Question - 1
Python: List of Even Integers Function

Implement a function that:

1. is named *even*.
2. takes 2 integer arguments, *start* and *n*.
3. returns a list of *n* smallest even integers greater than or equal to *start*, in ascending order.

Implementation of the function will be tested by a provided code stub on several input files. Each input file contains parameters for the function call. The function will be called with those parameters, and the result of its execution will be printed to the standard output by the provided code.

Constraints

- $1 \le start, n \le 100$

▼ Input Format Format for Custom Testing

In the first and only line, there are two space-separated integers, *start* and *n*.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
2 4     →   start = 2, n = 4
```

Sample Output

```
2 4 6 8
```

Explanation

The function must return a list of the 4 smallest even integers that are greater than or equal to 2, sorted into ascending order: 2, 4, 6, and 8.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
5 7     →   start = 5, n = 7
```

Sample Output

```
6 8 10 12 14 16 18
```

Explanation

The function must return a list of the 7 smallest even integers that are greater than or equal to 5, sorted in ascending order: 6, 8, 10, 12, 14, 16, 18.

Help

Question - 2
Python: Return or raise ValueError

Implement a function, *multiply,* that takes 3 integer arguments: *a*, *b* and *bound*.

- If the result of multiplying *a* and *b* is less than or equal to *bound*, the function returns the result.
- If the result of multiplying *a* and *b* is greater than *bound*, the function raises a ValueError exception with the following message: if *a=2*, *b=5*, and *bound=8*, then the message must be: "multiplication of 2 and 5 with bound 8 not possible"

Implementation of the function will be tested by a provided code stub on several input files. Each input file contains several queries, and each query contains parameters for the function call. The function will be called with those parameters and the result of its execution will be printed to the standard output by the provided code.

Constraints

- 1 ≤ the number of queries in one test file ≤ 1000
- 1 ≤ *a, b* ≤ *bound* ≤ 10^5

▼ Input Format Format for Custom Testing

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

In the first line, there is a single integer *q*.
Then, *q* lines follow. In the i^{th} of them, there are 3 space-separated integers that denote the values for *a*, *b*, and *bound* respectively.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
3        → number of queries q = 3
5 2 23   → query[0]: a = 5, b = 2, bound = 23
5 2 10   → query[1]: a = 5, b = 2, bound = 10
5 2 9    → query[2]: a = 5, b = 2, bound = 9
```

Sample Output

```
10
10
multiplication of 5 and 2 with bound 9 not
possible
```

Explanation
There are 3 queries. In all of them, a=5 and b=2 but the bound parameter is different. In the first query, bound is 23 and since 5*2=10 is not greater than 23, 10 is returned by the function. Similarly, in the second query, 5*2=10 is not greater than 10, so 10 is returned. In the last query, bound is 9, and since 5*2 is greater than

9, the function raises an exception with the message "multiplication of 2 and 5 with bound 9 not possible".

▼ **Sample Case 1**

Sample Input

```
STDIN        Function
-----        --------
2          → number of queries q = 2
11 11 120  → query[0]: a = 11, b = 11, bound =
120
7 8 100    → query[1]: a = 7, b = 8, bound = 100
```

Sample Output

