Problem 71: Simplify Path

Problem Information

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

Problem Description

You are given an

absolute

path for a Unix-style file system, which always begins with a slash

'/'

. Your task is to transform this absolute path into its

simplified canonical path

.

The

rules

of a Unix-style file system are as follows:

A single period

'.'

represents the current directory.

A double period

'..'

represents the previous/parent directory.

Multiple consecutive slashes such as

'//'

and

'///'

are treated as a single slash

'/'

.

Any sequence of periods that does

not match

the rules above should be treated as a

valid directory or

file

name

. For example,

'...'

and

'....'

are valid directory or file names.

The simplified canonical path should follow these

rules

:

The path must start with a single slash

'/'

.

Directories within the path must be separated by exactly one slash

'/'

.

The path must not end with a slash

'/'

, unless it is the root directory.

The path must not have any single or double periods (

'.'

and

'..'

) used to denote current or parent directories.

Return the

simplified canonical path

.

Example 1:

Input:

path = "/home/"

Output:

"/home"

Explanation:

The trailing slash should be removed.

Example 2:

Input:

path = "/home//foo/"

Output:

"/home/foo"

Explanation:

Multiple consecutive slashes are replaced by a single one.

Example 3:

Input:

path = "/home/user/Documents/../Pictures"

Output:

"/home/user/Pictures"

Explanation:

A double period

".."

refers to the directory up a level (the parent directory).

Example 4:

Input:

path = "/../"

Output:

"/"

Explanation:

Going one level up from the root directory is not possible.

Example 5:

Input:

path = "/.../a/../b/c/../d/./"

Output:

"/.../b/d"

Explanation:

"..."

is a valid name for a directory in this problem.

Constraints:

1 <= path.length <= 3000

path

consists of English letters, digits, period

'.'

, slash

'/'

or

'_'

.

path

is a valid absolute Unix path.

Code Snippets

C++:

```
class Solution {
public:
string simplifyPath(string path) {

}
};
```

Java:

```
class Solution {
public String simplifyPath(String path) {

}
}
```

Python3:

```
class Solution:
def simplifyPath(self, path: str) -> str:
```

Python:

```
class Solution(object):
def simplifyPath(self, path):
"""
:type path: str
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} path
* @return {string}
*/
var simplifyPath = function(path) {

};
```

TypeScript:

```
function simplifyPath(path: string): string {

};
```

C#:

```
public class Solution {
public string SimplifyPath(string path) {

}
}
```

C:

```
char* simplifyPath(char* path) {

}
```

Go:

```
func simplifyPath(path string) string {

}
```

Kotlin:

```
class Solution {
fun simplifyPath(path: String): String {

}
}
```

Swift:

```
class Solution {
func simplifyPath(_ path: String) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn simplify_path(path: String) -> String {

}
}
```

Ruby:

```
# @param {String} path
# @return {String}
def simplify_path(path)

end
```

PHP:

```
class Solution {

/**
* @param String $path
* @return String
*/
function simplifyPath($path) {

}
}
```

Dart:

```
class Solution {
String simplifyPath(String path) {

}
}
```

Scala:

```
object Solution {
def simplifyPath(path: String): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec simplify_path(path :: String.t) :: String.t
def simplify_path(path) do

end
end
```

Erlang:

```
-spec simplify_path(Path :: unicode:unicode_binary()) ->
unicode:unicode_binary().
simplify_path(Path) ->
```

```
.
```

Racket:

```
(define/contract (simplify-path path)
(-> string? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Simplify Path
* Difficulty: Medium
* 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:
string simplifyPath(string path) {

}
};
```

Java Solution:

```
/**
* Problem: Simplify Path
* Difficulty: Medium
* 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 String simplifyPath(String path) {

}
}
```

Python3 Solution:

```
"""
Problem: Simplify Path
Difficulty: Medium
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 simplifyPath(self, path: str) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def simplifyPath(self, path):
"""
:type path: str
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Simplify Path
* Difficulty: Medium
* 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} path
* @return {string}
*/
var simplifyPath = function(path) {

};
```

TypeScript Solution:

```
/**
* Problem: Simplify Path
* Difficulty: Medium
* 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 simplifyPath(path: string): string {

};
```

C# Solution:

```
/*
* Problem: Simplify Path
* Difficulty: Medium
* 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 string SimplifyPath(string path) {

}
}
```

C Solution:

```
/*
* Problem: Simplify Path
* Difficulty: Medium
* 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
*/

char* simplifyPath(char* path) {

}
```

Go Solution:

```
// Problem: Simplify Path
// Difficulty: Medium
// 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 simplifyPath(path string) string {

}
```

Kotlin Solution:

```
class Solution {
fun simplifyPath(path: String): String {

}
```

```
}
```

Swift Solution:

```
class Solution {
func simplifyPath(_ path: String) -> String {

}
}
```

Rust Solution:

```
// Problem: Simplify Path
// Difficulty: Medium
// 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 simplify_path(path: String) -> String {

}
}
```

Ruby Solution:

```
# @param {String} path
# @return {String}
def simplify_path(path)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $path
* @return String
```

```
*/
function simplifyPath($path) {

}
}
```

Dart Solution:

```
class Solution {
String simplifyPath(String path) {

}
}
```

Scala Solution:

```
object Solution {
def simplifyPath(path: String): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec simplify_path(path :: String.t) :: String.t
def simplify_path(path) do

end
end
```

Erlang Solution:

```
-spec simplify_path(Path :: unicode:unicode_binary()) ->
unicode:unicode_binary().
simplify_path(Path) ->
.
```

Racket Solution:

```
(define/contract (simplify-path path)
(-> string? string?)
)
```