Problem 355: Design Twitter

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user, and is able to see the

10

most recent tweets in the user's news feed.

Implement the

Twitter

class:

Twitter()

Initializes your twitter object.

void postTweet(int userId, int tweetId)

Composes a new tweet with ID

tweetId

by the user

userId

. Each call to this function will be made with a unique

tweetId

.

List<Integer> getNewsFeed(int userId)

Retrieves the

10

most recent tweet IDs in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user themself. Tweets must be

ordered from most recent to least recent

.

void follow(int followerId, int followeeId)

The user with ID

followerId

started following the user with ID

followeeId

.

void unfollow(int followerId, int followeeId)

The user with ID

followerId

started unfollowing the user with ID

followeeId

.

Example 1:

Input

["Twitter", "postTweet", "getNewsFeed", "follow", "postTweet", "getNewsFeed", "unfollow", "getNewsFeed"] [[], [1, 5], [1], [1, 2], [2, 6], [1], [1, 2], [1]]

Output

[null, null, [5], null, null, [6, 5], null, [5]]

Explanation

Twitter twitter = new Twitter(); twitter.postTweet(1, 5); // User 1 posts a new tweet (id = 5). twitter.getNewsFeed(1); // User 1's news feed should return a list with 1 tweet id -> [5]. return [5] twitter.follow(1, 2); // User 1 follows user 2. twitter.postTweet(2, 6); // User 2 posts a new tweet (id = 6). twitter.getNewsFeed(1); // User 1's news feed should return a list with 2 tweet ids -> [6, 5]. Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5. twitter.unfollow(1, 2); // User 1 unfollows user 2. twitter.getNewsFeed(1); // User 1's news feed should return a list with 1 tweet id -> [5], since user 1 is no longer following user 2.

Constraints:

1 <= userId, followerId, followeeId <= 500

0 <= tweetId <= 10

4

All the tweets have

unique

IDs.

At most

3 * 10

4

calls will be made to

postTweet

,

getNewsFeed

,

follow

, and

unfollow

.

A user cannot follow himself.

Code Snippets

C++:

```
class Twitter {
public:
Twitter() {

}

void postTweet(int userId, int tweetId) {

}
```

```
vector<int> getNewsFeed(int userId) {

}

void follow(int followerId, int followeeId) {

}

void unfollow(int followerId, int followeeId) {

}
};

/**
* Your Twitter object will be instantiated and called as such:
* Twitter* obj = new Twitter();
* obj->postTweet(userId,tweetId);
* vector<int> param_2 = obj->getNewsFeed(userId);
* obj->follow(followerId,followeeId);
* obj->unfollow(followerId,followeeId);
*/
```

Java:

```
class Twitter {

public Twitter() {

}

public void postTweet(int userId, int tweetId) {

}

public List<Integer> getNewsFeed(int userId) {

}

public void follow(int followerId, int followeeId) {

}
```

```
public void unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = new Twitter();
* obj.postTweet(userId,tweetId);
* List<Integer> param_2 = obj.getNewsFeed(userId);
* obj.follow(followerId,followeeId);
* obj.unfollow(followerId,followeeId);
*/
```

Python3:

```
class Twitter:

def __init__(self):


def postTweet(self, userId: int, tweetId: int) -> None:


def getNewsFeed(self, userId: int) -> List[int]:


def follow(self, followerId: int, followeeId: int) -> None:


def unfollow(self, followerId: int, followeeId: int) -> None:



# Your Twitter object will be instantiated and called as such:
# obj = Twitter()
# obj.postTweet(userId,tweetId)
# param_2 = obj.getNewsFeed(userId)
# obj.follow(followerId,followeeId)
# obj.unfollow(followerId,followeeId)
```

