Problem 382: Linked List Random Node

Problem Information

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

Problem Description

Given a singly linked list, return a random node's value from the linked list. Each node must have the

same probability

of being chosen.

Implement the

Solution

class:

Solution(ListNode head)

Initializes the object with the head of the singly-linked list

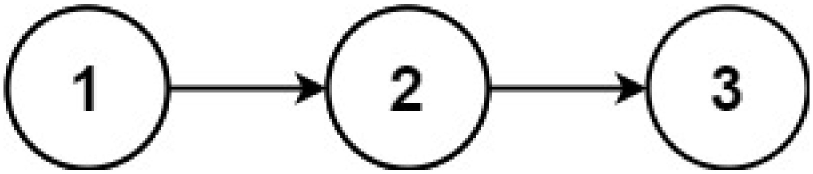
head

.

int getRandom()

Chooses a node randomly from the list and returns its value. All the nodes of the list should be equally likely to be chosen.

Example 1:



Input

["Solution", "getRandom", "getRandom", "getRandom", "getRandom", "getRandom"] [[[1, 2, 3]], [], [], [], [], []]

Output

[null, 1, 3, 2, 2, 3]

Explanation

Solution solution = new Solution([1, 2, 3]); solution.getRandom(); // return 1 solution.getRandom(); // return 3 solution.getRandom(); // return 2 solution.getRandom(); // return 2 solution.getRandom(); // return 3 // getRandom() should return either 1, 2, or 3 randomly. Each element should have equal probability of returning.

Constraints:

The number of nodes in the linked list will be in the range

[1, 10

4

]

.

-10

4

<= Node.val <= 10

4

At most

10

4

calls will be made to

getRandom

.

Follow up:

What if the linked list is extremely large and its length is unknown to you?

Could you solve this efficiently without using extra space?

Code Snippets

C++:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
Solution(ListNode* head) {

}
int getRandom() {

}
};

/**
* Your Solution object will be instantiated and called as such:
* Solution* obj = new Solution(head);
* int param_1 = obj->getRandom();
*/
```

Java:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {

public Solution(ListNode head) {

}

public int getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(head);
* int param_1 = obj.getRandom();
*/
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:

def __init__(self, head: Optional[ListNode]):


def getRandom(self) -> int:



# Your Solution object will be instantiated and called as such:
# obj = Solution(head)
# param_1 = obj.getRandom()
```

Python:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):

def __init__(self, head):
"""
:type head: Optional[ListNode]
"""


def getRandom(self):
"""
:rtype: int
"""



# Your Solution object will be instantiated and called as such:
```

```
# obj = Solution(head)
# param_1 = obj.getRandom()
```

JavaScript:

```
/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {ListNode} head
*/
var Solution = function(head) {

};

/**
* @return {number}
*/
Solution.prototype.getRandom = function() {

};

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(head)
* var param_1 = obj.getRandom()
*/
```

TypeScript:

```
/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

class Solution {
constructor(head: ListNode | null) {

}

getRandom(): number {

}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(head)
* var param_1 = obj.getRandom()
*/
```

C#:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {

public Solution(ListNode head) {

}

public int GetRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(head);
* int param_1 = obj.GetRandom();
*/
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/




typedef struct {

} Solution;



Solution* solutionCreate(struct ListNode* head) {

}

int solutionGetRandom(Solution* obj) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(head);
* int param_1 = solutionGetRandom(obj);
* solutionFree(obj);
*/
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
type Solution struct {

}



func Constructor(head *ListNode) Solution {

}



func (this *Solution) GetRandom() int {

}



/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(head);
* param_1 := obj.GetRandom();
*/
```

Kotlin:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution(head: ListNode?) {

fun getRandom(): Int {

}

}

/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(head)
* var param_1 = obj.getRandom()
*/
```

Swift:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next = 
next; }
* }
*/

class Solution {

init(_ head: ListNode?) {

}

func getRandom() -> Int {

}
}

/**
```

```
* Your Solution object will be instantiated and called as such:
* let obj = Solution(head)
* let ret_1: Int = obj.getRandom()
*/
```

Rust:

```
// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
struct Solution {

}



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

fn new(head: Option<Box<ListNode>>) -> Self {

}

fn get_random(&self) -> i32 {

}
}
```

```
/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(head);
* let ret_1: i32 = obj.get_random();
*/
```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
class Solution

=begin
:type head: ListNode
=end
def initialize(head)

end


=begin
:rtype: Integer
=end
def get_random()

end


end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(head)
# param_1 = obj.get_random()
```

PHP:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {
/**
* @param ListNode $head
*/
function __construct($head) {

}

/**
* @return Integer
*/
function getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($head);
* $ret_1 = $obj->getRandom();
*/
```

Dart:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {

Solution(ListNode? head) {

}

int getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(head);
* int param1 = obj.getRandom();
*/
```

Scala:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
class Solution(_head: ListNode) {

def getRandom(): Int = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(head)
* val param_1 = obj.getRandom()
*/
```

Elixir:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec init_(head :: ListNode.t | nil) :: any
def init_(head) do

end

@spec get_random() :: integer
def get_random() do

end
end

# Your functions will be called as such:
# Solution.init_(head)
# param_1 = Solution.get_random()

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

Erlang:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec solution_init_(Head :: #list_node{} | null) -> any().
solution_init_(Head) ->
.
```

```
-spec solution_get_random() -> integer().
solution_get_random() ->
.



%% Your functions will be called as such:
%% solution_init_(Head),
%% Param_1 = solution_get_random(),

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

Racket:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define solution%
(class object%
(super-new)

; head : (or/c list-node? #f)
(init-field
head)

; get-random : -> exact-integer?
(define/public (get-random)
)))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [head head]))
;; (define param_1 (send obj get-random))
```

Solutions

C++ Solution:

```
/*
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* ListNode(int x) : val(x), next(nullptr) {
// TODO: Implement optimized solution
return 0;
}
* ListNode(int x, ListNode *next) : val(x), next(next) {
// TODO: Implement optimized solution
return 0;
}
* };
*/
class Solution {
public:
Solution(ListNode* head) {

}
int getRandom() {

}
};

/**
* Your Solution object will be instantiated and called as such:
* Solution* obj = new Solution(head);
* int param_1 = obj->getRandom();
*/
```

Java Solution:

```
/**
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {
// TODO: Implement optimized solution
return 0;
}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {

public Solution(ListNode head) {
}

public int getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(head);
* int param_1 = obj.getRandom();
*/
```

Python3 Solution:

```
"""
Problem: Linked List Random Node
Difficulty: Medium
Tags: math, linked_list

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:

def __init__(self, head: Optional[ListNode]):


def getRandom(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):

def __init__(self, head):
"""
:type head: Optional[ListNode]
"""


def getRandom(self):
"""
:rtype: int
"""



# Your Solution object will be instantiated and called as such:
# obj = Solution(head)
# param_1 = obj.getRandom()
```

JavaScript Solution:

```
/**
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
```

```
*/
/**
* @param {ListNode} head
*/
var Solution = function(head) {

};

/**
* @return {number}
*/
Solution.prototype.getRandom = function() {

};

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(head)
* var param_1 = obj.getRandom()
*/
```

TypeScript Solution:

```
/**
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

class Solution {
constructor(head: ListNode | null) {

}

getRandom(): number {

}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(head)
* var param_1 = obj.getRandom()
*/
```

C# Solution:

```
/*
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {

public Solution(ListNode head) {

}

public int GetRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(head);
* int param_1 = obj.GetRandom();
*/
```

C Solution:

```
/*
* Problem: Linked List Random Node
* Difficulty: Medium
* Tags: math, linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
```

```
typedef struct {

} Solution;


Solution* solutionCreate(struct ListNode* head) {

}

int solutionGetRandom(Solution* obj) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(head);
* int param_1 = solutionGetRandom(obj);

* solutionFree(obj);
*/
```

Go Solution:

```
// Problem: Linked List Random Node
// Difficulty: Medium
// Tags: math, linked_list
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
```

```
*/
type Solution struct {

}


func Constructor(head *ListNode) Solution {

}


func (this *Solution) GetRandom() int {

}


/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(head);
* param_1 := obj.GetRandom();
*/
```

Kotlin Solution:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution(head: ListNode?) {

fun getRandom(): Int {

}

}
```

```
/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(head)
* var param_1 = obj.getRandom()
*/
```

Swift Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/

class Solution {

init(_ head: ListNode?) {

}

func getRandom() -> Int {

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution(head)
* let ret_1: Int = obj.getRandom()
*/
```

Rust Solution:

```
// Problem: Linked List Random Node
// Difficulty: Medium
// Tags: math, linked_list
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
struct Solution {

}


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

fn new(head: Option<Box<ListNode>>) -> Self {

}

fn get_random(&self) -> i32 {

}
}
```

```
/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(head);
* let ret_1: i32 = obj.get_random();
*/
```

Ruby Solution:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
class Solution

=begin
:type head: ListNode
=end
def initialize(head)

end


=begin
:rtype: Integer
=end
def get_random()

end


end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(head)
# param_1 = obj.get_random()
```

PHP Solution:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {
/**
* @param ListNode $head
*/
function __construct($head) {

}

/**
* @return Integer
*/
function getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($head);
* $ret_1 = $obj->getRandom();
*/
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
```

```
*/
class Solution {

Solution(ListNode? head) {

}

int getRandom() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(head);
* int param1 = obj.getRandom();
*/
```

Scala Solution:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
class Solution(_head: ListNode) {

def getRandom(): Int = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(head)
* val param_1 = obj.getRandom()
*/
```

Elixir Solution:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec init_(head :: ListNode.t | nil) :: any
def init_(head) do

end

@spec get_random() :: integer
def get_random() do

end
end

# Your functions will be called as such:
# Solution.init_(head)
# param_1 = Solution.get_random()

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

Erlang Solution:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec solution_init_(Head :: #list_node{} | null) -> any().
solution_init_(Head) ->
.

-spec solution_get_random() -> integer().
```

```
solution_get_random() ->
.



%% Your functions will be called as such:
%% solution_init_(Head),
%% Param_1 = solution_get_random(),

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

Racket Solution:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define solution%
(class object%
(super-new)

; head : (or/c list-node? #f)
(init-field
head)

; get-random : -> exact-integer?
(define/public (get-random)
)))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [head head]))
;; (define param_1 (send obj get-random))
```