```
multiplication of 11 and 11 with bound 120 not
possible
56
```

Explanation
There are 2 queries. In the first query, 11*11=121, is greater than 120, so the function raises the exception with the message "multiplication of 11 and 11 with bound 120 not possible". In the second query, 7*8=56 is not greater than 100, so 56 is returned.

Question - 3
Python: Increasing List

Implement a variation of a list type in Python, specifically a type IncreasingList that acts as an increasing list of elements. It must implement the following 3 methods:

- append(self, val): First, it removes all elements from the list that have greater values than *val*, starting from the last one. Once there are no greater elements in the list, it appends *val* to the end of the list
- pop(self): It removes the last element from the list if the list is not empty; otherwise, it does nothing.
- __len__(self): returns the number of elements in the list

Additional methods may be implemented as necessary. Notice that at any moment, by the definition of append and pop operations, the elements in the list are guaranteed to be sorted in non-decreasing order.

The implementations of the 3 required methods will be tested by a provided code stub on several input files. Each input file contains several operations, each one of the types below. Values returned by size operations are printed to the standard output by the provided code stub.

- append val: calls append(val) on the IncreasingList instance
- pop: calls pop() on the IncreasingList instance

- size: calls len(obj), where obj is an instance of IncreasingList, and prints the returned value to the standard output

Complete the class IncreasingList in the editor below. The class must have the 3 methods given above (append, pop, __len__).

Constraints

- 1 ≤ number of operations in one test file ≤ 10^5
- If *val* is a parameter of operation, then *val* is an integer and 1 ≤ val ≤ 10^5
- It is guaranteed that there is at least one operation of type *size* in every test file.

▼ Input Format Format for Custom Testing

In the first line, there is a single integer, *q*, the number of queries. Then, *q* lines follow. In the i^{th} of them, there is a string that denotes an operation and, optionally, an integer that denotes the parameter of the operation.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       --------
9         → number of queries, q = 9
size      → operations = ["size", "append 2", ...
, "pop", "size"]
append 2
append 4
append 5
size
append 2
size
pop
size
```

Sample Output

```
0
3
2
1
```

Explanation

There are 9 operations to be performed. We start with the empty list: *lst = []*.

1. The first operation asks for the size of the list, so 0 is printed to the output.
2. The second operation appends 2 to the list. Since there are no elements greater than 2 in the list, no elements are removed and 2 is appended: *lst = [2]*.
3. The third operation appends 4 to the list. Again, since there are no elements greater than 4 in the list, no elements are removed and 4 is appended: *lst = [2, 4]*.
4. The fourth operation appends 5 to the list. Again, since there are no elements greater than 5 in the list, no elements are removed and 5 is appended: *lst = [2, 4, 5]*.
5. The next operation asks for the size of the list, so 3 is printed to the output.

6. The next operation appends 3 to the list. There are two elements greater than 3 in the list, so they are removed first, starting from the end. After they are removed, 3 is appended: *lst = [2, 3]*.
7. The next operation asks for the size of the list, so 2 is printed to the output.
8. The next operation is pop, so it removes the last element from the list: *lst = [2]*.
9. Finally, the last operation asks for the size of the list, so 1 is printed to the output.

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       --------
4         → number of queries, q = 4
append 3 → operations = ["append 3", "pop",
"pop", "size"]
pop
pop
size
```

Sample Output

```
0
```

Explanation

There are 4 operations to be performed. We start with the empty list: *lst = []*.

1. The first operation appends 3 to the list. Since there are no elements greater than 3 in the list, no elements are removed and 3 is appended: *lst = [3]*.
2. The second operation removes the last element from the list: *lst = []*.
3. The next operation also tries to remove the last element from the list, but since the list is already empty, nothing happens: *lst = []*.
4. Finally, the last operation asks for the size of the list, so 0 is printed to the output.

Question - 4
Alphabet Filter

Given a string consisting of only lowercase characters, create two methods that remove all the consonants or vowels from the given word. They must retain the original order of the characters in the returned strings.

Example

s = 'onomatopoeia'

The *filter_vowels* method removes all vowels from *s* and returns the string 'nmtp'
The *filter_consonants* method removes all consonants from *s* and returns the string 'ooaooeia'

Function Description

For a given definition of a class *LetterFilter*, complete its methods *filter_vowels* and *filter_consonants*. The class takes a string in the constructor and stores it to its *s* attribute. The method *filter_vowels* must return a new string with all vowels removed from it. Similarly, the method *filter_consonants* must return a new string with all consonants removed from it.

Constraints

- The string contains only lowercase letters in the range ascii[a-z]
- The string contains at least one vowel and at least one consonant

▼ Input Format For Custom Testing

The first line contains a string, *s*, that denotes the string to be transformed.

▼ Sample Case 0

Sample Input 0

