Skip Station

ONLINE EDITOR (C)

Problem Description

Codu wants to travel from City A to City B. There are N stations between A and B. There are 3 kinds of trains that run from every station.

Train 1- Stops at every station

Train 2- Stops at every alternate station

Train 3- Stops at every third station

Codu can use any combination of the trains to reach from City A to City B. However, he cannot travel in the reverse direction during the course of his travel. He is allowed to change as many trains as needed to reach the destination.

You need to find how many ways Codu can reach City B from City A.

Constraints

1 <= T <= 5

0 <= N <= 10^2

Input

First line contains an integer T denoting the number of test cases.

Next T lines contains an integer N denoting the number of stations between A and B for each test case.

Output

For each test case print, no of combinations in a new line

Time Limit

1

Time Limit

1

Examples

Example

Input

2

0

1

Output

1

2

Explanation

For 0: No station between A and B. So only possible way to travel is by train1.

For 1: There is 1 station between A and B. So, Codu has two ways to travel. First way is to travel by train1 which halts at every station. Second way is to travel by train2 which starts from A and directly stops at B.

Upload Solution [Question : C]

☐ I, **nipun khandelwal** confirm that the answer submitted is my own. ☐ Took help from online sources (attributions)

Distinct SubString

ONLINE EDITOR (D)

Problem Description

Given a string and a set of rules, process the string in such a way, that it produces some unique substrings. Print all the longest possible substrings with distinct characters, following the below mentioned rules:

- String should be processed from left to right
- Substring should be longest possible consecutive stream of different characters (longest substring). As soon as a repeating character is encountered, consider the already processed characters as the longest substring achieved in that pass
- Once a pass is over, the next pass begins from the next index that was used in the previous pass. Simply put, first pass will always begin from first character, second pass will begin from second character, so on and so forth
- Just to make it explicit - a subsequent pass cannot process characters whose index is smaller than the beginning index of the current pass. For example, second pass cannot process first character, third pass cannot process first and second characters, so on and so forth
- A longest substring can be of two types - a *Proper substring* and an *Improper substring*
- A proper substring is one which satisfies two criteria

    1. All characters in a proper substring are fully contained in a previously identified longest substring and in same order
    2. Also, indices of proper substring are those that are completely overlapped by previously identified longest substring and in same order

        - E.g. if string is "abcdc", the first longest substring is "abcd" and its indices are from 1 to 4 (assume 1-based index)
        - Now, "bcd" is a proper substring of "a**bcd**" because not only "bcd" is completely contained in "a**bcd**", but also indices of "bcd" are 2 to 4, which completely lie in the range 1 - 4

2. Also, indices of proper substring are those that are completely overlapped by previously identified longest substring and in same order

- E.g. if string is "abcdc", the first longest substring is "abcd" and its indices are from 1 to 4 (assume 1-based index)
- Now, "bcd" is a proper substring of "**abcd**" because not only "bcd" is completely contained in "**abcd**", but also indices of "bcd" are 2 to 4, which completely lie in the range 1 - 4

- An Improper substring is one which satisfies only first criteria above.
  - E.g. Consider string "abcdc**bcd**", the first longest substring is "abcd" and its indices are from 1 to 4 (assume 1-based index)
  - Now, "**bcd**" at index 6 to 8 is an improper substring of "abcd" because even though it satisfies the first criteria of proper substring, it does not satisfy the second criteria
- In order to capture all possible longest substrings, ignore Proper substrings and retain Improper substrings
- Additional restrictions that exist on which longest substrings can be retained are as follows
  - When a longest substring is broken by a repeating character which is same as the first character of the substring, then two rules apply
    - If that substring is the first longest substring then retain that substring
    - Else, ignore that substring
      - Example - Consider string "abcdabca" - first longest substring "abcd" which is broken by 'a', is retained. However, second pass which generates second longest substring "bcda" is to be ignored because the substring starts and is also broken by the same character 'b'. Same logic applies for third longest substring "cdab", which is to be ignored
- When processing is complete, arrange all longest possible substring(s) in lexicographical order and print it as output

Constraints

$0 < N <= 10000$

String consist of all lowercase characters {a - z}

Input

First line contains an integer N which denotes the length of string

Second line contains the actual string

First line contains an integer N which denotes the length of string

Second line contains the actual string

Output

Print all the longest possible substring(s) with distinct characters, delimited by space character, while maintaining the rules mentioned above

