Problem 797: All Paths From Source to Target

Problem Information

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

Problem Description

Given a directed acyclic graph (

DAG

) of

n

nodes labeled from

0

to

n - 1

, find all possible paths from node

0

to node

n - 1

and return them in

any order

.

The graph is given as follows:

graph[i]

is a list of all nodes you can visit from node

i

(i.e., there is a directed edge from node

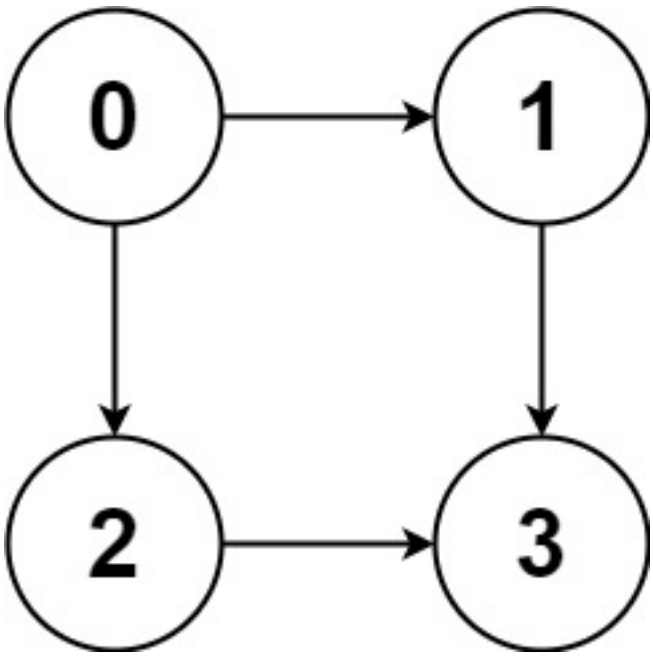
i

to node

graph[i][j]

).

Example 1:



Input:

graph = [[1,2],[3],[3],[]]

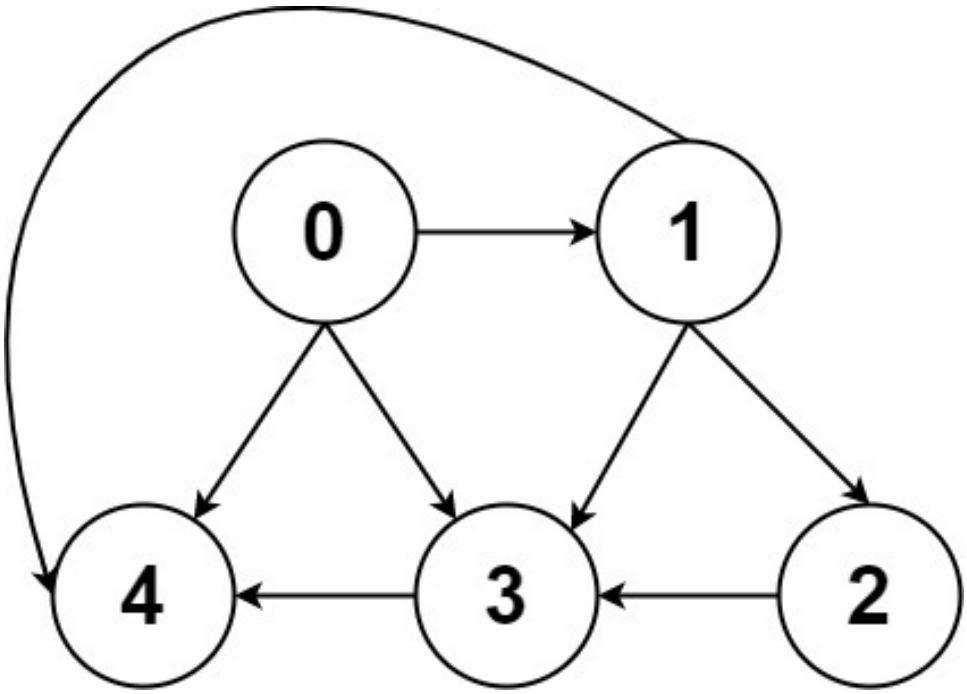
Output:

[[0,1,3],[0,2,3]]

Explanation:

There are two paths: 0 -> 1 -> 3 and 0 -> 2 -> 3.

