Problem 1348: Tweet Counts Per Frequency

Problem Information

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

Problem Description

A social media company is trying to monitor activity on their site by analyzing the number of tweets that occur in select periods of time. These periods can be partitioned into smaller

time chunks

based on a certain frequency (every

minute

,

hour

, or

day

).

For example, the period

[10, 10000]

(in

seconds

) would be partitioned into the following

time chunks

with these frequencies:

Every

minute

(60-second chunks):

[10,69]

,

[70,129]

,

[130,189]

,

...

,

[9970,10000]

Every

hour

(3600-second chunks):

[10,3609]

,

[3610,7209]

,

[7210,10000]

Every

day

(86400-second chunks):

[10,10000]

Notice that the last chunk may be shorter than the specified frequency's chunk size and will always end with the end time of the period (

10000

in the above example).

Design and implement an API to help the company with their analysis.

Implement the

TweetCounts

class:

TweetCounts()

Initializes the

TweetCounts

object.

void recordTweet(String tweetName, int time)

Stores the

tweetName

at the recorded

time

(in

seconds

).

List<Integer> getTweetCountsPerFrequency(String freq, String tweetName, int startTime, int endTime)

Returns a list of integers representing the number of tweets with

tweetName

in each

time chunk

for the given period of time

[startTime, endTime]

(in

seconds

) and frequency

freq

.

freq

is one of

"minute"

,

"hour"

, or

"day"

representing a frequency of every

minute

,

hour

, or

day

respectively.

Example:

Input

["TweetCounts","recordTweet","recordTweet","recordTweet","getTweetCountsPerFrequency","getTweetCountsPerFrequency","recordTweet","getTweetCountsPerFrequency"] [[],["tweet3",0],["tweet3",60],["tweet3",10],["minute","tweet3",0,59],["minute","tweet3",0,60],["tweet3",120],["hour","tweet3",0,210]]

Output

[null,null,null,null,[2],[2,1],null,[4]]

Explanation

TweetCounts tweetCounts = new TweetCounts(); tweetCounts.recordTweet("tweet3", 0); // New tweet "tweet3" at time 0 tweetCounts.recordTweet("tweet3", 60); // New tweet "tweet3" at time 60 tweetCounts.recordTweet("tweet3", 10); // New tweet "tweet3" at time 10 tweetCounts.getTweetCountsPerFrequency("minute", "tweet3", 0, 59); // return [2]; chunk [0,59] had 2 tweets tweetCounts.getTweetCountsPerFrequency("minute", "tweet3", 0, 60); // return [2,1]; chunk [0,59] had 2 tweets, chunk [60,60] had 1 tweet tweetCounts.recordTweet("tweet3", 120); // New tweet "tweet3" at time 120 tweetCounts.getTweetCountsPerFrequency("hour", "tweet3", 0, 210); // return [4]; chunk [0,210] had 4 tweets

Constraints:

0 <= time, startTime, endTime <= 10

9

0 <= endTime - startTime <= 10

4

There will be at most

10

4

calls

in total

to

recordTweet

and

getTweetCountsPerFrequency

.

Code Snippets

C++:

```
class TweetCounts {
public:
TweetCounts() {

}

void recordTweet(string tweetName, int time) {

}

vector<int> getTweetCountsPerFrequency(string freq, string tweetName, int
startTime, int endTime) {

}
};

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts* obj = new TweetCounts();
* obj->recordTweet(tweetName,time);
* vector<int> param_2 =
obj->getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Java:

```
class TweetCounts {

public TweetCounts() {
```

```
}

public void recordTweet(String tweetName, int time) {

}

public List<Integer> getTweetCountsPerFrequency(String freq, String
tweetName, int startTime, int endTime) {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts obj = new TweetCounts();
* obj.recordTweet(tweetName,time);
* List<Integer> param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Python3:

```
class TweetCounts:

def __init__(self):


def recordTweet(self, tweetName: str, time: int) -> None:


def getTweetCountsPerFrequency(self, freq: str, tweetName: str, startTime:
int, endTime: int) -> List[int]:




# Your TweetCounts object will be instantiated and called as such:
# obj = TweetCounts()
# obj.recordTweet(tweetName,time)
# param_2 = obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
```

Python:

```
class TweetCounts(object):

def __init__(self):


def recordTweet(self, tweetName, time):
"""
:type tweetName: str
:type time: int
:rtype: None
"""


def getTweetCountsPerFrequency(self, freq, tweetName, startTime, endTime):
"""
:type freq: str
:type tweetName: str
:type startTime: int
:type endTime: int
:rtype: List[int]
"""



# Your TweetCounts object will be instantiated and called as such:
# obj = TweetCounts()
# obj.recordTweet(tweetName,time)
# param_2 = obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
```

JavaScript:

```
var TweetCounts = function() {

};

/**
* @param {string} tweetName
* @param {number} time
* @return {void}
*/
TweetCounts.prototype.recordTweet = function(tweetName, time) {
```

```
};

/**
* @param {string} freq
* @param {string} tweetName
* @param {number} startTime
* @param {number} endTime
* @return {number[]}
*/
TweetCounts.prototype.getTweetCountsPerFrequency = function(freq, tweetName,
startTime, endTime) {

};

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = new TweetCounts()
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

TypeScript:

```
class TweetCounts {
constructor() {

}

recordTweet(tweetName: string, time: number): void {

}

getTweetCountsPerFrequency(freq: string, tweetName: string, startTime:
number, endTime: number): number[] {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = new TweetCounts()
```

```
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

C#:

```
public class TweetCounts {

public TweetCounts() {

}

public void RecordTweet(string tweetName, int time) {

}

public IList<int> GetTweetCountsPerFrequency(string freq, string tweetName,
int startTime, int endTime) {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts obj = new TweetCounts();
* obj.RecordTweet(tweetName,time);
* IList<int> param_2 =
obj.GetTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

C:

```
typedef struct {

} TweetCounts;


TweetCounts* tweetCountsCreate() {
```

```
}

void tweetCountsRecordTweet(TweetCounts* obj, char* tweetName, int time) {

}

int* tweetCountsGetTweetCountsPerFrequency(TweetCounts* obj, char* freq,
char* tweetName, int startTime, int endTime, int* retSize) {

}

void tweetCountsFree(TweetCounts* obj) {

}

/**
* Your TweetCounts struct will be instantiated and called as such:
* TweetCounts* obj = tweetCountsCreate();
* tweetCountsRecordTweet(obj, tweetName, time);

* int* param_2 = tweetCountsGetTweetCountsPerFrequency(obj, freq, tweetName,
startTime, endTime, retSize);

* tweetCountsFree(obj);
*/
```

Go:

```
type TweetCounts struct {

}


func Constructor() TweetCounts {

}


func (this *TweetCounts) RecordTweet(tweetName string, time int) {

}
```

```
func (this *TweetCounts) GetTweetCountsPerFrequency(freq string, tweetName
string, startTime int, endTime int) []int {

}


/**
* Your TweetCounts object will be instantiated and called as such:
* obj := Constructor();
* obj.RecordTweet(tweetName,time);
* param_2 :=
obj.GetTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Kotlin:

```
class TweetCounts() {

fun recordTweet(tweetName: String, time: Int) {

}

fun getTweetCountsPerFrequency(freq: String, tweetName: String, startTime:
Int, endTime: Int): List<Int> {

}

}

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = TweetCounts()
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

Swift:

```
class TweetCounts {

init() {

}

func recordTweet(_ tweetName: String, _ time: Int) {

}

func getTweetCountsPerFrequency(_ freq: String, _ tweetName: String, _
startTime: Int, _ endTime: Int) -> [Int] {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* let obj = TweetCounts()
* obj.recordTweet(tweetName, time)
* let ret_2: [Int] = obj.getTweetCountsPerFrequency(freq, tweetName,
startTime, endTime)
*/
```

Rust:

```
struct TweetCounts {

}


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

fn new() -> Self {

}

fn record_tweet(&self, tweet_name: String, time: i32) {
```

```
}

fn get_tweet_counts_per_frequency(&self, freq: String, tweet_name: String,
start_time: i32, end_time: i32) -> Vec<i32> {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* let obj = TweetCounts::new();
* obj.record_tweet(tweetName, time);
* let ret_2: Vec<i32> = obj.get_tweet_counts_per_frequency(freq, tweetName,
startTime, endTime);
*/
```

Ruby:

```
class TweetCounts
def initialize()

end


=begin
:type tweet_name: String
:type time: Integer
:rtype: Void
=end
def record_tweet(tweet_name, time)

end


=begin
:type freq: String
:type tweet_name: String
:type start_time: Integer
:type end_time: Integer
:rtype: Integer[]
=end
```

```
def get_tweet_counts_per_frequency(freq, tweet_name, start_time, end_time)

end


end

# Your TweetCounts object will be instantiated and called as such:
# obj = TweetCounts.new()
# obj.record_tweet(tweet_name, time)
# param_2 = obj.get_tweet_counts_per_frequency(freq, tweet_name, start_time, end_time)
```

PHP:

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

}

/**
* @param String $tweetName
* @param Integer $time
* @return NULL
*/
function recordTweet($tweetName, $time) {

}

/**
* @param String $freq
* @param String $tweetName
* @param Integer $startTime
* @param Integer $endTime
* @return Integer[]
*/
function getTweetCountsPerFrequency($freq, $tweetName, $startTime, $endTime)
{

}
```

```
}

/**
* Your TweetCounts object will be instantiated and called as such:
* $obj = TweetCounts();
* $obj->recordTweet($tweetName, $time);
* $ret_2 = $obj->getTweetCountsPerFrequency($freq, $tweetName, $startTime,
$endTime);
*/
```

Dart:

```
class TweetCounts {

TweetCounts() {

}

void recordTweet(String tweetName, int time) {

}

List<int> getTweetCountsPerFrequency(String freq, String tweetName, int
startTime, int endTime) {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts obj = TweetCounts();
* obj.recordTweet(tweetName,time);
* List<int> param2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Scala:

```
class TweetCounts() {

def recordTweet(tweetName: String, time: Int): Unit = {
```

```
}

def getTweetCountsPerFrequency(freq: String, tweetName: String, startTime:
Int, endTime: Int): List[Int] = {

}

}

/**
* Your TweetCounts object will be instantiated and called as such:
* val obj = new TweetCounts()
* obj.recordTweet(tweetName,time)
* val param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

Elixir:

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

end

@spec record_tweet(tweet_name :: String.t, time :: integer) :: any
def record_tweet(tweet_name, time) do

end

@spec get_tweet_counts_per_frequency(freq :: String.t, tweet_name ::
String.t, start_time :: integer, end_time :: integer) :: [integer]
def get_tweet_counts_per_frequency(freq, tweet_name, start_time, end_time) do

end
end

# Your functions will be called as such:
# TweetCounts.init_()
# TweetCounts.record_tweet(tweet_name, time)
# param_2 = TweetCounts.get_tweet_counts_per_frequency(freq, tweet_name,
start_time, end_time)
```

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

Erlang:

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

-spec tweet_counts_record_tweet(TweetName :: unicode:unicode_binary(), Time
:: integer()) -> any().
tweet_counts_record_tweet(TweetName, Time) ->
.

-spec tweet_counts_get_tweet_counts_per_frequency(Freq ::
unicode:unicode_binary(), TweetName :: unicode:unicode_binary(), StartTime ::
integer(), EndTime :: integer()) -> [integer()].
tweet_counts_get_tweet_counts_per_frequency(Freq, TweetName, StartTime,
EndTime) ->
.


%% Your functions will be called as such:
%% tweet_counts_init_(),
%% tweet_counts_record_tweet(TweetName, Time),
%% Param_2 = tweet_counts_get_tweet_counts_per_frequency(Freq, TweetName,
StartTime, EndTime),

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

Racket:

```
(define tweet-counts%
(class object%
(super-new)

(init-field)

; record-tweet : string? exact-integer? -> void?
(define/public (record-tweet tweet-name time)
```

```
)
; get-tweet-counts-per-frequency : string? string? exact-integer?
exact-integer? -> (listof exact-integer?)
(define/public (get-tweet-counts-per-frequency freq tweet-name start-time
end-time)
)))

;; Your tweet-counts% object will be instantiated and called as such:
;; (define obj (new tweet-counts%))
;; (send obj record-tweet tweet-name time)
;; (define param_2 (send obj get-tweet-counts-per-frequency freq tweet-name
start-time end-time))
```

Solutions

C++ Solution:

```
/*
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TweetCounts {
public:
TweetCounts() {

}

void recordTweet(string tweetName, int time) {

}

vector<int> getTweetCountsPerFrequency(string freq, string tweetName, int
startTime, int endTime) {
```

```
}
};

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts* obj = new TweetCounts();
* obj->recordTweet(tweetName,time);
* vector<int> param_2 =
obj->getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Java Solution:

```
/**
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TweetCounts {

public TweetCounts() {

}

public void recordTweet(String tweetName, int time) {

}

public List<Integer> getTweetCountsPerFrequency(String freq, String
tweetName, int startTime, int endTime) {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
```

```
* TweetCounts obj = new TweetCounts();
* obj.recordTweet(tweetName,time);
* List<Integer> param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Python3 Solution:

```
"""
Problem: Tweet Counts Per Frequency
Difficulty: Medium
Tags: string, hash, sort, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class TweetCounts:

def __init__(self):


def recordTweet(self, tweetName: str, time: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class TweetCounts(object):

def __init__(self):


def recordTweet(self, tweetName, time):
"""
:type tweetName: str
:type time: int
:rtype: None
"""
```

