Problem 677: Map Sum Pairs

Problem Information

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

Problem Description

Design a map that allows you to do the following:

Maps a string key to a given value.

Returns the sum of the values that have a key with a prefix equal to a given string.

Implement the

MapSum

class:

MapSum()

Initializes the

MapSum

object.

void insert(String key, int val)

Inserts the

key-val

pair into the map. If the

key

already existed, the original

key-value

pair will be overridden to the new one.

int sum(string prefix)

Returns the sum of all the pairs' value whose

key

starts with the

prefix

.

Example 1:

Input

["MapSum", "insert", "sum", "insert", "sum"] [[], ["apple", 3], ["ap"], ["app", 2], ["ap"]]

Output

[null, null, 3, null, 5]

Explanation

MapSum mapSum = new MapSum(); mapSum.insert("apple", 3); mapSum.sum("ap"); // return 3 (

ap

ple = 3) mapSum.insert("app", 2); mapSum.sum("ap"); // return 5 (

ap

ple +

ap

p = 3 + 2 = 5)

Constraints:

1 <= key.length, prefix.length <= 50

key

and

prefix

consist of only lowercase English letters.

1 <= val <= 1000

At most

50

calls will be made to

insert

and

sum

.

Code Snippets

C++:

```
class MapSum {
public:
MapSum() {

}

void insert(string key, int val) {

}

int sum(string prefix) {

}
};

/**
* Your MapSum object will be instantiated and called as such:
* MapSum* obj = new MapSum();
* obj->insert(key,val);
* int param_2 = obj->sum(prefix);
*/
```

Java:

```
class MapSum {

public MapSum() {

}

public void insert(String key, int val) {

}

public int sum(String prefix) {

}
}
```

```
/**
* Your MapSum object will be instantiated and called as such:
* MapSum obj = new MapSum();
* obj.insert(key,val);
* int param_2 = obj.sum(prefix);
*/
```

Python3:

```
class MapSum:

def __init__(self):


def insert(self, key: str, val: int) -> None:


def sum(self, prefix: str) -> int:



# Your MapSum object will be instantiated and called as such:
# obj = MapSum()
# obj.insert(key,val)
# param_2 = obj.sum(prefix)
```

Python:

```
class MapSum(object):

def __init__(self):


def insert(self, key, val):
"""
:type key: str
:type val: int
:rtype: None
"""


def sum(self, prefix):
```

```
"""
:type prefix: str
:rtype: int
"""



# Your MapSum object will be instantiated and called as such:
# obj = MapSum()
# obj.insert(key,val)
# param_2 = obj.sum(prefix)
```

JavaScript:

```
var MapSum = function() {

};

/**
* @param {string} key
* @param {number} val
* @return {void}
*/
MapSum.prototype.insert = function(key, val) {

};

/**
* @param {string} prefix
* @return {number}
*/
MapSum.prototype.sum = function(prefix) {

};

/**
* Your MapSum object will be instantiated and called as such:
* var obj = new MapSum()
* obj.insert(key,val)
* var param_2 = obj.sum(prefix)
*/
```

TypeScript:

```
class MapSum {
constructor() {

}

insert(key: string, val: number): void {

}

sum(prefix: string): number {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* var obj = new MapSum()
* obj.insert(key,val)
* var param_2 = obj.sum(prefix)
*/
```

C#:

```
public class MapSum {

public MapSum() {

}

public void Insert(string key, int val) {

}

public int Sum(string prefix) {

}
}

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

```
* MapSum obj = new MapSum();
* obj.Insert(key,val);
* int param_2 = obj.Sum(prefix);
*/
```

C:

```
typedef struct {

} MapSum;


MapSum* mapSumCreate() {

}

void mapSumInsert(MapSum* obj, char* key, int val) {

}

int mapSumSum(MapSum* obj, char* prefix) {

}

void mapSumFree(MapSum* obj) {

}

/**
* Your MapSum struct will be instantiated and called as such:
* MapSum* obj = mapSumCreate();
* mapSumInsert(obj, key, val);

* int param_2 = mapSumSum(obj, prefix);

* mapSumFree(obj);
*/
```

Go:

```
type MapSum struct {

}


func Constructor() MapSum {

}


func (this *MapSum) Insert(key string, val int) {

}


func (this *MapSum) Sum(prefix string) int {

}


/**
* Your MapSum object will be instantiated and called as such:
* obj := Constructor();
* obj.Insert(key,val);
* param_2 := obj.Sum(prefix);
*/
```

Kotlin:

```
class MapSum() {

fun insert(key: String, `val`: Int) {

}

fun sum(prefix: String): Int {

}

}
```

```
/**
* Your MapSum object will be instantiated and called as such:
* var obj = MapSum()
* obj.insert(key,`val`)
* var param_2 = obj.sum(prefix)
*/
```

Swift:

```
class MapSum {

init() {

}

func insert(_ key: String, _ val: Int) {

}

func sum(_ prefix: String) -> Int {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* let obj = MapSum()
* obj.insert(key, val)
* let ret_2: Int = obj.sum(prefix)
*/
```

Rust:

