Problem 1595: Minimum Cost to Connect Two Groups of Points

Problem Information

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

Problem Description

You are given two groups of points where the first group has

size

1

points, the second group has

size

2

points, and

size

1

>= size

2

.

The

cost

of the connection between any two points are given in an

size

1

x size

2

matrix where

cost[i][j]

is the cost of connecting point

i

of the first group and point

j

of the second group. The groups are connected if

each point in both groups is connected to one or more points in the opposite group

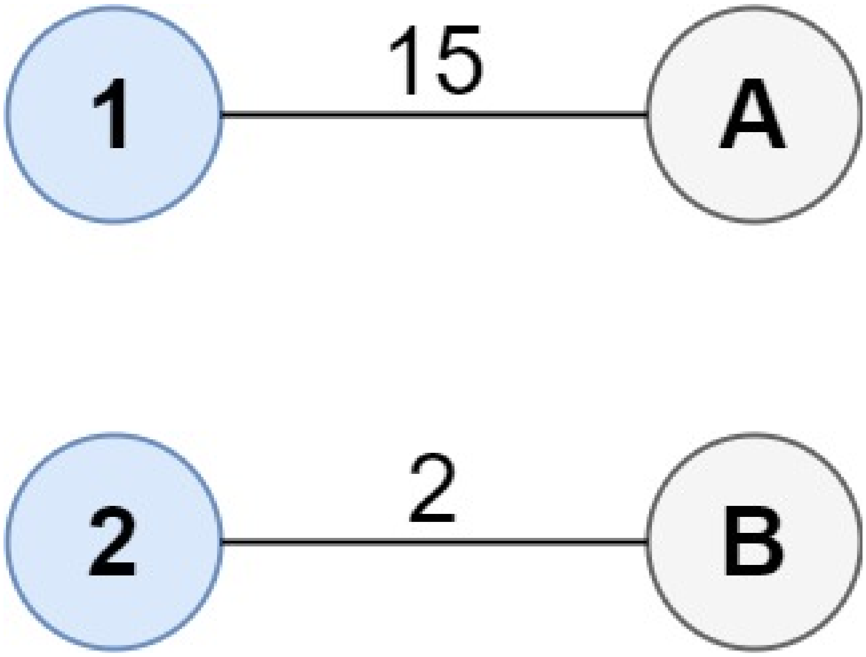
. In other words, each point in the first group must be connected to at least one point in the second group, and each point in the second group must be connected to at least one point in the first group.

Return

the minimum cost it takes to connect the two groups

.

Example 1:



Input:

cost = [[15, 96], [36, 2]]

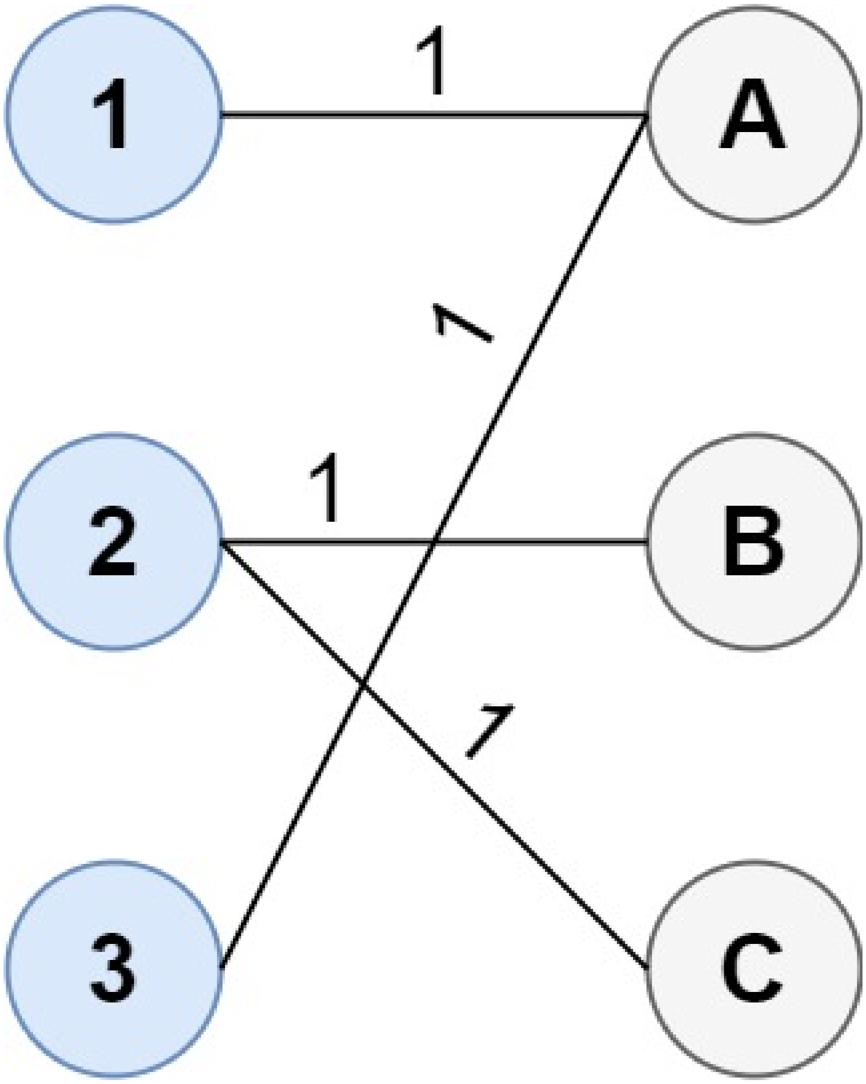
Output:

17

Explanation

: The optimal way of connecting the groups is: 1--A 2--B This results in a total cost of 17.

Example 2:



Input:

cost = [[1, 3, 5], [4, 1, 1], [1, 5, 3]]

Output:

4

Explanation

: The optimal way of connecting the groups is: 1--A 2--B 2--C 3--A This results in a total cost of 4. Note that there are multiple points connected to point 2 in the first group and point A in the second group. This does not matter as there is no limit to the number of points that can be connected. We only care about the minimum total cost.

Example 3:

Input:

cost = [[2, 5, 1], [3, 4, 7], [8, 1, 2], [6, 2, 4], [3, 8, 8]]

Output:

10

Constraints:

size

1

== cost.length

size

2

== cost[i].length

1 <= size

1

, size

2

<= 12

size

1

>= size

2

0 <= cost[i][j] <= 100

Code Snippets

C++:

```
class Solution {
public:
int connectTwoGroups(vector<vector<int>>& cost) {

}
};
```

Java:

```
class Solution {
public int connectTwoGroups(List<List<Integer>> cost) {

}
}
```

Python3:

```
class Solution:
def connectTwoGroups(self, cost: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function connectTwoGroups(cost: number[][]): number {

};
```

C#:

```
public class Solution {
public int ConnectTwoGroups(IList<IList<int>> cost) {

}
}
```

C:

```
int connectTwoGroups(int** cost, int costSize, int* costColSize) {

}
```

Go:

```
func connectTwoGroups(cost [][]int) int {

}
```

Kotlin:

```
class Solution {
fun connectTwoGroups(cost: List<List<Int>>): Int {

}
}
```

Swift:

```
class Solution {
func connectTwoGroups(_ cost: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn connect_two_groups(cost: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} cost
# @return {Integer}
def connect_two_groups(cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $cost
* @return Integer
*/
function connectTwoGroups($cost) {

}
}
```

Dart:

```
class Solution {
int connectTwoGroups(List<List<int>> cost) {

}
}
```

Scala:

```
object Solution {
def connectTwoGroups(cost: List[List[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec connect_two_groups(cost :: [[integer]]) :: integer
def connect_two_groups(cost) do

end
end
```

Erlang:

```
-spec connect_two_groups(Cost :: [[integer()]]) -> integer().
connect_two_groups(Cost) ->
.
```

Racket:

```
(define/contract (connect-two-groups cost)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int connectTwoGroups(vector<vector<int>>& cost) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int connectTwoGroups(List<List<Integer>> cost) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Connect Two Groups of Points
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} cost
* @return {number}
*/
var connectTwoGroups = function(cost) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

function connectTwoGroups(cost: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int ConnectTwoGroups(IList<IList<int>> cost) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Connect Two Groups of Points
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int connectTwoGroups(int** cost, int costSize, int* costColSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Connect Two Groups of Points
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func connectTwoGroups(cost [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun connectTwoGroups(cost: List<List<Int>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func connectTwoGroups(_ cost: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Connect Two Groups of Points
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn connect_two_groups(cost: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} cost
# @return {Integer}
def connect_two_groups(cost)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $cost
* @return Integer
*/
function connectTwoGroups($cost) {

}
}
```

Dart Solution:

```
class Solution {
int connectTwoGroups(List<List<int>> cost) {

}
}
```

Scala Solution:

```
object Solution {
def connectTwoGroups(cost: List[List[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec connect_two_groups(cost :: [[integer]]) :: integer
def connect_two_groups(cost) do

end
end
```

Erlang Solution:

```
-spec connect_two_groups(Cost :: [[integer()]]) -> integer().
connect_two_groups(Cost) ->
.
```

Racket Solution:

```
(define/contract (connect-two-groups cost)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```