Chapter 4

Sets

Chapter 4

Sets

In This Chapter:

1. Introduction to Sets
2. Symmetric Difference
3. No Idea!
4. Set .add()
5. Set .discard(), .remove() & .pop()
6. Set .union() Operation
7. Set .intersection() Operation
8. Set .difference() Operation
9. Set .symmetric_difference() Operation
10. Set Mutations
11. The Captain's Room
12. Check Subset
13. Check Strict Superset

4.1. Introduction to Sets

A *set* is an unordered collection of elements without duplicate entries. When printed, iterated or converted into a sequence, its elements will appear in an arbitrary order.

Example

```
>>> print set()
set([])

>>> print set('HackerRank')
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])

>>> print set([1,2,1,2,3,4,5,6,0,9,12,22,3])
set([0, 1, 2, 3, 4, 5, 6, 9, 12, 22])
```

```
>>> print set((1,2,3,4,5,5))
set([1, 2, 3, 4, 5])

>>> print set(set(['H','a','c','k','e','r','r','a','n','k']))
set(['a', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print set({'Hacker' : 'DOSHI', 'Rank' : 616 })
set(['Hacker', 'Rank'])

>>> print set(enumerate(['H','a','c','k','e','r','r','a','n','k']))
set([(6, 'r'), (7, 'a'), (3, 'k'), (4, 'e'), (5, 'r'), (9, 'k'), (2, 'c'), (0, 'H'), (1, 'a'), (8, 'n')])
```

Basically, sets are used for membership testing and eliminating duplicate entries.

Now, let's use our knowledge of sets and help Mickey.

Ms. Gabriel Williams is a botany professor at District College. One day, she asked her student Mickey to compute the average of all the plants with distinct heights in her greenhouse.

Formula used:

$$Average = \frac{Sum\ of\ Distinct\ Heights}{Total\ Number\ of\ Distinct\ Heights}$$

Input Format

The first line contains the integer, N, the total number of plants.
The second line contains the N space separated heights of the plants.

Constraints

$0 < N \leq 100$

Output Format

Output the average height value on a single line.

Sample Input

```
10
161 182 161 154 176 170 167 171 170 174
```

Sample Output

```
169.375
```

Explanation

Here, *set* is the set containing the distinct heights. Using the *sum()* and *len()* functions, we can compute the average.

Code

```
from __future__ import division

def average(array):
    # your code goes here
    heights = set(array)
    return '{:.3f}'.format(sum(heights) / len(heights))

if __name__ == '__main__':
    n = int(raw_input())
    arr = map(int, raw_input().split())
    result = average(arr)
```

4.2. Symmetric Difference

If the inputs are given on one line separated by a space character, use *split()* to get the separate values in the form of a list:

```
>> a = raw_input()
5 4 3 2
```

```
>> lis = a.split()
>> print (lis)
['5', '4', '3', '2']
```

If the list values are all integer types, use the *map()* method to convert all the strings to integers.

```
>> newlis = list(map(int, lis))
>> print (newlis)
[5, 4, 3, 2]
```

Sets are an unordered bag of unique values. A single set contains values of any immutable data type.

CREATING SETS

```
>> myset = {1, 2} # Directly assigning values to a set
>> myset = set()  # Initializing a set
>> myset = set(['a', 'b']) # Creating a set from a list
>> myset
{'a', 'b'}
```

MODIFYING SETS

Using the *add()* function:

```
>> myset.add('c')
>> myset
{'a', 'c', 'b'}
>> myset.add('a') # As 'a' already exists in the set, nothing happens
>> myset.add((5, 4))
>> myset
{'a', 'c', 'b', (5, 4)}
```

Using the *update()* function:

```
>> myset.update([1, 2, 3, 4]) # update() only works for iterable objects
>> myset
{'a', 1, 'c', 'b', 4, 2, (5, 4), 3}
>> myset.update({1, 7, 8})
```

```
>> myset
{'a', 1, 'c', 'b', 4, 7, 8, 2, (5, 4), 3}
>> myset.update({1, 6}, [5, 13])
>> myset
{'a', 1, 'c', 'b', 4, 5, 6, 7, 8, 2, (5, 4), 13, 3}
```