```
STDIN        Function
-----        --------
hackerrank → string s = 'hackerrank'
```

Sample Output 0

```
hckrrnk
aea
```

Explanation 0

The first result is after removing all vowels, {a,e,i,o,u}, from the string. The second result is after removing all consonants.

▼ Sample Case 1

Sample Input 1

```
STDIN          Function
-----          --------
programming →  string s = 'programming'
```

Sample Output 1

```
prgrmmng
oai
```

Explanation 1

The first result is after removing all vowels, {a,e,i,o,u}, from the string. The second result is after removing all consonants.

Question - 5
Get Author Articles

Write an *HTTP GET* method to retrieve information from an articles database. The query response is paginated and can be further accessed by appending to the query string &*page=num* where *num* is the page number.

Given a string of author, *getArticlesTitle* must perform the following tasks:

1. Query *https://jsonmock.hackerrank.com/api/articles?author=<authorName>* . (replace *<authorName>*).
2. Initialize the *titles* array to store a list of string elements.
3. Store the name of each article returned in the *data* field to the *titles* array using the following logic:
    - If both *title* and *story_title* are null, ignore the article.
    - Otherwise:
        - If *title* is not null, use *title* as the name.
        - If *title* is null, use *story_title* as the name.
4. Based on the *total_pages* count, fetch all the data (pagination) and perform step 3 for each.
5. Return the array of titles.

The query response from the website is a JSON response with the following five fields:

- *page*: The current page.
- *per_page*: The maximum number of results per page.
- *total*: The total number of records in the search result.
- *total_pages*: The total number of pages that must be queried to get all the results.
- *data*: An array of JSON objects containing article information

Function Description

Complete the function *getArticleTitles* in the editor below.

getArticlesTitles has the following parameter(s):

string *author:* the author string to search on

Returns:

string[]: a list of the articles as described

▼ Schema

You are provided 1 table: ARTICLES.

ARTICLES		
Name	Type	Description
title	STRING	This is the first column. It is the title of the article
url	STRING	URL of the article
author	STRING	the author name of the article
num_comments	LONG	total number of comments
story_id	LONG	unique identifier number for the article
story_title	STRING	an additional title for the article
story_url	STRING	an additional URL for the article
parent_id	LONG	unique identifier number of the parent article
created_at	LONG	created time of the article

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
epaga   →   author = 'epaga'
```

Sample Output

```
A Message to Our Customers
Apple’s declining software quality
```

Explanation

ARTICLES					
title	url	author	num_comments	story_id	st
A Message to Our Customers	null	epaga	967	null	
Google Is Eating Our Mail	null	saintamh	685	null	
null	null	epaga	705	null	Apple’s declir

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN         Function
-----         --------
saintamh  →   author = 'saintamh'
```

Sample Output