```
def getTweetCountsPerFrequency(self, freq, tweetName, startTime, endTime):
"""
:type freq: str
:type tweetName: str
:type startTime: int
:type endTime: int
:rtype: List[int]
"""



# Your TweetCounts object will be instantiated and called as such:
# obj = TweetCounts()
# obj.recordTweet(tweetName,time)
# param_2 = obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
```

JavaScript Solution:

```
/**
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/


var TweetCounts = function() {

};

/**
* @param {string} tweetName
* @param {number} time
* @return {void}
*/
TweetCounts.prototype.recordTweet = function(tweetName, time) {
```

```
};

/**
* @param {string} freq
* @param {string} tweetName
* @param {number} startTime
* @param {number} endTime
* @return {number[]}
*/
TweetCounts.prototype.getTweetCountsPerFrequency = function(freq, tweetName,
startTime, endTime) {

};

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = new TweetCounts()
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

TypeScript Solution:

```
/**
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TweetCounts {
constructor() {

}

recordTweet(tweetName: string, time: number): void {
```

```
}

getTweetCountsPerFrequency(freq: string, tweetName: string, startTime:
number, endTime: number): number[] {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = new TweetCounts()
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

C# Solution:

```
/*
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class TweetCounts {

public TweetCounts() {

}

public void RecordTweet(string tweetName, int time) {

}

public IList<int> GetTweetCountsPerFrequency(string freq, string tweetName,
int startTime, int endTime) {
```

```
}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* TweetCounts obj = new TweetCounts();
* obj.RecordTweet(tweetName,time);
* IList<int> param_2 =
obj.GetTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

C Solution:

```
/*
* Problem: Tweet Counts Per Frequency
* Difficulty: Medium
* Tags: string, hash, sort, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} TweetCounts;


TweetCounts* tweetCountsCreate() {

}

void tweetCountsRecordTweet(TweetCounts* obj, char* tweetName, int time) {

}

int* tweetCountsGetTweetCountsPerFrequency(TweetCounts* obj, char* freq,
char* tweetName, int startTime, int endTime, int* retSize) {
```

```
}

void tweetCountsFree(TweetCounts* obj) {

}

/**
* Your TweetCounts struct will be instantiated and called as such:
* TweetCounts* obj = tweetCountsCreate();
* tweetCountsRecordTweet(obj, tweetName, time);

* int* param_2 = tweetCountsGetTweetCountsPerFrequency(obj, freq, tweetName,
startTime, endTime, retSize);

* tweetCountsFree(obj);
*/
```

Go Solution:

```
// Problem: Tweet Counts Per Frequency
// Difficulty: Medium
// Tags: string, hash, sort, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type TweetCounts struct {

}


func Constructor() TweetCounts {

}


func (this *TweetCounts) RecordTweet(tweetName string, time int) {

}
```

```
func (this *TweetCounts) GetTweetCountsPerFrequency(freq string, tweetName
string, startTime int, endTime int) []int {

}


/**
* Your TweetCounts object will be instantiated and called as such:
* obj := Constructor();
* obj.RecordTweet(tweetName,time);
* param_2 :=
obj.GetTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Kotlin Solution:

```
class TweetCounts() {

fun recordTweet(tweetName: String, time: Int) {

}

fun getTweetCountsPerFrequency(freq: String, tweetName: String, startTime:
Int, endTime: Int): List<Int> {

}

}

/**
* Your TweetCounts object will be instantiated and called as such:
* var obj = TweetCounts()
* obj.recordTweet(tweetName,time)
* var param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

Swift Solution:

```
class TweetCounts {

init() {

}

func recordTweet(_ tweetName: String, _ time: Int) {

}

func getTweetCountsPerFrequency(_ freq: String, _ tweetName: String, _
startTime: Int, _ endTime: Int) -> [Int] {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* let obj = TweetCounts()
* obj.recordTweet(tweetName, time)
* let ret_2: [Int] = obj.getTweetCountsPerFrequency(freq, tweetName,
startTime, endTime)
*/
```

Rust Solution:

```
// Problem: Tweet Counts Per Frequency
// Difficulty: Medium
// Tags: string, hash, sort, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct TweetCounts {

}



/**
* `&self` means the method takes an immutable reference.
```

```
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl TweetCounts {

fn new() -> Self {

}

fn record_tweet(&self, tweet_name: String, time: i32) {

}

fn get_tweet_counts_per_frequency(&self, freq: String, tweet_name: String,
start_time: i32, end_time: i32) -> Vec<i32> {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* let obj = TweetCounts::new();
* obj.record_tweet(tweetName, time);
* let ret_2: Vec<i32> = obj.get_tweet_counts_per_frequency(freq, tweetName,
startTime, endTime);
*/
```

