangle that can be formed in the histogram.

```
input
6
2 1 5 6 2 3

output
10
```

I. Evaluate it!

You are given a *postfix notation* and it is your task to convert it to its corresponding *infix notation* (with complete termwise bracketing) and evaluate it. A term consists of 2 operands and a binary operator. Write a program to do the same. The given postfix notation will only contain digits (0-9) and operators (+, -, *, /). Assume the precedence order to be (/, *) > (+, -). implying that all of / and * have higher precedence over both + and -. / and * have the same precedence. Also, + and - have the same precedence.

Input

The only line of input contains a single string S ($1 \leq |S| \leq 5.10^6$) in postfix notation.

Output

In the first line, print a single integer M, denoting the result the given expression will evaluate to. In the next line print the infix notation of the given input string with complete termwise bracketing. Avoid putting unnecessary multiple brackets.

```
input
12+

output
3
(1+2)
```

```
input
442/-46/8-*

output
-16
((4-(4/2))*((4/6)-8))
```

J. Last Come First Serve

Consider the following last-come-first-serve (LCFS) scheduling algorithm. In this, consider the following steps for the algorithm:

- The process that arrives in the system last is scheduled first. If any process is executing on the processor, it is pre-empted (removed) from the processor and the latest incoming process is scheduled on the processor.
- If a certain process P is executing on the processor, it will be executing till its completion or till it is pre-empted by a new incoming process, say Q. P will be resumed only after Q and all other subsequently arriving processes, i.e., processes arriving later than Q have fully completed.
- If more than one process arrives at the same time, the processes will be scheduled according to increasing process IDs, i.e., the process with the lowest process ID will be put on the processor first.

With the above given description of the LCFS algorithm, can you get the finishing times of all the processes, given their arrival times and execution times?

Input

The first line contains a single integer N ($1 \leq N \leq 5.10^3$) denoting the number of processes in the system. Each of the following N lines contains three space-separated integers P_i ($1 \leq P_i \leq 10^4$), A_i ($0 \leq A_i \leq 5.10^3$) and B_i ($0 \leq B_i \leq 150$) denoting the process ID, arrival time and execution time of the i^{th} process respectively. Note that, in this case, the processes need not be given in order of arrival time or process ID.

Output

Print N lines, with each line containing two space-separated integers P_i and F_i with F_i denoting the finishing time of process with ID P_i. Note that, these pairs must be printed in ascending order of process IDs.

```
input
9
53 9 10
42 7 4
13 0 2
55 10 7
22 3 3
16 2 1
8 0 5
44 7 4
12 0 2

output
8 34
```

12 36
13 38
16 3
22 6
42 28
44 32
53 26
55 17