```
Google Is Eating Our Mail
```

Explanation

ARTICLES

title	url	author	num_comments	story_id	st
A Message to Our Customers	null	epaga	967	null	
Google Is Eating Our Mail	null	saintamh	685	null	
null	null	epaga	705	null	Apple’s declir

Question - 6
Root Threshold in IOT Devices

In this challenge, the REST API contains information about a collection of IoT devices. Given status query, threshold, year and month parameters, the goal is to use the API to get the total number of available IoT devices with the status containing given status query, having their root threshold parameter strictly greater than the given threshold value, and such that were added to the collection in the given month of the given year.

To access the collection of devices with some status perform HTTP GET request to:
https://jsonmock.hackerrank.com/api/iot_devices/search?status=<statusQuery>&page=<pageNumber>
where <statusQuery> is a string such that all devices having <statusQuery> as a substring of their status value (using case-insensitive matching) will be included in the result and <pageNumber> is integer denoting the page of the results to return.

For example, GET request to:
https://jsonmock.hackerrank.com/api/iot_devices/search?status=STOP&page=2

will return the second page of the devices with their status containing "STOP" as a substring (using case-insensitive matching) of their statuses. Pages are numbered from 1, so in order to access the first page, you need to ask for page number 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 devices returned per page.
- `total` : The total number of devices available on all pages of the result.
- `total_pages` : The total number of pages with results.
- `data` : An array of objects containing devices returned on the requested page

Each device object has the following schema:

- `id` : The unique ID of the device
- `timestamp` : The timestamp when the device was added to the collection, in UTC milliseconds
- `status` : The status of the device
- `operatingParams` : the object containing operating parameters of the device
- `asset` : The object containing information about the asset of the device
- `parent` : Optional. The object containing information about the parent of the device

The operating parameters object has the following schema:

- `rotorSpeed` : The rotor speed of the device
- `slack` : The slack in the device
- `rootThreshold` : The root threshold for the device

The asset object has the following schema:

- `id`: The unique ID of the asset
- `alias` : The alias for the asset

The parent object has the following schema:

- `id` : The unique ID of the parent of the asset
- `alias` : The alias for the parent of the asset

Given string *statusQuery*, numerical *threshold* value, and date in format MM-YYYY, e.g. 08-2012, the goal is to return the total number of devices that have *statusQuery* as a substring of their status (using case-insensitive matching), having their root threshold parameter strictly greater than the give *threshold* value, and such that were added to the collection in the given month of the given year.

Function Description

Complete the function *numDevices* in the editor below.

numDevices has the following parameter(s):

statusQuery: string denoting the substring of devices status to query for

threshold: integer denoting the threshold value for the devices root threshold parameter

dateStr: string in format MM-YYYY denoting the month and the year to query for

The function must return an integer denoting the total number of devices matching the given criteria.

▼ Input Format For Custom Testing

In the first line, there is a string *statusQuery*.
In the second line, there is an integer *threshold*.
In the third line, there is a string *dateStr*.

▼ Sample Case 0

Sample Input For Custom Testing

```
STOPPED
45
04-2019
```

Sample Output

```
3
```

Explanation
The status query is "STOPPED", the threshold value is 45 and we are interested in the devices added to the collection in April 2019. There are a total of 3 devices with their status containing "STOPPED" as a substring (using case-insensitive matching), having their root threshold strictly greater than 45 and such that were added to the collection in April 2019, therefore, the answer is 3.

Question - 7
Most Active Authors

In this challenge, the REST API contains information about a collection of users and articles they created. Given the threshold value, the goal is to use the API to get the list of most active authors. Specifically, the list of usernames of users with submission count strictly greater than the given threshold. The list of usernames must be returned in the order the users appear in the results.

To access the collection of users perform HTTP GET request to:
https://jsonmock.hackerrank.com/api/article_users?page=<pageNumber>
where <pageNumber> is an integer denoting the page of the results to return.

For example, GET request to:
https://jsonmock.hackerrank.com/api/article_users/search?page=2
will return the second page of the collection of users. Pages are numbered from 1, so in order to access the first page, you need to ask for page number 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:

- `id` : unique ID of the user
- `username` : the username of the user
- `about` : the about information of the user
- `submitted` : total number of articles submitted by the user
- `updated_at` : the date and time of the last update to this record
- `submission_count` : the number of submitted articles that are approved
- `comment_count` : the total number of comments the user made
- `created_at` : the date and time when the record was created

Function Description

Complete the function *getUsernames* in the editor below.
getUsernames has the following parameter(s):
threshold: integer denoting the threshold value for the number of submission count

The function must return an array of strings denoting the usernames of the users whose submission count is strictly greater than the given threshold. The usernames in the array must be ordered in the order they appear in the API response.

▼ Input Format For Custom Testing

In the first line, there is an integer *threshold*.

▼ Sample Case 0

Sample Input For Custom Testing