Python:

```
class Twitter(object):

def __init__(self):


def postTweet(self, userId, tweetId):
"""
:type userId: int
:type tweetId: int
:rtype: None
"""


def getNewsFeed(self, userId):
"""
:type userId: int
:rtype: List[int]
"""


def follow(self, followerId, followeeId):
"""
:type followerId: int
:type followeeId: int
:rtype: None
"""


def unfollow(self, followerId, followeeId):
"""
:type followerId: int
:type followeeId: int
:rtype: None
"""



# Your Twitter object will be instantiated and called as such:
# obj = Twitter()
# obj.postTweet(userId,tweetId)
```

```
# param_2 = obj.getNewsFeed(userId)
# obj.follow(followerId,followeeId)
# obj.unfollow(followerId,followeeId)
```

JavaScript:

```
var Twitter = function() {

};

/**
* @param {number} userId
* @param {number} tweetId
* @return {void}
*/
Twitter.prototype.postTweet = function(userId, tweetId) {

};

/**
* @param {number} userId
* @return {number[]}
*/
Twitter.prototype.getNewsFeed = function(userId) {

};

/**
* @param {number} followerId
* @param {number} followeeId
* @return {void}
*/
Twitter.prototype.follow = function(followerId, followeeId) {

};

/**
* @param {number} followerId
* @param {number} followeeId
* @return {void}
*/
```

```
Twitter.prototype.unfollow = function(followerId, followeeId) {

};

/**
* Your Twitter object will be instantiated and called as such:
* var obj = new Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

TypeScript:

```
class Twitter {
constructor() {

}

postTweet(userId: number, tweetId: number): void {

}

getNewsFeed(userId: number): number[] {

}

follow(followerId: number, followeeId: number): void {

}

unfollow(followerId: number, followeeId: number): void {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* var obj = new Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
```

```
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

C#:

```
public class Twitter {

public Twitter() {

}

public void PostTweet(int userId, int tweetId) {

}

public IList<int> GetNewsFeed(int userId) {

}

public void Follow(int followerId, int followeeId) {

}

public void Unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = new Twitter();
* obj.PostTweet(userId,tweetId);
* IList<int> param_2 = obj.GetNewsFeed(userId);
* obj.Follow(followerId,followeeId);
* obj.Unfollow(followerId,followeeId);
*/
```

C:

```
```

```
typedef struct {

} Twitter;


Twitter* twitterCreate() {

}

void twitterPostTweet(Twitter* obj, int userId, int tweetId) {

}

int* twitterGetNewsFeed(Twitter* obj, int userId, int* retSize) {

}

void twitterFollow(Twitter* obj, int followerId, int followeeId) {

}

void twitterUnfollow(Twitter* obj, int followerId, int followeeId) {

}

void twitterFree(Twitter* obj) {

}

/**
* Your Twitter struct will be instantiated and called as such:
* Twitter* obj = twitterCreate();
* twitterPostTweet(obj, userId, tweetId);

* int* param_2 = twitterGetNewsFeed(obj, userId, retSize);

* twitterFollow(obj, followerId, followeeId);

* twitterUnfollow(obj, followerId, followeeId);

* twitterFree(obj);
```

```
*/
```

Go:

```
type Twitter struct {

}


func Constructor() Twitter {

}


func (this *Twitter) PostTweet(userId int, tweetId int) {

}


func (this *Twitter) GetNewsFeed(userId int) []int {

}


func (this *Twitter) Follow(followerId int, followeeId int) {

}


func (this *Twitter) Unfollow(followerId int, followeeId int) {

}


/**
* Your Twitter object will be instantiated and called as such:
* obj := Constructor();
* obj.PostTweet(userId,tweetId);
* param_2 := obj.GetNewsFeed(userId);
* obj.Follow(followerId,followeeId);
* obj.Unfollow(followerId,followeeId);
*/
```

Kotlin:

```
class Twitter() {

fun postTweet(userId: Int, tweetId: Int) {

}

fun getNewsFeed(userId: Int): List<Int> {

}

fun follow(followerId: Int, followeeId: Int) {

}

fun unfollow(followerId: Int, followeeId: Int) {

}

}

/**
* Your Twitter object will be instantiated and called as such:
* var obj = Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

Swift:

