Problem 787: Cheapest Flights Within K Stops

Problem Information

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

Problem Description

There are

n

cities connected by some number of flights. You are given an array

flights

where

flights[i] = [from

i

, to

i

, price

i

]

indicates that there is a flight from city

from

i

to city

to

i

with cost

price

i

.

You are also given three integers

src

,

dst

, and

k

, return

the cheapest price

from

src

to

dst

with at most

k

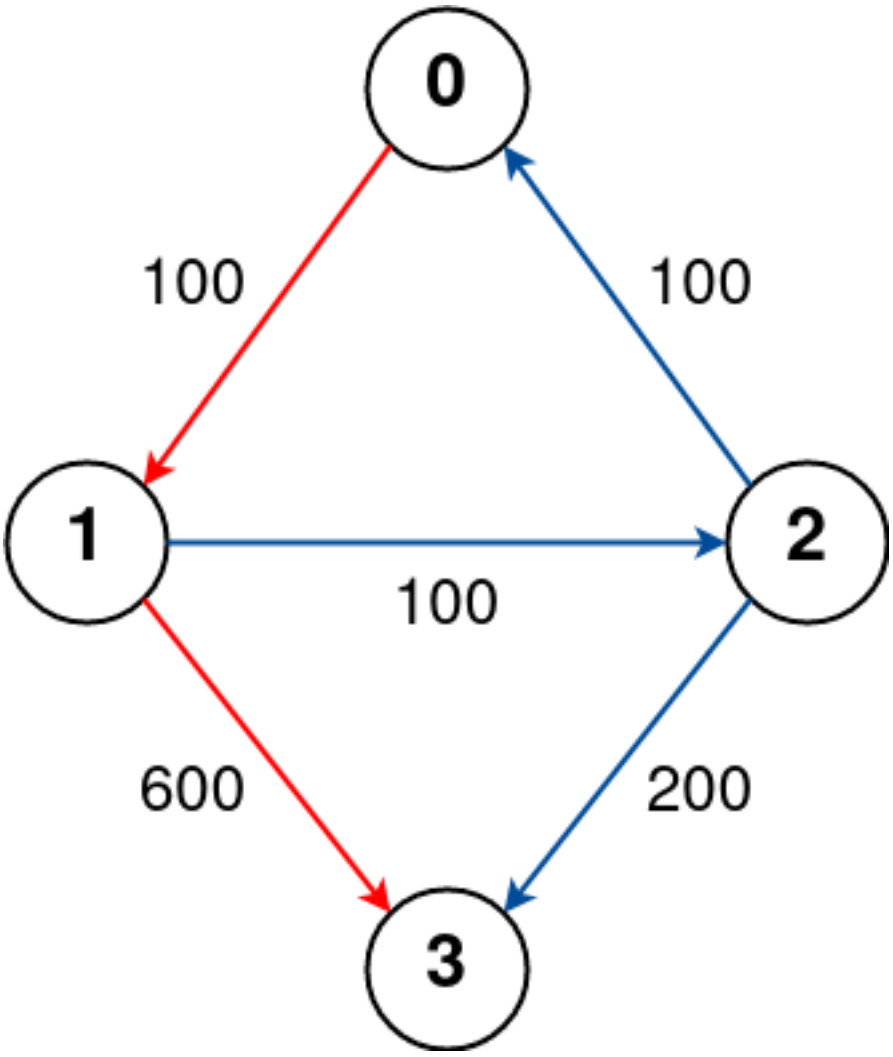
stops.

If there is no such route, return

-1

.

Example 1:



Input:

n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1

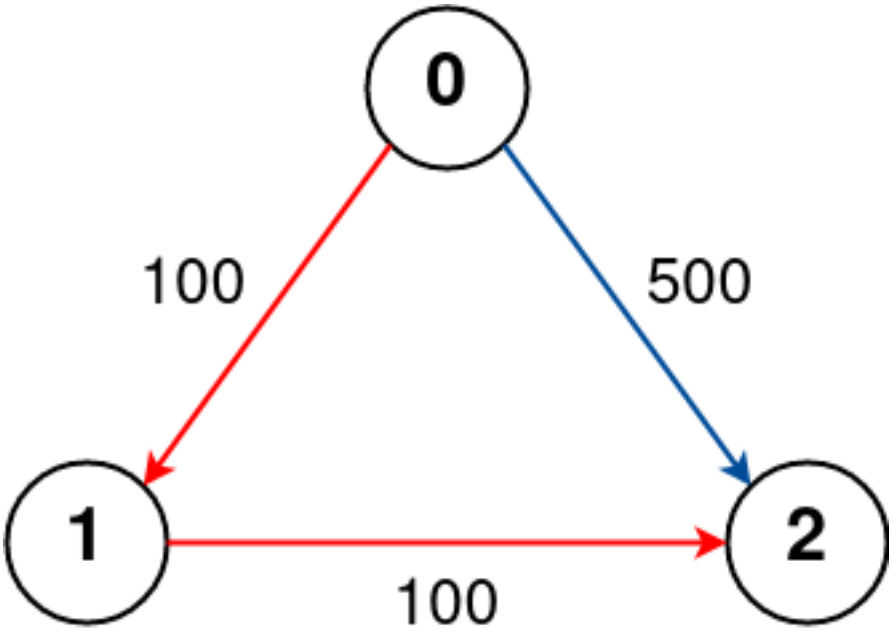
Output:

700

Explanation:

The graph is shown above. The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700. Note that the path through cities [0,1,2,3] is cheaper but is invalid because it uses 2 stops.

Example 2:



Input:

n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1

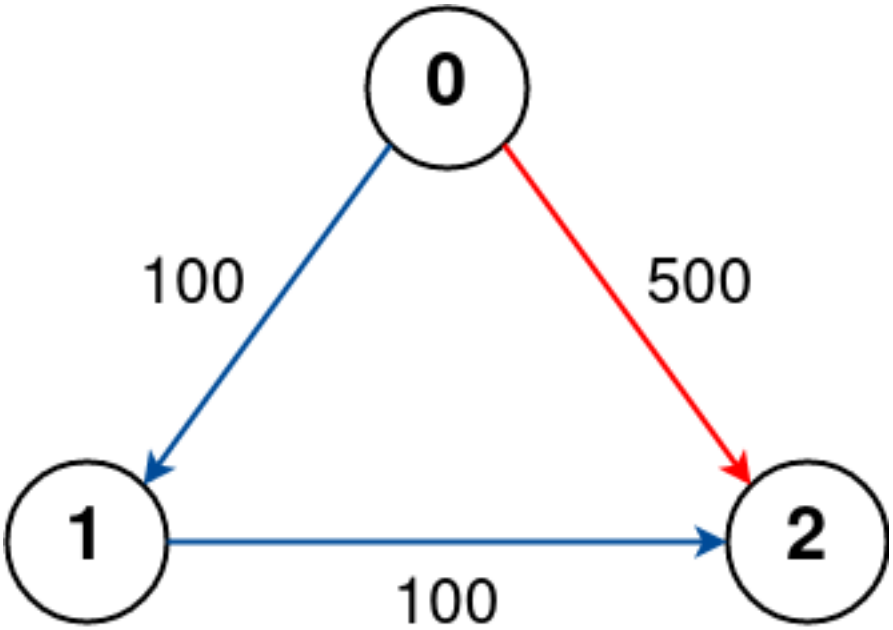
Output:

200

Explanation:

The graph is shown above. The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

Example 3:



Input:

n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0

Output:

500

Explanation:

The graph is shown above. The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.

Constraints:

2 <= n <= 100

0 <= flights.length <= (n * (n - 1) / 2)

flights[i].length == 3

0 <= from

i

, to

i

< n

from

i

!= to

i

1 <= price

i

<= 10

4

There will not be any multiple flights between two cities.

0 <= src, dst, k < n

src != dst

Code Snippets

C++:

```
class Solution {
public:
int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst,
int k) {

}
};
```

Java:

```
class Solution {
public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k)
{

}
}
```

Python3:

```
class Solution:
def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst:
int, k: int) -> int:
```

Python:

```
class Solution(object):
def findCheapestPrice(self, n, flights, src, dst, k):
"""
:type n: int
:type flights: List[List[int]]
:type src: int
:type dst: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} flights
```

```
* @param {number} src
* @param {number} dst
* @param {number} k
* @return {number}
*/
var findCheapestPrice = function(n, flights, src, dst, k) {

};
```

TypeScript:

```
function findCheapestPrice(n: number, flights: number[][], src: number, dst:
number, k: number): number {

};
```

C#:

```
public class Solution {
public int FindCheapestPrice(int n, int[][] flights, int src, int dst, int k)
{

}
}
```

C:

```
int findCheapestPrice(int n, int** flights, int flightsSize, int*
flightsColSize, int src, int dst, int k) {

}
```

Go:

```
func findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findCheapestPrice(n: Int, flights: Array<IntArray>, src: Int, dst: Int,
```

```
k: Int): Int {

}
}
```