```
10
```

Sample Output

```
epaga
panny
olalonde
WisNorCan
dmmalam
replicatorblog
vladikoff
mpweiher
coloneltcb
guelo
```

Explanation
The threshold value is 10, so the result must contain usernames of users with the submission count value greater than 10. There are 10 such users and their usernames in the order they appear in the API response are: panny, olalonde, WisNorCan, dmmalam, replicatorblog, vladikoff, mpweiher, coloneltcb, guelo.

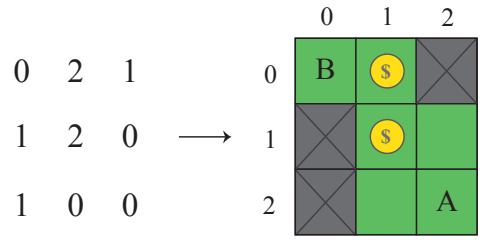
Question - 8

Bob Navigates a Maze

Bob and Alice have teamed up on a game show. After winning the first round, they now have access to a maze with hidden gold. If Bob can collect all the gold coins and deliver them to Alice's position, they can split the gold. Bob can move horizontally or vertically as long as he stays in the maze, and the cell is not blocked.

The maze is represented by an *n × m* array. Each cell has a value, where *0* is *open, 1 is blocked*, and *2* is *open with a gold coin*. Bob starts at the top left in cell in *(row, column) = (0, 0)*. Alice's position is given by *(x,y)*.
Determine the shortest path Bob can follow to collect all gold coins and deliver them to Alice. If Bob can't collect and give all the gold coins, return *-1*.

Example: *maze = [[0,2,1],[1,2,0],[1,0,0]]* with Alice at *(2,2)* is represented as follows:



B represents Bob's starting position at *(0,0)*, and A represents Alice's position (which will also be Bob's final position). As Bob starts at *(0,0)*, he has two possible paths to collect the gold and deliver to Alice*: (0, 0) → (0, 1) → (1, 1) → (2, 1) → (2, 2)* and *(0, 0) → (0, 1) → (1, 1) → (1, 2) → (2,2). Both paths have a length of 4 and could represent the shortest path.*

Function Description

Complete the function *minMoves* in the editor below. The function must return the integer length of Bob's shortest path, or *-1* if it's not possible.

minMoves has the following parameter(s):

maze[maze[0][0],...maze[n-1][m-1]]: a 2D array of integers
x: an integer denoting Alice's row coordinate
y: an integer denoting Alice's column coordinate

Constraints

- *1 ≤ n, m ≤ 100*
- *0 ≤ the number of coins ≤ 10*
- *1 ≤ x < n*
- *1 ≤ y < m*

▼ Input Format For Custom Testing

The first line contains an integer *n*, the numbers of rows in *maze*.
The second line contains an integer *m*, the number of columns in *maze*.
Each of the next *n* lines contains *m* space-separated integers that describe the cells of each row in *maze*.
The next line contains an integer *x*.

The next line contains an integer, *y*.

▼ Sample Case 0

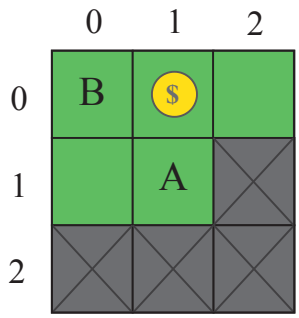
Sample Input 0

```
STDIN           Function Parameters
-----           -------------------
3       →       maze[][] number of rows n = 3
3       →       maze[][] number of columns m = 3
0 2 0   →       maze[][] = [ [0 2 0], [0 0 1], [1
1 1] ]
0 0 1
1 1 1
1       →       x = 1
1       →       y = 1
```

Sample Output 0