Time Limit

1

Examples

Example 1

Input

8

abcdcbac

Output

abcd bac dcba

Explanation:

String: abcdcbcd

For illustration purpose and better understanding, substrings of interest are marked in **bold** and are underlined

- In the first pass, longest substring is "abcd" (**abcd**cbac). Since there is 'c' which is already in the substring, we retain only "abcd" and move forward.
- In the second pass, longest substring is "bcd" (a**bcd**cbac). Since the next character is 'c' and "bcd" is a proper substring of previously captured longest substring "abcd", we discard this substring.
- In the third pass, longest substring is "cd" (ab**cd**cbac). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also. Also, substring "cd" is discarded because, the starting character of this substring and the first repetitive character is same. Hence "cd" is discarded because of two rules.
- In the fourth pass, longest substring is "dcba" (abc**dcba**c). Since the next character is 'c' and this substring is not contained in any previously retained longest substring, we retain "dcba" and move forward.

- In the third pass, longest substring is "cd" (abc**cd**cbac). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also. Also, substring "cd" is discarded because, the starting character of this substring and the first repetitive character is same. Hence "cd" is discarded because of two rules.
- In the fourth pass, longest substring is "dcba" (abc**dcba**c). Since the next character is 'c' and this substring is not contained in any previously retained longest substring, we retain "dcba" and move forward.
- In the fifth pass, longest substring is "cba" (abcd**cba**c), Since the next character is 'c' and "cba" is a proper substring of "dcba", we discard this substring.
- In the sixth pass, longest substring is "bac" (abcdc**bac**). We consider this substring because it is not a part of any previously retained longest substrings.
- In the seventh pass, longest substring is "ac" (abcdcb**ac**). Since "ac" is a proper substring of "bac", we discard this substring.
- In the eighth pass, longest substring is "c" (abcdcba**c**). Since "c" is a proper substring of "bac", we discard this substring too.

Since the length of the string is 8, we made eight passes over the string to compute longest substrings. Notice that in the current pass, all the characters whose index is lower than the index of the beginning character from where the current pass begins, are not considered.

Hence, final output with retained longest substrings sorted in lexicographical order is - "abcd bac dcba".

Example 2

Input

8

abcdcbcd

Output

abcd bcd dcb

Explanation:

String: abcdcbcd

For illustration purpose and better understanding, substrings of interest are marked **bold** and underlined

- In the first pass, longest substring is "abcd" (**abcd**cbcd). Since there is 'c' which is already in the substring, we retain only "abcd" and move forward.
- In the second pass, longest substring is "bcd" (a**bcd**cbcd). Since the next character is 'c' and "bcd" is a proper substring of previously captured longest substring "abcd", we discard this substring.
- In the third pass, longest substring is "cd" (ab**cd**cbcd). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also.

- In the first pass, longest substring is "abcd" (**abcd**cbcd). Since there is 'c' which is already in the substring, we retain only "abcd" and move forward.
- In the second pass, longest substring is "bcd" (a**bcd**cbcd). Since the next character is 'c' and "bcd" is a proper substring of previously captured longest substring "abcd", we discard this substring.
- In the third pass, longest substring is "cd" (ab**cd**cbcd). Since the next character is 'c' and "cd" is already a proper substring of "abcd", we discard this substring also.
- In the fourth pass, longest substring is "dcb" (abc**dcb**cd). Since the next character is 'c' and this substring is not contained in any previously retained longest substring, we retain "dcb" and move forward.
- In the fifth pass, longest substring is "cb" (abcd**cb**cd), Since the next character is 'c' and also "cb" is a proper substring of "dcb", we discard this substring.
- In the sixth pass, longest substring is "bcd" (abcdc**bcd**). We consider this substring because although it is a part of previously retained longest substring "abcd", it is an Improper substring. Hence, we retain this substring.
- In the seventh pass, longest substring is "cd" (abcdcb**cd**). Since "cd" is a proper substring of "bcd", we discard this substring.
- In the eighth pass, longest substring is "d" (abcdcbc**d**). Since "d" is a proper substring of "bcd", we discard this substring too.

Since the length of the string is 8, we made eight passes over the string to compute longest substrings. Notice that in the current pass, all the characters whose index is lower than the index of the beginning character from where the current pass begins, are not considered.

Hence, final output with retained longest substrings sorted in lexicographical order is - "abcd bcd dcb".

