Problem 1928: Minimum Cost to Reach Destination in Time

Problem Information

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

Problem Description

There is a country of

n

cities numbered from

0

to

n - 1

where

all the cities are connected

by bi-directional roads. The roads are represented as a 2D integer array

edges

where

edges[i] = [x

i

, y

i

, time

i

]

denotes a road between cities

x

i

and

y

i

that takes

time

i

minutes to travel. There may be multiple roads of differing travel times connecting the same two cities, but no road connects a city to itself.

Each time you pass through a city, you must pay a passing fee. This is represented as a

0-indexed

integer array

passingFees

of length

n

where

passingFees[j]

is the amount of dollars you must pay when you pass through city

j

.

In the beginning, you are at city

0

and want to reach city

n - 1

in

maxTime

minutes or less

. The

cost

of your journey is the

summation of passing fees

for each city that you passed through at some moment of your journey (

including

the source and destination cities).

Given

maxTime

,

edges

, and

passingFees

, return

the

minimum cost

to complete your journey, or

-1

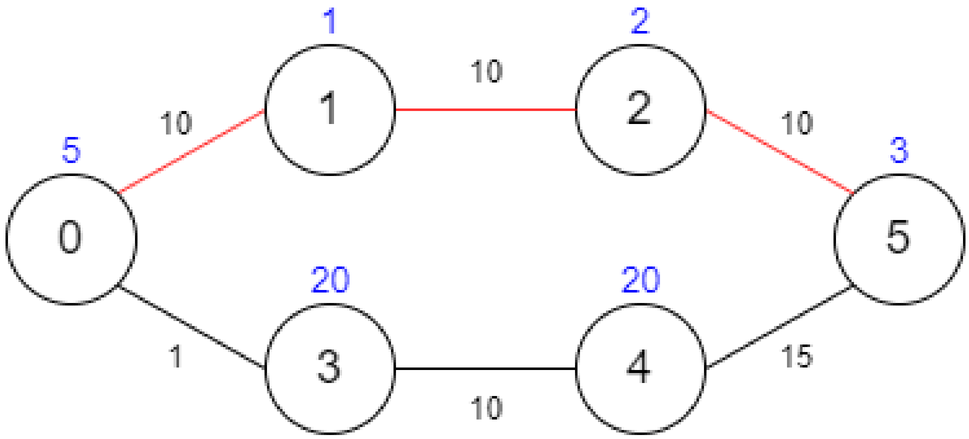
if you cannot complete it within

maxTime

minutes

.

Example 1:



Input:

maxTime = 30, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,15]], passingFees = [5,1,2,20,20,3]

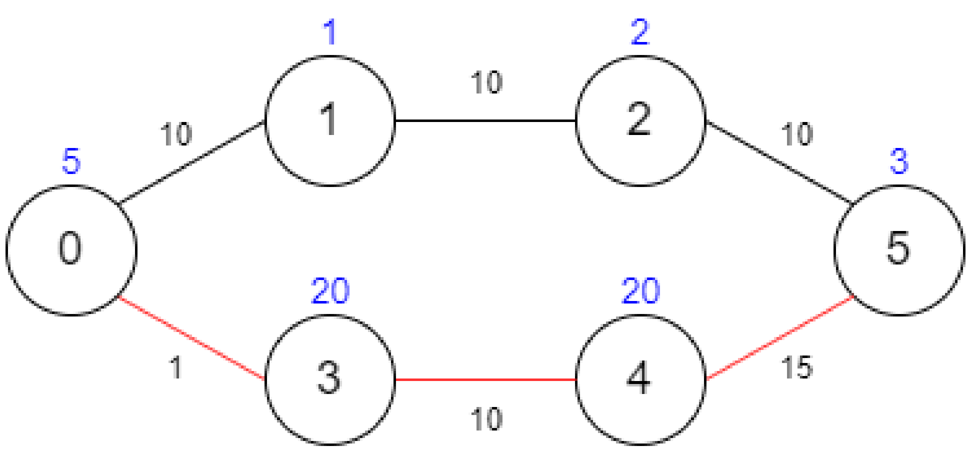
Output:

11

Explanation:

The path to take is 0 -> 1 -> 2 -> 5, which takes 30 minutes and has $11 worth of passing fees.

Example 2:



Input:

maxTime = 29, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,15]], passingFees = [5,1,2,20,20,3]

Output:

48

Explanation:

The path to take is 0 -> 3 -> 4 -> 5, which takes 26 minutes and has $48 worth of passing fees. You cannot take path 0 -> 1 -> 2 -> 5 since it would take too long.

Example 3:

Input:

maxTime = 25, edges = [[0,1,10],[1,2,10],[2,5,10],[0,3,1],[3,4,10],[4,5,15]], passingFees = [5,1,2,20,20,3]

Output:

-1

Explanation:

There is no way to reach city 5 from city 0 within 25 minutes.

Constraints:

1 <= maxTime <= 1000

n == passingFees.length

2 <= n <= 1000

n - 1 <= edges.length <= 1000

0 <= x

i

, y

i

<= n - 1

1 <= time

i

<= 1000

1 <= passingFees[j] <= 1000

The graph may contain multiple edges between two nodes.

The graph does not contain self loops.

Code Snippets

C++:

```
class Solution {
public:
int minCost(int maxTime, vector<vector<int>>& edges, vector<int>&
passingFees) {

}
};
```

Java:

```
class Solution {
public int minCost(int maxTime, int[][] edges, int[] passingFees) {

}
}
```

Python3:

```
class Solution:
def minCost(self, maxTime: int, edges: List[List[int]], passingFees:
List[int]) -> int:
```

Python:

```
class Solution(object):
def minCost(self, maxTime, edges, passingFees):
"""
:type maxTime: int
:type edges: List[List[int]]
:type passingFees: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} maxTime
* @param {number[][]} edges
* @param {number[]} passingFees
* @return {number}
*/
var minCost = function(maxTime, edges, passingFees) {

};
```

TypeScript:

```
function minCost(maxTime: number, edges: number[][], passingFees: number[]):
number {

};
```

C#:

```
public class Solution {
public int MinCost(int maxTime, int[][] edges, int[] passingFees) {

}
}
```

C:

```
int minCost(int maxTime, int** edges, int edgesSize, int* edgesColSize, int*
passingFees, int passingFeesSize) {

}
```

Go:

```
func minCost(maxTime int, edges [][]int, passingFees []int) int {

}
```

Kotlin:

```
class Solution {
fun minCost(maxTime: Int, edges: Array<IntArray>, passingFees: IntArray): Int
{

}
}
```

Swift:

```
class Solution {
func minCost(_ maxTime: Int, _ edges: [[Int]], _ passingFees: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost(max_time: i32, edges: Vec<Vec<i32>>, passing_fees: Vec<i32>)
-> i32 {

}
}
```

Ruby:

```
# @param {Integer} max_time
# @param {Integer[][]} edges
```

```
# @param {Integer[]} passing_fees
# @return {Integer}
def min_cost(max_time, edges, passing_fees)

end
```

PHP:

```
class Solution {

/**
* @param Integer $maxTime
* @param Integer[][] $edges
* @param Integer[] $passingFees
* @return Integer
*/
function minCost($maxTime, $edges, $passingFees) {

}
}
```

Dart:

```
class Solution {
int minCost(int maxTime, List<List<int>> edges, List<int> passingFees) {

}
}
```

Scala:

```
object Solution {
def minCost(maxTime: Int, edges: Array[Array[Int]], passingFees: Array[Int]):
Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost(max_time :: integer, edges :: [[integer]], passing_fees ::
```

```
[integer]) :: integer
def min_cost(max_time, edges, passing_fees) do

end
end
```

Erlang:

```
-spec min_cost(MaxTime :: integer(), Edges :: [[integer()]], PassingFees ::
[integer()]) -> integer().
min_cost(MaxTime, Edges, PassingFees) ->
.
```

Racket:

```
(define/contract (min-cost maxTime edges passingFees)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Reach Destination in Time
* 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 minCost(int maxTime, vector<vector<int>>& edges, vector<int>&
passingFees) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Reach Destination in Time
* 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 minCost(int maxTime, int[][] edges, int[] passingFees) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Reach Destination in Time
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 minCost(self, maxTime: int, edges: List[List[int]], passingFees:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minCost(self, maxTime, edges, passingFees):
"""
:type maxTime: int
```

```
:type edges: List[List[int]]
:type passingFees: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Reach Destination in Time
* 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} maxTime
* @param {number[][]} edges
* @param {number[]} passingFees
* @return {number}
*/
var minCost = function(maxTime, edges, passingFees) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Reach Destination in Time
* 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 minCost(maxTime: number, edges: number[][], passingFees: number[]):
number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Reach Destination in Time
* 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 MinCost(int maxTime, int[][] edges, int[] passingFees) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Reach Destination in Time
* 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 minCost(int maxTime, int** edges, int edgesSize, int* edgesColSize, int*
passingFees, int passingFeesSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Reach Destination in Time
// 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 minCost(maxTime int, edges [][]int, passingFees []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCost(maxTime: Int, edges: Array<IntArray>, passingFees: IntArray): Int
{

}
}
```

Swift Solution:

```
class Solution {
func minCost(_ maxTime: Int, _ edges: [[Int]], _ passingFees: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Reach Destination in Time
// 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 min_cost(max_time: i32, edges: Vec<Vec<i32>>, passing_fees: Vec<i32>)
-> i32 {
```

```
}
}
```

Ruby Solution:

```ruby
# @param {Integer} max_time
# @param {Integer[][]} edges
# @param {Integer[]} passing_fees
# @return {Integer}
def min_cost(max_time, edges, passing_fees)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer $maxTime
* @param Integer[][] $edges
* @param Integer[] $passingFees
* @return Integer
*/
function minCost($maxTime, $edges, $passingFees) {

}
}
```

Dart Solution:

```dart
class Solution {
int minCost(int maxTime, List<List<int>> edges, List<int> passingFees) {

}
}
```

Scala Solution:

```scala
object Solution {
def minCost(maxTime: Int, edges: Array[Array[Int]], passingFees: Array[Int]):
```

```
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost(max_time :: integer, edges :: [[integer]], passing_fees ::
[integer]) :: integer
def min_cost(max_time, edges, passing_fees) do

end
end
```

Erlang Solution:

```
-spec min_cost(MaxTime :: integer(), Edges :: [[integer()]], PassingFees ::
[integer()]) -> integer().
min_cost(MaxTime, Edges, PassingFees) ->
.
```

Racket Solution:

```
(define/contract (min-cost maxTime edges passingFees)
(-> exact-integer? (listof (listof exact-integer?)) (listof exact-integer?)
exact-integer?)
)
```