1h 57m left

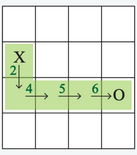
ALL

1. Laser Grid Ver.2

A player stands on a cell within a grid. The player can move to one of four adjacent cells, but the motion is constrained by lasers. To move from one position to another involves a cost: the cost to move between row *row i + 1* is *costRows[i]* and the cost to move between column *j* and column *j + 1* is *costCols[j]*. Find the min cost to move from a starting point to an ending point within the grid.

Example

rows = 4
cols = 4
initR = 1
initC = 0
finalR = 2
finalC = 3
costRows = [1, 2, 3]
costCols = [4, 5, 6]



1h 57m left

The player starts at position *(row, col) = (1, 0)* (marked with an X) and must reach position *(2, 3)* (marked wit O).

- To move from *row 1* from *row 2*, the cost is *costRows[1] = 2*.
- To move from column *0* to column *3*, the player must move to column *1*, then *2*, then *3* at a cost of *costCols[0] costCols[1] + costCols[2] = 4 + 5 + 6 = 15*.
- The total cost is *2 + 15 = 17*.

Function Description

Complete the function *minCost* in the editor below.

minCost has the following parameters:
int rows: the number of rows in the grid
int cols: the number of columns in the grid
int initR: the player's starting row
int initC: the player's starting column
int finalR: the goal's row
int finalC: the goal's column
int costRows[n]: each *costRows[i]* denotes the cost to move between rows *i* and *i + 1*.
int costCols[m]: each *costCols[j]* denotes the cost to move between columns *j* and *j + 1*.

Returns:
int: the minimum cost to move from the starting position to the goal

Constraints

- $1 \le rows, cols \le 10^5$
- *0 ≤ initR, finalR< rows*
- *0 ≤ initC, finalC < cols*
- $0 \le costRows[i] \le 10^4$ *(0 ≤ i < rows-1)*
- $0 \le costCols[j] \le 10^4$ *(0 ≤ j < cols-1)*

Input Format For Custom Testing

1h 57m left

The first line contains an integer, *rows*, the number of rows in the grid.

The next line contains an integer, *cols*, the number of columns in the grid.

The next line contains an integer, *initR*, the row where the player starts.

The next line contains an integer, *initC*, the column where the player starts.

The next line contains an integer, *finalR*, the row where the player ends.

The next line contains an integer, *finalC*, the final column where the player ends.

The next line contains the integer, *n = rows-1*, the number of elements in *costRows[]*.

Each line *i* of the *rows-1* subsequent lines (where *0 ≤ i < rows-1*) contains an integer that describes *costRo*

The next line contains the integer, *m = cols-1*, the number of elements in *costCols[]*.

Each line *j* of the *cols-1* subsequent lines (where *0 ≤ j < cols-1*) contains an integer that describes *costCols[*

Sample Case 0

Sample Input

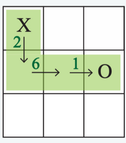
```
STDIN       Function
-----       -----
3       →   rows = 3
3       →   cols = 3
0       →   initial row initR = 0
0       →   initial column initC = 0
1       →   finalR = 1
2       →   finalC = 2
2       →   costRows[] size n = 2
2       →   costRows = [2, 5]
5
2       →   costCols[] size m = 2
6       →   costCols = [6, 1]
1
```

1h 57m left

Sample Output

```
9
```

Explanation



The player must move down one row (*cost = 2*) and over two columns (*cost = 6 + 1 = 7*) for a total cost of *2 = 9*.

Sample Case 1

1h 57m left

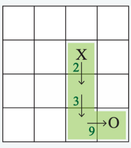
Sample Input

```
STDIN       Function
-----       -----
4       →   rows = 4
4       →   cols = 4
1       →   initial row initR = 1
2       →   initial column initC = 2
3       →   finalR = 3
3       →   finalC = 3
3       →   costRows[] size n = 3
1       →   costRows = [1, 2, 3]
2
3
3       →   costCols[] size m = 3
7       →   costCols = [7, 8, 9]
8
9
```

Sample Output

```
14
```

Explanation



The player must move from row *1* to row *3* for a cost = *2 + 3 = 5* and then from column *2* to column *3* for a of *9*. The total cost is *5 + 9 = 14*.

ALL

2. Which is the right answer to the following?

What is the running time of the following algorithm:

```
Procedure A(n)
begin
    if (n < 3)
        return 1
```

kakao [카카오/추천팀] 2021 봄 인턴십 1차 코딩테스트 Answered: 0 / 5 01 hour 57 mins

```
end
```

ALL

is best described by

Pick **ONE** option

- O (n)
- O (log n)
- O (log log n)
- O (1)

Clear Selection

1 2 3 4 5

Continue

3. Maximum Earnings

A taxi driver knows the pick-up and drop-off locations of people who are requesting taxi services. All the locations are in *km* from the starting point. The starting point is at *0 km*.
For each *km* travelled by a passenger, the driver charges *1* unit of money per passenger. Moreover, the taxi is very fancy. Therefore some people are even willing to pay an extra tip if they get to travel in the taxi. At a point of time, the taxi can only accommodate one passenger. Determine the maximum amount the driver earn.