```

class Twitter {

init() {

}

func postTweet(_ userId: Int, _ tweetId: Int) {
```

```
}

func getNewsFeed(_ userId: Int) -> [Int] {

}

func follow(_ followerId: Int, _ followeeId: Int) {

}

func unfollow(_ followerId: Int, _ followeeId: Int) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* let obj = Twitter()
* obj.postTweet(userId, tweetId)
* let ret_2: [Int] = obj.getNewsFeed(userId)
* obj.follow(followerId, followeeId)
* obj.unfollow(followerId, followeeId)
*/
```

Rust:

```
struct Twitter {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Twitter {

fn new() -> Self {

}

fn post_tweet(&self, user_id: i32, tweet_id: i32) {
```

```
}

fn get_news_feed(&self, user_id: i32) -> Vec<i32> {

}

fn follow(&self, follower_id: i32, followee_id: i32) {

}

fn unfollow(&self, follower_id: i32, followee_id: i32) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* let obj = Twitter::new();
* obj.post_tweet(userId, tweetId);
* let ret_2: Vec<i32> = obj.get_news_feed(userId);
* obj.follow(followerId, followeeId);
* obj.unfollow(followerId, followeeId);
*/
```

Ruby:

```
class Twitter
def initialize()

end


=begin
:type user_id: Integer
:type tweet_id: Integer
:rtype: Void
=end
def post_tweet(user_id, tweet_id)

end
```

```
=begin
:type user_id: Integer
:rtype: Integer[]
=end
def get_news_feed(user_id)

end


=begin
:type follower_id: Integer
:type followee_id: Integer
:rtype: Void
=end
def follow(follower_id, followee_id)

end


=begin
:type follower_id: Integer
:type followee_id: Integer
:rtype: Void
=end
def unfollow(follower_id, followee_id)

end


end

# Your Twitter object will be instantiated and called as such:
# obj = Twitter.new()
# obj.post_tweet(user_id, tweet_id)
# param_2 = obj.get_news_feed(user_id)
# obj.follow(follower_id, followee_id)
# obj.unfollow(follower_id, followee_id)
```

PHP:

```
class Twitter {
/**
*/
function __construct() {

}

/**
* @param Integer $userId
* @param Integer $tweetId
* @return NULL
*/
function postTweet($userId, $tweetId) {

}

/**
* @param Integer $userId
* @return Integer[]
*/
function getNewsFeed($userId) {

}

/**
* @param Integer $followerId
* @param Integer $followeeId
* @return NULL
*/
function follow($followerId, $followeeId) {

}

/**
* @param Integer $followerId
* @param Integer $followeeId
* @return NULL
*/
function unfollow($followerId, $followeeId) {

}
}
```

```
/**
* Your Twitter object will be instantiated and called as such:
* $obj = Twitter();
* $obj->postTweet($userId, $tweetId);
* $ret_2 = $obj->getNewsFeed($userId);
* $obj->follow($followerId, $followeeId);
* $obj->unfollow($followerId, $followeeId);
*/
```

Dart:

```
class Twitter {

Twitter() {

}

void postTweet(int userId, int tweetId) {

}

List<int> getNewsFeed(int userId) {

}

void follow(int followerId, int followeeId) {

}

void unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = Twitter();
* obj.postTweet(userId,tweetId);
* List<int> param2 = obj.getNewsFeed(userId);
* obj.follow(followerId,followeeId);
* obj.unfollow(followerId,followeeId);
*/
```

Scala:

```
class Twitter() {

def postTweet(userId: Int, tweetId: Int): Unit = {

}

def getNewsFeed(userId: Int): List[Int] = {

}

def follow(followerId: Int, followeeId: Int): Unit = {

}

def unfollow(followerId: Int, followeeId: Int): Unit = {

}

}

/**
* Your Twitter object will be instantiated and called as such:
* val obj = new Twitter()
* obj.postTweet(userId,tweetId)
* val param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

Elixir:

```
defmodule Twitter do
@spec init_() :: any
def init_() do

end

@spec post_tweet(user_id :: integer, tweet_id :: integer) :: any
def post_tweet(user_id, tweet_id) do
```

```
end

@spec get_news_feed(user_id :: integer) :: [integer]
def get_news_feed(user_id) do

end

@spec follow(follower_id :: integer, followee_id :: integer) :: any
def follow(follower_id, followee_id) do

end

@spec unfollow(follower_id :: integer, followee_id :: integer) :: any
def unfollow(follower_id, followee_id) do

end
end

# Your functions will be called as such:
# Twitter.init_()
# Twitter.post_tweet(user_id, tweet_id)
# param_2 = Twitter.get_news_feed(user_id)
# Twitter.follow(follower_id, followee_id)
# Twitter.unfollow(follower_id, followee_id)

# Twitter.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang:

```
-spec twitter_init_() -> any().
twitter_init_() ->
.

-spec twitter_post_tweet(UserId :: integer(), TweetId :: integer()) -> any().
twitter_post_tweet(UserId, TweetId) ->
.

-spec twitter_get_news_feed(UserId :: integer()) -> [integer()].
twitter_get_news_feed(UserId) ->
.
```

```
-spec twitter_follow(FollowerId :: integer(), FolloweeId :: integer()) ->
any().
twitter_follow(FollowerId, FolloweeId) ->
.

-spec twitter_unfollow(FollowerId :: integer(), FolloweeId :: integer()) ->
any().
twitter_unfollow(FollowerId, FolloweeId) ->
.


%% Your functions will be called as such:
%% twitter_init_(),
%% twitter_post_tweet(UserId, TweetId),
%% Param_2 = twitter_get_news_feed(UserId),
%% twitter_follow(FollowerId, FolloweeId),
%% twitter_unfollow(FollowerId, FolloweeId),

%% twitter_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket:

```
(define twitter%
(class object%
(super-new)

(init-field)

; post-tweet : exact-integer? exact-integer? -> void?
(define/public (post-tweet user-id tweet-id)
)
; get-news-feed : exact-integer? -> (listof exact-integer?)
(define/public (get-news-feed user-id)
)
; follow : exact-integer? exact-integer? -> void?
(define/public (follow follower-id followee-id)
)
; unfollow : exact-integer? exact-integer? -> void?
(define/public (unfollow follower-id followee-id)
)))
```

```
;; Your twitter% object will be instantiated and called as such:
;; (define obj (new twitter%))
;; (send obj post-tweet user-id tweet-id)
;; (define param_2 (send obj get-news-feed user-id))
;; (send obj follow follower-id followee-id)
;; (send obj unfollow follower-id followee-id)
```

Solutions

C++ Solution:

```
/*
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Twitter {
public:
Twitter() {

}

void postTweet(int userId, int tweetId) {

}

vector<int> getNewsFeed(int userId) {

}

void follow(int followerId, int followeeId) {

}

void unfollow(int followerId, int followeeId) {
```

```
}
};

/**
* Your Twitter object will be instantiated and called as such:
* Twitter* obj = new Twitter();
* obj->postTweet(userId,tweetId);
* vector<int> param_2 = obj->getNewsFeed(userId);
* obj->follow(followerId,followeeId);
* obj->unfollow(followerId,followeeId);
*/
```

Java Solution:

```
/**
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Twitter {

public Twitter() {

}

public void postTweet(int userId, int tweetId) {

}

public List<Integer> getNewsFeed(int userId) {

}

public void follow(int followerId, int followeeId) {
```

```
}

public void unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = new Twitter();
* obj.postTweet(userId,tweetId);
* List<Integer> param_2 = obj.getNewsFeed(userId);
* obj.follow(followerId,followeeId);
* obj.unfollow(followerId,followeeId);
*/
```

