Max Funds

Problem Description

An NGO wants to arrange funds for flood relief. It has divided volunteers into groups. A volunteer can only be a part of single group. Your task is to find the maximum funds collected by a group.

You will be given the funds collected by each volunteer and grouping pairs of the volunteers. You need to group the volunteers through these pairs.

Constraints

0 < N, P <= 10000

0 < A, B <= N

Input

First line contains one integer N, denoting number of volunteers.

Second line contains N space separated integers, representing the amount collected by each volunteer. The index of integer is the volunteer number starting from 1.

Third line contains the number of pairs, P.

Next P lines contain two space separated integers, A and B where A represents the first person and B represents the second person in the pair.

Output

One line containing an integer, representing the maximum funds collected by the group.

Time Limit

2

Examples

Example 1

Input

5

23 43 123 54 2

3

1 3

2 3

1 2

Output

189

Explanation

In the above example, we have five volunteer [1, 2, 3, 4, 5] who have collected [23, 43, 123, 54, 2] respectively.

We have three groups that consists of [1, 2, 3], [4], [5]. First group collects 189 units of money, second group collects 54 units of money and third group collects 2 units of money. The maximum funds collected by any group is 189. Hence, the output is 189

Example 2

Input

9

34 54 65 76 88 23 56 76 43

7

1 3

2 3

1 2

6 8

5 4

5 7

8 9

Output

220

Explanation

In the above example, we have three groups that consists of [1, 2, 3], [4, 5, 7], [6, 8, 9]. Each group collects 153, 220, 142 units of money respectively. The maximum funds collected by a group is 220. Hence the output is 220.

Infected Cells

Problem Description

An animal has an infected tissue; the tissue is infected by a virus. The virus is activated only at a pH value of 6.

John, a veterinary doctor, suspects all the cells to be infected and hence, have a pH value of 6. However he needs to confirm his suspicion with the help of a specialized machine which finds the pH value of cells in the tissue.

The machine finds the pH value and returns the number of virus infected spots. If the adjacent cells are of same pH and they form either square or a rectangle then and only then the said infected cells will be counted as 1 infected spot. Thus it is easy to see that the infected spots can be of different sizes. (Refer Examples for better clarity)

The machine first scans the infected cells in row order and determines the number of infected spots. Then scans the infected cells in column order and determines the number of infected spots and returns the minimum of the two counts. (Refer Example 2 for better understanding)

The tissue will be given in the form of a matrix. Every element of the matrix provides the pH value of the cell.

You need to find the infected spots as determined by the machine. An infected spot as explained earlier consists of one or more cells which form a rectangle or a square relative to the direction of the scan.

Constraints

0 < R, C <= 10000

Input

First line contains two integers R and C, where R represents number of rows and C represents number of columns. Next R lines contain C space separated integers, representing the pH values.

Output

One line containing an integer, representing the number of infected spots in the tissue determined by the machine.

Time Limit

3

Examples

Example 1

Input

3 3
0 6 6
9 6 6
9 7 4

3 3

0 6 6

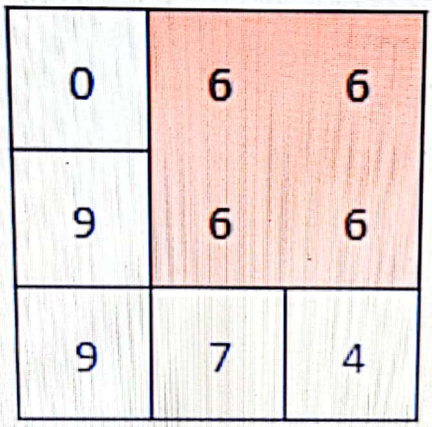
9 6 6

9 7 4

Output

1

Explanation



There will only be 1 infected spot as determined by both ways of scanning. Hence the machine will return 1. Hence, the output is 1.

Example 2

Input

5 4

6 6 6 6

3 6 4 6

6 6 6 6

3 6 4 6

6 6 6 6

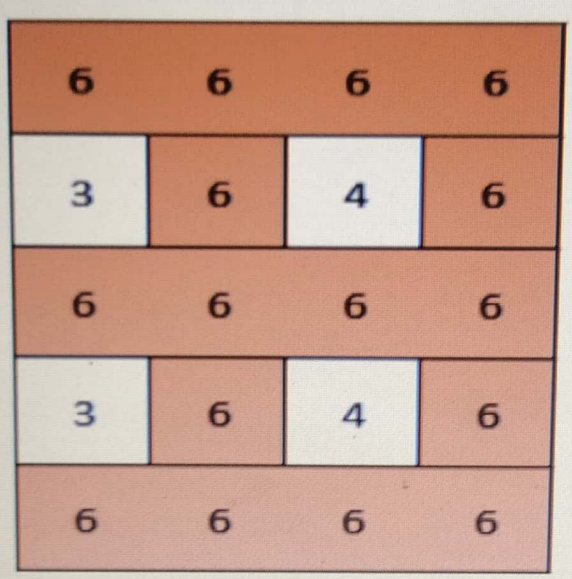
6 6 6 6

Output

7

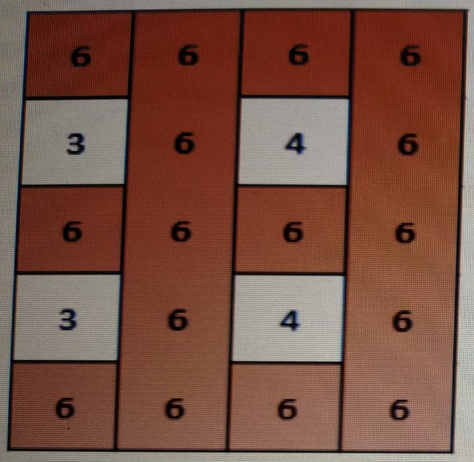
Explanation

For row based scanning the number of infected spots is 7 as depicted below.



Similarly, for column based scanning the number of infected spots is 8 as depicted below

Similarly, for column based scanning the number of infected spots is 8 as depicted below



Since, row based scanning count is less than column based scanning count, the machine will return the number of infected spots as 7. Hence, the output is 7.

Skip Station

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

Examples

Example

Input

2

0

1

Output

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

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.

ONLINE EDITOR (D)

Distinct SubString

— 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 "**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
    a. 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

- 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 "**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

Output

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

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 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 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 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 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.

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 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".

ONLINE EDITOR (E)

Bus Travel

— 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:

- 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

For better understanding, please refer to example section.

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
- 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.

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).

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

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 I -
C I B
D I A C
- I D
- I R
5
1 2
1 4
2 4
2 3
3 4
```

Output

-D

-I 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.

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.

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.

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.