Problem 2247: Maximum Cost of Trip With K Highways

Problem Information

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

Problem Description

A series of highways connect

n

cities numbered from

0

to

n - 1

. You are given a 2D integer array

highways

where

highways[i] = [city1

i

, city2

i

, toll

i

]

indicates that there is a highway that connects

city1

i

and

city2

i

, allowing a car to go from

city1

i

to

city2

i

and

vice versa

for a cost of

toll

i

.

You are also given an integer

k

. You are going on a trip that crosses

exactly

k

highways. You may start at any city, but you may only visit each city

at most

once during your trip.

Return

the

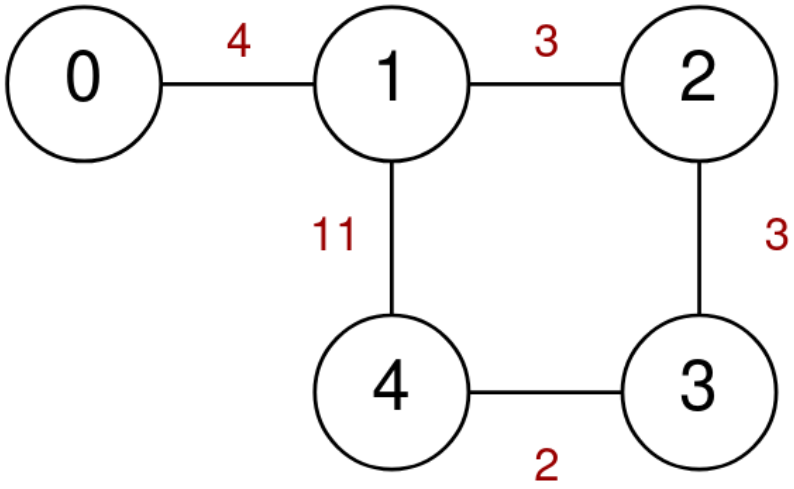
maximum

cost of your trip. If there is no trip that meets the requirements, return

-1

.

Example 1:



Input:

n = 5, highways = [[0,1,4],[2,1,3],[1,4,11],[3,2,3],[3,4,2]], k = 3

Output:

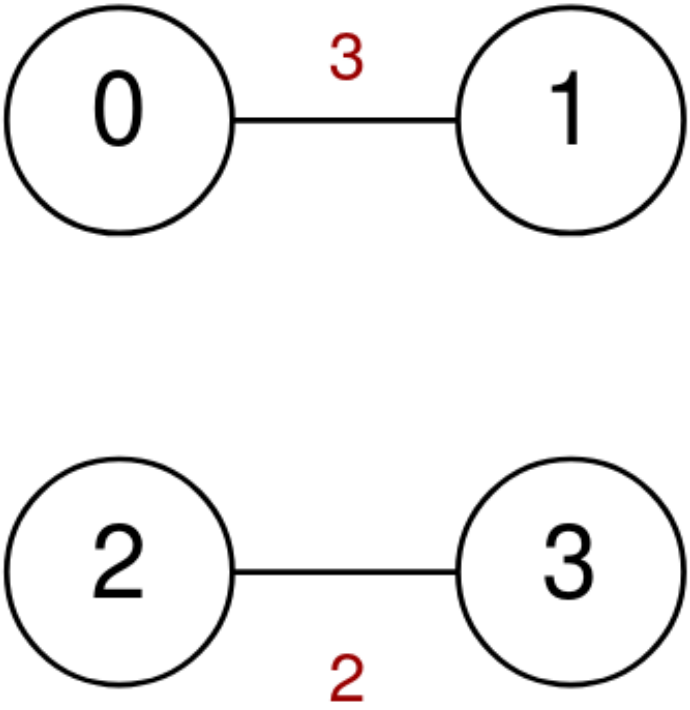
17

Explanation:

One possible trip is to go from 0 -> 1 -> 4 -> 3. The cost of this trip is 4 + 11 + 2 = 17. Another possible trip is to go from 4 -> 1 -> 2 -> 3. The cost of this trip is 11 + 3 + 3 = 17. It can be proven that 17 is the maximum possible cost of any valid trip.

