Problem 1472: Design Browser History

Problem Information

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

Problem Description

You have a

browser

of one tab where you start on the

homepage

and you can visit another

url

, get back in the history number of

steps

or move forward in the history number of

steps

.

Implement the

BrowserHistory

class:

BrowserHistory(string homepage)

Initializes the object with the

homepage

of the browser.

void visit(string url)

Visits

url

from the current page. It clears up all the forward history.

string back(int steps)

Move

steps

back in history. If you can only return

x

steps in the history and

steps > x

, you will return only

x

steps. Return the current

url

after moving back in history

at most

steps

.

string forward(int steps)

Move

steps

forward in history. If you can only forward

x

steps in the history and

steps > x

, you will forward only

x

steps. Return the current

url

after forwarding in history

at most

steps

.

Example:

Input:

["BrowserHistory","visit","visit","visit","back","back","forward","visit","forward","back","back"] [["leetcode.com"],["google.com"],["facebook.com"],["youtube.com"],[1],[1],[1],["linkedin.com"],[2],[2],[7]]

Output:

[null,null,null,null,"facebook.com","google.com","facebook.com",null,"linkedin.com","google.com","leetcode.com"]

Explanation:

BrowserHistory browserHistory = new BrowserHistory("leetcode.com"); browserHistory.visit("google.com"); // You are in "leetcode.com". Visit "google.com" browserHistory.visit("facebook.com"); // You are in "google.com". Visit "facebook.com" browserHistory.visit("youtube.com"); // You are in "facebook.com". Visit "youtube.com" browserHistory.back(1); // You are in "youtube.com", move back to "facebook.com" return "facebook.com" browserHistory.back(1); // You are in "facebook.com", move back to "google.com" return "google.com" browserHistory.forward(1); // You are in "google.com", move forward to "facebook.com" return "facebook.com" browserHistory.visit("linkedin.com"); // You are in "facebook.com". Visit "linkedin.com" browserHistory.forward(2); // You are in "linkedin.com", you cannot move forward any steps. browserHistory.back(2); // You are in "linkedin.com", move back two steps to "facebook.com" then to "google.com". return "google.com" browserHistory.back(7); // You are in "google.com", you can move back only one step to "leetcode.com". return "leetcode.com"

Constraints:

1 <= homepage.length <= 20

1 <= url.length <= 20

1 <= steps <= 100

homepage

and

url

consist of '.' or lower case English letters.

At most

5000

calls will be made to

visit

,

back

, and

forward

.

Code Snippets

C++:

```
class BrowserHistory {
public:
BrowserHistory(string homepage) {

}

void visit(string url) {

}

string back(int steps) {

}
```

```
string forward(int steps) {

}
};

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory* obj = new BrowserHistory(homepage);
* obj->visit(url);
* string param_2 = obj->back(steps);
* string param_3 = obj->forward(steps);
*/
```

Java:

```
class BrowserHistory {

public BrowserHistory(String homepage) {

}

public void visit(String url) {

}

public String back(int steps) {

}

public String forward(int steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = new BrowserHistory(homepage);
* obj.visit(url);
* String param_2 = obj.back(steps);
* String param_3 = obj.forward(steps);
*/
```

Python3:

```
class BrowserHistory:

def __init__(self, homepage: str):


def visit(self, url: str) -> None:


def back(self, steps: int) -> str:


def forward(self, steps: int) -> str:



# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

Python:

```
class BrowserHistory(object):

def __init__(self, homepage):
"""
:type homepage: str
"""



def visit(self, url):
"""
:type url: str
:rtype: None
"""



def back(self, steps):
"""
```

```
        :type steps: int
        :rtype: str
        """


    def forward(self, steps):
        """
        :type steps: int
        :rtype: str
        """



# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

JavaScript:

```
/**
* @param {string} homepage
*/
var BrowserHistory = function(homepage) {

};

/**
* @param {string} url
* @return {void}
*/
BrowserHistory.prototype.visit = function(url) {

};

/**
* @param {number} steps
* @return {string}
*/
BrowserHistory.prototype.back = function(steps) {
```

```
};

/**
* @param {number} steps
* @return {string}
*/
BrowserHistory.prototype.forward = function(steps) {

};

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = new BrowserHistory(homepage)
* obj.visit(url)
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

TypeScript:

```
class BrowserHistory {
constructor(homepage: string) {

}

visit(url: string): void {

}

back(steps: number): string {

}

forward(steps: number): string {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = new BrowserHistory(homepage)
* obj.visit(url)
```

```
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

C#:

```
public class BrowserHistory {

public BrowserHistory(string homepage) {

}

public void Visit(string url) {

}

public string Back(int steps) {

}

public string Forward(int steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = new BrowserHistory(homepage);
* obj.Visit(url);
* string param_2 = obj.Back(steps);
* string param_3 = obj.Forward(steps);
*/
```

C:

```
typedef struct {

} BrowserHistory;
```

```
BrowserHistory* browserHistoryCreate(char* homepage) {

}

void browserHistoryVisit(BrowserHistory* obj, char* url) {

}

char* browserHistoryBack(BrowserHistory* obj, int steps) {

}

char* browserHistoryForward(BrowserHistory* obj, int steps) {

}

void browserHistoryFree(BrowserHistory* obj) {

}

/**
* Your BrowserHistory struct will be instantiated and called as such:
* BrowserHistory* obj = browserHistoryCreate(homepage);
* browserHistoryVisit(obj, url);

* char* param_2 = browserHistoryBack(obj, steps);

* char* param_3 = browserHistoryForward(obj, steps);

* browserHistoryFree(obj);
*/
```

Go:

```
type BrowserHistory struct {

}


func Constructor(homepage string) BrowserHistory {
```

```
}


func (this *BrowserHistory) Visit(url string) {

}


func (this *BrowserHistory) Back(steps int) string {

}


func (this *BrowserHistory) Forward(steps int) string {

}


/**
* Your BrowserHistory object will be instantiated and called as such:
* obj := Constructor(homepage);
* obj.Visit(url);
* param_2 := obj.Back(steps);
* param_3 := obj.Forward(steps);
*/
```

Kotlin:

```
class BrowserHistory(homepage: String) {

fun visit(url: String) {

}

fun back(steps: Int): String {

}

fun forward(steps: Int): String {

}
```

```
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = BrowserHistory(homepage)
* obj.visit(url)
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

Swift:

```
class BrowserHistory {

init(_ homepage: String) {

}

func visit(_ url: String) {

}

func back(_ steps: Int) -> String {

}

func forward(_ steps: Int) -> String {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* let obj = BrowserHistory(homepage)
* obj.visit(url)
* let ret_2: String = obj.back(steps)
* let ret_3: String = obj.forward(steps)
*/
```

Rust:

```
struct BrowserHistory {

}


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

fn new(homepage: String) -> Self {

}

fn visit(&self, url: String) {

}

fn back(&self, steps: i32) -> String {

}

fn forward(&self, steps: i32) -> String {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* let obj = BrowserHistory::new(homepage);
* obj.visit(url);
* let ret_2: String = obj.back(steps);
* let ret_3: String = obj.forward(steps);
*/
```

Ruby:

```
class BrowserHistory

=begin
:type homepage: String
=end
```

```
def initialize(homepage)

end


=begin
:type url: String
:rtype: Void
=end
def visit(url)

end


=begin
:type steps: Integer
:rtype: String
=end
def back(steps)

end


=begin
:type steps: Integer
:rtype: String
=end
def forward(steps)

end


end

# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory.new(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

