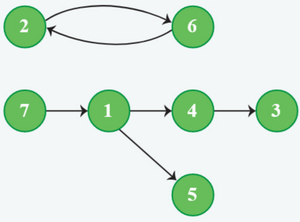
3. Task Master

Eric is the most methodical employee at the Acme company. His manager assigned him a number of tasks for the quarter, and gave him a list of notes regarding the order they must be performed. Each note states that some task must be completed before some related task. If he goes to perform some task and sees that a rule exists requiring that this task be performed *before* an already completed task, then he cannot perform the task. Help Eric determine the maximum number of tasks he can complete.

For example, Eric has *n=7* tasks to complete. His manager gives him *m=6* notes on the order tasks must be performed. Here is a graph of the dependencies.The dependent array, *a = [1, 2, 3, 4, 6, 5]*. His principal tasks array, *b = [7, 6, 4, 1, 2, 1]*. Here is a graph of the dependencies.

```
a = [1,2,3,4,6,5]
b = [7,6,4,1,2,1]
```



From the graph, it is easy to see that task 6 must be performed *before* task *2* and vice versa. He can only complete one of those two tasks before the other, so he must choose either task *6* or *2*. He can complete *7 - 1 = 6* tasks.

Function Description
Complete the function *tasks* in the editor below. It must return an integer denoting the maximum number of tasks that Eric can complete.

tasks has the following parameter(s):
n: integer, the number of tasks
a[a[0],...a[n-1]]: an array of integers, the dependent tasks
b[b[0],...b[n-1]]: an array of integers, the primary tasks

Constraints

- $1 \le n \le 10^5$
- $0 \le m \le n$
- $1 \le a[i], b[i] \le n$
- Each task depends on at most one other task.

▾ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer *n*, the number of tasks.
The next line contains an integer *m*, the size of the array *a*.
Each of the next *m* lines contains an integer *a[i]* where $0 \le i < n$.
The next line contains the integer *m*, the size of the array *a*.
Each of the next *m* lines contains an integer *b[i]* where $0 \le i < n$.

▾ Sample Case 0

Sample Input 0

```
2
0
0
```

Sample Output 0

```
2
```

Explanation 0
The following arguments are passed to your function:

n = 2
a = {}
b = {}

There are two tasks and no dependencies. Eric can complete *2* tasks.

▾ Sample Case 1

Sample Input 1

```
2
1
1
1
2
```

Sample Output 1

```
2
```

Explanation 1
The following arguments are passed to your function:

n = 2
a = {1}
b = {2}

There is only one dependency, so it can be satisfied. Eric can complete task *2* then task *1*.

▾ Sample Case 2

Sample Input 2

```
2
2
1
2
2
2
1
```

Sample Output 2

```
1
```

Explanation 2

There are two tasks and two dependencies. Each of the tasks must be performed before the other. Eric can choose only *1* task to complete.