Problem 1436: Destination City

Problem Information

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

Problem Description

You are given the array

paths

, where

paths[i] = [cityA

i

, cityB

i

]

means there exists a direct path going from

cityA

i

to

cityB

i

.

Return the destination city, that is, the city without any path outgoing to another city.

It is guaranteed that the graph of paths forms a line without any loop, therefore, there will be exactly one destination city.

Example 1:

Input:

paths = [["London","New York"],["New York","Lima"],["Lima","Sao Paulo"]]

Output:

"Sao Paulo"

Explanation:

Starting at "London" city you will reach "Sao Paulo" city which is the destination city. Your trip consist of: "London" -> "New York" -> "Lima" -> "Sao Paulo".

Example 2:

Input:

paths = [["B","C"],["D","B"],["C","A"]]

Output:

"A"

Explanation:

All possible trips are: "D" -> "B" -> "C" -> "A". "B" -> "C" -> "A". "C" -> "A". "A". Clearly the destination city is "A".

Example 3:

Input:

paths = [["A","Z"]]

Output:

"Z"

Constraints:

1 <= paths.length <= 100

paths[i].length == 2

1 <= cityA

i

.length, cityB

i

.length <= 10

cityA

i

!= cityB

i

All strings consist of lowercase and uppercase English letters and the space character.

Code Snippets

C++:

```
class Solution {
public:
string destCity(vector<vector<string>>& paths) {

}
};
```

Java:

```
class Solution {
public String destCity(List<List<String>> paths) {

}
}
```

Python3:

```
class Solution:
def destCity(self, paths: List[List[str]]) -> str:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function destCity(paths: string[][]): string {

};
```

C#:

```
public class Solution {
public string DestCity(IList<IList<string>> paths) {

}
}
```

C:

```
char* destCity(char*** paths, int pathsSize, int* pathsColSize) {

}
```

Go:

```
func destCity(paths [][]string) string {

}
```

Kotlin:

```
class Solution {
fun destCity(paths: List<List<String>>): String {

}
}
```

Swift:

```
class Solution {
func destCity(_ paths: [[String]]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn dest_city(paths: Vec<Vec<String>>) -> String {

}
}
```

Ruby:

```
# @param {String[][]} paths
# @return {String}
def dest_city(paths)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $paths
* @return String
*/
function destCity($paths) {

}
}
```

Dart:

```
class Solution {
String destCity(List<List<String>> paths) {

}
}
```

Scala:

```
object Solution {
def destCity(paths: List[List[String]]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec dest_city(paths :: [[String.t]]) :: String.t
def dest_city(paths) do

end
end
```

Erlang:

```
-spec dest_city(Paths :: [[unicode:unicode_binary()]]) ->
unicode:unicode_binary().
dest_city(Paths) ->
.
```

Racket:

```
(define/contract (dest-city paths)
(-> (listof (listof string?)) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
string destCity(vector<vector<string>>& paths) {

}
};
```

Java Solution:

```
/**
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public String destCity(List<List<String>> paths) {

}
}
```

Python3 Solution:

```
"""
Problem: Destination City
Difficulty: Easy
Tags: array, string, graph, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def destCity(self, paths: List[List[str]]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string[][]} paths
* @return {string}
*/
var destCity = function(paths) {

};
```

TypeScript Solution:

```
/**
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function destCity(paths: string[][]): string {

};
```

C# Solution:

```
/*
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public string DestCity(IList<IList<string>> paths) {

}
}
```

C Solution:

```
/*
* Problem: Destination City
* Difficulty: Easy
* Tags: array, string, graph, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

char* destCity(char*** paths, int pathsSize, int* pathsColSize) {

}
```

Go Solution:

```
// Problem: Destination City
// Difficulty: Easy
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func destCity(paths [][]string) string {

}
```

Kotlin Solution:

```
class Solution {
fun destCity(paths: List<List<String>>): String {

}
}
```

Swift Solution:

```
class Solution {
func destCity(_ paths: [[String]]) -> String {

}
}
```

Rust Solution:

```
// Problem: Destination City
// Difficulty: Easy
// Tags: array, string, graph, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn dest_city(paths: Vec<Vec<String>>) -> String {

}
}
```

Ruby Solution:

```
# @param {String[][]} paths
# @return {String}
def dest_city(paths)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $paths
* @return String
*/
function destCity($paths) {

}
}
```

Dart Solution:

```
class Solution {
String destCity(List<List<String>> paths) {

}
}
```

Scala Solution:

```
object Solution {
def destCity(paths: List[List[String]]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec dest_city(paths :: [[String.t]]) :: String.t
def dest_city(paths) do

end
end
```

Erlang Solution:

```
-spec dest_city(Paths :: [[unicode:unicode_binary()]]) ->
unicode:unicode_binary().
dest_city(Paths) ->
.
```

Racket Solution:

```
(define/contract (dest-city paths)
(-> (listof (listof string?)) string?)
)
```