PHP:

```
class BrowserHistory {
/**
* @param String $homepage
*/
function __construct($homepage) {

}

/**
* @param String $url
* @return NULL
*/
function visit($url) {

}

/**
* @param Integer $steps
* @return String
*/
function back($steps) {

}

/**
* @param Integer $steps
* @return String
*/
function forward($steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* $obj = BrowserHistory($homepage);
* $obj->visit($url);
* $ret_2 = $obj->back($steps);
* $ret_3 = $obj->forward($steps);
*/
```

Dart:

```
class BrowserHistory {

BrowserHistory(String homepage) {

}

void visit(String url) {

}

String back(int steps) {

}

String forward(int steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = BrowserHistory(homepage);
* obj.visit(url);
* String param2 = obj.back(steps);
* String param3 = obj.forward(steps);
*/
```

Scala:

```
class BrowserHistory(_homepage: String) {

def visit(url: String): Unit = {

}

def back(steps: Int): String = {

}

def forward(steps: Int): String = {

}
```

```
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* val obj = new BrowserHistory(homepage)
* obj.visit(url)
* val param_2 = obj.back(steps)
* val param_3 = obj.forward(steps)
*/
```

Elixir:

```
defmodule BrowserHistory do
@spec init_(homepage :: String.t) :: any
def init_(homepage) do

end

@spec visit(url :: String.t) :: any
def visit(url) do

end

@spec back(steps :: integer) :: String.t
def back(steps) do

end

@spec forward(steps :: integer) :: String.t
def forward(steps) do

end
end

# Your functions will be called as such:
# BrowserHistory.init_(homepage)
# BrowserHistory.visit(url)
# param_2 = BrowserHistory.back(steps)
# param_3 = BrowserHistory.forward(steps)

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

Erlang:

```
-spec browser_history_init_(Homepage :: unicode:unicode_binary()) -> any().
browser_history_init_(Homepage) ->
.

-spec browser_history_visit(Url :: unicode:unicode_binary()) -> any().
browser_history_visit(Url) ->
.

-spec browser_history_back(Steps :: integer()) -> unicode:unicode_binary().
browser_history_back(Steps) ->
.

-spec browser_history_forward(Steps :: integer()) ->
unicode:unicode_binary().
browser_history_forward(Steps) ->
.


%% Your functions will be called as such:
%% browser_history_init_(Homepage),
%% browser_history_visit(Url),
%% Param_2 = browser_history_back(Steps),
%% Param_3 = browser_history_forward(Steps),

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

Racket:

```
(define browser-history%
(class object%
(super-new)

; homepage : string?
(init-field
homepage)

; visit : string? -> void?
(define/public (visit url)
)
```

```
; back : exact-integer? -> string?
(define/public (back steps)
)
; forward : exact-integer? -> string?
(define/public (forward steps)
)))

;; Your browser-history% object will be instantiated and called as such:
;; (define obj (new browser-history% [homepage homepage]))
;; (send obj visit url)
;; (define param_2 (send obj back steps))
;; (define param_3 (send obj forward steps))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class BrowserHistory {
public:
BrowserHistory(string homepage) {

}

void visit(string url) {

}

string back(int steps) {

}
```

```
string forward(int steps) {

}
};

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory* obj = new BrowserHistory(homepage);
* obj->visit(url);
* string param_2 = obj->back(steps);
* string param_3 = obj->forward(steps);
*/
```

Java Solution:

```
/**
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class BrowserHistory {

public BrowserHistory(String homepage) {

}

public void visit(String url) {

}

public String back(int steps) {

}

public String forward(int steps) {
```

```
}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = new BrowserHistory(homepage);
* obj.visit(url);
* String param_2 = obj.back(steps);
* String param_3 = obj.forward(steps);
*/
```

Python3 Solution:

```
"""
Problem: Design Browser History
Difficulty: Medium
Tags: array, string, linked_list, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class BrowserHistory:

def __init__(self, homepage: str):


def visit(self, url: str) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class BrowserHistory(object):

def __init__(self, homepage):
"""
:type homepage: str
"""
```

```
def visit(self, url):
"""
:type url: str
:rtype: None
"""


def back(self, steps):
"""
:type steps: int
:rtype: str
"""


def forward(self, steps):
"""
:type steps: int
:rtype: str
"""



# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

JavaScript Solution:

