Problem 1629: Slowest Key

Problem Information

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

Problem Description

A newly designed keypad was tested, where a tester pressed a sequence of

n

keys, one at a time.

You are given a string

keysPressed

of length

n

, where

keysPressed[i]

was the

i

th

key pressed in the testing sequence, and a sorted list

releaseTimes

, where

releaseTimes[i]

was the time the

i

th

key was released. Both arrays are

0-indexed

. The

0

th

key was pressed at the time

0

, and every subsequent key was pressed at the

exact

time the previous key was released.

The tester wants to know the key of the keypress that had the

longest duration

. The

i

th

keypress had a

duration

of

releaseTimes[i] - releaseTimes[i - 1]

, and the

0

th

keypress had a duration of

releaseTimes[0]

.

Note that the same key could have been pressed multiple times during the test, and these multiple presses of the same key

may not

have had the same

duration

.

Return the key of the keypress that had the

longest duration

. If there are multiple such keypresses, return the lexicographically largest key of the keypresses.

Example 1:

Input:

releaseTimes = [9,29,49,50], keysPressed = "cbcd"

Output:

"c"

Explanation:

The keypresses were as follows: Keypress for 'c' had a duration of 9 (pressed at time 0 and released at time 9). Keypress for 'b' had a duration of 29 - 9 = 20 (pressed at time 9 right after the release of the previous character and released at time 29). Keypress for 'c' had a duration of 49 - 29 = 20 (pressed at time 29 right after the release of the previous character and released at time 49). Keypress for 'd' had a duration of 50 - 49 = 1 (pressed at time 49 right after the release of the previous character and released at time 50). The longest of these was the keypress for 'b' and the second keypress for 'c', both with duration 20. 'c' is lexicographically larger than 'b', so the answer is 'c'.

Example 2:

Input:

releaseTimes = [12,23,36,46,62], keysPressed = "spuda"

Output:

"a"

Explanation:

The keypresses were as follows: Keypress for 's' had a duration of 12. Keypress for 'p' had a duration of 23 - 12 = 11. Keypress for 'u' had a duration of 36 - 23 = 13. Keypress for 'd' had a duration of 46 - 36 = 10. Keypress for 'a' had a duration of 62 - 46 = 16. The longest of these was the keypress for 'a' with duration 16.

Constraints:

releaseTimes.length == n

keysPressed.length == n

2 <= n <= 1000

1 <= releaseTimes[i] <= 10

9

releaseTimes[i] < releaseTimes[i+1]

keysPressed

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
char slowestKey(vector<int>& releaseTimes, string keysPressed) {

}
};
```

Java:

```
class Solution {
public char slowestKey(int[] releaseTimes, String keysPressed) {

}
}
```

Python3:

```
class Solution:
def slowestKey(self, releaseTimes: List[int], keysPressed: str) -> str:
```

Python:

```
class Solution(object):
def slowestKey(self, releaseTimes, keysPressed):
"""
:type releaseTimes: List[int]
:type keysPressed: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number[]} releaseTimes
* @param {string} keysPressed
* @return {character}
*/
var slowestKey = function(releaseTimes, keysPressed) {

};
```

TypeScript:

```
function slowestKey(releaseTimes: number[], keysPressed: string): string {

};
```

C#:

```
public class Solution {
public char SlowestKey(int[] releaseTimes, string keysPressed) {

}
}
```

C:

```
char slowestKey(int* releaseTimes, int releaseTimesSize, char* keysPressed) {

}
```

Go:

```
func slowestKey(releaseTimes []int, keysPressed string) byte {

}
```

Kotlin:

```
class Solution {
fun slowestKey(releaseTimes: IntArray, keysPressed: String): Char {

}
}
```

Swift:

```
class Solution {
func slowestKey(_ releaseTimes: [Int], _ keysPressed: String) -> Character {

}
}
```

Rust:

```
impl Solution {
pub fn slowest_key(release_times: Vec<i32>, keys_pressed: String) -> char {

}
}
```

Ruby:

```
# @param {Integer[]} release_times
# @param {String} keys_pressed
# @return {Character}
def slowest_key(release_times, keys_pressed)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $releaseTimes
```

```
* @param String $keysPressed
* @return String
*/
function slowestKey($releaseTimes, $keysPressed) {

}
}
```

Dart:

```
class Solution {
String slowestKey(List<int> releaseTimes, String keysPressed) {

}
}
```

Scala:

```
object Solution {
def slowestKey(releaseTimes: Array[Int], keysPressed: String): Char = {

}
}
```

Elixir:

```
defmodule Solution do
@spec slowest_key(release_times :: [integer], keys_pressed :: String.t) ::
char
def slowest_key(release_times, keys_pressed) do

end
end
```

Erlang:

```
-spec slowest_key(ReleaseTimes :: [integer()], KeysPressed ::
unicode:unicode_binary()) -> char().
slowest_key(ReleaseTimes, KeysPressed) ->
.
```

Racket:

```
(define/contract (slowest-key releaseTimes keysPressed)
(-> (listof exact-integer?) string? char?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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 Solution {
public:
char slowestKey(vector<int>& releaseTimes, string keysPressed) {

}
};
```

Java Solution:

```
/**
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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 Solution {
public char slowestKey(int[] releaseTimes, String keysPressed) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Slowest Key
Difficulty: Easy
Tags: array, string, graph, sort

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 Solution:
def slowestKey(self, releaseTimes: List[int], keysPressed: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def slowestKey(self, releaseTimes, keysPressed):
"""
:type releaseTimes: List[int]
:type keysPressed: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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 {number[]} releaseTimes
* @param {string} keysPressed
* @return {character}
*/
var slowestKey = function(releaseTimes, keysPressed) {

};
```

TypeScript Solution:

```
/**
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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
*/

function slowestKey(releaseTimes: number[], keysPressed: string): string {

};
```

C# Solution:

```
/*
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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 Solution {
public char SlowestKey(int[] releaseTimes, string keysPressed) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Slowest Key
* Difficulty: Easy
* Tags: array, string, graph, sort
*
* 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
*/

char slowestKey(int* releaseTimes, int releaseTimesSize, char* keysPressed) {

}
```

Go Solution:

```
// Problem: Slowest Key
// Difficulty: Easy
// Tags: array, string, graph, sort
//
// 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

func slowestKey(releaseTimes []int, keysPressed string) byte {

}
```

Kotlin Solution:

```
class Solution {
fun slowestKey(releaseTimes: IntArray, keysPressed: String): Char {

}
}
```

Swift Solution:

```
class Solution {
func slowestKey(_ releaseTimes: [Int], _ keysPressed: String) -> Character {

}
}
```

Rust Solution:

```
// Problem: Slowest Key
// Difficulty: Easy
// Tags: array, string, graph, sort
//
// 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

impl Solution {
pub fn slowest_key(release_times: Vec<i32>, keys_pressed: String) -> char {

}
}
```

Ruby Solution:

```
# @param {Integer[]} release_times
# @param {String} keys_pressed
# @return {Character}
def slowest_key(release_times, keys_pressed)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $releaseTimes
* @param String $keysPressed
* @return String
*/
function slowestKey($releaseTimes, $keysPressed) {
```

```
}
}
```

Dart Solution:

```
class Solution {
String slowestKey(List<int> releaseTimes, String keysPressed) {

}
}
```

Scala Solution:

```
object Solution {
def slowestKey(releaseTimes: Array[Int], keysPressed: String): Char = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec slowest_key(release_times :: [integer], keys_pressed :: String.t) ::
char
def slowest_key(release_times, keys_pressed) do

end
end
```

Erlang Solution:

```
-spec slowest_key(ReleaseTimes :: [integer()], KeysPressed ::
unicode:unicode_binary()) -> char().
slowest_key(ReleaseTimes, KeysPressed) ->
.
```

Racket Solution:

```
(define/contract (slowest-key releaseTimes keysPressed)
(-> (listof exact-integer?) string? char?)
)
```