REMOVING ITEMS

Both the *discard()* and *remove()* functions take a single value as an argument and removes that value from the set. If that value is not present, *discard()* does nothing, but *remove()* will raise a KeyError exception.

```
>> myset.discard(10)
>> myset
{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 13, 11, 3}
>> myset.remove(13)
>> myset
{'a', 1, 'c', 'b', 4, 5, 7, 8, 2, 12, (5, 4), 11, 3}
```

COMMON SET OPERATIONS

Using *union()*, *intersection()* and *difference()* functions.

```
>> a = {2, 4, 5, 9}
>> b = {2, 4, 11, 12}
>> a.union(b) # Values which exist in a or b
{2, 4, 5, 9, 11, 12}
>> a.intersection(b) # Values which exist in a and b
{2, 4}
>> a.difference(b) # Values which exist in a but not in b
{9, 5}
```

The *union()* and *intersection()* functions are symmetric methods:

```
>> a.union(b) == b.union(a)
True
>> a.intersection(b) == b.intersection(a)
True
>> a.difference(b) == b.difference(a)
False
```

For more detailed examples on other set operations, please refer to http://bit.ly/2nluAEz

Which will redirect you to
http://www.thelearningpoint.net/computer-science/learning-python-programming-and-data-structures/learning-python-programming-and-data-structures--tutorial-4--built-in-data-structures-strings-lists-tuples-dictionaries-mutability

Given 2 sets of integers, M and N, print their symmetric difference in ascending order. The term *symmetric difference* indicates those values that exist in either M or N but do not exist in both.

Input Format

The first line of input contains an integer,M .
The second line contains M space-separated integers.
The third line contains an integer,N .
The fourth line contains N space-separated integers.

Output Format

Output the symmetric difference integers in ascending order, one per line.

Sample Input

```
4
2 4 5 9
4
2 4 11 12
```

Sample Output

```
5
9
11
12
```

Code

```
_m = int(raw_input())
M = set (map (int, raw_input().split()))
_n = int(raw_input())
N = set (map (int, raw_input().split()))

# R = M ^ N
R = M.symmetric_difference(N)

Result = sorted(R, key=int)

for iterator in Result:
    print iterator
```

4.3. No Idea!

There is an array of n integers. There are also 2 **disjoint sets**, A and B, each containing m integers. You like all the integers in set A and dislike all the integers in set B. Your initial happiness is 0. For each integer i in the array, if i in A, you add 1 to your happiness. If i in B, you add -1 to your happiness. Otherwise, your happiness does not change. Output your final happiness at the end.
Note: Since A and B are sets, they have no repeated elements. However, the array might contain duplicate elements.

Constraints

$1 \leq n \leq 10^5$
$1 \leq m \leq 10^5$
$1 \leq Any\ integer\ in\ the\ input \leq 10^9$

Input Format

The first line contains integers n and m separated by a space.
The second line contains n integers, the elements of the array.
The third and fourth lines contain m integers, A and B, respectively.

Output Format

Output a single integer, your total happiness.

Sample Input

```
3 2
1 5 3
3 1
5 7
```

Sample Output

```
1
```

Code

```
(n, m) = map(int, raw_input().split())
array = map(int, raw_input().split())
assert len(array) == n
A = set(map(int, raw_input().split()))
assert len(A) == m
B = set(map(int, raw_input().split()))
assert len(B) == m
happiness = 0
for i in array:
    if i in A:
        happiness += 1
    elif i in B:
        happiness -= 1

print happiness
```

4.4. Set .add()

If we want to add a single element to an existing set, we can use the *.add()* operation.

It adds the element to the set and returns '**None**'.

Example

```
>>> s = set('HackerRank')
>>> s.add('H')
>>> print s
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])
>>> print s.add('HackerRank')
None
>>> print s
set(['a', 'c', 'e', 'HackerRank', 'H', 'k', 'n', 'r', 'R'])
```

Rupal has a huge collection of country stamps. She decided to count the total number of distinct country stamps in her collection. She asked for your help. You pick the stamps one by one from a stack of N country stamps. Find total number of distinct country stamps.

Input Format

The first line contains integer N, the total number of country stamps. The next N lines contains the name of the country where the stamp is from.

Constraints

$0 < N < 1000$

Output Format

Output the total number of distinct country stamps on a single line.