```
/**
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
/**
* @param {string} homepage
*/
var BrowserHistory = function(homepage) {

};

/**
* @param {string} url
* @return {void}
*/
BrowserHistory.prototype.visit = function(url) {

};

/**
* @param {number} steps
* @return {string}
*/
BrowserHistory.prototype.back = function(steps) {

};

/**
* @param {number} steps
* @return {string}
*/
BrowserHistory.prototype.forward = function(steps) {

};

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = new BrowserHistory(homepage)
* obj.visit(url)
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class BrowserHistory {
constructor(homepage: string) {

}

visit(url: string): void {

}

back(steps: number): string {

}

forward(steps: number): string {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = new BrowserHistory(homepage)
* obj.visit(url)
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

C# Solution:

```
/*
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class BrowserHistory {

public BrowserHistory(string homepage) {

}

public void Visit(string url) {

}

public string Back(int steps) {

}

public string Forward(int steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = new BrowserHistory(homepage);
* obj.Visit(url);
* string param_2 = obj.Back(steps);
* string param_3 = obj.Forward(steps);
*/
```

C Solution:

```
/*
* Problem: Design Browser History
* Difficulty: Medium
* Tags: array, string, linked_list, stack
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/



typedef struct {

} BrowserHistory;


BrowserHistory* browserHistoryCreate(char* homepage) {

}

void browserHistoryVisit(BrowserHistory* obj, char* url) {

}

char* browserHistoryBack(BrowserHistory* obj, int steps) {

}

char* browserHistoryForward(BrowserHistory* obj, int steps) {

}

void browserHistoryFree(BrowserHistory* obj) {

}

/**
* Your BrowserHistory struct will be instantiated and called as such:
* BrowserHistory* obj = browserHistoryCreate(homepage);
* browserHistoryVisit(obj, url);

* char* param_2 = browserHistoryBack(obj, steps);

* char* param_3 = browserHistoryForward(obj, steps);

* browserHistoryFree(obj);
```

```
*/
```

Go Solution:

```
// Problem: Design Browser History
// Difficulty: Medium
// Tags: array, string, linked_list, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type BrowserHistory struct {

}


func Constructor(homepage string) BrowserHistory {

}


func (this *BrowserHistory) Visit(url string) {

}


func (this *BrowserHistory) Back(steps int) string {

}


func (this *BrowserHistory) Forward(steps int) string {

}


/**
* Your BrowserHistory object will be instantiated and called as such:
* obj := Constructor(homepage);
* obj.Visit(url);
```

```
* param_2 := obj.Back(steps);
* param_3 := obj.Forward(steps);
*/
```

Kotlin Solution:

```
class BrowserHistory(homepage: String) {

fun visit(url: String) {

}

fun back(steps: Int): String {

}

fun forward(steps: Int): String {

}

}

/**
* Your BrowserHistory object will be instantiated and called as such:
* var obj = BrowserHistory(homepage)
* obj.visit(url)
* var param_2 = obj.back(steps)
* var param_3 = obj.forward(steps)
*/
```

Swift Solution:

```
class BrowserHistory {

init(_ homepage: String) {

}

func visit(_ url: String) {
```

```
}

func back(_ steps: Int) -> String {

}

func forward(_ steps: Int) -> String {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* let obj = BrowserHistory(homepage)
* obj.visit(url)
* let ret_2: String = obj.back(steps)
* let ret_3: String = obj.forward(steps)
*/
```

Rust Solution:

```
// Problem: Design Browser History
// Difficulty: Medium
// Tags: array, string, linked_list, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct BrowserHistory {

}


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

fn new(homepage: String) -> Self {
```

```
}

fn visit(&self, url: String) {

}

fn back(&self, steps: i32) -> String {

}

fn forward(&self, steps: i32) -> String {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* let obj = BrowserHistory::new(homepage);
* obj.visit(url);
* let ret_2: String = obj.back(steps);
* let ret_3: String = obj.forward(steps);
*/
```

