# Problem 2714: Find Shortest Path with K Hops

## Problem Information

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

## Problem Description

You are given a positive integer

n

which is the number of nodes of a

0-indexed undirected weighted connected

graph and a

0-indexed

2D array

edges

where

edges[i] = [u

i

, v

i

, w

i

]

indicates that there is an edge between nodes

u

i

and

v

i

with weight

w

i

.

You are also given two nodes

s

and

d

, and a positive integer

k

, your task is to find the

shortest

path from

s

to

d

, but you can hop over

at most

k

edges. In other words, make the weight of

at most

k

edges

0

and then find the

shortest

path from

s

to

d

.

Return

the length of the

shortest

path from

s

to

d

with the given condition

.

Example 1:

Input:

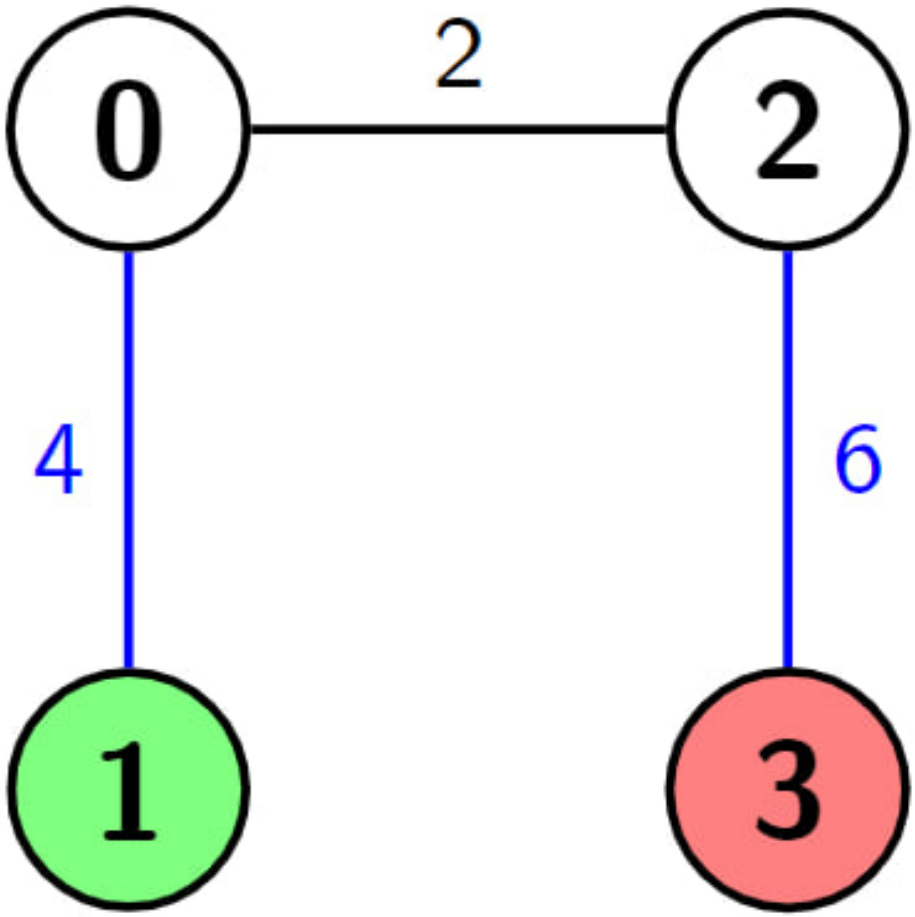
n = 4, edges = [[0,1,4],[0,2,2],[2,3,6]], s = 1, d = 3, k = 2

Output:

2

Explanation:

In this example there is only one path from node 1 (the green node) to node 3 (the red node), which is (1->0->2->3) and the length of it is 4 + 2 + 6 = 12. Now we can make weight of two edges 0, we make weight of the blue edges 0, then we have 0 + 2 + 0 = 2. It can be shown that 2 is the minimum length of a path we can achieve with the given condition.




Example 2:

Input:

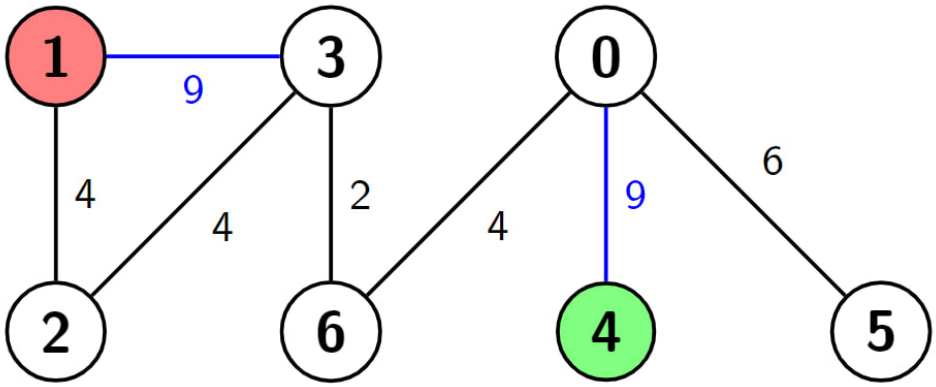
n = 7, edges = [[3,1,9],[3,2,4],[4,0,9],[0,5,6],[3,6,2],[6,0,4],[1,2,4]], s = 4, d = 1, k = 2

Output:

6

Explanation:

In this example there are 2 paths from node 4 (the green node) to node 1 (the red node), which are (4->0->6->3->2->1) and (4->0->6->3->1). The first one has the length 9 + 4 + 2 + 4 + 4 = 23, and the second one has the length 9 + 4 + 2 + 9 = 24. Now if we make weight of the blue edges 0, we get the shortest path with the length 0 + 4 + 2 + 0 = 6. It can be shown that 6 is the minimum length of a path we can achieve with the given condition.




Example 3:

Input:

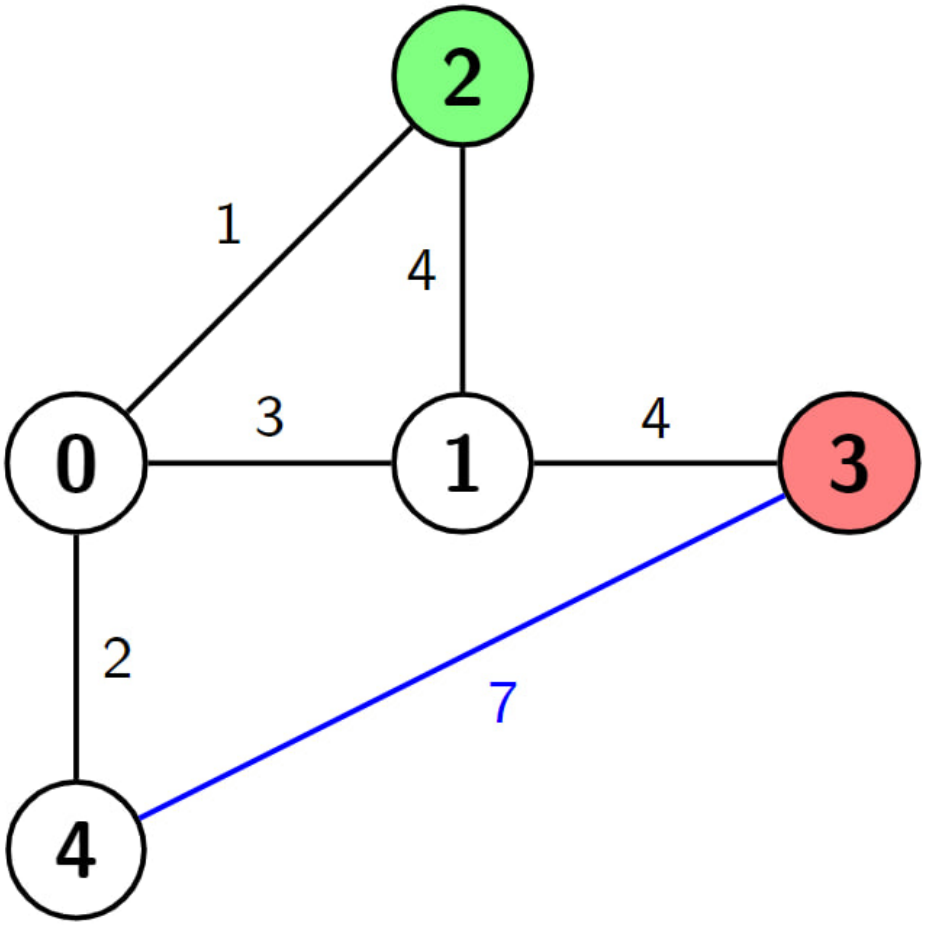
n = 5, edges = [[0,4,2],[0,1,3],[0,2,1],[2,1,4],[1,3,4],[3,4,7]], s = 2, d = 3, k = 1