Sample Input

```
7
UK
China
USA
France
New Zealand
UK
France
```

Sample Output

```
5
```

Code

```
N = int(raw_input())

countries = set()

for i in range(N):
    countries.add(raw_input())

print len(countries)
```

4.5. Set .discard(), .remove() & .pop()

.remove(x)

This operation removes element x from the set.
If element x does not exist, it raises a **KeyError**.
The *.remove(x)* operation returns **None**.

Example

```
>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s.remove(5)
>>> print s
set([1, 2, 3, 4, 6, 7, 8, 9])
>>> print s.remove(4)
None
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
>>> s.remove(0)
KeyError: 0
```

.discard(x)

This operation also removes element x from the set.
If element x does not exist, it **does not** raise a KeyError.
The *.discard(x)* operation returns **None**.

Example

```
>>> s = set([1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> s.discard(5)
>>> print s
set([1, 2, 3, 4, 6, 7, 8, 9])
>>> print s.discard(4)
None
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
>>> s.discard(0)
>>> print s
set([1, 2, 3, 6, 7, 8, 9])
```

.pop()

pop() removes and return an arbitrary element from the set.
If there are no elements to remove, it raises a **KeyError**.

Example

```
>>> s = set([1])
>>> print s.pop()
1
>>> print s
set([])
>>> print s.pop()
KeyError: pop from an empty set
```

You have a non-empty set s, and you have to execute N commands given in N lines. The commands will be *pop*, *remove* and *discard*.

Input Format

The first line contains integer n, the number of elements in set s.
The second line contains n space separated elements of set s. All of the elements are non-negative integers, less than or equal to 9.
The third line contains integer N, the number of commands.
The next N lines contains either *pop*,
remove and/or *discard* commands followed by their associated value.

Constraints

$0 < n < 20$
$0 < N < 20$

Output Format

Print the sum of the elements of set s on a single line.

Sample Input

```
9
1 2 3 4 5 6 7 8 9
10
pop
remove 9
discard 9
discard 8
remove 7
pop
discard 6
remove 5
pop
discard 5
```

Sample Output

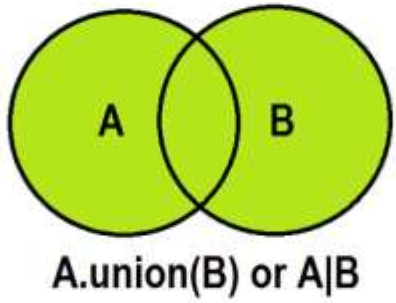
```
4
```

Code

```
n = input()
s = set(map(int, raw_input().split()))

N = int(raw_input())

for i in range(N):
    command = raw_input().split()

    if command[0] == 'pop':
        s.pop()
    elif command[0] == 'remove':
        s.remove(int(command[1]))
    elif command[0] == 'discard':
        s.discard(int(command[1]))
    else:
        assert False

print sum(s)
```

4.6. Set .union() Operation
4.7. Set .intersection() Operation
4.8. Set .difference() Operation
4.9. Set .symmetric_difference() Operation

These 4 tasks are soved together



.union()

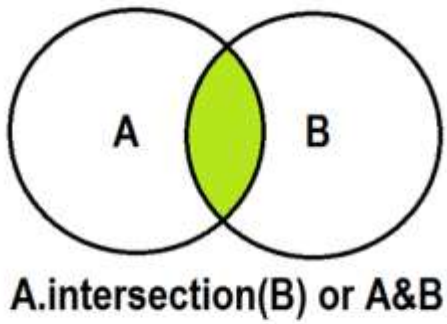
The *.union()* operator returns the union of a set and the set of elements in an iterable.

Sometimes, the | operator is used in place of *.union()* operator, but it operates only on the set of elements in *set*.

Set is immutable to the *.union()* operation (or | operation).