Ruby Solution:

```
class TweetCounts
def initialize()

end


=begin
:type tweet_name: String
:type time: Integer
:rtype: Void
=end
def record_tweet(tweet_name, time)
```

```
end


=begin
:type freq: String
:type tweet_name: String
:type start_time: Integer
:type end_time: Integer
:rtype: Integer[]
=end
def get_tweet_counts_per_frequency(freq, tweet_name, start_time, end_time)

end



end

# Your TweetCounts object will be instantiated and called as such:
# obj = TweetCounts.new()
# obj.record_tweet(tweet_name, time)
# param_2 = obj.get_tweet_counts_per_frequency(freq, tweet_name, start_time,
end_time)
```

PHP Solution:

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

}

/**
* @param String $tweetName
* @param Integer $time
* @return NULL
*/
function recordTweet($tweetName, $time) {

}
```

```
/**
* @param String $freq
* @param String $tweetName
* @param Integer $startTime
* @param Integer $endTime
* @return Integer[]
*/
function getTweetCountsPerFrequency($freq, $tweetName, $startTime, $endTime)
{

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
* $obj = TweetCounts();
* $obj->recordTweet($tweetName, $time);
* $ret_2 = $obj->getTweetCountsPerFrequency($freq, $tweetName, $startTime,
$endTime);
*/
```

Dart Solution:

```
class TweetCounts {

TweetCounts() {

}

void recordTweet(String tweetName, int time) {

}

List<int> getTweetCountsPerFrequency(String freq, String tweetName, int
startTime, int endTime) {

}
}

/**
* Your TweetCounts object will be instantiated and called as such:
```

```
* TweetCounts obj = TweetCounts();
* obj.recordTweet(tweetName,time);
* List<int> param2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime);
*/
```

Scala Solution:

```
class TweetCounts() {

def recordTweet(tweetName: String, time: Int): Unit = {

}

def getTweetCountsPerFrequency(freq: String, tweetName: String, startTime:
Int, endTime: Int): List[Int] = {

}

}

/**
* Your TweetCounts object will be instantiated and called as such:
* val obj = new TweetCounts()
* obj.recordTweet(tweetName,time)
* val param_2 =
obj.getTweetCountsPerFrequency(freq,tweetName,startTime,endTime)
*/
```

Elixir Solution:

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

end

@spec record_tweet(tweet_name :: String.t, time :: integer) :: any
def record_tweet(tweet_name, time) do

end
```

```
@spec get_tweet_counts_per_frequency(freq :: String.t, tweet_name ::
String.t, start_time :: integer, end_time :: integer) :: [integer]
def get_tweet_counts_per_frequency(freq, tweet_name, start_time, end_time) do

end
end

# Your functions will be called as such:
# TweetCounts.init_()
# TweetCounts.record_tweet(tweet_name, time)
# param_2 = TweetCounts.get_tweet_counts_per_frequency(freq, tweet_name,
start_time, end_time)

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

Erlang Solution:

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

-spec tweet_counts_record_tweet(TweetName :: unicode:unicode_binary(), Time
:: integer()) -> any().
tweet_counts_record_tweet(TweetName, Time) ->
.

-spec tweet_counts_get_tweet_counts_per_frequency(Freq ::
unicode:unicode_binary(), TweetName :: unicode:unicode_binary(), StartTime ::
integer(), EndTime :: integer()) -> [integer()].
tweet_counts_get_tweet_counts_per_frequency(Freq, TweetName, StartTime,
EndTime) ->
.


%% Your functions will be called as such:
%% tweet_counts_init_(),
%% tweet_counts_record_tweet(TweetName, Time),
%% Param_2 = tweet_counts_get_tweet_counts_per_frequency(Freq, TweetName,
StartTime, EndTime),
```

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

Racket Solution:

```
(define tweet-counts%
(class object%
(super-new)

(init-field)

; record-tweet : string? exact-integer? -> void?
(define/public (record-tweet tweet-name time)
)
; get-tweet-counts-per-frequency : string? string? exact-integer? exact-integer? -> (listof exact-integer?)
(define/public (get-tweet-counts-per-frequency freq tweet-name start-time end-time)
)))

;; Your tweet-counts% object will be instantiated and called as such:
;; (define obj (new tweet-counts%))
;; (send obj record-tweet tweet-name time)
;; (define param_2 (send obj get-tweet-counts-per-frequency freq tweet-name start-time end-time))
```