Problem 1601: Maximum Number of Achievable Transfer Requests

Problem Information

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

Problem Description

We have

n

buildings numbered from

0

to

n - 1

. Each building has a number of employees. It's transfer season, and some employees want to change the building they reside in.

You are given an array

requests

where

requests[i] = [from

i

, to

i

]

represents an employee's request to transfer from building

from

i

to building

to

i

.

All buildings are full

, so a list of requests is achievable only if for each building, the

net change in employee transfers is zero

. This means the number of employees

leaving

is

equal

to the number of employees

moving in

. For example if

n = 3

and two employees are leaving building

0

, one is leaving building

1

, and one is leaving building

2

, there should be two employees moving to building

0

, one employee moving to building

1

, and one employee moving to building

2

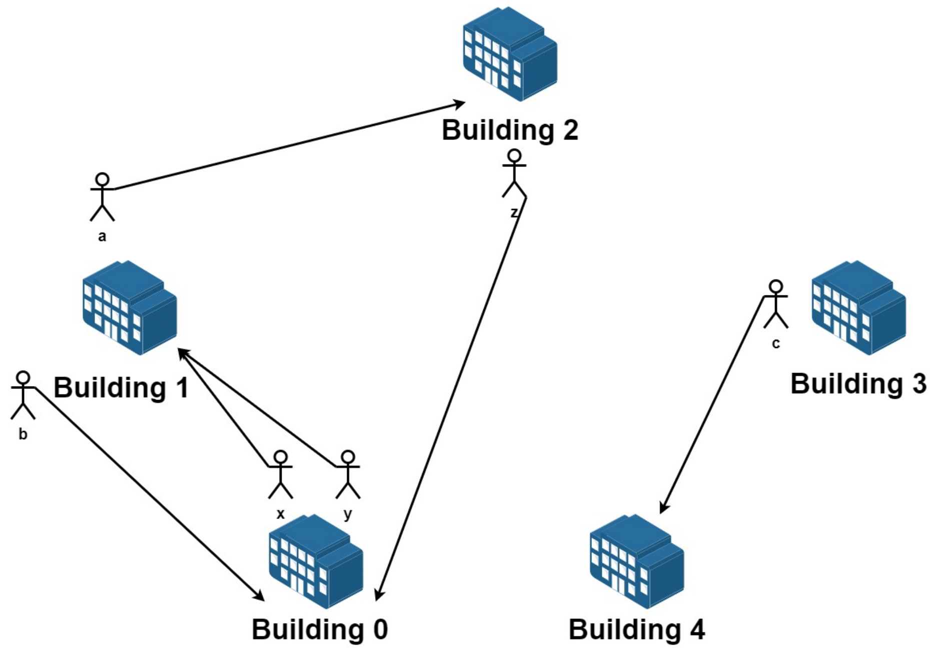
.

Return

the maximum number of achievable requests

.

Example 1:



Input:

n = 5, requests = [[0,1],[1,0],[0,1],[1,2],[2,0],[3,4]]

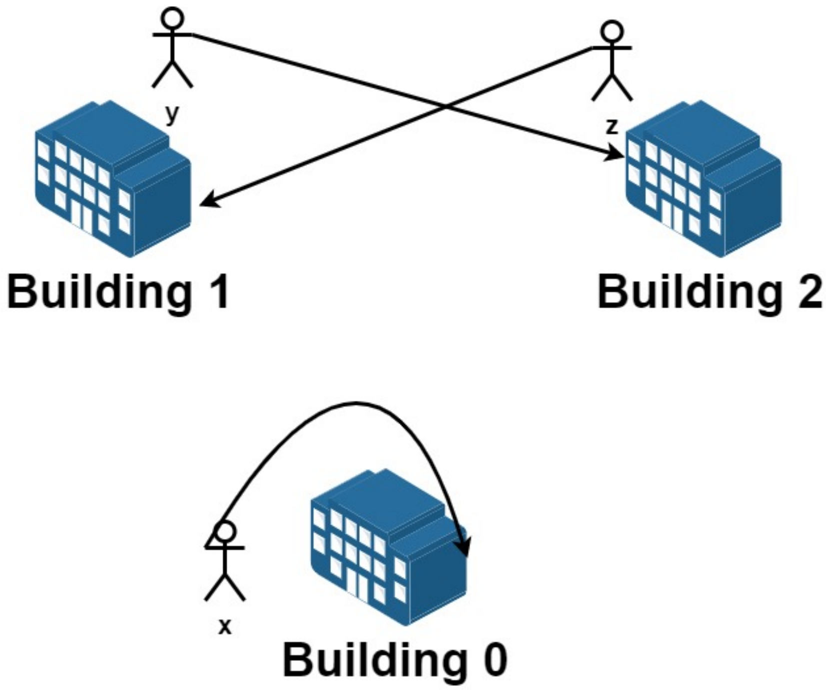
Output:

5

Explantion:

Let's see the requests: From building 0 we have employees x and y and both want to move to building 1. From building 1 we have employees a and b and they want to move to buildings 2 and 0 respectively. From building 2 we have employee z and they want to move to building 0. From building 3 we have employee c and they want to move to building 4. From building 4 we don't have any requests. We can achieve the requests of users x and b by swapping their places. We can achieve the requests of users y, a and z by swapping the places in the 3 buildings.

Example 2:



Input:

n = 3, requests = [[0,0],[1,2],[2,1]]

Output:

3

Explantion:

Let's see the requests: From building 0 we have employee x and they want to stay in the same building 0. From building 1 we have employee y and they want to move to building 2. From building 2 we have employee z and they want to move to building 1. We can achieve all the requests.

Example 3:

Input:

n = 4, requests = [[0,3],[3,1],[1,2],[2,0]]

Output:

4

Constraints:

1 <= n <= 20

1 <= requests.length <= 16

requests[i].length == 2

0 <= from

i

, to

i

< n

Code Snippets

C++:

```
class Solution {
public:
int maximumRequests(int n, vector<vector<int>>& requests) {

}
};
```

Java:

```
class Solution {
public int maximumRequests(int n, int[][] requests) {

}
}
```

Python3:

```
class Solution:
def maximumRequests(self, n: int, requests: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maximumRequests(self, n, requests):
"""
:type n: int
:type requests: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} requests
* @return {number}
*/
var maximumRequests = function(n, requests) {

};
```

TypeScript:

```
function maximumRequests(n: number, requests: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaximumRequests(int n, int[][] requests) {

}
}
```

C:

```
int maximumRequests(int n, int** requests, int requestsSize, int*
requestsColSize) {

}
```

Go:

```
func maximumRequests(n int, requests [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maximumRequests(n: Int, requests: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maximumRequests(_ n: Int, _ requests: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_requests(n: i32, requests: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} requests
# @return {Integer}
def maximum_requests(n, requests)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $requests
* @return Integer
*/
function maximumRequests($n, $requests) {

}
}
```

Dart:

```
class Solution {
int maximumRequests(int n, List<List<int>> requests) {

}
}
```

Scala:

```
object Solution {
def maximumRequests(n: Int, requests: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_requests(n :: integer, requests :: [[integer]]) :: integer
def maximum_requests(n, requests) do

end
end
```

Erlang:

```
-spec maximum_requests(N :: integer(), Requests :: [[integer()]]) ->
integer().
```

```
maximum_requests(N, Requests) ->
.
```

Racket:

```
(define/contract (maximum-requests n requests)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
*
* 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 maximumRequests(int n, vector<vector<int>>& requests) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
*
* 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 maximumRequests(int n, int[][] requests) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Achievable Transfer Requests
Difficulty: Hard
Tags: array

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 maximumRequests(self, n: int, requests: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumRequests(self, n, requests):
"""
:type n: int
:type requests: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
```

```
*
* 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[][]} requests
* @return {number}
*/
var maximumRequests = function(n, requests) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
*
* 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 maximumRequests(n: number, requests: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
*
* 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 MaximumRequests(int n, int[][] requests) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Achievable Transfer Requests
* Difficulty: Hard
* Tags: array
*
* 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 maximumRequests(int n, int** requests, int requestsSize, int*
requestsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Achievable Transfer Requests
// Difficulty: Hard
// Tags: array
//
// 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 maximumRequests(n int, requests [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumRequests(n: Int, requests: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumRequests(_ n: Int, _ requests: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Achievable Transfer Requests
// Difficulty: Hard
// Tags: array
//
// 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 maximum_requests(n: i32, requests: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} requests
# @return {Integer}
def maximum_requests(n, requests)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $requests
* @return Integer
*/
function maximumRequests($n, $requests) {

}
}
```

Dart Solution:

```
class Solution {
int maximumRequests(int n, List<List<int>> requests) {

}
}
```

Scala Solution:

```
object Solution {
def maximumRequests(n: Int, requests: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_requests(n :: integer, requests :: [[integer]]) :: integer
def maximum_requests(n, requests) do

end
end
```

Erlang Solution:

```
-spec maximum_requests(N :: integer(), Requests :: [[integer()]]) ->
integer().
maximum_requests(N, Requests) ->
```

```
.
```

Racket Solution:

```
(define/contract (maximum-requests n requests)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```