Ruby Solution:

```
class BrowserHistory

=begin
:type homepage: String
=end
def initialize(homepage)

end


=begin
:type url: String
:rtype: Void
=end
def visit(url)
```

```
end


=begin
:type steps: Integer
:rtype: String
=end
def back(steps)

end


=begin
:type steps: Integer
:rtype: String
=end
def forward(steps)

end


end

# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory.new(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

PHP Solution:

```
class BrowserHistory {
/**
* @param String $homepage
*/
function __construct($homepage) {

}

/**
```

```
* @param String $url
* @return NULL
*/
function visit($url) {

}

/**
* @param Integer $steps
* @return String
*/
function back($steps) {

}

/**
* @param Integer $steps
* @return String
*/
function forward($steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* $obj = BrowserHistory($homepage);
* $obj->visit($url);
* $ret_2 = $obj->back($steps);
* $ret_3 = $obj->forward($steps);
*/
```

Dart Solution:

```
class BrowserHistory {

BrowserHistory(String homepage) {

}

void visit(String url) {
```

```
}

String back(int steps) {

}

String forward(int steps) {

}
}

/**
* Your BrowserHistory object will be instantiated and called as such:
* BrowserHistory obj = BrowserHistory(homepage);
* obj.visit(url);
* String param2 = obj.back(steps);
* String param3 = obj.forward(steps);
*/
```

Scala Solution:

```
class BrowserHistory(_homepage: String) {

def visit(url: String): Unit = {

}

def back(steps: Int): String = {

}

def forward(steps: Int): String = {

}

}

/**
* Your BrowserHistory object will be instantiated and called as such:
* val obj = new BrowserHistory(homepage)
```

```
* obj.visit(url)
* val param_2 = obj.back(steps)
* val param_3 = obj.forward(steps)
*/
```

Elixir Solution:

```
defmodule BrowserHistory do
@spec init_(homepage :: String.t) :: any
def init_(homepage) do

end

@spec visit(url :: String.t) :: any
def visit(url) do

end

@spec back(steps :: integer) :: String.t
def back(steps) do

end

@spec forward(steps :: integer) :: String.t
def forward(steps) do

end
end

# Your functions will be called as such:
# BrowserHistory.init_(homepage)
# BrowserHistory.visit(url)
# param_2 = BrowserHistory.back(steps)
# param_3 = BrowserHistory.forward(steps)

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

Erlang Solution:

```
-spec browser_history_init_(Homepage :: unicode:unicode_binary()) -> any().
browser_history_init_(Homepage) ->
.

-spec browser_history_visit(Url :: unicode:unicode_binary()) -> any().
browser_history_visit(Url) ->
.

-spec browser_history_back(Steps :: integer()) -> unicode:unicode_binary().
browser_history_back(Steps) ->
.

-spec browser_history_forward(Steps :: integer()) ->
unicode:unicode_binary().
browser_history_forward(Steps) ->
.


%% Your functions will be called as such:
%% browser_history_init_(Homepage),
%% browser_history_visit(Url),
%% Param_2 = browser_history_back(Steps),
%% Param_3 = browser_history_forward(Steps),

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

Racket Solution:

```
(define browser-history%
(class object%
(super-new)

; homepage : string?
(init-field
homepage)

; visit : string? -> void?
(define/public (visit url)
)
; back : exact-integer? -> string?
(define/public (back steps)
```

```
)
; forward : exact-integer? -> string?
(define/public (forward steps)
)))

;; Your browser-history% object will be instantiated and called as such:
;; (define obj (new browser-history% [homepage homepage]))
;; (send obj visit url)
;; (define param_2 (send obj back steps))
;; (define param_3 (send obj forward steps))
```