Problem 2194: Cells in a Range on an Excel Sheet

Problem Information

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

Problem Description

A cell

(r, c)

of an excel sheet is represented as a string

"<col><row>"

where:

<col>

denotes the column number

c

of the cell. It is represented by

alphabetical letters

.

For example, the

1

st

column is denoted by

'A'

, the

2

nd

by

'B'

, the

3

rd

by

'C'

, and so on.

<row>

is the row number

r

of the cell. The

r

th

row is represented by the

integer

r

.

You are given a string

s

in the format

"<col1><row1>:<col2><row2>"

, where

<col1>

represents the column

c1

,

<row1>

represents the row

r1

,

<col2>

represents the column

c2

, and

<row2>

represents the row

r2

, such that

r1 <= r2

and

c1 <= c2

.

Return

the

list of cells

(x, y)

such that

r1 <= x <= r2

and

c1 <= y <= c2

. The cells should be represented as

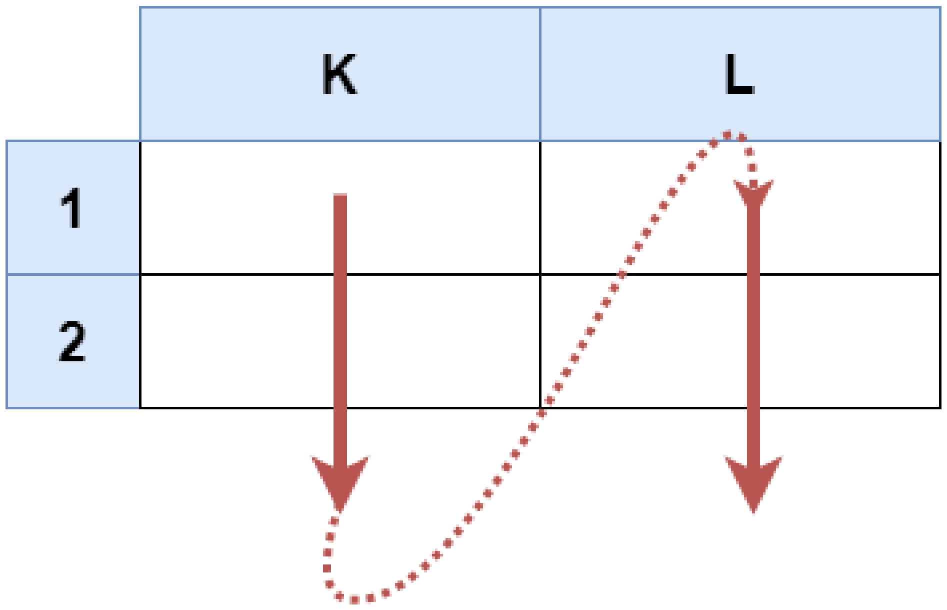
strings

in the format mentioned above and be sorted in

non-decreasing

order first by columns and then by rows.

Example 1:



Input:

s = "K1:L2"

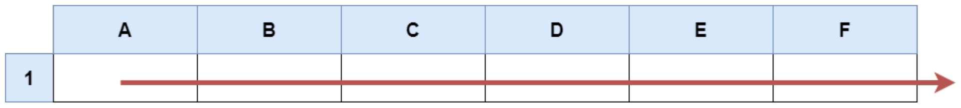
Output:

["K1","K2","L1","L2"]

Explanation:

The above diagram shows the cells which should be present in the list. The red arrows denote the order in which the cells should be presented.

Example 2:



Input:

s = "A1:F1"

Output:

["A1","B1","C1","D1","E1","F1"]

Explanation:

The above diagram shows the cells which should be present in the list. The red arrow denotes the order in which the cells should be presented.

Constraints:

s.length == 5

'A' <= s[0] <= s[3] <= 'Z'

'1' <= s[1] <= s[4] <= '9'

s

consists of uppercase English letters, digits and

':'

.

Code Snippets

C++:

```
class Solution {
public:
```

```
vector<string> cellsInRange(string s) {

}
};
```

Java:

```
class Solution {
public List<String> cellsInRange(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<string> CellsInRange(string s) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun cellsInRange(s: String): List<String> {

}
}
```

Swift:

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

}
}
```

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def cellsInRange(s: String): List[String] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (cells-in-range s)
(-> string? (listof string?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
*
* 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> cellsInRange(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
*
* 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 List<String> cellsInRange(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Cells in a Range on an Excel Sheet
Difficulty: Easy
Tags: string, sort

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 cellsInRange(self, s: str) -> List[str]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
*
* 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 cellsInRange = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
*
* 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 cellsInRange(s: string): string[] {

};
```

C# Solution:

```
/*
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
```

```
*
* 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 IList<string> CellsInRange(string s) {

}
}
```

C Solution:

```
/*
* Problem: Cells in a Range on an Excel Sheet
* Difficulty: Easy
* Tags: string, sort
*
* 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** cellsInRange(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Cells in a Range on an Excel Sheet
// Difficulty: Easy
// Tags: string, sort
//
// 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 cellsInRange(s string) []string {

}
```

Kotlin Solution:

```
class Solution {
fun cellsInRange(s: String): List<String> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Cells in a Range on an Excel Sheet
// Difficulty: Easy
// Tags: string, sort
//
// 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 cells_in_range(s: String) -> Vec<String> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def cellsInRange(s: String): List[String] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (cells-in-range s)
(-> string? (listof string?))
)
```