Problem 756: Pyramid Transition Matrix

Problem Information

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

Problem Description

You are stacking blocks to form a pyramid. Each block has a color, which is represented by a single letter. Each row of blocks contains

one less block

than the row beneath it and is centered on top.

To make the pyramid aesthetically pleasing, there are only specific

triangular patterns

that are allowed. A triangular pattern consists of a

single block

stacked on top of

two blocks

. The patterns are given as a list of three-letter strings

allowed

, where the first two characters of a pattern represent the left and right bottom blocks respectively, and the third character is the top block.

For example,

"ABC"

represents a triangular pattern with a

'C'

block stacked on top of an

'A'

(left) and

'B'

(right) block. Note that this is different from

"BAC"

where

'B'

is on the left bottom and

'A'

is on the right bottom.

You start with a bottom row of blocks

bottom

, given as a single string, that you

must

use as the base of the pyramid.

Given

bottom

and

allowed

, return

true

if you can build the pyramid all the way to the top such that

every triangular pattern

in the pyramid is in

allowed

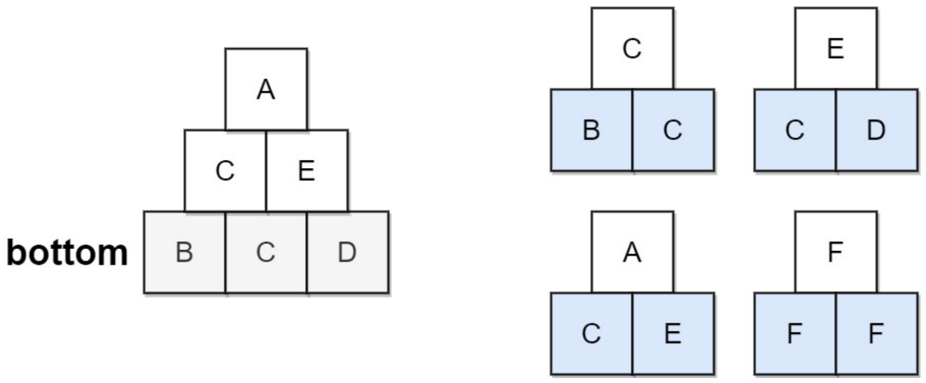
, or

false

otherwise

.

Example 1:



Input:

bottom = "BCD", allowed = ["BCC","CDE","CEA","FFF"]

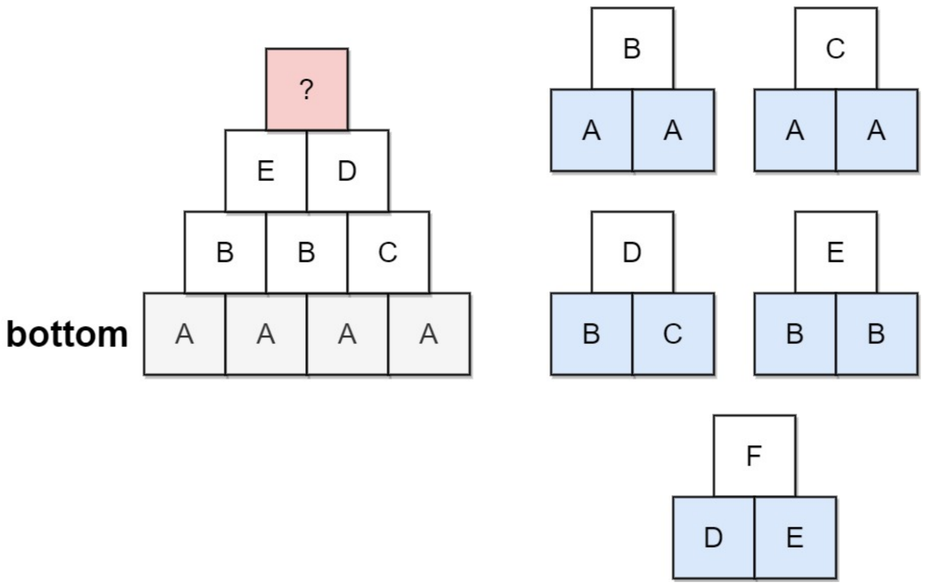
Output:

true

Explanation:

The allowed triangular patterns are shown on the right. Starting from the bottom (level 3), we can build "CE" on level 2 and then build "A" on level 1. There are three triangular patterns in the pyramid, which are "BCC", "CDE", and "CEA". All are allowed.

Example 2:



Input:

bottom = "AAAA", allowed = ["AAB","AAC","BCD","BBE","DEF"]

Output:

false

Explanation:

The allowed triangular patterns are shown on the right. Starting from the bottom (level 4), there are multiple ways to build level 3, but trying all the possibilites, you will get always stuck before building level 1.

Constraints:

2 <= bottom.length <= 6

0 <= allowed.length <= 216

allowed[i].length == 3

The letters in all input strings are from the set

{'A', 'B', 'C', 'D', 'E', 'F'}

.