```
2
```

Explanation 0



The shortest path Bob can take is *(0, 0) → (0, 1) → (1, 1)*.

▼ Sample Case 1

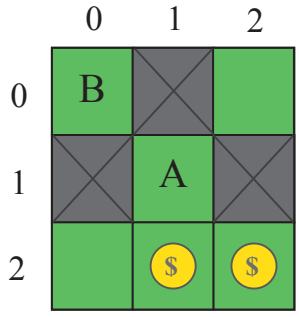
Sample Input 1

```
STDIN         Function Parameters
-----         -------------------
3       →     maze[][] number of rows n = 3
3       →     maze[][] number of columns m = 3
0 1 0   →     maze[][] = [ [0 1 0], [1 0 1], [0 2
2] ]
1 0 1
0 2 2
1       →      x = 1
1       →      y = 1
```

Sample Output 1

```
-1
```

Explanation 1



It is not possible for Bob to reach Alice, so return *–1*.

▼ Sample Case 2

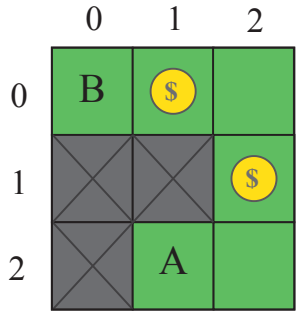
Sample Input 2

```
STDIN       Function Parameters
-----       -------------------
3       →   maze[][] number of rows n = 3
3       →   maze[][] number of columns m = 3
0 2 0   →   maze[][] = [ [0 2 0] , [1 1 2] , [1 0
0] ]
1 1 2
1 0 0
2       →   x = 2
1       →   y = 1
```

Sample Output 2

```
5
```

Explanation 2



The shortest path Bob can take is *(0, 0) → (0, 1) → (0, 2) → (1, 2) → (2, 2) → (2, 1).*

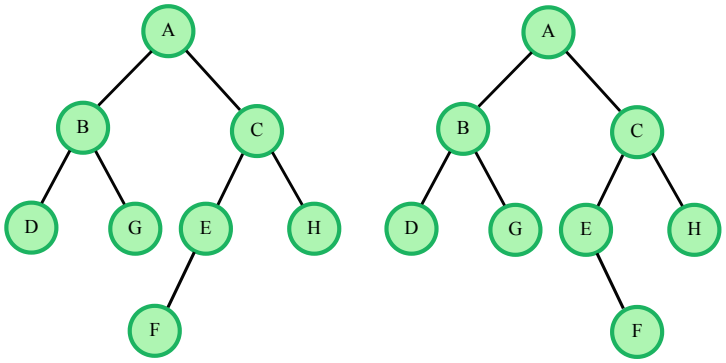
Question - 9

Is this a tree?

A binary tree is represented as a sequence of parent-child pairs, for example:

```
(A,B) (A,C) (B,G) (C,H) (E,F) (B,D) (C,E)
```

A tree with those edges may be illustrated in many ways. Here are two:



The following is a recursive definition for the S-expression of a tree:

```
S-exp(node) = ( node->val (S-exp(node->first_child))(S-exp(node->second_child))), if node != NULL = "", node == NULL
```

where, first_child->val < second_child->val (first_child->val is lexicographically smaller than second_child-> val)

This tree can be represented in an S-expression in multiple ways. The lexicographically smallest way of expressing it is as follows:

```
(A(B(D)(G))(C(E(F))(H)))
```

Translate the node-pair representation into its lexicographically smallest S-expression or report any errors that do not conform to the definition of a binary tree.

The list of errors with their codes is as follows:

Error Code	Type of error
E1	More than 2 children
E2	Duplicate Edges
E3	Cycle present (node is direct descendant of more than one node)
E4	Multiple roots
E5	Any other error

Function Description
Complete the function *sExpression* in the editor below. The function must return either the lexicographically lowest S-expression or the lexicographically lowest error code as a string.

sExpression has the following parameter(s):
nodes: a string of space-separated parenthetical elements, each of which contains the names of two connected nodes separated by a comma

Constraints:

1. All node names are single characters in the range *ascii[A-Z]*
2. The maximum node count is *26*.
3. There is no specific order to the input (parent, child) pairs.

▼ Input Format for Custom Testing

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

The first line contains a string *nodes*.

▼ Sample Case 0

Sample Input 0

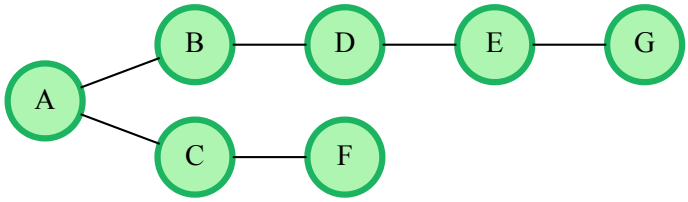
```
(B,D) (D,E) (A,B) (C,F) (E,G) (A,C)
```

Sample Output 0

```
(A(B(D(E(G))))(C(F)))
```

Explanation 0

A representation of the tree is as follows:



▼ Sample Case 1

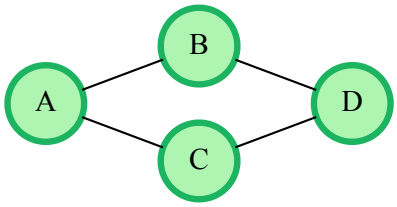
Sample Input 1

```
(A,B) (A,C) (B,D) (D,C)
```

Sample Output 1

```
E3
```

Explanation 1



Node *C* is a child of nodes *A* and *D*. Since *D* tries to attach itself as parent to a node already above it in the tree, this forms a cycle.

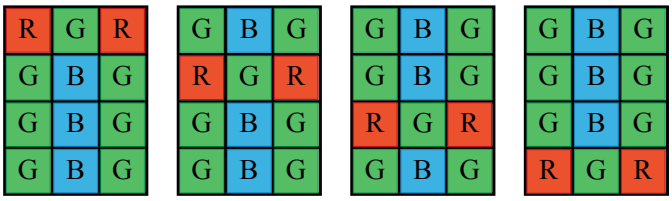
Question - 10
Coloring a Grid

An automated painting system needs a program that can paint an n x 3 grid in red, green, and blue such that no row or column contains cells that are all the same color. Determine the number of valid patterns that can be painted given *n* rows. Since the number of patterns can be large, return the value modulo (10^9+7).

Example

n = 4

Some examples of how the colors can be arranged for *4 x 3* grid are shown below. The total number of valid patterns is *296490* for a grid with 4 rows. Samples with 2 and 3 rows are given below.



Function Description

Complete the *countPatterns* function in the editor below.

countPatterns has only one parameter:

int n: the number of columns of the n *x 3* grid.

Return

int: the number of ways in which the gid can be colored, calculated as a modulo of (10^9+7)

Constraints

- $2 \leq n \leq 20000$

▼ Input Format For Custom Testing

The only line of input contains an integer, *n*, the number of rows of the *n x 3* grid.

▼ Sample Case 0

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
2       →   number of rows n = 2
```

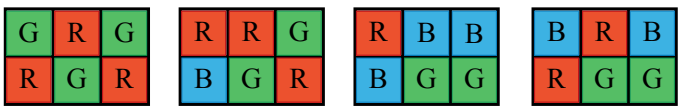
Sample Output

```
174
```

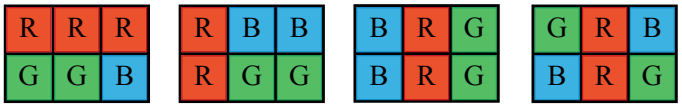
Explanation

Examples of valid patterns are shown below, as well as some invalid arrangements for comparison. There are 174 valid patterns.

Few Valid Arrangements



Few Invalid Arrangements



▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN       Function
-----       --------
```

```
3     →   number of rows n = 3
```

Sample Output

```
9750
```

Explanation

n = 3

Some valid ways to fill the grid: