Problem 1087: Brace Expansion

Problem Information

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

Problem Description

You are given a string

s

representing a list of words. Each letter in the word has one or more options.

If there is one option, the letter is represented as is.

If there is more than one option, then curly braces delimit the options. For example,

"{a,b,c}"

represents options

["a", "b", "c"]

.

For example, if

s = "a{b,c}"

, the first character is always

'a'

, but the second character can be

'b'

or

'c'

. The original list is

["ab", "ac"]

.

Return all words that can be formed in this manner,

sorted

in lexicographical order.

Example 1:

Input:

s = "{a,b}c{d,e}f"

Output:

["acdf","acef","bcdf","bcef"]

Example 2:

Input:

s = "abcd"

Output:

["abcd"]

Constraints:

1 <= s.length <= 50

s

consists of curly brackets

'{}'

, commas

','

, and lowercase English letters.

s

is guaranteed to be a valid input.

There are no nested curly brackets.

All characters inside a pair of consecutive opening and ending curly brackets are different.

Code Snippets

C++:

```
class Solution {
public:
vector<string> expand(string s) {

}
};
```

Java:

```
class Solution {
public String[] expand(String s) {
```

```
}
}
```

Python3:

```
class Solution:
def expand(self, s: str) -> List[str]:
```

Python:

```
class Solution(object):
def expand(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {string[]}
*/
var expand = function(s) {

};
```

TypeScript:

```
function expand(s: string): string[] {

};
```

C#:

```
public class Solution {
public string[] Expand(string s) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** expand(char* s, int* returnSize) {

}
```

Go:

```
func expand(s string) []string {

}
```

Kotlin:

```
class Solution {
fun expand(s: String): Array<String> {

}
}
```

Swift:

```
class Solution {
func expand(_ s: String) -> [String] {

}
}
```

Rust:

```
impl Solution {
pub fn expand(s: String) -> Vec<String> {

}
}
```

Ruby:

```
# @param {String} s
# @return {String[]}
```

```
def expand(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return String[]
*/
function expand($s) {

}
}
```

Dart:

```
class Solution {
List<String> expand(String s) {

}
}
```

Scala:

```
object Solution {
def expand(s: String): Array[String] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec expand(s :: String.t) :: [String.t]
def expand(s) do

end
end
```

Erlang:

```
-spec expand(S :: unicode:unicode_binary()) -> [unicode:unicode_binary()].
expand(S) ->
.
```

Racket:

```
(define/contract (expand s)
(-> string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Brace Expansion
* Difficulty: Medium
* Tags: string, graph, sort, search, 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:
vector<string> expand(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Brace Expansion
* Difficulty: Medium
* Tags: string, graph, sort, search, 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[] expand(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Brace Expansion
Difficulty: Medium
Tags: string, graph, sort, search, 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 expand(self, s: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def expand(self, s):
"""
:type s: str
:rtype: List[str]
"""
```

JavaScript Solution:

```
/**
* Problem: Brace Expansion
* Difficulty: Medium
```

```
* Tags: string, graph, sort, search, 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} s
* @return {string[]}
*/
var expand = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Brace Expansion
* Difficulty: Medium
* Tags: string, graph, sort, search, 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 expand(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Brace Expansion
* Difficulty: Medium
* Tags: string, graph, sort, search, 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[] Expand(string s) {

}
}
```

C Solution:

```
/*
* Problem: Brace Expansion
* Difficulty: Medium
* Tags: string, graph, sort, search, 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
char** expand(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Brace Expansion
// Difficulty: Medium
// Tags: string, graph, sort, search, 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 expand(s string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun expand(s: String): Array<String> {

}
}
```

Swift Solution:

```
class Solution {
func expand(_ s: String) -> [String] {

}
}
```

Rust Solution:

```
// Problem: Brace Expansion
// Difficulty: Medium
// Tags: string, graph, sort, search, 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 expand(s: String) -> Vec<String> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {String[]}
def expand(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return String[]
*/
function expand($s) {

}
}
```

Dart Solution:

```
class Solution {
List<String> expand(String s) {

}
}
```

Scala Solution:

```
object Solution {
def expand(s: String): Array[String] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec expand(s :: String.t) :: [String.t]
def expand(s) do

end
end
```

Erlang Solution:

```
-spec expand(S :: unicode:unicode_binary()) -> [unicode:unicode_binary()].
expand(S) ->
.
```

Racket Solution:

```
(define/contract (expand s)
(-> string? (listof string?))
)
```