Upload Solution [Question : D]

☐ I, **nipun khandelwal** confirm that the answer submitted is my own.

☐ Took help from online sources (attributions)

Choose a File ...

Bus Travel

ONLINE EDITOR (E)

Problem Description

Amidst stiff competition the transport operators in the city are not in a position to increase the bus fare. One such public transport operator is you, who is suspecting some revenue loss despite the popularity of the bus service. Your reliable sources tipped you off that there are some passengers who take the advantage of overly crowded buses and take partial tickets (They take a ticket from a bus stop which is later than their actual start stop; or they exit bus much later than their destination stop on their ticket). Thanks to the in-out sensors you placed at the bus entry and exit gates which provides you the exact details of the traveler's journey. Your objective is to find the bus stops from which these kinds of malpractices occur. You have to find out which passengers travelled with invalid tickets, and when they have over travelled (OT) or under travelled (UT).

A passenger is said to have over travelled if he does not have a valid ticket for all portions of his travel. Similarly, if a passenger has a valid ticket but alights the bus before his actual destination then he is said to have under travelled.

You have the data about every passengers' source and destination at which they boarded and alighted the bus respectively. You also have data about how many tickets were issued from which source to which destination. Using this information, you have to find out the passengers who OT or UT. For this purpose, you will have to adopt a ticket allocation policy which will map a ticket to a given passenger. Rules of ticket allocation policy in order of their importance are mentioned below. They have to be applied sequentially to do the ticket to passenger mapping.

1. For all tickets whose source and destination match with passenger data i.e., the stops at which they boarded and alighted should be allocated first. Passenger order is preserved for ticket allocation i.e. First matching ticket is allocated to the person who boarded first. Refer Example 1 in the examples section to get a better understanding.
2. For all tickets whose source matches with the source stop in the ticket data, try and do allocation if possible.
3. For all tickets whose destination matches with the destination stop in the ticket data, try and do allocation if possible.
4. If neither source nor destination data matches with any tickets data, then sequentially allocate the remaining tickets in the order of passenger boarding.

For better understanding, please refer to example section.

Note:

Note:

- None of the passenger can travel in the same bus again
- Assume none of the passenger will get ticket before boarding
- Assume passenger names are unique

Constraints

$0 < N < 50$

$0 < T < 100$

Input

- First line contains an integer N denoting the number of bus stops in a bus journey.
- Next N lines contain a string which provides passenger boarding and alighting information per stop. Format is as follows:
  - There is an alphabetical string containing a pipe (|) character. The left-hand side of the pipe provides names of passenger boarding the bus and the right-hand side provides names of passengers alighting the bus.
  - Passenger names are space delimited.
  - When no passengers have boarded or alighted at a bus stop it would be denoted by (-).
- Next line contains an integer T denoting the total number of tickets issued.
- Next T lines contain two space delimited integers which denotes source and destination of each ticket respectively.

Output

Print the output in lexicographically sorted order of passenger names.

Each line of output should contain four things:

- First output will be passenger name
- Second output will be a number corresponding to start bus stop number
- Third output will be a number corresponding to end bus stop number

Third output will be a number corresponding to end bus stop number

Print "UT" if the travel type is under travel or "OT" if the travel type is over travel

OR

Print "VALID TRAVEL" when all the tickets are valid.

Time Limit

1

Examples

Example **1**

Input

4

A B | -

C | -

- | A B

- | C

3

1 3

1 2

2 3

Output

B 2 3 OT

C 3 4 OT

Explanation

Here we have 4 bus stops and 3 passengers (A, B, C).

Explanation

Here we have 4 bus stops and 3 passengers (A, B, C).

A and B have boarded from bus stop 1 and nobody has alighted at bus stop 1.

C has boarded from bus stop 2 and nobody has alighted at bus stop 2.

Nobody has boarded from bus stop 3 and A & B has alighted at bus stop 3.

Nobody has boarded from bus stop 4 and C has alighted at bus stop 4.

Total 3 tickets are issued.

First, A is allocated ticket #1 with source 1 and destination 3 because source and destination are matching.

Then, B is allocated ticket #2 with source 1 and destination 2 because only source is matching. B has alighted at stop 3. So, B has over travelled from 2 to 3.

Then, C is allocated ticket #3 with source 2 and destination 3 because only source is matching. C has alighted at stop 4. So, C has over travelled from 3 to 4.

Hence the output is printed in lexicographically sorted order of passenger name as depicted above.

Example 2

Input

```
5
A B R | -
C | B
D | A C
- | D
- | R
5
1 2
1 4
```

- | R

5

1 2

1 4

2 4

2 3

3 4

Output

A 3 4 UT

R 1 2 OT

R 4 5 OT

Explanation

Here we have 5 bus stops and 5 passengers (A, B, C, D, R).

A,B and R has boarded from bus stop 1 and nobody has alighted at bus stop 1.

C has boarded from bus stop 2 and B has alighted at bus stop 2.

D has boarded from bus stop 3 and A & C has alighted at bus stop 3.

Nobody has boarded from bus stop 4 and D has alighted at bus stop 4.

Nobody has boarded from bus stop 5 and R has alighted at bus stop 5.

Total 5 tickets are issued.

First, B is allocated ticket #1 with source 1 and destination 2 because source and destination are matching.

Then, C is allocated ticket #4 with source 2 and destination 3 because source and destination are matching.

Then, D is allocated ticket #5 with source 3 and destination 4 because source and destination are matching.

Nobody has boarded from bus stop 5 and R has alighted at bus stop 5.

Total 5 tickets are issued.

First, B is allocated ticket #1 with source 1 and destination 2 because source and destination are matching.

Then, C is allocated ticket #4 with source 2 and destination 3 because source and destination are matching.

Then, D is allocated ticket #5 with source 3 and destination 4 because source and destination are matching.

Then, A is allocated ticket #2 with source 1 and destination 4 because only source is matching. A has alighted at stop 3. So, A has under travelled from 3 to 4.

Now, R is allocated remaining ticket i.e., ticket #3 with source 2 and destination 4. R has boarded from bus stop 1 and alighted at bus stop 5. So, R has over travelled from 1 to 2 and 4 to 5.

Hence the output is printed in lexicographically sorted order of passenger name as depicted above.

Upload Solution [Question : E]

☐ I, **nipun khandelwal** confirm that the answer submitted is my own.

☐ Took help from online sources (attributions)

Choose a File ...

A B C D E F G H

ONLINE EDITOR (F)

Train Track

Problem Description

Station master of Codington railway station is working on automation of trains to platform allocation. This allocation will not only prevent manual errors but also bring about a consistency in train to platform allocation. This will improve the passenger experience. Your job is to help the station master achieve the objectives.

The data and the rules that need to be considered for this automation are as follows-

- Codington is a large railway junction. Hence assume that there is always a platform at which the incoming train can be accommodated (infinitely many platforms). However, operating using a less number of platforms is economical and hence preferred.
- Safety is of paramount importance. Hence, safety cannot be traded off for cost. For example, if Train A's departure time is x and Train B's arrival time is x, then we can't accommodate Train B on the same platform as Train A.
- The trains to platforms allocation rules are as follows-
  - The platform which is being freed earlier has to be considered earlier for allocation to the next train. For example, if platform #2 is freed at t=5 and platform #1 is freed at t=6, in that case the next train arriving at t=7 will arrive at platform #2.
  - If two or more platforms are freed at the same time, then the train arrives on the platform with the lowest number. For example, if platform #1 and #2 are freed at t=5 and the train is arriving at time t=6, then the train needs to arrive at platform #1.
  - If two or more trains are arriving at the same time then above two rules should be applied first for a train with smaller train number. This train's allocation will be decided first and the next qualified platform will be allocated to the other train. For example, consider the table below

Case No.	Platform #1 free since	Platform #2 free since	Train number 1 arriving at	Train number 2 arriving at	Resulting allocation
Case 1	t=5	t=6	t=7	t=7	Train 1 is allocated platform #1 and Train 2 is allocated platform #2
Case 2	t=5	t=5	t=7	t=7	Train 1 is allocated platform #1 and Train

Case 2	t=5	t=5	t=7	t=7	Train 1 is allocated platform #1 and Train 2 is allocated platform #2
Case 3	t=6	t=5	t=7	t=7	Train 1 is allocated platform #2 and Train 2 is allocated platform #1

Constraints

1 <= N <= 5.10^4

0 <= A <= 86400

0 < I <= 86400

1000 < T < 10^8

Number of platforms > 0

Input

First line contains an integer N denoting number of trains.

Next N lines contain three integers T, A and I denoting the train number(T), the arrival time(A) and stoppage interval(I) of train respectively.