Example

```
>>> s = set("Hacker")
>>> print s.union("Rank")
set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(set(['R', 'a', 'n', 'k']))
set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(['R', 'a', 'n', 'k'])
set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])

>>> print s.union(enumerate(['R', 'a', 'n', 'k']))
set(['a', 'c', 'r', 'e', (1, 'a'), (2, 'n'), 'H', 'k', (3, 'k'),
(0, 'R')])

>>> print s.union({"Rank":1})
set(['a', 'c', 'r', 'e', 'H', 'k', 'Rank'])

>>> s | set("Rank")
set(['a', 'R', 'c', 'r', 'e', 'H', 'k', 'n'])
```



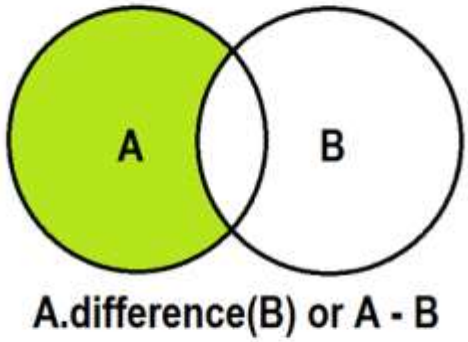
.intersection()

The *.intersection()* operator returns the intersection of a set and the set of elements in an iterable.

Sometimes, the *&* operator is used in place of the *.intersection()* operator, but it only operates on the set of elements in *set*.

The set is immutable to the *.intersection()* operation (or *&* operation).

```
>>> s = set("Hacker")
>>> print s.intersection("Rank")
set(['a', 'k'])

>>> print s.intersection(set(['R', 'a', 'n', 'k']))
set(['a', 'k'])

>>> print s.intersection(['R', 'a', 'n', 'k'])
set(['a', 'k'])

>>> print s.intersection(enumerate(['R', 'a', 'n', 'k']))
set([])

>>> print s.intersection({"Rank":1})
set([])

>>> s & set("Rank")
set(['a', 'k'])
```



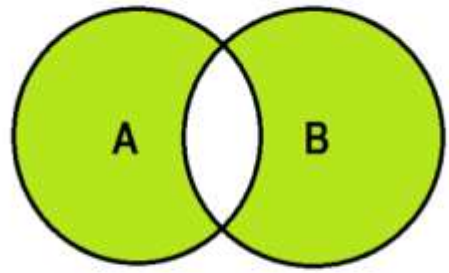
.difference()

The tool *.difference()* returns a set with all the elements from the set that are not in an iterable.

Sometimes the `-` operator is used in place of the *.difference()* tool, but it only operates on the set of elements in *set*.

Set is immutable to the *.difference()* operation (or the `-` operation).

```
>>> s = set("Hacker")
>>> print s.difference("Rank")
set(['c', 'r', 'e', 'H'])

>>> print s.difference(set(['R', 'a', 'n', 'k']))
set(['c', 'r', 'e', 'H'])

>>> print s.difference(['R', 'a', 'n', 'k'])
set(['c', 'r', 'e', 'H'])

>>> print s.difference(enumerate(['R', 'a', 'n', 'k']))
set(['a', 'c', 'r', 'e', 'H', 'k'])

>>> print s.difference({"Rank":1})
set(['a', 'c', 'e', 'H', 'k', 'r'])

>>> s - set("Rank")
set(['H', 'c', 'r', 'e'])
```



A.symmetric_difference(B) or A^B

by DOSHI

.symmetric_difference()

The *.symmetric_difference()* operator returns a set with all the elements that are in the set and the iterable but not both.

Sometimes, a `^` operator is used in place of the *.symmetric_difference()* tool, but it only operates on the set of elements in *set*.

The set is immutable to the *.symmetric_difference()* operation (or `^` operation).

```
>>> s = set("Hacker")
>>> print s.symmetric_difference("Rank")
set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric_difference(set(['R', 'a', 'n', 'k']))
set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric_difference(['R', 'a', 'n', 'k'])
set(['c', 'e', 'H', 'n', 'R', 'r'])

>>> print s.symmetric_difference(enumerate(['R', 'a', 'n', 'k']))
set(['a', 'c', 'e', 'H', (0, 'R'), 'r', (2, 'n'), 'k', (1, 'a'),
(3, 'k')])

>>> print s.symmetric_difference({"Rank":1})
set(['a', 'c', 'e', 'H', 'k', 'Rank', 'r'])

>>> s ^ set("Rank")
set(['c', 'e', 'H', 'n', 'R', 'r'])
```

The students of District College have subscriptions to *English* and *French* newspapers. Some students have subscribed only to *English*, some have subscribed to only *French* and some have subscribed to both newspapers.