Example 2:



Input:

graph = [[4,3,1],[3,2,4],[3],[4],[]]

Output:

[[0,4],[0,3,4],[0,1,3,4],[0,1,2,3,4],[0,1,4]]

Constraints:

n == graph.length

2 <= n <= 15

0 <= graph[i][j] < n

graph[i][j] != i

(i.e., there will be no self-loops).

All the elements of

graph[i]

are

unique

.

The input graph is

guaranteed

to be a

DAG

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {
```

```
}
};
```

Java:

```
class Solution {
public List<List<Integer>> allPathsSourceTarget(int[][] graph) {

}
}
```

Python3:

```
class Solution:
def allPathsSourceTarget(self, graph: List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def allPathsSourceTarget(self, graph):
"""
:type graph: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} graph
* @return {number[][]}
*/
var allPathsSourceTarget = function(graph) {

};
```

TypeScript:

```
function allPathsSourceTarget(graph: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> AllPathsSourceTarget(int[][] graph) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func allPathsSourceTarget(graph [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun allPathsSourceTarget(graph: Array<IntArray>): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func allPathsSourceTarget(_ graph: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn all_paths_source_target(graph: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} graph
# @return {Integer[][]}
def all_paths_source_target(graph)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer[][]
*/
function allPathsSourceTarget($graph) {

}
}
```

Dart:

```
class Solution {
List<List<int>> allPathsSourceTarget(List<List<int>> graph) {

}
}
```

Scala:

```
object Solution {
def allPathsSourceTarget(graph: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec all_paths_source_target(graph :: [[integer]]) :: [[integer]]
def all_paths_source_target(graph) do

end
end
```

Erlang:

```
-spec all_paths_source_target(Graph :: [[integer()]]) -> [[integer()]].
all_paths_source_target(Graph) ->
.
```

Racket:

```
(define/contract (all-paths-source-target graph)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> allPathsSourceTarget(vector<vector<int>>& graph) {

}
};
```

Java Solution:

```
/**
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<List<Integer>> allPathsSourceTarget(int[][] graph) {

}
}
```

Python3 Solution:

```
"""
Problem: All Paths From Source to Target
Difficulty: Medium
Tags: graph, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def allPathsSourceTarget(self, graph: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def allPathsSourceTarget(self, graph):
"""
:type graph: List[List[int]]
:rtype: List[List[int]]
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} graph
* @return {number[][]}
*/
var allPathsSourceTarget = function(graph) {

};
```

TypeScript Solution:

```
/**
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function allPathsSourceTarget(graph: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<IList<int>> AllPathsSourceTarget(int[][] graph) {

}
}
```

C Solution:

```
/*
* Problem: All Paths From Source to Target
* Difficulty: Medium
* Tags: graph, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** allPathsSourceTarget(int** graph, int graphSize, int* graphColSize,
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: All Paths From Source to Target
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func allPathsSourceTarget(graph [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun allPathsSourceTarget(graph: Array<IntArray>): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func allPathsSourceTarget(_ graph: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: All Paths From Source to Target
// Difficulty: Medium
// Tags: graph, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn all_paths_source_target(graph: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} graph
# @return {Integer[][]}
def all_paths_source_target(graph)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $graph
* @return Integer[][]
*/
function allPathsSourceTarget($graph) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> allPathsSourceTarget(List<List<int>> graph) {

}
}
```

Scala Solution:

```
object Solution {
def allPathsSourceTarget(graph: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec all_paths_source_target(graph :: [[integer]]) :: [[integer]]
def all_paths_source_target(graph) do

end
end
```

Erlang Solution:

```
-spec all_paths_source_target(Graph :: [[integer()]]) -> [[integer()]].
all_paths_source_target(Graph) ->
.
```

Racket Solution:

```
(define/contract (all-paths-source-target graph)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```