Problem 2434: Using a Robot to Print the Lexicographically Smallest String

Problem Information

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

Problem Description

You are given a string

s

and a robot that currently holds an empty string

t

. Apply one of the following operations until

s

and

t

are both empty

:

Remove the

first

character of a string

s

and give it to the robot. The robot will append this character to the string

t

.

Remove the

last

character of a string

t

and give it to the robot. The robot will write this character on paper.

Return

the lexicographically smallest string that can be written on the paper.

Example 1:

Input:

s = "zza"

Output:

"azz"

Explanation:

Let p denote the written string. Initially p="", s="zza", t="". Perform first operation three times p="", s="", t="zza". Perform second operation three times p="azz", s="", t="".

Example 2:

Input:

s = "bac"

Output:

"abc"

Explanation:

Let p denote the written string. Perform first operation twice p="", s="c", t="ba". Perform second operation twice p="ab", s="c", t="". Perform first operation p="ab", s="", t="c". Perform second operation p="abc", s="", t="".

Example 3:

Input:

s = "bdda"

Output:

"addb"

Explanation:

Let p denote the written string. Initially p="", s="bdda", t="". Perform first operation four times p="", s="", t="bdda". Perform second operation four times p="addb", s="", t="".

Constraints:

1 <= s.length <= 10

5

s

consists of only English lowercase letters.

Code Snippets

C++:

```
class Solution {
public:
string robotWithString(string s) {

}
};
```

Java:

```
class Solution {
public String robotWithString(String s) {

}
}
```

Python3:

```
class Solution:
def robotWithString(self, s: str) -> str:
```

Python:

```
class Solution(object):
def robotWithString(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string}
*/
var robotWithString = function(s) {

};
```

TypeScript:

```
function robotWithString(s: string): string {

};
```

C#:

```
public class Solution {
public string RobotWithString(string s) {

}
}
```

C:

```
char* robotWithString(char* s) {

}
```

Go:

```
func robotWithString(s string) string {

}
```

Kotlin:

```
class Solution {
fun robotWithString(s: String): String {

}
}
```

Swift:

```
class Solution {
func robotWithString(_ s: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn robot_with_string(s: String) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @return {String}
def robot_with_string(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String
*/
function robotWithString($s) {

}
}
```

Dart:

```
class Solution {
String robotWithString(String s) {

}
}
```

Scala:

```
object Solution {
def robotWithString(s: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec robot_with_string(s :: String.t) :: String.t
def robot_with_string(s) do

end
end
```

Erlang:

```
-spec robot_with_string(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
robot_with_string(S) ->
.
```

Racket:

```
(define/contract (robot-with-string s)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
```

```
string robotWithString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String robotWithString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Using a Robot to Print the Lexicographically Smallest String
Difficulty: Medium
Tags: string, graph, greedy, hash, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def robotWithString(self, s: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def robotWithString(self, s):
"""
:type s: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @return {string}
*/
var robotWithString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function robotWithString(s: string): string {

};
```

C# Solution:

```
/*
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string RobotWithString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Using a Robot to Print the Lexicographically Smallest String
* Difficulty: Medium
* Tags: string, graph, greedy, hash, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* robotWithString(char* s) {

}
```

Go Solution:

```
// Problem: Using a Robot to Print the Lexicographically Smallest String
// Difficulty: Medium
// Tags: string, graph, greedy, hash, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func robotWithString(s string) string {

}
```

Kotlin Solution:

```
class Solution {
fun robotWithString(s: String): String {

}
}
```

Swift Solution:

```
class Solution {
func robotWithString(_ s: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Using a Robot to Print the Lexicographically Smallest String
// Difficulty: Medium
// Tags: string, graph, greedy, hash, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn robot_with_string(s: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String}
def robot_with_string(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String
*/
function robotWithString($s) {

}
}
```

Dart Solution:

```
class Solution {
String robotWithString(String s) {

}
}
```

Scala Solution:

```
object Solution {
def robotWithString(s: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec robot_with_string(s :: String.t) :: String.t
def robot_with_string(s) do

end
```

```
end
```

Erlang Solution:

```
-spec robot_with_string(S :: unicode:unicode_binary()) ->
unicode:unicode_binary().
robot_with_string(S) ->
.
```

Racket Solution:

```
(define/contract (robot-with-string s)
(-> string? string?)
)
```