Output:

3

Explanation:

In this example there are 4 paths from node 2 (the green node) to node 3 (the red node), which are (2->1->3), (2->0->1->3), (2->1->0->4->3) and (2->0->4->3). The first two have the length 4 + 4 = 1 + 3 + 4 = 8, the third one has the length 4 + 3 + 2 + 7 = 16 and the last one has the length 1 + 2 + 7 = 10. Now if we make weight of the blue edge 0, we get the shortest path with the length 1 + 2 + 0 = 3. It can be shown that 3 is the minimum length of a path we can achieve with the given condition.




Constraints:

2 <= n <= 500

n - 1 <= edges.length <= min(10

4

, n * (n - 1) / 2)

edges[i].length = 3

0 <= edges[i][0], edges[i][1] <= n - 1

1 <= edges[i][2] <= 10

6

0 <= s, d, k <= n - 1

s != d

The input is generated such that the graph is

connected

and has

no

repeated edges

or

self-loops

## Code Snippets

**C++:**

```
class Solution {
public:
int shortestPathWithHops(int n, vector<vector<int>>& edges, int s, int d, int
k) {

}
};
```

**Java:**

```
class Solution {
public int shortestPathWithHops(int n, int[][] edges, int s, int d, int k) {

}
}
```

**Python3:**

```
class Solution:
def shortestPathWithHops(self, n: int, edges: List[List[int]], s: int, d:
int, k: int) -> int:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function shortestPathWithHops(n: number, edges: number[][], s: number, d:
number, k: number): number {

};
```

**C#:**

```
public class Solution {
public int ShortestPathWithHops(int n, int[][] edges, int s, int d, int k) {
```

```
}
}
```

**C:**

```
int shortestPathWithHops(int n, int** edges, int edgesSize, int*
edgesColSize, int s, int d, int k) {

}
```

**Go:**

```
func shortestPathWithHops(n int, edges [][]int, s int, d int, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun shortestPathWithHops(n: Int, edges: Array<IntArray>, s: Int, d: Int, k:
Int): Int {

}
}
```

**Swift:**

```
class Solution {
func shortestPathWithHops(_ n: Int, _ edges: [[Int]], _ s: Int, _ d: Int, _
k: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn shortest_path_with_hops(n: i32, edges: Vec<Vec<i32>>, s: i32, d: i32,
k: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} s
# @param {Integer} d
# @param {Integer} k
# @return {Integer}
def shortest_path_with_hops(n, edges, s, d, k)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $s
* @param Integer $d
* @param Integer $k
* @return Integer
*/
function shortestPathWithHops($n, $edges, $s, $d, $k) {

}
}
```

**Dart:**

```
class Solution {
int shortestPathWithHops(int n, List<List<int>> edges, int s, int d, int k) {

}
}
```

**Scala:**

```
object Solution {
def shortestPathWithHops(n: Int, edges: Array[Array[Int]], s: Int, d: Int, k:
Int): Int = {
```

```
}
}
```

**Elixir:**

```
defmodule Solution do
@spec shortest_path_with_hops(n :: integer, edges :: [[integer]], s ::
integer, d :: integer, k :: integer) :: integer
def shortest_path_with_hops(n, edges, s, d, k) do

end
end
```

**Erlang:**

```
-spec shortest_path_with_hops(N :: integer(), Edges :: [[integer()]], S ::
integer(), D :: integer(), K :: integer()) -> integer().
shortest_path_with_hops(N, Edges, S, D, K) ->
.
```

**Racket:**

```
(define/contract (shortest-path-with-hops n edges s d k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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 shortestPathWithHops(int n, vector<vector<int>>& edges, int s, int d, int
k) {

}
};
```

### Java Solution:

```
/**
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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 shortestPathWithHops(int n, int[][] edges, int s, int d, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find Shortest Path with K Hops
Difficulty: Hard
Tags: array, graph, queue, heap

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 shortestPathWithHops(self, n: int, edges: List[List[int]], s: int, d:
```

```
int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

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

**JavaScript Solution:**

```
/**
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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} n
* @param {number[][]} edges
* @param {number} s
* @param {number} d
* @param {number} k
* @return {number}
*/
var shortestPathWithHops = function(n, edges, s, d, k) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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 shortestPathWithHops(n: number, edges: number[][], s: number, d:
number, k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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 ShortestPathWithHops(int n, int[][] edges, int s, int d, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Find Shortest Path with K Hops
* Difficulty: Hard
* Tags: array, graph, queue, heap
*
* 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 shortestPathWithHops(int n, int** edges, int edgesSize, int*
edgesColSize, int s, int d, int k) {

}
```

### Go Solution:

```
// Problem: Find Shortest Path with K Hops
// Difficulty: Hard
// Tags: array, graph, queue, heap
//
// 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 shortestPathWithHops(n int, edges [][]int, s int, d int, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun shortestPathWithHops(n: Int, edges: Array<IntArray>, s: Int, d: Int, k:
Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func shortestPathWithHops(_ n: Int, _ edges: [[Int]], _ s: Int, _ d: Int, _
k: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Find Shortest Path with K Hops
// Difficulty: Hard
// Tags: array, graph, queue, heap
//
// 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 shortest_path_with_hops(n: i32, edges: Vec<Vec<i32>>, s: i32, d: i32,
k: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} edges
# @param {Integer} s
# @param {Integer} d
# @param {Integer} k
# @return {Integer}
def shortest_path_with_hops(n, edges, s, d, k)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $edges
* @param Integer $s
* @param Integer $d
* @param Integer $k
* @return Integer
*/
function shortestPathWithHops($n, $edges, $s, $d, $k) {

}
```

```
}
```

**Dart Solution:**

```
class Solution {
int shortestPathWithHops(int n, List<List<int>> edges, int s, int d, int k) {

}
}
```

**Scala Solution:**

```
object Solution {
def shortestPathWithHops(n: Int, edges: Array[Array[Int]], s: Int, d: Int, k:
Int): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec shortest_path_with_hops(n :: integer, edges :: [[integer]], s ::
integer, d :: integer, k :: integer) :: integer
def shortest_path_with_hops(n, edges, s, d, k) do

end
end
```

**Erlang Solution:**

```
-spec shortest_path_with_hops(N :: integer(), Edges :: [[integer()]], S ::
integer(), D :: integer(), K :: integer()) -> integer().
shortest_path_with_hops(N, Edges, S, D, K) ->
.
```

**Racket Solution:**

```
(define/contract (shortest-path-with-hops n edges s d k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
```

```
)
```