Python3 Solution:

```
"""
Problem: Design Twitter
Difficulty: Medium
Tags: hash, linked_list, queue, heap

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class Twitter:

def __init__(self):


def postTweet(self, userId: int, tweetId: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Twitter(object):
```

```
def __init__(self):


def postTweet(self, userId, tweetId):
"""
:type userId: int
:type tweetId: int
:rtype: None
"""


def getNewsFeed(self, userId):
"""
:type userId: int
:rtype: List[int]
"""


def follow(self, followerId, followeeId):
"""
:type followerId: int
:type followeeId: int
:rtype: None
"""


def unfollow(self, followerId, followeeId):
"""
:type followerId: int
:type followeeId: int
:rtype: None
"""



# Your Twitter object will be instantiated and called as such:
# obj = Twitter()
# obj.postTweet(userId,tweetId)
# param_2 = obj.getNewsFeed(userId)
# obj.follow(followerId,followeeId)
# obj.unfollow(followerId,followeeId)
```

JavaScript Solution:

```
/**
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/


var Twitter = function() {

};

/**
* @param {number} userId
* @param {number} tweetId
* @return {void}
*/
Twitter.prototype.postTweet = function(userId, tweetId) {

};

/**
* @param {number} userId
* @return {number[]}
*/
Twitter.prototype.getNewsFeed = function(userId) {

};

/**
* @param {number} followerId
* @param {number} followeeId
* @return {void}
*/
Twitter.prototype.follow = function(followerId, followeeId) {

};
```

```
/**
* @param {number} followerId
* @param {number} followeeId
* @return {void}
*/
Twitter.prototype.unfollow = function(followerId, followeeId) {

};

/**
* Your Twitter object will be instantiated and called as such:
* var obj = new Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Twitter {
constructor() {

}

postTweet(userId: number, tweetId: number): void {

}

getNewsFeed(userId: number): number[] {
```

```
}

follow(followerId: number, followeeId: number): void {

}

unfollow(followerId: number, followeeId: number): void {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* var obj = new Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

C# Solution:

```
/*
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class Twitter {

public Twitter() {

}

public void PostTweet(int userId, int tweetId) {
```

```
}

public IList<int> GetNewsFeed(int userId) {

}

public void Follow(int followerId, int followeeId) {

}

public void Unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = new Twitter();
* obj.PostTweet(userId,tweetId);
* IList<int> param_2 = obj.GetNewsFeed(userId);
* obj.Follow(followerId,followeeId);
* obj.Unfollow(followerId,followeeId);
*/
```

C Solution:

```
/*
* Problem: Design Twitter
* Difficulty: Medium
* Tags: hash, linked_list, queue, heap
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/




typedef struct {
```

```
} Twitter;


Twitter* twitterCreate() {

}

void twitterPostTweet(Twitter* obj, int userId, int tweetId) {

}

int* twitterGetNewsFeed(Twitter* obj, int userId, int* retSize) {

}

void twitterFollow(Twitter* obj, int followerId, int followeeId) {

}

void twitterUnfollow(Twitter* obj, int followerId, int followeeId) {

}

void twitterFree(Twitter* obj) {

}

/**
* Your Twitter struct will be instantiated and called as such:
* Twitter* obj = twitterCreate();
* twitterPostTweet(obj, userId, tweetId);

* int* param_2 = twitterGetNewsFeed(obj, userId, retSize);

* twitterFollow(obj, followerId, followeeId);

* twitterUnfollow(obj, followerId, followeeId);

* twitterFree(obj);
*/
```

Go Solution:

```
// Problem: Design Twitter
// Difficulty: Medium
// Tags: hash, linked_list, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

type Twitter struct {

}


func Constructor() Twitter {

}


func (this *Twitter) PostTweet(userId int, tweetId int) {

}


func (this *Twitter) GetNewsFeed(userId int) []int {

}


func (this *Twitter) Follow(followerId int, followeeId int) {

}


func (this *Twitter) Unfollow(followerId int, followeeId int) {

}


/**
* Your Twitter object will be instantiated and called as such:
* obj := Constructor();
```