All the values of

allowed

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
bool pyramidTransition(string bottom, vector<string>& allowed) {

}
};
```

Java:

```
class Solution {
public boolean pyramidTransition(String bottom, List<String> allowed) {

}
}
```

Python3:

```
class Solution:
def pyramidTransition(self, bottom: str, allowed: List[str]) -> bool:
```

Python:

```
class Solution(object):
def pyramidTransition(self, bottom, allowed):
"""
:type bottom: str
:type allowed: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} bottom
* @param {string[]} allowed
* @return {boolean}
*/
var pyramidTransition = function(bottom, allowed) {

};
```

TypeScript:

```
function pyramidTransition(bottom: string, allowed: string[]): boolean {

};
```

C#:

}

```
public class Solution {
public bool PyramidTransition(string bottom, IList<string> allowed) {

}
}
```

C:

```
bool pyramidTransition(char* bottom, char** allowed, int allowedSize) {

}
```

Go:

```
func pyramidTransition(bottom string, allowed []string) bool {

}
```

Kotlin:

```
class Solution {
fun pyramidTransition(bottom: String, allowed: List<String>): Boolean {

}
}
```

Swift:

```
class Solution {
func pyramidTransition(_ bottom: String, _ allowed: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn pyramid_transition(bottom: String, allowed: Vec<String>) -> bool {

}
}
```

Ruby:

```
# @param {String} bottom
# @param {String[]} allowed
# @return {Boolean}
def pyramid_transition(bottom, allowed)

end
```

PHP:

```
class Solution {

/**
* @param String $bottom
* @param String[] $allowed
* @return Boolean
*/
function pyramidTransition($bottom, $allowed) {

}
}
```

Dart:

```
class Solution {
bool pyramidTransition(String bottom, List<String> allowed) {

}
}
```

Scala:

```
object Solution {
def pyramidTransition(bottom: String, allowed: List[String]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec pyramid_transition(bottom :: String.t, allowed :: [String.t]) ::
boolean
def pyramid_transition(bottom, allowed) do
```

```
end
end
```

Erlang:

```
-spec pyramid_transition(Bottom :: unicode:unicode_binary(), Allowed ::
[unicode:unicode_binary()]) -> boolean().
pyramid_transition(Bottom, Allowed) ->
.
```

Racket:

```
(define/contract (pyramid-transition bottom allowed)
(-> string? (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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:
bool pyramidTransition(string bottom, vector<string>& allowed) {

}
};
```

Java Solution:

```
/**
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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 boolean pyramidTransition(String bottom, List<String> allowed) {

}
}
```

Python3 Solution:

```
"""
Problem: Pyramid Transition Matrix
Difficulty: Medium
Tags: string, 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 pyramidTransition(self, bottom: str, allowed: List[str]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def pyramidTransition(self, bottom, allowed):
"""
:type bottom: str
:type allowed: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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} bottom
* @param {string[]} allowed
* @return {boolean}
*/
var pyramidTransition = function(bottom, allowed) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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 pyramidTransition(bottom: string, allowed: string[]): boolean {

};
```

C# Solution:

```csharp
/*
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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 bool PyramidTransition(string bottom, IList<string> allowed) {

}
}
```

C Solution:

```
/*
* Problem: Pyramid Transition Matrix
* Difficulty: Medium
* Tags: string, 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
*/

bool pyramidTransition(char* bottom, char** allowed, int allowedSize) {

}
```

Go Solution:

```
// Problem: Pyramid Transition Matrix
// Difficulty: Medium
// Tags: string, 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 pyramidTransition(bottom string, allowed []string) bool {
}
```

Kotlin Solution:

```
class Solution {
fun pyramidTransition(bottom: String, allowed: List<String>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func pyramidTransition(_ bottom: String, _ allowed: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Pyramid Transition Matrix
// Difficulty: Medium
// Tags: string, 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 pyramid_transition(bottom: String, allowed: Vec<String>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} bottom
# @param {String[]} allowed
# @return {Boolean}
def pyramid_transition(bottom, allowed)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $bottom
* @param String[] $allowed
* @return Boolean
*/
function pyramidTransition($bottom, $allowed) {

}
}
```

Dart Solution:

```
class Solution {
bool pyramidTransition(String bottom, List<String> allowed) {

}
}
```

Scala Solution:

```
object Solution {
def pyramidTransition(bottom: String, allowed: List[String]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pyramid_transition(bottom :: String.t, allowed :: [String.t]) ::
boolean
def pyramid_transition(bottom, allowed) do

end
end
```

Erlang Solution:

```
-spec pyramid_transition(Bottom :: unicode:unicode_binary(), Allowed ::
[unicode:unicode_binary()]) -> boolean().
pyramid_transition(Bottom, Allowed) ->
.
```

Racket Solution:

```
(define/contract (pyramid-transition bottom allowed)
(-> string? (listof string?) boolean?)
)
```