Problem 3543: Maximum Weighted K-Edge Path

Problem Information

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

Problem Description

You are given an integer

n

and a

Directed Acyclic Graph (DAG)

with

n

nodes labeled from 0 to

n - 1

. This is represented by a 2D array

edges

, where

edges[i] = [u

i

, v_i, w_i] indicates a directed edge from node u_i to v_i with weight w_i.

You are also given two integers, k and t.

Your task is to determine the

maximum

possible sum of edge weights for any path in the graph such that:

The path contains

exactly

k

edges.

The total sum of edge weights in the path is

strictly

less than

t

.

Return the

maximum

possible sum of weights for such a path. If no such path exists, return

-1

.

Example 1:

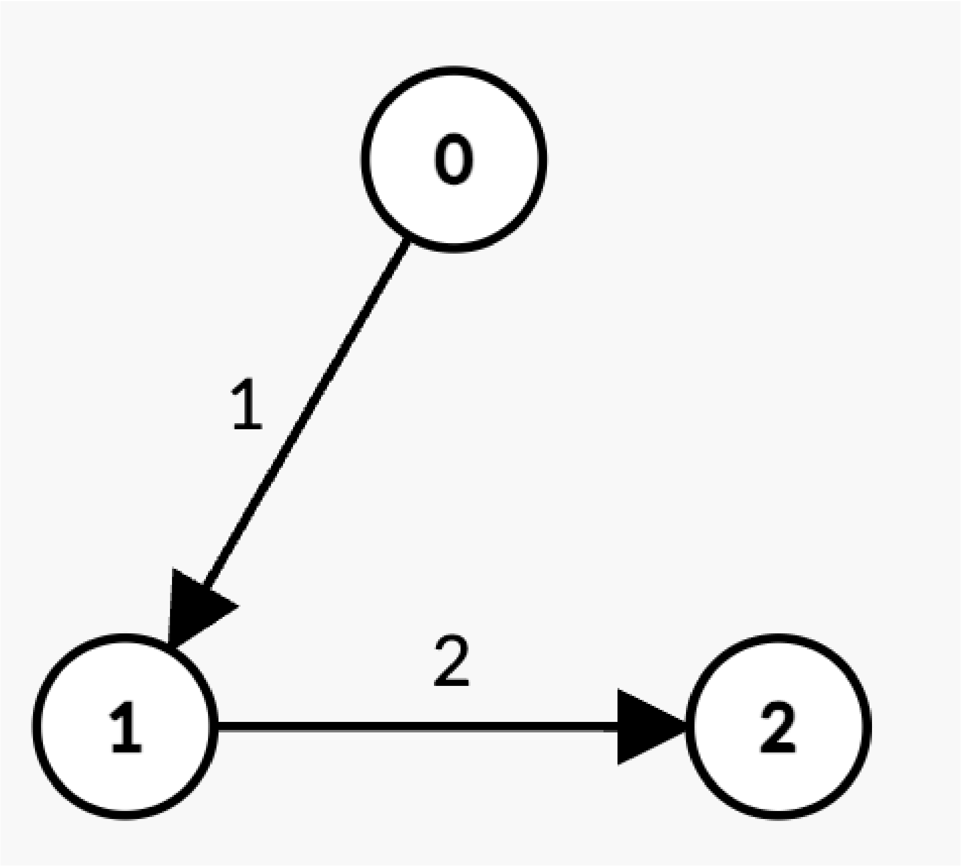
Input:

n = 3, edges = [[0,1,1],[1,2,2]], k = 2, t = 4

Output:

3

Explanation:



The only path with

k = 2

edges is

0 -> 1 -> 2

with weight

1 + 2 = 3 < t

.

Thus, the maximum possible sum of weights less than

t

is 3.

Example 2:

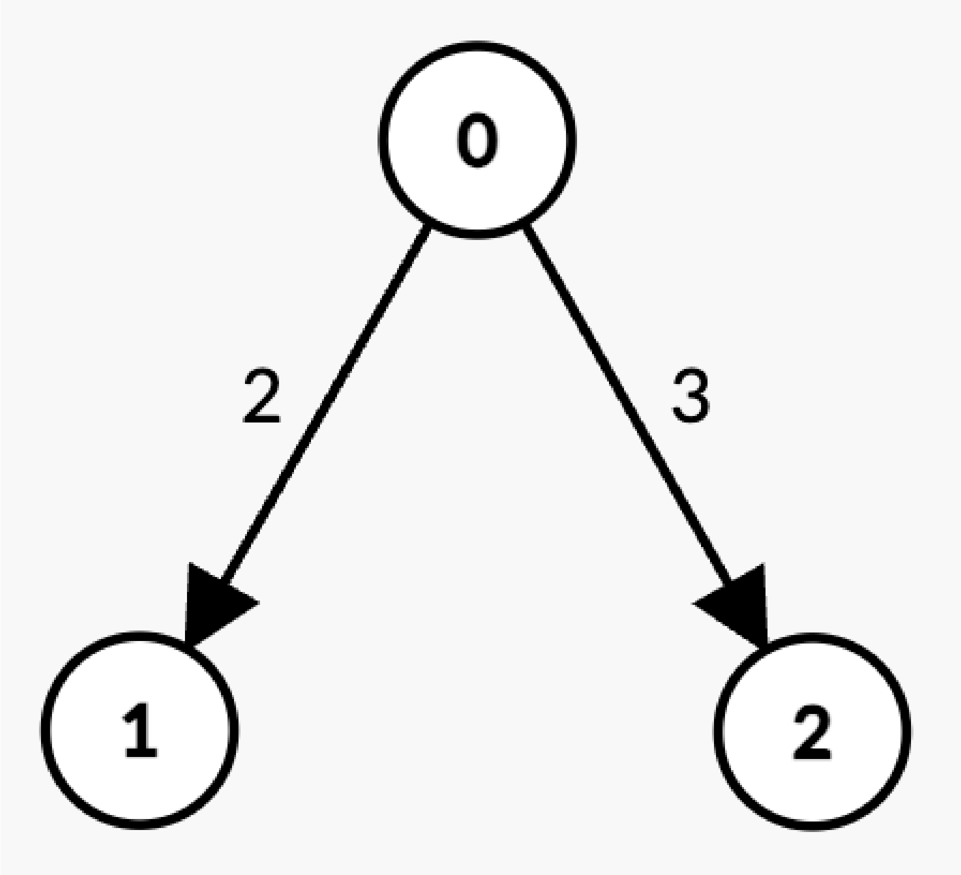
Input:

n = 3, edges = [[0,1,2],[0,2,3]], k = 1, t = 3

Output:

2

Explanation:



There are two paths with

k = 1

edge:

0 -> 1

with weight

2 < t

.

0 -> 2

with weight

3 = t

, which is not strictly less than

t

.

Thus, the maximum possible sum of weights less than

t

is 2.

Example 3:

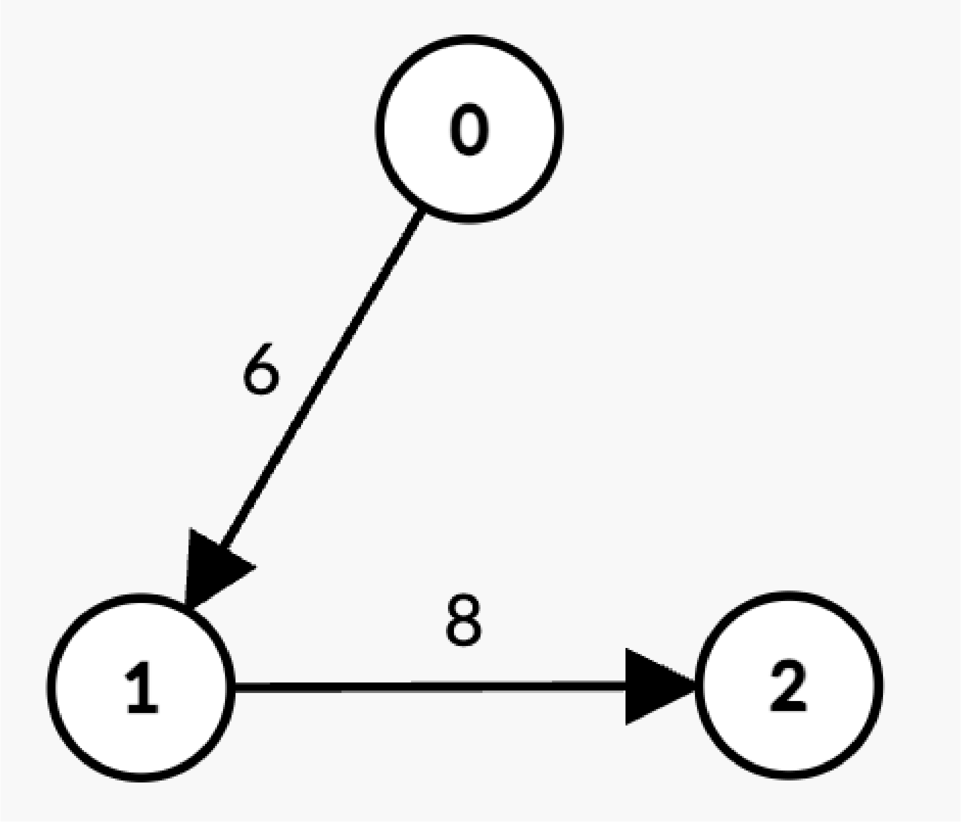
Input:

n = 3, edges = [[0,1,6],[1,2,8]], k = 1, t = 6

Output:

-1

Explanation:



There are two paths with k = 1 edge:

0 -> 1

with weight

6 = t

, which is not strictly less than

t

.

1 -> 2

with weight

8 > t

, which is not strictly less than

t

.

Since there is no path with sum of weights strictly less than

t

, the answer is -1.

