

# Problem 3508: Implement Router

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Design a data structure that can efficiently manage data packets in a network router. Each data packet consists of the following attributes:

source

: A unique identifier for the machine that generated the packet.

destination

: A unique identifier for the target machine.

timestamp

: The time at which the packet arrived at the router.

Implement the

Router

class:

Router(int memoryLimit)

: Initializes the Router object with a fixed memory limit.

memoryLimit

is the

maximum

number of packets the router can store at any given time.

If adding a new packet would exceed this limit, the

oldest

packet must be removed to free up space.

```
bool addPacket(int source, int destination, int timestamp)
```

: Adds a packet with the given attributes to the router.

A packet is considered a duplicate if another packet with the same

source

,

destination

, and

timestamp

already exists in the router.

Return

true

if the packet is successfully added (i.e., it is not a duplicate); otherwise return

false

`int[] forwardPacket()`

: Forwards the next packet in FIFO (First In First Out) order.

Remove the packet from storage.

Return the packet as an array

`[source, destination, timestamp]`

If there are no packets to forward, return an empty array.

`int getCount(int destination, int startTime, int endTime)`

:

Returns the number of packets currently stored in the router (i.e., not yet forwarded) that have the specified destination and have timestamps in the inclusive range

`[startTime, endTime]`

Note

that queries for

`addPacket`

will be made in non-decreasing order of

timestamp

Example 1:

Input:

```
["Router", "addPacket", "addPacket", "addPacket", "addPacket", "addPacket",
 "forwardPacket", "addPacket", "getCount"]
```

```
[[3], [1, 4, 90], [2, 5, 90], [1, 4, 90], [3, 5, 95], [4, 5, 105], [], [5, 2, 110], [5, 100, 110]]
```

Output:

```
[null, true, true, false, true, true, [2, 5, 90], true, 1]
```

Explanation

```
Router router = new Router(3); // Initialize Router with memoryLimit of 3.
```

```
router.addPacket(1, 4, 90); // Packet is added. Return True.
```

```
router.addPacket(2, 5, 90); // Packet is added. Return True.
```

```
router.addPacket(1, 4, 90); // This is a duplicate packet. Return False.
```

```
router.addPacket(3, 5, 95); // Packet is added. Return True
```

```
router.addPacket(4, 5, 105); // Packet is added,
```

```
[1, 4, 90]
```

is removed as number of packets exceeds memoryLimit. Return True.

```
router.forwardPacket(); // Return
```

```
[2, 5, 90]
```

and remove it from router.

```
router.addPacket(5, 2, 110); // Packet is added. Return True.
```

```
router.getCount(5, 100, 110); // The only packet with destination 5 and timestamp in the inclusive range
```

[100, 110]

is

[4, 5, 105]

. Return 1.

Example 2:

Input:

```
["Router", "addPacket", "forwardPacket", "forwardPacket"]
```

```
[[2], [7, 4, 90], [], []]
```

Output:

```
[null, true, [7, 4, 90], []]
```

Explanation

```
Router router = new Router(2); // Initialize
```

Router

with

memoryLimit

of 2.

```
router.addPacket(7, 4, 90); // Return True.
```

```
router.forwardPacket(); // Return
```

[7, 4, 90]

router.forwardPacket(); // There are no packets left, return

[]

Constraints:

$2 \leq \text{memoryLimit} \leq 10$

5

$1 \leq \text{source, destination} \leq 2 * 10$

5

$1 \leq \text{timestamp} \leq 10$

9

$1 \leq \text{startTime} \leq \text{endTime} \leq 10$

9

At most

10

5

calls will be made to

addPacket

,

forwardPacket

, and

getCount

methods altogether.

queries for

addPacket

will be made in non-decreasing order of

timestamp

## Code Snippets

C++:

```
class Router {
public:
    Router(int memoryLimit) {

    }

    bool addPacket(int source, int destination, int timestamp) {

    }

    vector<int> forwardPacket() {

    }

    int getCount(int destination, int startTime, int endTime) {

    }
}
```

```
};

/**
 * Your Router object will be instantiated and called as such:
 * Router* obj = new Router(memoryLimit);
 * bool param_1 = obj->addPacket(source,destination,timestamp);
 * vector<int> param_2 = obj->forwardPacket();
 * int param_3 = obj->getCount(destination,startTime,endTime);
 */
```

### Java:

```
class Router {

    public Router(int memoryLimit) {

    }

    public boolean addPacket(int source, int destination, int timestamp) {

    }

    public int[] forwardPacket() {

    }

    public int getCount(int destination, int startTime, int endTime) {

    }
}

/**
 * Your Router object will be instantiated and called as such:
 * Router obj = new Router(memoryLimit);
 * boolean param_1 = obj.addPacket(source,destination,timestamp);
 * int[] param_2 = obj.forwardPacket();
 * int param_3 = obj.getCount(destination,startTime,endTime);
 */
```