Swift:

```
class Solution {
func findCheapestPrice(_ n: Int, _ flights: [[Int]], _ src: Int, _ dst: Int,
_ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_cheapest_price(n: i32, flights: Vec<Vec<i32>>, src: i32, dst:
i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} flights
# @param {Integer} src
# @param {Integer} dst
# @param {Integer} k
# @return {Integer}
def find_cheapest_price(n, flights, src, dst, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $flights
* @param Integer $src
```

```
* @param Integer $dst
* @param Integer $k
* @return Integer
*/
function findCheapestPrice($n, $flights, $src, $dst, $k) {

}
}
```

Dart:

```
class Solution {
int findCheapestPrice(int n, List<List<int>> flights, int src, int dst, int
k) {

}
}
```

Scala:

```
object Solution {
def findCheapestPrice(n: Int, flights: Array[Array[Int]], src: Int, dst: Int,
k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_cheapest_price(n :: integer, flights :: [[integer]], src ::
integer, dst :: integer, k :: integer) :: integer
def find_cheapest_price(n, flights, src, dst, k) do

end
end
```

Erlang:

```
-spec find_cheapest_price(N :: integer(), Flights :: [[integer()]], Src ::
integer(), Dst :: integer(), K :: integer()) -> integer().
find_cheapest_price(N, Flights, Src, Dst, K) ->
```

```
.
```

Racket:

```
(define/contract (find-cheapest-price n flights src dst k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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 findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst,
int k) {

}
};
```

Java Solution:

```
/**
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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 findCheapestPrice(int n, int[][] flights, int src, int dst, int k)
{

}
}
```

Python3 Solution:

```
"""
Problem: Cheapest Flights Within K Stops
Difficulty: Medium
Tags: array, graph, dp, search, queue, heap

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 findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst:
int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findCheapestPrice(self, n, flights, src, dst, k):
"""
:type n: int
:type flights: List[List[int]]
:type src: int
:type dst: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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[][]} flights
* @param {number} src
* @param {number} dst
* @param {number} k
* @return {number}
*/
var findCheapestPrice = function(n, flights, src, dst, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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 findCheapestPrice(n: number, flights: number[][], src: number, dst:
number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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 FindCheapestPrice(int n, int[][] flights, int src, int dst, int k)
{

}
}
```

C Solution:

```
/*
* Problem: Cheapest Flights Within K Stops
* Difficulty: Medium
* Tags: array, graph, dp, search, queue, heap
*
* 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 findCheapestPrice(int n, int** flights, int flightsSize, int*
flightsColSize, int src, int dst, int k) {

}
```

Go Solution:

```
// Problem: Cheapest Flights Within K Stops
// Difficulty: Medium
// Tags: array, graph, dp, search, queue, heap
//
// 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 findCheapestPrice(n int, flights [][]int, src int, dst int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findCheapestPrice(n: Int, flights: Array<IntArray>, src: Int, dst: Int,
k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findCheapestPrice(_ n: Int, _ flights: [[Int]], _ src: Int, _ dst: Int,
_ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Cheapest Flights Within K Stops
// Difficulty: Medium
// Tags: array, graph, dp, search, queue, heap
//
// 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 find_cheapest_price(n: i32, flights: Vec<Vec<i32>>, src: i32, dst:
i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} flights
# @param {Integer} src
# @param {Integer} dst
# @param {Integer} k
# @return {Integer}
def find_cheapest_price(n, flights, src, dst, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $flights
* @param Integer $src
* @param Integer $dst
* @param Integer $k
* @return Integer
*/
function findCheapestPrice($n, $flights, $src, $dst, $k) {

}
}
```

Dart Solution:

```
class Solution {
int findCheapestPrice(int n, List<List<int>> flights, int src, int dst, int
k) {

}
}
```

Scala Solution:

```
object Solution {
def findCheapestPrice(n: Int, flights: Array[Array[Int]], src: Int, dst: Int,
```

```
k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_cheapest_price(n :: integer, flights :: [[integer]], src ::
integer, dst :: integer, k :: integer) :: integer
def find_cheapest_price(n, flights, src, dst, k) do

end
end
```

Erlang Solution:

```
-spec find_cheapest_price(N :: integer(), Flights :: [[integer()]], Src ::
integer(), Dst :: integer(), K :: integer()) -> integer().
find_cheapest_price(N, Flights, Src, Dst, K) ->
.
```

Racket Solution:

```
(define/contract (find-cheapest-price n flights src dst k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer? exact-integer? exact-integer?)
)
```