Constraints:

1 <= n <= 300

0 <= edges.length <= 300

edges[i] = [u

i

, v

i

, w

i

]

0 <= u

i

, v

i

< n

u

i

!= v

i

1 <= w

i

<= 10

0 <= k <= 300

1 <= t <= 600

The input graph is

guaranteed

to be a

DAG

.

There are no duplicate edges.

Code Snippets

C++:

```
class Solution {
public:
int maxWeight(int n, vector<vector<int>>& edges, int k, int t) {

}
};
```

Java:

```
class Solution {
public int maxWeight(int n, int[][] edges, int k, int t) {

}
}
```

Python3:

```
class Solution:
def maxWeight(self, n: int, edges: List[List[int]], k: int, t: int) -> int:
```

Python:

```
class Solution(object):
def maxWeight(self, n, edges, k, t):
"""
:type n: int
:type edges: List[List[int]]
:type k: int
:type t: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} edges
* @param {number} k
* @param {number} t
* @return {number}
*/
var maxWeight = function(n, edges, k, t) {

};
```

TypeScript:

```
function maxWeight(n: number, edges: number[][], k: number, t: number):
number {

};
```

C#:

```
public class Solution {
public int MaxWeight(int n, int[][] edges, int k, int t) {

}
}
```

C:

```
int maxWeight(int n, int** edges, int edgesSize, int* edgesColSize, int k,
int t) {

}
```

Go:

```
func maxWeight(n int, edges [][]int, k int, t int) int {

}
```

Kotlin:

```
class Solution {
fun maxWeight(n: Int, edges: Array<IntArray>, k: Int, t: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxWeight(_ n: Int, _ edges: [[Int]], _ k: Int, _ t: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_weight(n: i32, edges: Vec<Vec<i32>>, k: i32, t: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} k
# @param {Integer} t
# @return {Integer}
def max_weight(n, edges, k, t)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $k
* @param Integer $t
* @return Integer
*/
function maxWeight($n, $edges, $k, $t) {

}
}
```

Dart:

```
class Solution {
int maxWeight(int n, List<List<int>> edges, int k, int t) {

}
}
```

Scala:

```
object Solution {
def maxWeight(n: Int, edges: Array[Array[Int]], k: Int, t: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_weight(n :: integer, edges :: [[integer]], k :: integer, t ::
integer) :: integer
def max_weight(n, edges, k, t) do

end
end
```

Erlang:

```
-spec max_weight(N :: integer(), Edges :: [[integer()]], K :: integer(), T ::
integer()) -> integer().
max_weight(N, Edges, K, T) ->
.
```

Racket:

```
(define/contract (max-weight n edges k t)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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 maxWeight(int n, vector<vector<int>>& edges, int k, int t) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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 maxWeight(int n, int[][] edges, int k, int t) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Weighted K-Edge Path
Difficulty: Medium
Tags: array, graph, dp, hash
```

```
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 maxWeight(self, n: int, edges: List[List[int]], k: int, t: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxWeight(self, n, edges, k, t):
"""
:type n: int
:type edges: List[List[int]]
:type k: int
:type t: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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[][]} edges
* @param {number} k
* @param {number} t
* @return {number}
```

```
*/
var maxWeight = function(n, edges, k, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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 maxWeight(n: number, edges: number[][], k: number, t: number):
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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 MaxWeight(int n, int[][] edges, int k, int t) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Weighted K-Edge Path
* Difficulty: Medium
* Tags: array, graph, dp, hash
*
* 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 maxWeight(int n, int** edges, int edgesSize, int* edgesColSize, int k,
int t) {

}
```

Go Solution:

```
// Problem: Maximum Weighted K-Edge Path
// Difficulty: Medium
// Tags: array, graph, dp, hash
//
// 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 maxWeight(n int, edges [][]int, k int, t int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxWeight(n: Int, edges: Array<IntArray>, k: Int, t: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxWeight(_ n: Int, _ edges: [[Int]], _ k: Int, _ t: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Weighted K-Edge Path
// Difficulty: Medium
// Tags: array, graph, dp, hash
//
// 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 max_weight(n: i32, edges: Vec<Vec<i32>>, k: i32, t: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} k
# @param {Integer} t
# @return {Integer}
def max_weight(n, edges, k, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $k
* @param Integer $t
* @return Integer
*/
function maxWeight($n, $edges, $k, $t) {

}
}
```

Dart Solution:

```
class Solution {
int maxWeight(int n, List<List<int>> edges, int k, int t) {

}
}
```

Scala Solution:

```
object Solution {
def maxWeight(n: Int, edges: Array[Array[Int]], k: Int, t: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_weight(n :: integer, edges :: [[integer]], k :: integer, t ::
integer) :: integer
def max_weight(n, edges, k, t) do

end
end
```

Erlang Solution:

```
-spec max_weight(N :: integer(), Edges :: [[integer()]], K :: integer(), T ::
integer()) -> integer().
max_weight(N, Edges, K, T) ->
.
```

Racket Solution:

```
(define/contract (max-weight n edges k t)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer?)
)
```