Next line contains an integer F denoting the train number for whom the allocated platform is to be reported in the output.

Output

Next line contains an integer F denoting the train number for whom the allocated platform is to be reported in the output.

Output

First line contains an integer denoting the platform number on which the train F arrived.

Second line contains an integer denoting the busiest platform number on which the maximum trains arrived.

OR

If there are more than one platform having same traffic, sort all such platforms and print them in ascending order with each platform number on a new line. For example, if platform #2 and #3 are the busiest platforms then print as given below -

2

3

Time Limit

2

Examples

Example 1

Input

3

12121 15 5

12311 5 10

17215 2 3

12311 5 10

17215 2 3

12311

Output

2

1

Explanation

The earliest arriving train (17215) is at time t = 2 will arrive at platform# 1. Since it will stay there till t = 5, train (12311) arriving at time t = 5 w
arrive at platform# 2 as the previous train (17215) will leave at t=5. Since current train (12311) will depart at time t = 15, next train (12121) arrivin
at time t = 15 will arrive at platform# 1 as the previous train (17215) left at t=5.

Here, maximum traffic came on platform #1. And train 12311 arrived at platform #2.

Example 2

Input

4

12102 2 4

12144 7 2

02812 3 2

13411 5 5

13411

Output

4

12106 2 4

12144 7 2

02812 3 2

13411 5 5

13411

Output

3

2

Explanation

The earliest arriving train (12106) at time t = 2 will arrive at platform# 1. Since it will stay there till t = 6, train (02812) arriving at time t = 3 will arrive at platform# 2. Since it will stay there till t = 5, train (13411) arriving at time t = 5 will arrive at platform# 3 Since it will stay there till t = 10, train (12144) arriving at time t = 7 will arrive at platform #2 as the previous train (02812) left at t=5.

Here, maximum traffic came on platform #2. And train 13411 arrived at platform #3.

Upload Solution [Question : F]

☐ I, **nipun khandelwal** confirm that the answer submitted is my own.

☐ Took help from online sources (attributions)

ONLINE EDITOR (G)

Paste Reduction

Problem Description

Some keys in Codu's computer keyboard are not working. Fortunately for Codu, these characters are present in a previously existing text file.

He wants to write a paragraph which involves typing those characters whose keys are defunct in Codu's keyboard. Only option left for Codu is to copy-paste those characters from the previously existing text files. However, copy-pasting keys is a laborious operation since one has to switch windows and also previously copied items are lost once a new set of characters are copied. Hence, Codu wants to minimize the number of times he needs to copy-paste from that text file. Fortunately there can be situations where previously copied characters are readily available for pasting. Help Codu devise a method to minimize the number of times a paste operation is needed, given - the text he intends to type and the faulty keys on his keyboard.

Constraints

0 < Length of paragraph <= 1000 characters

0 < number of faulty keys in keyboard <= 26

Only a to z can be faulty.

Input paragraph does not contain any upper-case character, special characters and punctuation marks.

Input

First line contains a paragraph that is to be written.

Input

First line contains a paragraph that is to be written.

Second line contains a string. This string has to be interpreted in the following fashion

All characters in that string correspond to faulty keys

That string is available for copy as-is from the other text file that Codu is referring

Also, individual characters can always be copied from the same text file

Output

Single integer denoting minimum number of paste operations required

Time Limit

1

Examples

Example 1

Input

supreme court is the highest judicial court

su

Output

1

Examples

Example 1

Input

supreme court is the highest judicial court

su

Output

4

Explanation

Codu will first paste *su* from the file when typing characters *su* in the word supreme.

In the second instance, Codu will need to copy character *u* and paste it when typing character *u* in the word court.

In the third instance, Codu will need to copy character *su* and paste *su* when typing character s in the word "is". Codu will back track using the left arrow key and type the characters " the highe".

In the fourth instance, Codu will again paste *su*. The overall string typed until this moment is "supreme court is the highesuu". Codu will then back track again using left arrow key and type characters "t j". At this point the typed string is "supreme court is the highest juu". Cursor is after character j. Codu will use right arrow key and now the cursor will be after ju. Codu will now type "dicial co". String typed till this point is "supreme court is the highest judicial cou". Cursor is after character o. Codu will use right arrow and the cursor will be after character *u*. Finally Codu will type "rt".

Final string - "supreme court is the highest judicial court" - is thus fully typed. Here, the number of paste operations are 4.