Example
pickup = [0, 2, 9, 10, 11, 12]
drop = [5, 9, 11, 11, 14, 17]
tip = [1, 2, 3, 2, 2, 1]

The way to earn the most money is by accepting passengers at indices 1, 2 and 5.

- The amount paid by the passenger at index 1: 9 - 2 + 2 = 9
- The amount paid by the passenger at index 2: 11 - 9 + 3 = 5
- The amount paid by the passenger at index 5: 17 - 12 + 1 =6
- The total amount paid by the passengers is 9 + 5 + 6 = 20

Therefore, the return value is 20.

Function Description

Complete the function *taxiDriver* in the editor below. The function must return an integer denoting the maximum amount that can be earned by the driver.

taxiDriver has the following parameter(s):

pickup[pickup[0],...pickup[n-1]]: an array of *n* integers that denote the pickup location of the potential ride
drop[drop[0],...drop[n-1]]: an array of *n* integers that denote the drop-off locations of the potential riders
tip[tip[0],...tip[n-1]]: an array of *n* integers that denote the tips offered by each person if they are accepted ride

Constraints

- $0 < |pickup|, |drop|, |tip| \leq 10^4$
- $0 \leq pickup_i, drop_i \leq 10^{12}$
- $pickup_i < drop_i$
- $0 \leq tip \leq 10^5$

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *pickup*.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, $pickup_i$.
The next line contains the same integer, *n*, the number of elements in *drop*.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, $drop_i$.
The next line contains the same integer, *n*, the number of elements in *tip*.
Each line *i* of the *n* subsequent lines (where $0 \leq i < n$) contains an integer, tip_i.

Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
2         → pickup[] size n = 2
1         → pickup[] = [ 1, 4 ]
4
2         → drop[] size n = 2
5         → drop[] = [ 5, 6 ]
6
2         → tip[] size n = 2
2         → tip[] = [ 2, 5 ]
5
```

Sample Output

```
7
```

Explanation
There are two people, and locations are overlapping so only one of them can be accepted.
If person 1 is picked, the amount made is 5-1+2 = 6
If person 2 is picked, the amount made is 6-4+5 = 7

It is best to pick person 2 and earn 7.

Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
3         → pickup[] size n = 3
0         → pickup[] = [ 0, 4, 5 ]
4
5
3         → drop[] size n = 3
3         → drop[] = [ 3, 5, 7 ]
5
7
3         → tip[] size n = 2
1         → tip[] = [ 1, 2, 2 ]
2
2
```

Sample Output

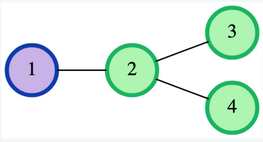
```
11
```

Explanation
All three passengers can be accepted because they do not overlap.

4. Delivery Management System

A manufacturing company is located in a certain city. Their goods need to be shipped to other cities that ar connected with bidirectional roads, though some cities may not be accessible because roads don't connect them. The order of deliveries is determined first by distance, then by priority. Given the number of cities, th connections via roads, and what city the manufacturing company is located in, determine the order of cities where the goods will be delivered.

For example, let's say that the number of cities is *cityNodes = 4*, where *cityFrom = [1, 2, 2]*, *cityTo = [2, 3, 4]*, *company = 1. In other words, the manufacturing company is located in city 1, and the roads run between c and 2, cities 2 and 3, and cities 2 and 4, like so:*



In this case, the cities would be visited based on the following logic:

- The closest city (or cities) is visited first. This is city 2, which is 1 unit from the manufacturing company.
- The next-closest city (or cities) is visited next. This is city 3 and city 4, which are both 2 units from the manufact company.
  - In this case, priority is then calculated, visiting the smaller-numbered city first (city 3) and continuing in ascending order (city 4).

Therefore, the order is [2, 3, 4], which is the answer you would return.