### Python3:

```

class Router:

def __init__(self, memoryLimit: int):

    def addPacket(self, source: int, destination: int, timestamp: int) -> bool:

        def forwardPacket(self) -> List[int]:

            def getCount(self, destination: int, startTime: int, endTime: int) -> int:

                # Your Router object will be instantiated and called as such:
                # obj = Router(memoryLimit)
                # param_1 = obj.addPacket(source,destination,timestamp)
                # param_2 = obj.forwardPacket()
                # param_3 = obj.getCount(destination,startTime,endTime)

```

## Python:

```

class Router(object):

    def __init__(self, memoryLimit):
        """
        :type memoryLimit: int
        """

    def addPacket(self, source, destination, timestamp):
        """
        :type source: int
        :type destination: int
        :type timestamp: int
        :rtype: bool
        """

    def forwardPacket(self):
        """
        :rtype: List[int]

```

```

"""
def getCount(self, destination, startTime, endTime):
    """
    :type destination: int
    :type startTime: int
    :type endTime: int
    :rtype: int
"""

# Your Router object will be instantiated and called as such:
# obj = Router(memoryLimit)
# param_1 = obj.addPacket(source,destination,timestamp)
# param_2 = obj.forwardPacket()
# param_3 = obj.getCount(destination,startTime,endTime)

```

### JavaScript:

```

/**
 * @param {number} memoryLimit
 */
var Router = function(memoryLimit) {

};

/**
 * @param {number} source
 * @param {number} destination
 * @param {number} timestamp
 * @return {boolean}
 */
Router.prototype.addPacket = function(source, destination, timestamp) {

};

/**
 * @return {number[]}
 */
Router.prototype.forwardPacket = function() {

```

```

};

/**
* @param {number} destination
* @param {number} startTime
* @param {number} endTime
* @return {number}
*/
Router.prototype.getCount = function(destination, startTime, endTime) {

};

/**
* Your Router object will be instantiated and called as such:
* var obj = new Router(memoryLimit)
* var param_1 = obj.addPacket(source,destination,timestamp)
* var param_2 = obj.forwardPacket()
* var param_3 = obj.getCount(destination,startTime,endTime)
*/

```

### TypeScript:

```

class Router {
constructor(memoryLimit: number) {

}

addPacket(source: number, destination: number, timestamp: number): boolean {

}

forwardPacket(): number[] {

}

getCount(destination: number, startTime: number, endTime: number): number {

}
}

/**

```

```
* Your Router object will be instantiated and called as such:  
* var obj = new Router(memoryLimit)  
* var param_1 = obj.addPacket(source,destination,timestamp)  
* var param_2 = obj.forwardPacket()  
* var param_3 = obj.getCount(destination,startTime,endTime)  
*/
```

## C#:

```
public class Router {  
  
    public Router(int memoryLimit) {  
  
    }  
  
    public bool AddPacket(int source, int destination, int timestamp) {  
  
    }  
  
    public int[] ForwardPacket() {  
  
    }  
  
    public int GetCount(int destination, int startTime, int endTime) {  
  
    }  
}  
  
/**  
 * Your Router object will be instantiated and called as such:  
 * Router obj = new Router(memoryLimit);  
 * bool param_1 = obj.AddPacket(source,destination,timestamp);  
 * int[] param_2 = obj.ForwardPacket();  
 * int param_3 = obj.GetCount(destination,startTime,endTime);  
 */
```

## C:

```
typedef struct {
```

```

} Router;

Router* routerCreate(int memoryLimit) {

}

bool routerAddPacket(Router* obj, int source, int destination, int timestamp)
{

}

int* routerForwardPacket(Router* obj, int* retSize) {

}

int routerGetCount(Router* obj, int destination, int startTime, int endTime)
{

}

void routerFree(Router* obj) {

}

/**
 * Your Router struct will be instantiated and called as such:
 * Router* obj = routerCreate(memoryLimit);
 * bool param_1 = routerAddPacket(obj, source, destination, timestamp);
 *
 * int* param_2 = routerForwardPacket(obj, retSize);
 *
 * int param_3 = routerGetCount(obj, destination, startTime, endTime);
 *
 * routerFree(obj);
 */

```

## Go:

```
type Router struct {
```

```

}

func Constructor(memoryLimit int) Router {

}

func (this *Router) AddPacket(source int, destination int, timestamp int)
bool {

}

func (this *Router) ForwardPacket() []int {

}

func (this *Router) GetCount(destination int, startTime int, endTime int) int
{
}

/***
* Your Router object will be instantiated and called as such:
* obj := Constructor(memoryLimit);
* param_1 := obj.AddPacket(source,destination,timestamp);
* param_2 := obj.ForwardPacket();
* param_3 := obj.GetCount(destination,startTime,endTime);
*/

```

## Kotlin:

```

class Router(memoryLimit: Int) {

    fun addPacket(source: Int, destination: Int, timestamp: Int): Boolean {

    }

    fun forwardPacket(): IntArray {

```

```

}

fun getCount(destination: Int, startTime: Int, endTime: Int): Int {

}

}

/***
* Your Router object will be instantiated and called as such:
* var obj = Router(memoryLimit)
* var param_1 = obj.addPacket(source,destination,timestamp)
* var param_2 = obj.forwardPacket()
* var param_3 = obj.getCount(destination,startTime,endTime)
*/

