Problem 3062: Winner of the Linked List Game

Problem Information

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

Problem Description

You are given the

head

of a linked list of

even

length containing integers.

Each

odd-indexed

node contains an odd integer and each

even-indexed

node contains an even integer.

We call each even-indexed node and its next node a

pair

, e.g., the nodes with indices

0

and

1

are a pair, the nodes with indices

2

and

3

are a pair, and so on.

For every

pair

, we compare the values of the nodes in the pair:

If the odd-indexed node is higher, the

"Odd"

team gets a point.

If the even-indexed node is higher, the

"Even"

team gets a point.

Return

the name of the team with the

higher

points, if the points are equal, return

"Tie"

.

Example 1:

Input:

head = [2,1]

Output:

"Even"

Explanation:

There is only one pair in this linked list and that is

(2,1)

. Since

2 > 1

, the Even team gets the point.

Hence, the answer would be

"Even"

.

Example 2:

Input:

head = [2,5,4,7,20,5]

Output:

"Odd"

Explanation:

There are

3

pairs in this linked list. Let's investigate each pair individually:

(2,5)

-> Since

2 < 5

, The Odd team gets the point.

(4,7)

-> Since

4 < 7

, The Odd team gets the point.

(20,5)

-> Since

20 > 5

, The Even team gets the point.

The Odd team earned

2

points while the Even team got

1

point and the Odd team has the higher points.

Hence, the answer would be

"Odd"

.

Example 3:

Input:

head = [4,5,2,1]

Output:

"Tie"

Explanation:

There are

2

pairs in this linked list. Let's investigate each pair individually:

(4,5)

-> Since

4 < 5

, the Odd team gets the point.

(2,1)

-> Since

2 > 1

, the Even team gets the point.

Both teams earned

1

point.

Hence, the answer would be

"Tie"

.

Constraints:

The number of nodes in the list is in the range

[2, 100]

.

The number of nodes in the list is even.

1 <= Node.val <= 100

The value of each odd-indexed node is odd.

The value of each even-indexed node is even.

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:
string gameResult(ListNode* head) {

}
};
```

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 String gameResult(ListNode head) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
```

```
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def gameResult(self, head: Optional[ListNode]) -> str:
```

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 gameResult(self, head):
"""
:type head: Optional[ListNode]
:rtype: str
"""
```

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
* @return {string}
*/
var gameResult = function(head) {

};
```

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)
* }
* }
*/

function gameResult(head: ListNode | null): string {

};
```

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 string GameResult(ListNode head) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
char* gameResult(struct ListNode* head) {



}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func gameResult(head *ListNode) string {

}
```

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 {
fun gameResult(head: ListNode?): String {

}
}
```

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 {
func gameResult(_ head: ListNode?) -> String {

}
}
```

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
// }
// }
// }
impl Solution {
pub fn game_result(head: Option<Box<ListNode>>) -> String {

}
}
```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} head
# @return {String}
def game_result(head)

end
```

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 $head
* @return String
*/
function gameResult($head) {

}
}
```

Dart:

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

```
class Solution {
String gameResult(ListNode? head) {

}
}
```

Scala:

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

}
}
```

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 game_result(head :: ListNode.t | nil) :: String.t
def game_result(head) do

end
end
```

Erlang:

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

-spec game_result(Head :: #list_node{} | null) -> unicode:unicode_binary().
game_result(Head) ->
.
```

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/contract (game-result head)
(-> (or/c list-node? #f) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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:
string gameResult(ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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 String gameResult(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Winner of the Linked List Game
Difficulty: Easy
Tags: 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 gameResult(self, head: Optional[ListNode]) -> str:
# 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 gameResult(self, head):
"""
:type head: Optional[ListNode]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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
* @return {string}
*/
var gameResult = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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)
* }
* }
*/

function gameResult(head: ListNode | null): string {

};
```

C# Solution:

```
/*
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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 string GameResult(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Winner of the Linked List Game
* Difficulty: Easy
* Tags: 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;
* };
*/
char* gameResult(struct ListNode* head) {

}
```

Go Solution:

```
// Problem: Winner of the Linked List Game
// Difficulty: Easy
// Tags: 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
* }
*/
func gameResult(head *ListNode) string {

}
```

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 {
fun gameResult(head: ListNode?): String {

}
}
```

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 {
func gameResult(_ head: ListNode?) -> String {

}
}
```

Rust Solution:

```
// Problem: Winner of the Linked List Game
// Difficulty: Easy
// Tags: 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
// }
// }
// }
impl Solution {
pub fn game_result(head: Option<Box<ListNode>>) -> String {

}

}
```

Ruby Solution:

```ruby
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} head
# @return {String}
def game_result(head)

end
```

PHP Solution:

```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 $head
* @return String
*/
function gameResult($head) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

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 game_result(head :: ListNode.t | nil) :: String.t
def game_result(head) do

end
end
```

Erlang Solution:

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

-spec game_result(Head :: #list_node{} | null) -> unicode:unicode_binary().
game_result(Head) ->
.
```

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/contract (game-result head)
(-> (or/c list-node? #f) string?)
)
```