```
* obj.PostTweet(userId,tweetId);
* param_2 := obj.GetNewsFeed(userId);
* obj.Follow(followerId,followeeId);
* obj.Unfollow(followerId,followeeId);
*/
```

Kotlin Solution:

```
class Twitter() {

fun postTweet(userId: Int, tweetId: Int) {

}

fun getNewsFeed(userId: Int): List<Int> {

}

fun follow(followerId: Int, followeeId: Int) {

}

fun unfollow(followerId: Int, followeeId: Int) {

}

}

/**
* Your Twitter object will be instantiated and called as such:
* var obj = Twitter()
* obj.postTweet(userId,tweetId)
* var param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

Swift Solution:

```
class Twitter {
```

```
init() {

}

func postTweet(_ userId: Int, _ tweetId: Int) {

}

func getNewsFeed(_ userId: Int) -> [Int] {

}

func follow(_ followerId: Int, _ followeeId: Int) {

}

func unfollow(_ followerId: Int, _ followeeId: Int) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* let obj = Twitter()
* obj.postTweet(userId, tweetId)
* let ret_2: [Int] = obj.getNewsFeed(userId)
* obj.follow(followerId, followeeId)
* obj.unfollow(followerId, followeeId)
*/
```

Rust Solution:

```
// Problem: Design Twitter
// Difficulty: Medium
// Tags: hash, linked_list, queue, heap
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map
```

```
struct Twitter {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Twitter {

fn new() -> Self {

}

fn post_tweet(&self, user_id: i32, tweet_id: i32) {

}

fn get_news_feed(&self, user_id: i32) -> Vec<i32> {

}

fn follow(&self, follower_id: i32, followee_id: i32) {

}

fn unfollow(&self, follower_id: i32, followee_id: i32) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* let obj = Twitter::new();
* obj.post_tweet(userId, tweetId);
* let ret_2: Vec<i32> = obj.get_news_feed(userId);
* obj.follow(followerId, followeeId);
* obj.unfollow(followerId, followeeId);
*/
```

Ruby Solution:

```
class Twitter
def initialize()

end


=begin
:type user_id: Integer
:type tweet_id: Integer
:rtype: Void
=end
def post_tweet(user_id, tweet_id)

end


=begin
:type user_id: Integer
:rtype: Integer[]
=end
def get_news_feed(user_id)

end


=begin
:type follower_id: Integer
:type followee_id: Integer
:rtype: Void
=end
def follow(follower_id, followee_id)

end


=begin
:type follower_id: Integer
:type followee_id: Integer
:rtype: Void
=end
def unfollow(follower_id, followee_id)
```

```
end


end

# Your Twitter object will be instantiated and called as such:
# obj = Twitter.new()
# obj.post_tweet(user_id, tweet_id)
# param_2 = obj.get_news_feed(user_id)
# obj.follow(follower_id, followee_id)
# obj.unfollow(follower_id, followee_id)
```

PHP Solution:

```
class Twitter {
/**
*/
function __construct() {

}

/**
* @param Integer $userId
* @param Integer $tweetId
* @return NULL
*/
function postTweet($userId, $tweetId) {

}

/**
* @param Integer $userId
* @return Integer[]
*/
function getNewsFeed($userId) {

}

/**
* @param Integer $followerId
```

```
* @param Integer $followeeId
* @return NULL
*/
function follow($followerId, $followeeId) {

}

/**
* @param Integer $followerId
* @param Integer $followeeId
* @return NULL
*/
function unfollow($followerId, $followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* $obj = Twitter();
* $obj->postTweet($userId, $tweetId);
* $ret_2 = $obj->getNewsFeed($userId);
* $obj->follow($followerId, $followeeId);
* $obj->unfollow($followerId, $followeeId);
*/
```

