

Problem 3383: Minimum Runes to Add to Cast Spell

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Alice has just graduated from wizard school, and wishes to cast a magic spell to celebrate. The magic spell contains certain

focus points

where magic needs to be concentrated, and some of these focus points contain

magic crystals

which serve as the spell's energy source. Focus points can be linked through

directed runes

, which channel magic flow from one focus point to another.

You are given a integer

n

denoting the

number

of focus points and an array of integers

crystals

where

crystals[i]

indicates a focus point which holds a magic crystal. You are also given two integer arrays

flowFrom

and

flowTo

, which represent the existing

directed runes

. The

i

th

rune allows magic to freely flow from focus point

flowFrom[i]

to focus point

flowTo[i]

You need to find the number of directed runes Alice must add to her spell, such that

each

focus point either:

Contains

a magic crystal.

Receives

magic flow

from

another focus point.

Return the

minimum

number of directed runes that she should add.

Example 1:

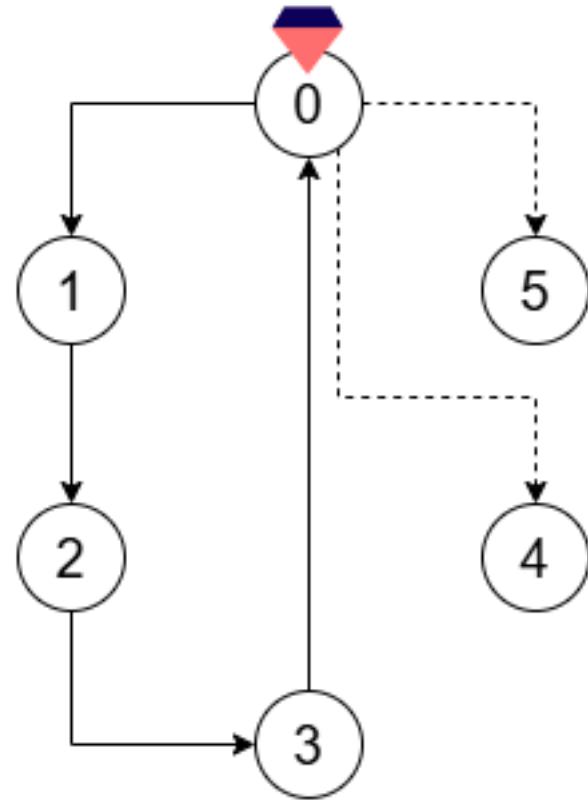
Input:

$n = 6$, crystals = [0], flowFrom = [0,1,2,3], flowTo = [1,2,3,0]

Output:

2

Explanation:



Add two directed runes:

From focus point 0 to focus point 4.

From focus point 0 to focus point 5.

Example 2:

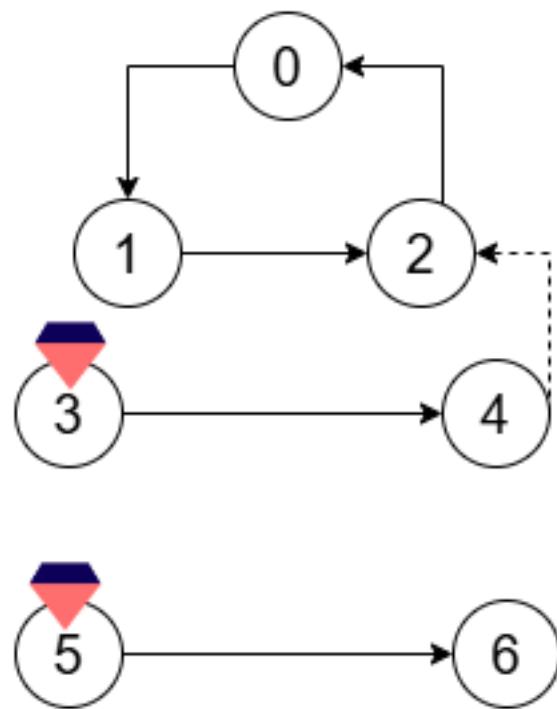
Input:

$n = 7$, crystals = [3,5], flowFrom = [0,1,2,3,5], flowTo = [1,2,0,4,6]

Output:

1

Explanation:



Add a directed rune from focus point 4 to focus point 2.

Constraints:

$$2 \leq n \leq 10$$

$$5$$

$$1 \leq \text{crystals.length} \leq n$$

$$0 \leq \text{crystals}[i] \leq n - 1$$

$$1 \leq \text{flowFrom.length} == \text{flowTo.length} \leq \min(2 * 10)$$

$$5$$

$$, (n * (n - 1)) / 2$$

$$0 \leq \text{flowFrom}[i], \text{flowTo}[i] \leq n - 1$$

$$\text{flowFrom}[i] \neq \text{flowTo}[i]$$

All pre-existing directed runes are

distinct

Code Snippets

C++:

```
class Solution {  
public:  
    int minRunesToAdd(int n, vector<int>& crystals, vector<int>& flowFrom,  
                      vector<int>& flowTo) {  
  
    }  
};
```

Java:

```
class Solution {  
    public int minRunesToAdd(int n, int[] crystals, int[] flowFrom, int[] flowTo)  
    {  
  
    }  
}
```

Python3:

```
class Solution:  
    def minRunesToAdd(self, n: int, crystals: List[int], flowFrom: List[int],  
                      flowTo: List[int]) -> int:
```

Python:

```
class Solution(object):  
    def minRunesToAdd(self, n, crystals, flowFrom, flowTo):  
        """  
        :type n: int  
        :type crystals: List[int]
```

```
:type flowFrom: List[int]
:type flowTo: List[int]
:rtype: int
"""

```

JavaScript:

```
/** 
 * @param {number} n
 * @param {number[]} crystals
 * @param {number[]} flowFrom
 * @param {number[]} flowTo
 * @return {number}
 */
var minRunesToAdd = function(n, crystals, flowFrom, flowTo) {
}
```

TypeScript:

```
function minRunesToAdd(n: number, crystals: number[], flowFrom: number[],
flowTo: number[]): number {

};


```

C#:

```
public class Solution {
public int MinRunesToAdd(int n, int[] crystals, int[] flowFrom, int[] flowTo)
{
}

}
```

C:

```
int minRunesToAdd(int n, int* crystals, int crystalsSize, int* flowFrom, int
flowFromSize, int* flowTo, int flowToSize) {
}
```

Go:

```
func minRunesToAdd(n int, crystals []int, flowFrom []int, flowTo []int) int {  
    }  
}
```

Kotlin:

```
class Solution {  
    fun minRunesToAdd(n: Int, crystals: IntArray, flowFrom: IntArray, flowTo:  
        IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func minRunesToAdd(_ n: Int, _ crystals: [Int], _ flowFrom: [Int], _ flowTo:  
        [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn min_runes_to_add(n: i32, crystals: Vec<i32>, flow_from: Vec<i32>,  
        flow_to: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer} n  
# @param {Integer[]} crystals  
# @param {Integer[]} flow_from  
# @param {Integer[]} flow_to  
# @return {Integer}  
def min_runes_to_add(n, crystals, flow_from, flow_to)  
  
end
```

PHP:

```

class Solution {

    /**
     * @param Integer $n
     * @param Integer[] $crystals
     * @param Integer[] $flowFrom
     * @param Integer[] $flowTo
     * @return Integer
     */
    function minRunesToAdd($n, $crystals, $flowFrom, $flowTo) {

    }
}

```

Dart:

```

class Solution {
int minRunesToAdd(int n, List<int> crystals, List<int> flowFrom, List<int>
flowTo) {

}
}

```

Scala:

```

object Solution {
def minRunesToAdd(n: Int, crystals: Array[Int], flowFrom: Array[Int], flowTo:
Array[Int]): Int = {

}
}

```

Elixir:

```

defmodule Solution do
@spec min_runes_to_add(n :: integer, crystals :: [integer], flow_from :: [integer],
flow_to :: [integer]) :: integer
def min_runes_to_add(n, crystals, flow_from, flow_to) do

end
end

```

Erlang:

```

-spec min_runes_to_add(N :: integer(), Crystals :: [integer()], FlowFrom :: [integer()], FlowTo :: [integer()]) -> integer().
min_runes_to_add(N, Crystals, FlowFrom, FlowTo) ->
    .

```

Racket:

```

(define/contract (min-runes-to-add n crystals flowFrom flowTo)
  (-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
  exact-integer?) exact-integer?))

```

Solutions

C++ Solution:

```

/*
 * Problem: Minimum Runes to Add to Cast Spell
 * Difficulty: Hard
 * Tags: array, graph, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    int minRunesToAdd(int n, vector<int>& crystals, vector<int>& flowFrom,
                     vector<int>& flowTo) {

    }
};

```

Java Solution:

```

/**
 * Problem: Minimum Runes to Add to Cast Spell
 * Difficulty: Hard
 * Tags: array, graph, sort, search
 *

```

```

* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/



class Solution {
public int minRunesToAdd(int n, int[] crystals, int[] flowFrom, int[] flowTo)
{
}

}
}

```

Python3 Solution:

```

"""
Problem: Minimum Runes to Add to Cast Spell
Difficulty: Hard
Tags: array, graph, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:

def minRunesToAdd(self, n: int, crystals: List[int], flowFrom: List[int],
flowTo: List[int]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def minRunesToAdd(self, n, crystals, flowFrom, flowTo):
"""

:type n: int
:type crystals: List[int]
:type flowFrom: List[int]
:type flowTo: List[int]
:rtype: int
"""

```

JavaScript Solution:

```
/**  
 * Problem: Minimum Runes to Add to Cast Spell  
 * Difficulty: Hard  
 * Tags: array, graph, sort, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
/**  
 * @param {number} n  
 * @param {number[]} crystals  
 * @param {number[]} flowFrom  
 * @param {number[]} flowTo  
 * @return {number}  
 */  
var minRunesToAdd = function(n, crystals, flowFrom, flowTo) {  
};
```

TypeScript Solution:

```
/**  
 * Problem: Minimum Runes to Add to Cast Spell  
 * Difficulty: Hard  
 * Tags: array, graph, sort, search  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(1) to O(n) depending on approach  
 */  
  
function minRunesToAdd(n: number, crystals: number[], flowFrom: number[],  
flowTo: number[]): number {  
};
```

C# Solution:

```

/*
 * Problem: Minimum Runes to Add to Cast Spell
 * Difficulty: Hard
 * Tags: array, graph, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public int MinRunesToAdd(int n, int[] crystals, int[] flowFrom, int[] flowTo)
    {

    }
}

```

C Solution:

```

/*
 * Problem: Minimum Runes to Add to Cast Spell
 * Difficulty: Hard
 * Tags: array, graph, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

int minRunesToAdd(int n, int* crystals, int crystalssSize, int* flowFrom, int
flowFromSize, int* flowTo, int flowToSize) {

}

```

Go Solution:

```

// Problem: Minimum Runes to Add to Cast Spell
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)

```

```

// Space Complexity: O(1) to O(n) depending on approach

func minRunesToAdd(n int, crystals []int, flowFrom []int, flowTo []int) int {
}

```

Kotlin Solution:

```

class Solution {
    fun minRunesToAdd(n: Int, crystals: IntArray, flowFrom: IntArray, flowTo:
        IntArray): Int {
        }
    }
}

```

Swift Solution:

```

class Solution {
    func minRunesToAdd(_ n: Int, _ crystals: [Int], _ flowFrom: [Int], _ flowTo:
        [Int]) -> Int {
        }
    }
}

```

Rust Solution:

```

// Problem: Minimum Runes to Add to Cast Spell
// Difficulty: Hard
// Tags: array, graph, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn min_runes_to_add(n: i32, crystals: Vec<i32>, flow_from: Vec<i32>,
                           flow_to: Vec<i32>) -> i32 {
        }
    }
}

```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} crystals
# @param {Integer[]} flow_from
# @param {Integer[]} flow_to
# @return {Integer}

def min_runes_to_add(n, crystals, flow_from, flow_to)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer $n
     * @param Integer[] $crystals
     * @param Integer[] $flowFrom
     * @param Integer[] $flowTo
     * @return Integer
     */
    function minRunesToAdd($n, $crystals, $flowFrom, $flowTo) {

    }
}
```

Dart Solution:

```
class Solution {
int minRunesToAdd(int n, List<int> crystals, List<int> flowFrom, List<int>
flowTo) {
}
```

Scala Solution:

```
object Solution {
def minRunesToAdd(n: Int, crystals: Array[Int], flowFrom: Array[Int], flowTo:
Array[Int]): Int = {
```

```
}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_runes_to_add(n :: integer, crystals :: [integer], flow_from :: [integer], flow_to :: [integer]) :: integer
def min_runes_to_add(n, crystals, flow_from, flow_to) do
end
end
```

Erlang Solution:

```
-spec min_runes_to_add(N :: integer(), Crystals :: [integer()], FlowFrom :: [integer()], FlowTo :: [integer()]) -> integer().
min_runes_to_add(N, Crystals, FlowFrom, FlowTo) ->
.
```

Racket Solution:

```
(define/contract (min-runes-to-add n crystals flowFrom flowTo)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
exact-integer?) exact-integer?))
)
```