Problem 1106: Parsing A Boolean Expression

Problem Information

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

Problem Description

A

boolean expression

is an expression that evaluates to either

true

or

false

. It can be in one of the following shapes:

't'

that evaluates to

true

.

'f'

that evaluates to

false

.

'!(subExpr)'

that evaluates to

the logical NOT

of the inner expression

subExpr

.

'&(subExpr

1

, subExpr

2

, ..., subExpr

n

)'

that evaluates to

the logical AND

of the inner expressions

subExpr

1

, subExpr

2

, ..., subExpr

n

where

n >= 1

.

'|(subExpr

1

, subExpr

2

, ..., subExpr

n

)'

that evaluates to

the logical OR

of the inner expressions

subExpr

1

, subExpr

2

, ..., subExpr

n

where

n >= 1

.

Given a string

expression

that represents a

boolean expression

, return

the evaluation of that expression

.

It is

guaranteed

that the given expression is valid and follows the given rules.

Example 1:

Input:

expression = "&(|(f))"

Output:

false

Explanation:

First, evaluate |(f) --> f. The expression is now "&(f)". Then, evaluate &(f) --> f. The expression is now "f". Finally, return false.

Example 2:

Input:

expression = "|(f,f,f,t)"

Output:

true

Explanation:

The evaluation of (false OR false OR false OR true) is true.

Example 3:

Input:

expression = "!(&(f,t))"

Output:

true

Explanation:

First, evaluate &(f,t) --> (false AND true) --> false --> f. The expression is now "!(f)". Then, evaluate !(f) --> NOT false --> true. We return true.

Constraints:

1 <= expression.length <= 2 * 10

4

expression[i] is one following characters:

'('

,

')'

,

'&'

,

'|'

,

'!'

,

't'

,

'f'

, and

','

.

Code Snippets

C++:

```
class Solution {
public:
bool parseBoolExpr(string expression) {

}
};
```

Java:

```
class Solution {
public boolean parseBoolExpr(String expression) {

}
}
```

Python3:

```
class Solution:
def parseBoolExpr(self, expression: str) -> bool:
```

Python:

```
class Solution(object):
def parseBoolExpr(self, expression):
"""
:type expression: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} expression
* @return {boolean}
*/
var parseBoolExpr = function(expression) {

};
```

TypeScript:

```
function parseBoolExpr(expression: string): boolean {

};
```

C#:

```
public class Solution {
public bool ParseBoolExpr(string expression) {

}
}
```

C:

```
bool parseBoolExpr(char* expression) {

}
```

Go:

```
func parseBoolExpr(expression string) bool {

}
```

Kotlin:

```
class Solution {
fun parseBoolExpr(expression: String): Boolean {

}
}
```

Swift:

```
class Solution {
func parseBoolExpr(_ expression: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn parse_bool_expr(expression: String) -> bool {

}
}
```

Ruby:

```
# @param {String} expression
# @return {Boolean}
def parse_bool_expr(expression)

end
```

PHP:

```
class Solution {

/**
* @param String $expression
* @return Boolean
*/
function parseBoolExpr($expression) {

}
}
```

Dart:

```
class Solution {
bool parseBoolExpr(String expression) {

}
}
```

Scala:

```
object Solution {
def parseBoolExpr(expression: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec parse_bool_expr(expression :: String.t) :: boolean
def parse_bool_expr(expression) do

end
end
```

Erlang:

```
-spec parse_bool_expr(Expression :: unicode:unicode_binary()) -> boolean().
parse_bool_expr(Expression) ->
.
```

Racket:

```
(define/contract (parse-bool-expr expression)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool parseBoolExpr(string expression) {
}
};
```

Java Solution:

```
/**
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean parseBoolExpr(String expression) {

}
}
```

Python3 Solution:

```
"""
Problem: Parsing A Boolean Expression
Difficulty: Hard
Tags: string, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def parseBoolExpr(self, expression: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def parseBoolExpr(self, expression):
"""
:type expression: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} expression
* @return {boolean}
*/
var parseBoolExpr = function(expression) {

};
```

TypeScript Solution:

```
/**
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function parseBoolExpr(expression: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool ParseBoolExpr(string expression) {

}
}
```

C Solution:

```
/*
* Problem: Parsing A Boolean Expression
* Difficulty: Hard
* Tags: string, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool parseBoolExpr(char* expression) {

}
```

Go Solution:

```
// Problem: Parsing A Boolean Expression
// Difficulty: Hard
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func parseBoolExpr(expression string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun parseBoolExpr(expression: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func parseBoolExpr(_ expression: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Parsing A Boolean Expression
// Difficulty: Hard
// Tags: string, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn parse_bool_expr(expression: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} expression
# @return {Boolean}
def parse_bool_expr(expression)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $expression
* @return Boolean
*/
function parseBoolExpr($expression) {

}
}
```

Dart Solution:

```
class Solution {
bool parseBoolExpr(String expression) {

}
}
```

Scala Solution:

```
object Solution {
def parseBoolExpr(expression: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec parse_bool_expr(expression :: String.t) :: boolean
def parse_bool_expr(expression) do

end
```

```


end
```

Erlang Solution:

```
-spec parse_bool_expr(Expression :: unicode:unicode_binary()) -> boolean().
parse_bool_expr(Expression) ->
.
```

Racket Solution:

```
(define/contract (parse-bool-expr expression)
(-> string? boolean?)
)
```