```
struct MapSum {

}


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

```
*/
impl MapSum {

fn new() -> Self {

}

fn insert(&self, key: String, val: i32) {

}

fn sum(&self, prefix: String) -> i32 {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* let obj = MapSum::new();
* obj.insert(key, val);
* let ret_2: i32 = obj.sum(prefix);
*/
```

Ruby:

```
class MapSum
def initialize()

end


=begin
:type key: String
:type val: Integer
:rtype: Void
=end
def insert(key, val)

end


=begin
```

```
:type prefix: String
:rtype: Integer
=end
def sum(prefix)

end



end

# Your MapSum object will be instantiated and called as such:
# obj = MapSum.new()
# obj.insert(key, val)
# param_2 = obj.sum(prefix)
```

PHP:

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

}

/**
* @param String $key
* @param Integer $val
* @return NULL
*/
function insert($key, $val) {

}

/**
* @param String $prefix
* @return Integer
*/
function sum($prefix) {

}
}
```

```
/**
* Your MapSum object will be instantiated and called as such:
* $obj = MapSum();
* $obj->insert($key, $val);
* $ret_2 = $obj->sum($prefix);
*/
```

Dart:

```
class MapSum {

MapSum() {

}

void insert(String key, int val) {

}

int sum(String prefix) {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* MapSum obj = MapSum();
* obj.insert(key,val);
* int param2 = obj.sum(prefix);
*/
```

Scala:

```
class MapSum() {

def insert(key: String, `val`: Int): Unit = {

}

def sum(prefix: String): Int = {

}
```

```
}

/**
* Your MapSum object will be instantiated and called as such:
* val obj = new MapSum()
* obj.insert(key,`val`)
* val param_2 = obj.sum(prefix)
*/
```

Elixir:

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

end

@spec insert(key :: String.t, val :: integer) :: any
def insert(key, val) do

end

@spec sum(prefix :: String.t) :: integer
def sum(prefix) do

end
end

# Your functions will be called as such:
# MapSum.init_()
# MapSum.insert(key, val)
# param_2 = MapSum.sum(prefix)

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

Erlang:

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

```
-spec map_sum_insert(Key :: unicode:unicode_binary(), Val :: integer()) ->
any().
map_sum_insert(Key, Val) ->
.

-spec map_sum_sum(Prefix :: unicode:unicode_binary()) -> integer().
map_sum_sum(Prefix) ->
.


%% Your functions will be called as such:
%% map_sum_init_(),
%% map_sum_insert(Key, Val),
%% Param_2 = map_sum_sum(Prefix),

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

Racket:

```
(define map-sum%
(class object%
(super-new)

(init-field)

; insert : string? exact-integer? -> void?
(define/public (insert key val)
)
; sum : string? -> exact-integer?
(define/public (sum prefix)
)))

;; Your map-sum% object will be instantiated and called as such:
;; (define obj (new map-sum%))
;; (send obj insert key val)
;; (define param_2 (send obj sum prefix))
```

Solutions

C++ Solution:

```
/*
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 MapSum {
public:
MapSum() {

}

void insert(string key, int val) {

}

int sum(string prefix) {

}
};

/**
* Your MapSum object will be instantiated and called as such:
* MapSum* obj = new MapSum();
* obj->insert(key,val);
* int param_2 = obj->sum(prefix);
*/
```

Java Solution:

```
/**
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 MapSum {

public MapSum() {

}

public void insert(String key, int val) {

}

public int sum(String prefix) {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* MapSum obj = new MapSum();
* obj.insert(key,val);
* int param_2 = obj.sum(prefix);
*/
```

Python3 Solution:

```
"""
Problem: Map Sum Pairs
Difficulty: Medium
Tags: string, hash

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 MapSum:

def __init__(self):
```

```
def insert(self, key: str, val: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MapSum(object):

def __init__(self):


def insert(self, key, val):
"""
:type key: str
:type val: int
:rtype: None
"""


def sum(self, prefix):
"""
:type prefix: str
:rtype: int
"""




# Your MapSum object will be instantiated and called as such:
# obj = MapSum()
# obj.insert(key,val)
# param_2 = obj.sum(prefix)
```

JavaScript Solution:

```
/**
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 MapSum = function() {

};

/**
* @param {string} key
* @param {number} val
* @return {void}
*/
MapSum.prototype.insert = function(key, val) {

};

/**
* @param {string} prefix
* @return {number}
*/
MapSum.prototype.sum = function(prefix) {

};

/**
* Your MapSum object will be instantiated and called as such:
* var obj = new MapSum()
* obj.insert(key,val)
* var param_2 = obj.sum(prefix)
*/
```

TypeScript Solution:

```
/**
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 MapSum {
constructor() {

}

insert(key: string, val: number): void {

}

sum(prefix: string): number {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* var obj = new MapSum()
* obj.insert(key,val)
* var param_2 = obj.sum(prefix)
*/
```

C# Solution:

```
/*
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 MapSum {

public MapSum() {
```

```
}

public void Insert(string key, int val) {

}

public int Sum(string prefix) {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* MapSum obj = new MapSum();
* obj.Insert(key,val);
* int param_2 = obj.Sum(prefix);
*/
```

C Solution:

```
/*
* Problem: Map Sum Pairs
* Difficulty: Medium
* Tags: string, hash
*
* 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 {

} MapSum;


MapSum* mapSumCreate() {

}
```

```
void mapSumInsert(MapSum* obj, char* key, int val) {

}

int mapSumSum(MapSum* obj, char* prefix) {

}

void mapSumFree(MapSum* obj) {

}

/**
* Your MapSum struct will be instantiated and called as such:
* MapSum* obj = mapSumCreate();
* mapSumInsert(obj, key, val);

* int param_2 = mapSumSum(obj, prefix);

* mapSumFree(obj);
*/
```

Go Solution:

```
// Problem: Map Sum Pairs
// Difficulty: Medium
// Tags: string, hash
//
// 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 MapSum struct {

}


func Constructor() MapSum {

}
```

```
func (this *MapSum) Insert(key string, val int) {

}


func (this *MapSum) Sum(prefix string) int {

}


/**
* Your MapSum object will be instantiated and called as such:
* obj := Constructor();
* obj.Insert(key,val);
* param_2 := obj.Sum(prefix);
*/
```

Kotlin Solution:

```
class MapSum() {

fun insert(key: String, `val`: Int) {

}

fun sum(prefix: String): Int {

}

}

/**
* Your MapSum object will be instantiated and called as such:
* var obj = MapSum()
* obj.insert(key,`val`)
* var param_2 = obj.sum(prefix)
*/
```

Swift Solution:

```
class MapSum {

init() {

}

func insert(_ key: String, _ val: Int) {

}

func sum(_ prefix: String) -> Int {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* let obj = MapSum()
* obj.insert(key, val)
* let ret_2: Int = obj.sum(prefix)
*/
```

Rust Solution:

```
// Problem: Map Sum Pairs
// Difficulty: Medium
// Tags: string, hash
//
// 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 MapSum {

}


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

```
impl MapSum {

fn new() -> Self {

}

fn insert(&self, key: String, val: i32) {

}

fn sum(&self, prefix: String) -> i32 {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* let obj = MapSum::new();
* obj.insert(key, val);
* let ret_2: i32 = obj.sum(prefix);
*/
```

Ruby Solution:

```
class MapSum
def initialize()

end


=begin
:type key: String
:type val: Integer
:rtype: Void
=end
def insert(key, val)

end


=begin
```

```
:type prefix: String
:rtype: Integer
=end
def sum(prefix)

end



end

# Your MapSum object will be instantiated and called as such:
# obj = MapSum.new()
# obj.insert(key, val)
# param_2 = obj.sum(prefix)
```

PHP Solution:

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

}

/**
* @param String $key
* @param Integer $val
* @return NULL
*/
function insert($key, $val) {

}

/**
* @param String $prefix
* @return Integer
*/
function sum($prefix) {

}
}
```

```
/**
* Your MapSum object will be instantiated and called as such:
* $obj = MapSum();
* $obj->insert($key, $val);
* $ret_2 = $obj->sum($prefix);
*/
```

Dart Solution:

```
class MapSum {

MapSum() {

}

void insert(String key, int val) {

}

int sum(String prefix) {

}
}

/**
* Your MapSum object will be instantiated and called as such:
* MapSum obj = MapSum();
* obj.insert(key,val);
* int param2 = obj.sum(prefix);
*/
```

Scala Solution:

```
class MapSum() {

def insert(key: String, `val`: Int): Unit = {

}

def sum(prefix: String): Int = {
```

```
}

}

/**
* Your MapSum object will be instantiated and called as such:
* val obj = new MapSum()
* obj.insert(key,`val`)
* val param_2 = obj.sum(prefix)
*/
```

Elixir Solution:

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

end

@spec insert(key :: String.t, val :: integer) :: any
def insert(key, val) do

end

@spec sum(prefix :: String.t) :: integer
def sum(prefix) do

end
end

# Your functions will be called as such:
# MapSum.init_()
# MapSum.insert(key, val)
# param_2 = MapSum.sum(prefix)

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

Erlang Solution:

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

-spec map_sum_insert(Key :: unicode:unicode_binary(), Val :: integer()) ->
any().
map_sum_insert(Key, Val) ->
.

-spec map_sum_sum(Prefix :: unicode:unicode_binary()) -> integer().
map_sum_sum(Prefix) ->
.


%% Your functions will be called as such:
%% map_sum_init_(),
%% map_sum_insert(Key, Val),
%% Param_2 = map_sum_sum(Prefix),

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

Racket Solution:

```
(define map-sum%
(class object%
(super-new)

(init-field)

; insert : string? exact-integer? -> void?
(define/public (insert key val)
)
; sum : string? -> exact-integer?
(define/public (sum prefix)
)))

;; Your map-sum% object will be instantiated and called as such:
;; (define obj (new map-sum%))
;; (send obj insert key val)
;; (define param_2 (send obj sum prefix))
```