Note that the trip 4 -> 1 -> 0 -> 1 is not allowed because you visit the city 1 twice.

Example 2:



Input:

n = 4, highways = [[0,1,3],[2,3,2]], k = 2

Output:

-1

Explanation:

There are no valid trips of length 2, so return -1.

Constraints:

2 <= n <= 15

1 <= highways.length <= 50

highways[i].length == 3

0 <= city1

i

, city2

i

<= n - 1

city1

i

!= city2

i

0 <= toll

i

<= 100

1 <= k <= 50

There are no duplicate highways.

Code Snippets

C++:

```
class Solution {
public:
int maximumCost(int n, vector<vector<int>>& highways, int k) {

}
};
```

Java:

```
class Solution {
public int maximumCost(int n, int[][] highways, int k) {

}
}
```

Python3:

```
class Solution:
def maximumCost(self, n: int, highways: List[List[int]], k: int) -> int:
```

Python:

```
class Solution(object):
def maximumCost(self, n, highways, k):
"""
:type n: int
:type highways: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} highways
* @param {number} k
* @return {number}
*/
var maximumCost = function(n, highways, k) {

};
```

TypeScript:

```
function maximumCost(n: number, highways: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximumCost(int n, int[][] highways, int k) {

}
}
```

C:

```
int maximumCost(int n, int** highways, int highwaysSize, int*
highwaysColSize, int k) {

}
```

Go:

```
func maximumCost(n int, highways [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximumCost(n: Int, highways: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumCost(_ n: Int, _ highways: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_cost(n: i32, highways: Vec<Vec<i32>>, k: i32) -> i32 {

}
```

```
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} highways
# @param {Integer} k
# @return {Integer}
def maximum_cost(n, highways, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $highways
* @param Integer $k
* @return Integer
*/
function maximumCost($n, $highways, $k) {

}
}
```

Dart:

```
class Solution {
int maximumCost(int n, List<List<int>> highways, int k) {

}
}
```

Scala:

```
object Solution {
def maximumCost(n: Int, highways: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_cost(n :: integer, highways :: [[integer]], k :: integer) ::
integer
def maximum_cost(n, highways, k) do

end
end
```

Erlang:

```
-spec maximum_cost(N :: integer(), Highways :: [[integer()]], K :: integer())
-> integer().
maximum_cost(N, Highways, K) ->
.
```

Racket:

```
(define/contract (maximum-cost n highways k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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 maximumCost(int n, vector<vector<int>>& highways, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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 maximumCost(int n, int[][] highways, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Cost of Trip With K Highways
Difficulty: Hard
Tags: array, graph, 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 maximumCost(self, n: int, highways: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumCost(self, n, highways, k):
"""
:type n: int
:type highways: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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} n
* @param {number[][]} highways
* @param {number} k
* @return {number}
*/
var maximumCost = function(n, highways, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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 maximumCost(n: number, highways: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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 MaximumCost(int n, int[][] highways, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Cost of Trip With K Highways
* Difficulty: Hard
* Tags: array, graph, 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 maximumCost(int n, int** highways, int highwaysSize, int*
highwaysColSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Cost of Trip With K Highways
// Difficulty: Hard
// Tags: array, graph, 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 maximumCost(n int, highways [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumCost(n: Int, highways: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumCost(_ n: Int, _ highways: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Cost of Trip With K Highways
// Difficulty: Hard
// Tags: array, graph, 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 maximum_cost(n: i32, highways: Vec<Vec<i32>>, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} highways
# @param {Integer} k
# @return {Integer}
def maximum_cost(n, highways, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $highways
* @param Integer $k
* @return Integer
*/
function maximumCost($n, $highways, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumCost(int n, List<List<int>> highways, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumCost(n: Int, highways: Array[Array[Int]], k: Int): Int = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_cost(n :: integer, highways :: [[integer]], k :: integer) ::
integer
def maximum_cost(n, highways, k) do

end
end
```

Erlang Solution:

```
-spec maximum_cost(N :: integer(), Highways :: [[integer()]], K :: integer())
-> integer().
maximum_cost(N, Highways, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-cost n highways k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```