Dart Solution:

```
class Twitter {

Twitter() {

}

void postTweet(int userId, int tweetId) {

}

List<int> getNewsFeed(int userId) {

}
```

```
void follow(int followerId, int followeeId) {

}

void unfollow(int followerId, int followeeId) {

}
}

/**
* Your Twitter object will be instantiated and called as such:
* Twitter obj = Twitter();
* obj.postTweet(userId,tweetId);
* List<int> param2 = obj.getNewsFeed(userId);
* obj.follow(followerId,followeeId);
* obj.unfollow(followerId,followeeId);
*/
```

Scala Solution:

```
class Twitter() {

def postTweet(userId: Int, tweetId: Int): Unit = {

}

def getNewsFeed(userId: Int): List[Int] = {

}

def follow(followerId: Int, followeeId: Int): Unit = {

}

def unfollow(followerId: Int, followeeId: Int): Unit = {

}

}
```

```
/**
* Your Twitter object will be instantiated and called as such:
* val obj = new Twitter()
* obj.postTweet(userId,tweetId)
* val param_2 = obj.getNewsFeed(userId)
* obj.follow(followerId,followeeId)
* obj.unfollow(followerId,followeeId)
*/
```

Elixir Solution:

```
defmodule Twitter do
@spec init_() :: any
def init_() do

end

@spec post_tweet(user_id :: integer, tweet_id :: integer) :: any
def post_tweet(user_id, tweet_id) do

end

@spec get_news_feed(user_id :: integer) :: [integer]
def get_news_feed(user_id) do

end

@spec follow(follower_id :: integer, followee_id :: integer) :: any
def follow(follower_id, followee_id) do

end

@spec unfollow(follower_id :: integer, followee_id :: integer) :: any
def unfollow(follower_id, followee_id) do

end
end

# Your functions will be called as such:
# Twitter.init_()
# Twitter.post_tweet(user_id, tweet_id)
```

```
# param_2 = Twitter.get_news_feed(user_id)
# Twitter.follow(follower_id, followee_id)
# Twitter.unfollow(follower_id, followee_id)

# Twitter.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec twitter_init_() -> any().
twitter_init_() ->
.

-spec twitter_post_tweet(UserId :: integer(), TweetId :: integer()) -> any().
twitter_post_tweet(UserId, TweetId) ->
.

-spec twitter_get_news_feed(UserId :: integer()) -> [integer()].
twitter_get_news_feed(UserId) ->
.

-spec twitter_follow(FollowerId :: integer(), FolloweeId :: integer()) ->
any().
twitter_follow(FollowerId, FolloweeId) ->
.

-spec twitter_unfollow(FollowerId :: integer(), FolloweeId :: integer()) ->
any().
twitter_unfollow(FollowerId, FolloweeId) ->
.


%% Your functions will be called as such:
%% twitter_init_(),
%% twitter_post_tweet(UserId, TweetId),
%% Param_2 = twitter_get_news_feed(UserId),
%% twitter_follow(FollowerId, FolloweeId),
%% twitter_unfollow(FollowerId, FolloweeId),

%% twitter_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket Solution:

```
(define twitter%
(class object%
(super-new)

(init-field)

; post-tweet : exact-integer? exact-integer? -> void?
(define/public (post-tweet user-id tweet-id)
)
; get-news-feed : exact-integer? -> (listof exact-integer?)
(define/public (get-news-feed user-id)
)
; follow : exact-integer? exact-integer? -> void?
(define/public (follow follower-id followee-id)
)
; unfollow : exact-integer? exact-integer? -> void?
(define/public (unfollow follower-id followee-id)
)))

;; Your twitter% object will be instantiated and called as such:
;; (define obj (new twitter%))
;; (send obj post-tweet user-id tweet-id)
;; (define param_2 (send obj get-news-feed user-id))
;; (send obj follow follower-id followee-id)
;; (send obj unfollow follower-id followee-id)
```