Function Description
Complete the function *order* in the editor below.

order has the following parameters:
int *cityNodes:* the number of cities
int *cityFrom[n]:* an array of integers denoting the first city node where there is a bidirectional node
int *cityTo[n]:* an array of integers denoting the second city node where there is a bidirectional node
int *company:* employee who invites all other employees, the node where the route starts
Returns:
int[]: an array of integers denoting the cities where the goods will be delivered in the order they will be delivered

Constraints

- $2 \le cityNodes \le 10^5$
- $1 \le n \le \min(^{(cityNodes \times (cityNodes - 1))}/_{2}, 10^5)$
- $1 \le cityFrom[i], cityTo[i], company \le n$
- $cityFrom[i] \ne cityTo[i]$

Input Format For Custom Testing

The first line contains two space-separated integers: *cityNodes,* denoting the number of cities, and *n,* deno the number of roads.
Each line *i* of the *n* subsequent lines (where $0 \le i < n$) contains two space-separated integers, *cityFrom[i]* a *cityTo[i]*
The next line contains an integer, *company.*

Sample Case 0

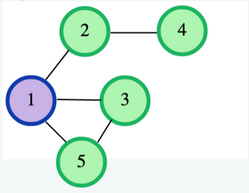
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
5 5   →   cityNodes = 5, n = 5
1 2   →   cityFrom = 1, cityTo = 2
1 3   →   cityFrom = 1, cityTo = 3
2 4   →   cityFrom = 2, cityTo = 4
3 5   →   cityFrom = 3, cityTo = 5
1 5   →   cityFrom = 1, cityTo = 5
1     →   company = 1
```

Sample Output

```
2
3
5
4
```

Explanation



Cities 2, 3, and 5 are all 1 unit of distance away from the manufacturing company. These are visited based priority in ascending order, so [2, 3, 5]. City 4 is 2 units of distance away from the manufacturing company is visited next. Therefore, the final order is [2, 3, 5, 4].

Sample Case 1

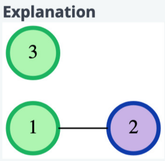
Sample Input For Custom Testing

```
STDIN     Function
-----     --------
3 1   →   cityNodes = 3, n = 1
1 2   →   cityFrom = 1, cityTo = 2
2     →   company = 2
```

Sample Output

```
1
```

Explanation



City 1 is located 1 unit of distance away from the manufacturing company. City 3 is not accessible because there are no roads connecting it to the manufacturing company's city. Therefore, the answer is [1].

5. Top Articles for Author

In this challenge, the REST API contains information about a collection of articles. Given a username and a l value, the goal is to return articles authored by the user with the given username, ordered by the number c comments they have and limited to the first ones in that order by the given limit value.

To access the collection of users perform HTTP GET request to *https://jsonmock.hackerrank.com /api/articles?author=<username>&page=<pageNumber>* where *<username>* is the username of the autho articles to search for and *<pageNumber>* is an integer that denotes the page of results to return.

For example, a GET request to *https://jsonmock.hackerrank.com/api/articles?author=epaga&page=1* will re the first page of the collection of articles authored by the user with username "epaga". Pages are numbered 1.

The response to such request is a JSON with the following 5 fields:

- *page*: the current page of the results
- *per_page*: the maximum number of users returned per page.
- *total*: the total number of users on all pages of the result.
- *total_pages*: the total number of pages with results.
- *data*: an array of objects containing users returned on the requested page

Each user record has the following schema:

- *title*: the title of the article, can be null
- *url*: the URL of the article
- *author*: the username of the author of the article
- *num_comments*: **the number of comments the article has, can be null which means it has 0 comments**
- *story_id*: identifier of the story related to the article, can be null
- *story_title*: the title of the story related to the article, can be null
- *story_url*: the URL of the story related to the article, can be null
- *parent_id*: identifier of the parent of the article, can be null
- *created_at*: the date and time when the record was created

When considering an article:

- If both the title and the story title of an article are null, then ignore the article.
- Otherwise:
  - If the *title* is not null, the *name* of the article is *title.*
  - If the *title* is null, the name of the article is *story_title.*

Given *username* and *limit* values, return an array of names of articles authored by the user with the given username, ordered by the number of comments they have, descending. Limit the list to *limit* records. If mu articles have the same number of comments, order them ascending by *name.*

Function Description
Complete the function *topArticles* in the editor below.

topArticles has the following parameter(s):
  string *username:* the username of the author
  int *limit*: the maximum number of articles to return
Returns:
  string[]: the names of articles retrieved, selected and ordered as described

Input Format For Custom Testing

In the first line, there is a string *username.*
In the second line, there is an integer *limit.*

Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
olalonde →  username = 'olalonde'
1        →  limit = 1
```

Sample Output

```
Show HN: This up votes itself
```

Explanation
We are interested in the top 1 articles authored by the user with username *olalonde*. There are 2 articles authored by *olalonde* available in the API response:

1. *title*: null, *story_title*: Guacamole – A clientless remote desktop gateway, *num_comments*: null
2. *title*: Show HN: This up votes itself, *story_title*: null, *num_comments*: 83

The name of the first article is its story title because its title is null, while the name of the second article is title. The number of comments in the first article is null, so we treat is as having 0 comments. When those articles are ordered ascending by comment count, only the second article appears in the top *limit = 1.* Therefore, the name of the second article is the only one appearing in the result.