Problem 2606: Find the Substring With Maximum Cost

Problem Information

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

Problem Description

You are given a string

s

, a string

chars

of

distinct

characters and an integer array

vals

of the same length as

chars

.

The

cost of the substring

is the sum of the values of each character in the substring. The cost of an empty string is considered

0

.

The

value of the character

is defined in the following way:

If the character is not in the string

chars

, then its value is its corresponding position

(1-indexed)

in the alphabet.

For example, the value of

'a'

is

1

, the value of

'b'

is

2

, and so on. The value of

'z'

is

26

.

Otherwise, assuming

i

is the index where the character occurs in the string

chars

, then its value is

vals[i]

.

Return

the maximum cost among all substrings of the string

s

.

Example 1:

Input:

s = "adaa", chars = "d", vals = [-1000]

Output:

2

Explanation:

The value of the characters "a" and "d" is 1 and -1000 respectively. The substring with the maximum cost is "aa" and its cost is 1 + 1 = 2. It can be proven that 2 is the maximum cost.

Example 2:

Input:

s = "abc", chars = "abc", vals = [-1,-1,-1]

Output:

0

Explanation:

The value of the characters "a", "b" and "c" is -1, -1, and -1 respectively. The substring with the maximum cost is the empty substring "" and its cost is 0. It can be proven that 0 is the maximum cost.

Constraints:

1 <= s.length <= 10

5

s

consist of lowercase English letters.

1 <= chars.length <= 26

chars

consist of

distinct

lowercase English letters.

vals.length == chars.length

-1000 <= vals[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maximumCostSubstring(string s, string chars, vector<int>& vals) {

}
};
```

Java:

```
class Solution {
public int maximumCostSubstring(String s, String chars, int[] vals) {

}
}
```

Python3:

```
class Solution:
def maximumCostSubstring(self, s: str, chars: str, vals: List[int]) -> int:
```

Python:

```
class Solution(object):
def maximumCostSubstring(self, s, chars, vals):
"""
:type s: str
```

```
:type chars: str
:type vals: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {string} chars
* @param {number[]} vals
* @return {number}
*/
var maximumCostSubstring = function(s, chars, vals) {

};
```

TypeScript:

```
function maximumCostSubstring(s: string, chars: string, vals: number[]):
number {

};
```

C#:

```
public class Solution {
public int MaximumCostSubstring(string s, string chars, int[] vals) {

}
}
```

C:

```
int maximumCostSubstring(char* s, char* chars, int* vals, int valsSize) {

}
```

Go:

```
func maximumCostSubstring(s string, chars string, vals []int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun maximumCostSubstring(s: String, chars: String, vals: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumCostSubstring(_ s: String, _ chars: String, _ vals: [Int]) -> Int
{

}
}
```

Rust:

```
impl Solution {
pub fn maximum_cost_substring(s: String, chars: String, vals: Vec<i32>) ->
i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} chars
# @param {Integer[]} vals
# @return {Integer}
def maximum_cost_substring(s, chars, vals)

end
```

PHP:

```
class Solution {
```

```
/**
* @param String $s
* @param String $chars
* @param Integer[] $vals
* @return Integer
*/
function maximumCostSubstring($s, $chars, $vals) {

}
}
```

Dart:

```
class Solution {
int maximumCostSubstring(String s, String chars, List<int> vals) {

}
}
```

Scala:

```
object Solution {
def maximumCostSubstring(s: String, chars: String, vals: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_cost_substring(s :: String.t, chars :: String.t, vals ::
[integer]) :: integer
def maximum_cost_substring(s, chars, vals) do

end
end
```

Erlang:

```
-spec maximum_cost_substring(S :: unicode:unicode_binary(), Chars ::
unicode:unicode_binary(), Vals :: [integer()]) -> integer().
maximum_cost_substring(S, Chars, Vals) ->
```

```
.
```

Racket:

```
(define/contract (maximum-cost-substring s chars vals)
(-> string? string? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maximumCostSubstring(string s, string chars, vector<int>& vals) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public int maximumCostSubstring(String s, String chars, int[] vals) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Substring With Maximum Cost
Difficulty: Medium
Tags: array, string, tree, dp, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maximumCostSubstring(self, s: str, chars: str, vals: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumCostSubstring(self, s, chars, vals):
"""
:type s: str
:type chars: str
:type vals: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {string} chars
* @param {number[]} vals
* @return {number}
*/
var maximumCostSubstring = function(s, chars, vals) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumCostSubstring(s: string, chars: string, vals: number[]):
number {

};
```

C# Solution:

```
/*
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaximumCostSubstring(string s, string chars, int[] vals) {

}
}
```

C Solution:

```
/*
* Problem: Find the Substring With Maximum Cost
* Difficulty: Medium
* Tags: array, string, tree, dp, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maximumCostSubstring(char* s, char* chars, int* vals, int valsSize) {

}
```

Go Solution:

```
// Problem: Find the Substring With Maximum Cost
// Difficulty: Medium
// Tags: array, string, tree, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumCostSubstring(s string, chars string, vals []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumCostSubstring(s: String, chars: String, vals: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumCostSubstring(_ s: String, _ chars: String, _ vals: [Int]) -> Int
{

}
}
```

Rust Solution:

```
// Problem: Find the Substring With Maximum Cost
// Difficulty: Medium
// Tags: array, string, tree, dp, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn maximum_cost_substring(s: String, chars: String, vals: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} chars
# @param {Integer[]} vals
# @return {Integer}
def maximum_cost_substring(s, chars, vals)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $chars
* @param Integer[] $vals
* @return Integer
*/
function maximumCostSubstring($s, $chars, $vals) {

}
}
```

Dart Solution:

```
class Solution {
int maximumCostSubstring(String s, String chars, List<int> vals) {

}
}
```

Scala Solution:

```
object Solution {
def maximumCostSubstring(s: String, chars: String, vals: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_cost_substring(s :: String.t, chars :: String.t, vals ::
[integer]) :: integer
def maximum_cost_substring(s, chars, vals) do

end
end
```

Erlang Solution:

```
-spec maximum_cost_substring(S :: unicode:unicode_binary(), Chars ::
unicode:unicode_binary(), Vals :: [integer()]) -> integer().
maximum_cost_substring(S, Chars, Vals) ->
.
```

Racket Solution:

```
(define/contract (maximum-cost-substring s chars vals)
(-> string? string? (listof exact-integer?) exact-integer?)
)
```