You are given two sets of student roll numbers. One set has subscribed to the *English* newspaper, and the other set is subscribed to the *French* newspaper. The same student could be in both sets. Your task is to find the total number of students who have subscribed

to *at least one* newspaper ➔ union

to *both* newspaper ➔ intersection

to *only* English newspapers. ➔ difference

to either *English* or *French* but *not both* ➔ symmetric difference

Input Format

The first line contains an integer, n, the number of students who have subscribed to the *English* newspaper.

The second line contains n space separated roll numbers of those students.

The third line contains b, the number of students who have subscribed to the *French* newspaper.

The fourth line contains b space separated roll numbers of those students.

Constraints

$0 < Total\ number\ of\ students\ in\ college < 1000$

Code for union

```
_n = int(raw_input())
En = set(map(int, raw_input().split()))
_b = int(raw_input())
Fr = set(map(int, raw_input().split()))

R = En | Fr
# R = En.union(Fr)

print len(R)
```

```
_n = int(raw_input())

En = set(map(int, raw_input().split()))

_b = int(raw_input())

Fr = set(map(int, raw_input().split()))

R = En | Fr     # R = En.union(Fr)

R = En & Fr    # R = En.intersection(Fr)

R = En - Fr     # R = En.difference(Fr)

R = En ^ Fr    # R = En.symmetric_difference(Fr)

print len(R)
```

4.10. Set Mutations

We have seen the applications of *union*, *intersection*, *difference* and *symmetric difference* operations, but these operations do not make any changes or mutations to the set.

We can use the following operations to create mutations to a set:

.update() or |=
Update the set by adding elements from an iterable/another set.

```
>>> H = set("Hacker")
>>> R = set("Rank")
>>> H.update(R)
>>> print H
set(['a', 'c', 'e', 'H', 'k', 'n', 'r', 'R'])
```

.intersection_update() or **&=**

Update the set by keeping only the elements found in it and an iterable/another set.

```
>>> H = set("Hacker")
>>> R = set("Rank")
>>> H.intersection_update(R)
>>> print H
set(['a', 'k'])
```

.difference_update() or **-=**

Update the set by removing elements found in an iterable/another set.

```
>>> H = set("Hacker")
>>> R = set("Rank")
>>> H.difference_update(R)
>>> print H
set(['c', 'e', 'H', 'r'])
```

.symmetric_difference_update() or **^=**

Update the set by only keeping the elements found in either set, but not in both.

```
>>> H = set("Hacker")
>>> R = set("Rank")
>>> H.symmetric_difference_update(R)
>>> print H
set(['c', 'e', 'H', 'n', 'r', 'R'])
```

You are given a set A and N number of other sets. These N number of sets have to perform some specific mutation operations on set A. Your task is to execute those operations and print the sum of elements from set A.

Input Format

The first line contains the number of elements in set A.
The second line contains the space separated list of elements in set A.
The third line contains integer N, the number of other sets.
The next 2N lines are divided into N parts containing two lines each.
The first line of each part contains the space separated entries of the *operation name* and the *length of the other set*.
The second line of each part contains space separated list of elements in the other set.

$0 < len(set(\boldsymbol{A})) < 1000$
$0 < len(otherSets) < 100$
$0 < N < 100$

Output Format

Output the sum of elements in set A.

Sample Input

```
16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 24 52
4
intersection_update 10
2 3 5 6 8 9 1 4 7 11
update 2
55 66
symmetric_difference_update 5
22 7 35 62 58
difference_update 7
11 22 35 55 58 62 66
```

Sample Output

```
38
```

Code

```
length = int(raw_input())
A = set(map(int, raw_input().split()))
N = int(raw_input())

for i in range(N):
    (operation_name, _set_length) = raw_input().split()
    other_set = set(map(int, raw_input().split()))

    if operation_name == 'intersection_update':
        A.intersection_update(other_set)
    elif operation_name == 'update':
        A.update(other_set)
    elif operation_name == 'symmetric_difference_update':
        A.symmetric_difference_update(other_set)
    elif operation_name == 'difference_update':
        A.difference_update(other_set)
    else:
        assert False

print sum(A)
```

Code using sysmols (|=, &=, …)

```
length = int(raw_input())
A = set(map(int, raw_input().split()))
N = int(raw_input())

for i in range(N):
    (operation_name, _set_length) = raw_input().split()
    other_set = set(map(int, raw_input().split()))

    if operation_name == 'intersection_update':
        A &= other_set
    elif operation_name == 'update':
        A |= other_set
    elif operation_name == 'symmetric_difference_update':
        A ^= other_set
    elif operation_name == 'difference_update':
        A -= other_set
    else:
        assert False

print sum(A)
```

4.11. The Captain's Room

Mr. Anant Asankhya is the manager at the *INFINITE* hotel. The hotel has an infinite amount of rooms.
One fine day, a *finite* number of tourists come to stay at the hotel.
The tourists consist of:
→ A Captain.
→ An unknown group of families consisting of K members per group where K ≠ 1.