```

## Swift:

```

class Router {

init(_ memoryLimit: Int) {

}

func addPacket(_ source: Int, _ destination: Int, _ timestamp: Int) -> Bool {

}

func forwardPacket() -> [Int] {

}

func getCount(_ destination: Int, _ startTime: Int, _ endTime: Int) -> Int {

}

}

/***
* Your Router object will be instantiated and called as such:
* let obj = Router(memoryLimit)
*/

```

```
* let ret_1: Bool = obj.addPacket(source, destination, timestamp)
* let ret_2: [Int] = obj.forwardPacket()
* let ret_3: Int = obj.getCount(destination, startTime, endTime)
*/
```

## Rust:

```
struct Router {

}

/***
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Router {

    fn new(memoryLimit: i32) -> Self {

    }

    fn add_packet(&self, source: i32, destination: i32, timestamp: i32) -> bool {

    }

    fn forward_packet(&self) -> Vec<i32> {

    }

    fn get_count(&self, destination: i32, start_time: i32, end_time: i32) -> i32
    {
    }
}

/***
* Your Router object will be instantiated and called as such:
* let obj = Router::new(memoryLimit);
* let ret_1: bool = obj.add_packet(source, destination, timestamp);
* let ret_2: Vec<i32> = obj.forward_packet();
* let ret_3: i32 = obj.get_count(destination, startTime, endTime);
*/
```

```
* /
```

## Ruby:

```
class Router

=begin
:type memory_limit: Integer
=end
def initialize(memory_limit)

end

=begin
:type source: Integer
:type destination: Integer
:type timestamp: Integer
:rtype: Boolean
=end
def add_packet(source, destination, timestamp)

end

=begin
:rtype: Integer[]
=end
def forward_packet()

end

=begin
:type destination: Integer
:type start_time: Integer
:type end_time: Integer
:rtype: Integer
=end
def get_count(destination, start_time, end_time)

end
```

```
end

# Your Router object will be instantiated and called as such:
# obj = Router.new(memory_limit)
# param_1 = obj.add_packet(source, destination, timestamp)
# param_2 = obj.forward_packet()
# param_3 = obj.get_count(destination, start_time, end_time)
```

## PHP:

```
class Router {

    /**
     * @param Integer $memoryLimit
     */

    function __construct($memoryLimit) {

    }

    /**
     * @param Integer $source
     * @param Integer $destination
     * @param Integer $timestamp
     * @return Boolean
     */
    function addPacket($source, $destination, $timestamp) {

    }

    /**
     * @return Integer[]
     */
    function forwardPacket() {

    }

    /**
     * @param Integer $destination
     * @param Integer $startTime
     * @param Integer $endTime
     * @return Integer
     */
}
```

```

        */

    function getCount($destination, $startTime, $endTime) {

    }

}

/***
* Your Router object will be instantiated and called as such:
* $obj = Router($memoryLimit);
* $ret_1 = $obj->addPacket($source, $destination, $timestamp);
* $ret_2 = $obj->forwardPacket();
* $ret_3 = $obj->getCount($destination, $startTime, $endTime);
*/

```

### Dart:

```

class Router {

    Router(int memoryLimit) {

    }

    bool addPacket(int source, int destination, int timestamp) {

    }

    List<int> forwardPacket() {

    }

    int getCount(int destination, int startTime, int endTime) {

    }

}

/***
* Your Router object will be instantiated and called as such:
* Router obj = Router(memoryLimit);
* bool param1 = obj.addPacket(source,destination,timestamp);
* List<int> param2 = obj.forwardPacket();
* int param3 = obj.getCount(destination,startTime,endTime);
*/

```

## Scala:

```
class Router(_memoryLimit: Int) {  
  
    def addPacket(source: Int, destination: Int, timestamp: Int): Boolean = {  
  
    }  
  
    def forwardPacket(): Array[Int] = {  
  
    }  
  
    def getCount(destination: Int, startTime: Int, endTime: Int): Int = {  
  
    }  
  
    }  
  
/**  
 * Your Router object will be instantiated and called as such:  
 * val obj = new Router(memoryLimit)  
 * val param_1 = obj.addPacket(source,destination,timestamp)  
 * val param_2 = obj.forwardPacket()  
 * val param_3 = obj.getCount(destination,startTime,endTime)  
 */
```

## Elixir:

```
defmodule Router do  
  @spec init_(memory_limit :: integer) :: any  
  def init_(memory_limit) do  
  
  end  
  
  @spec add_packet(source :: integer, destination :: integer, timestamp ::  
  integer) :: boolean  
  def add_packet(source, destination, timestamp) do  
  
  end  
  
  @spec forward_packet() :: [integer]  
  def forward_packet() do  
  
  end
```

```

end

@spec get_count(destination :: integer, start_time :: integer, end_time :: integer) :: integer
def get_count(destination, start_time, end_time) do

end
end

# Your functions will be called as such:
# Router.init_(memory_limit)
# param_1 = Router.add_packet(source, destination, timestamp)
# param_2 = