Problem 2065: Maximum Path Quality of a Graph

Problem Information

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

Problem Description

There is an

undirected

graph with

n

nodes numbered from

0

to

n - 1

(

inclusive

). You are given a

0-indexed

integer array

values

where

values[i]

is the

value

of the

i

th

node. You are also given a

0-indexed

2D integer array

edges

, where each

edges[j] = [u

j

, v

j

, time

j

]

indicates that there is an undirected edge between the nodes

u

j

and

v

j

,

and it takes

time

j

seconds to travel between the two nodes. Finally, you are given an integer

maxTime

.

A

valid

path

in the graph is any path that starts at node

0

, ends at node

0

, and takes

at most

maxTime

seconds to complete. You may visit the same node multiple times. The

quality

of a valid path is the

sum

of the values of the

unique nodes

visited in the path (each node's value is added

at most once

to the sum).

Return

the

maximum

quality of a valid path

.

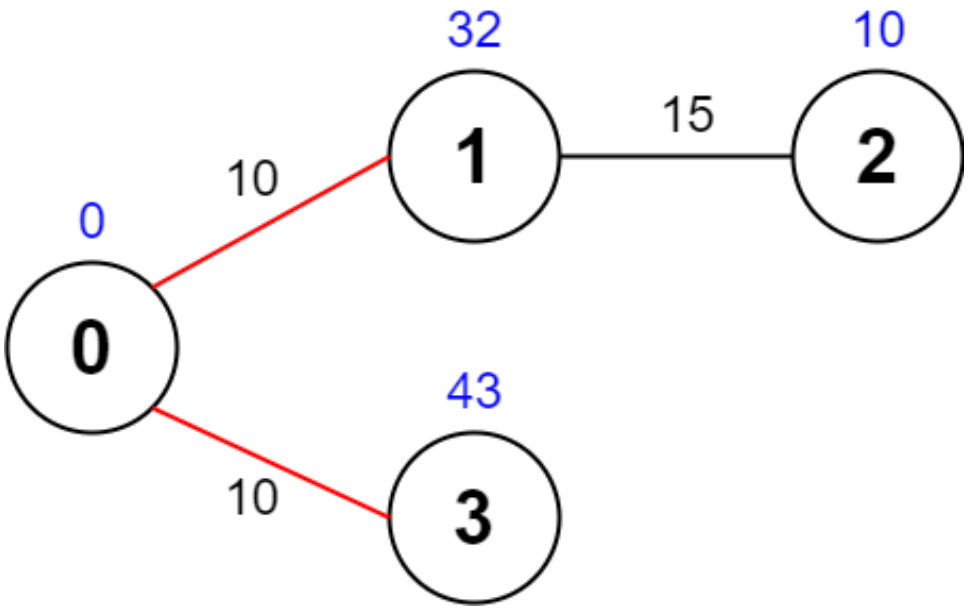
Note:

There are

at most four

edges connected to each node.

Example 1:



Input:

values = [0,32,10,43], edges = [[0,1,10],[1,2,15],[0,3,10]], maxTime = 49

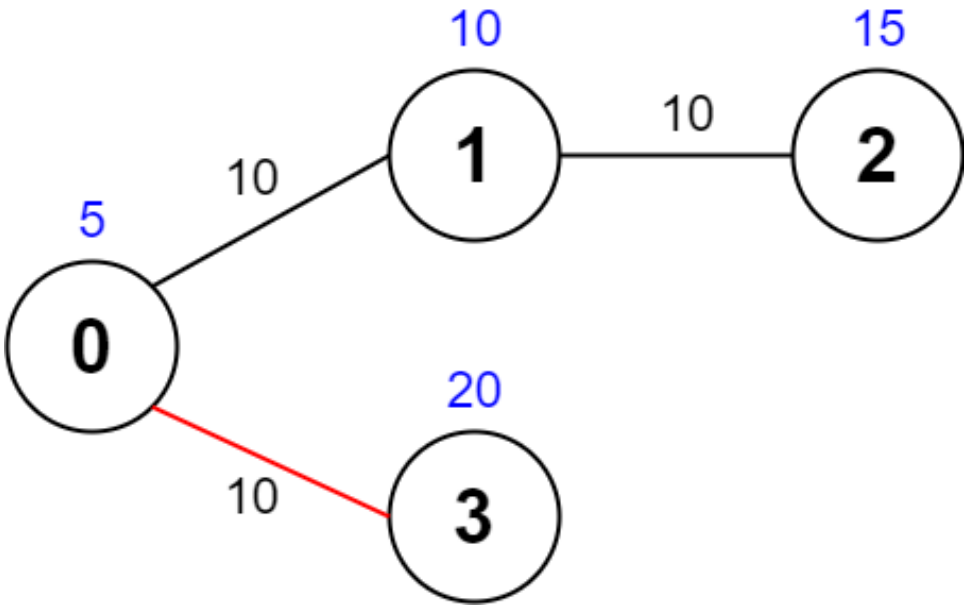
Output:

75

Explanation:

One possible path is 0 -> 1 -> 0 -> 3 -> 0. The total time taken is 10 + 10 + 10 + 10 = 40 <= 49. The nodes visited are 0, 1, and 3, giving a maximal path quality of 0 + 32 + 43 = 75.

Example 2:



Input:

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

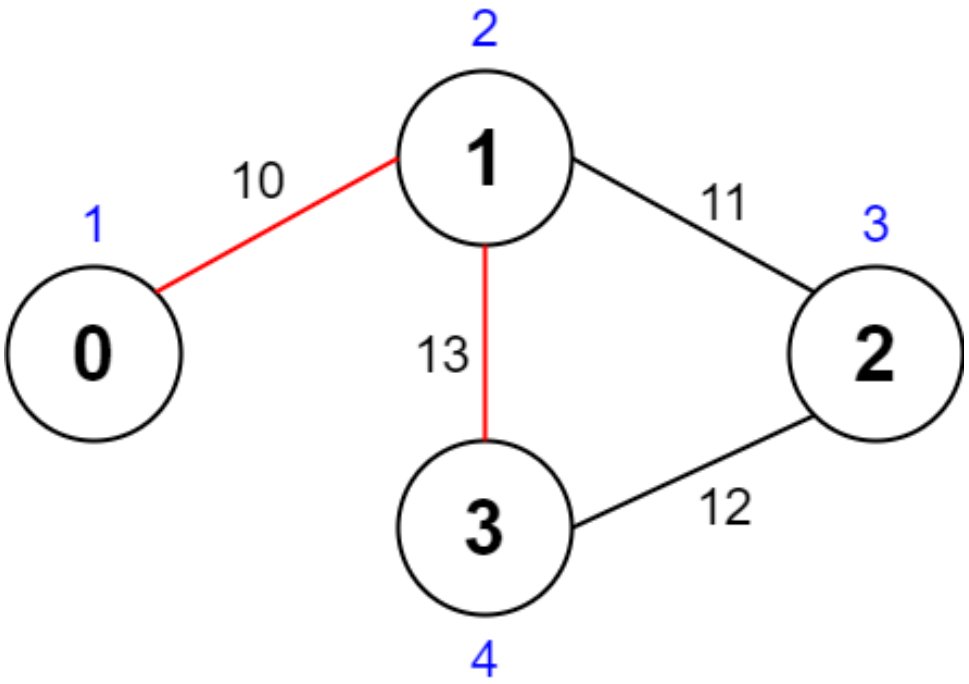
Output:

25

Explanation:

One possible path is 0 -> 3 -> 0. The total time taken is 10 + 10 = 20 <= 30. The nodes visited are 0 and 3, giving a maximal path quality of 5 + 20 = 25.

Example 3:



Input:

values = [1,2,3,4], edges = [[0,1,10],[1,2,11],[2,3,12],[1,3,13]], maxTime = 50

Output:

7

Explanation:

One possible path is 0 -> 1 -> 3 -> 1 -> 0. The total time taken is 10 + 13 + 13 + 10 = 46 <= 50. The nodes visited are 0, 1, and 3, giving a maximal path quality of 1 + 2 + 4 = 7.

Constraints:

n == values.length

1 <= n <= 1000

0 <= values[i] <= 10

0 <= edges.length <= 2000

edges[j].length == 3

0 <= u

j

< v

j

<= n - 1

10 <= time

j

, maxTime <= 100

All the pairs

[u

j

, v

j

]

are

unique

.

There are

at most four

edges connected to each node.

The graph may not be connected.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int maximalPathQuality(int[] values, int[][] edges, int maxTime) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def maximalPathQuality(self, values, edges, maxTime):
"""
:type values: List[int]
:type edges: List[List[int]]
:type maxTime: int
```

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maximalPathQuality(values: number[], edges: number[][], maxTime:
number): number {

};
```

C#:

```
public class Solution {
public int MaximalPathQuality(int[] values, int[][] edges, int maxTime) {

}
}
```

C:

```
int maximalPathQuality(int* values, int valuesSize, int** edges, int
edgesSize, int* edgesColSize, int maxTime) {

}
```

Go:

```
func maximalPathQuality(values []int, edges [][]int, maxTime int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $values
```

```
* @param Integer[][] $edges
* @param Integer $maxTime
* @return Integer
*/
function maximalPathQuality($values, $edges, $maxTime) {

}
}
```

Dart:

```
class Solution {
int maximalPathQuality(List<int> values, List<List<int>> edges, int maxTime)
{

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec maximal_path_quality(Values :: [integer()], Edges :: [[integer()]],
MaxTime :: integer()) -> integer().
maximal_path_quality(Values, Edges, MaxTime) ->
```

```
.
```

Racket:

```
(define/contract (maximal-path-quality values edges maxTime)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
* Tags: array, graph
*
* 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 maximalPathQuality(vector<int>& values, vector<vector<int>>& edges, int
maxTime) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
* Tags: array, graph
*
* 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 maximalPathQuality(int[] values, int[][] edges, int maxTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Path Quality of a Graph
Difficulty: Hard
Tags: array, graph

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 maximalPathQuality(self, values: List[int], edges: List[List[int]],
maxTime: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
* Tags: array, graph
*
* 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[]} values
* @param {number[][]} edges
* @param {number} maxTime
* @return {number}
*/
var maximalPathQuality = function(values, edges, maxTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
* Tags: array, graph
*
* 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 maximalPathQuality(values: number[], edges: number[][], maxTime:
number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
```

```
* Tags: array, graph
*
* 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 MaximalPathQuality(int[] values, int[][] edges, int maxTime) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Path Quality of a Graph
* Difficulty: Hard
* Tags: array, graph
*
* 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 maximalPathQuality(int* values, int valuesSize, int** edges, int
edgesSize, int* edgesColSize, int maxTime) {

}
```

Go Solution:

```
// Problem: Maximum Path Quality of a Graph
// Difficulty: Hard
// Tags: array, graph
//
// 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 maximalPathQuality(values []int, edges [][]int, maxTime int) int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Path Quality of a Graph
// Difficulty: Hard
// Tags: array, graph
//
// 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 maximal_path_quality(values: Vec<i32>, edges: Vec<Vec<i32>>, max_time:
i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int maximalPathQuality(List<int> values, List<List<int>> edges, int maxTime)
{

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec maximal_path_quality(Values :: [integer()], Edges :: [[integer()]],
MaxTime :: integer()) -> integer().
maximal_path_quality(Values, Edges, MaxTime) ->
.
```

Racket Solution:

```
(define/contract (maximal-path-quality values edges maxTime)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```