The Captain was given a separate room, and the rest were given one room per group.

Mr. Anant has an unordered list of randomly arranged room entries. The list consists of the room numbers for all of the tourists. The room numbers will appear K times per group except for the Captain's room.

Mr. Anant needs you to help him find the Captain's room number.
The total number of tourists or the total number of groups of families is not known to you.
You only know the value of K and the room number list.

Input Format

The first line consists of an integer,K , the size of each group.
The second line contains the unordered elements of the room number list.

Constraints

$1 < K < 1000$

Output Format

Output the Captain's room number.

Sample Input

```
5
1 2 3 6 5 4 4 2 5 3 6 1 6 5 3 2 4 1 2 5 1 4 3 6 8 4 3 1 5 6 2
```

Sample Output

```
8
```

Explanation

The list of room numbers contains 31 elements. Since K is 5, there must be 6 groups of families. In the given list, all of the numbers repeat 5 times except for room number 8.
Hence, 8 is the Captain's room number.

Code

```
K = int(raw_input())
room_number_list = map(int, raw_input().split())
rooms = set(room_number_list)

print (sum(rooms) * K - sum(room_number_list)) / (K - 1)
```

4.12. Check Subset

You are given two sets,A and B.
Your job is to find whether set A is a subset of set B.
If set A is subset of set B, print **True**.
If set A is not a subset of set B, print **False**.

Input Format

The first line will contain the number of test cases, T.
The first line of each test case contains the number of elements in set A. The second line of each test case contains the space separated elements of set A.
The third line of each test case contains the number of elements in set B. The fourth line of each test case contains the space separated elements of set B.

Constraints

$0 < T < 21$

$0 < Number\ of\ elements\ in\ each\ set < 1001$

Output Format

Output **True** or **False** for each test case on separate lines.

Sample Input

```
3
5
1 2 3 5 6
9
9 8 5 6 3 2 1 4 7
1
2
5
3 6 5 4 1
7
1 2 3 5 6 8 9
3
9 8 2
```

Sample Output

```
True
False
False
```

Code

```
for i in range(int(raw_input())): #More than 4 lines will result in 0
    a = int(raw_input()); A = set(raw_input().split())
    b = int(raw_input()); B = set(raw_input().split())

    if A.issubset(B): print True
    else: print False
```

4.13. Check Strict Superset

You are given one set A and a number of other sets,N .
Your job is to find whether set A is a strict superset of all the N sets.
Print **True**, if A is a *strict superset* of all of the N sets. Otherwise, print **False**.
A strict superset has at least one element that does not exist in its subset.

Example:

$\text{Set}([1,3,4])$ is a *strict superset* of $\text{set}([1,3])$.

$\text{Set}([1,3,4])$ is **not** a *strict superset* of $\text{set}([1,3,4])$.

$\text{Set}([1,3,4])$ is **not** a *strict superset* of $\text{set}([1,3,5])$.

Input Format

The first line contains the space separated elements of set A.
The second line contains integer N, the number of other sets.
The next N lines contains the space separated elements of the other sets.

Constraints

$0 < len(set(A)) < 501$

$0 < N < 21$

$0 < len(otherSets) < 101$

Output Format

Print **True** if set A is a *strict superset* of all other N sets. Otherwise, print **False**.

Sample Input

```
1 2 3 4 5 6 7 8 9 10 11 12 23 45 84 78
2
1 2 3 4 5
100 11 12
```

Sample Output

```
False
```

Code

```
A = set(map(int, raw_input().split()))
N = int(raw_input())

res = True

for i in range(N):
    s = set(map(int, raw_input().split()))
    if not s.issubset(A):
        res = False
    if len(s) >= len(A):
        res = False

print res
```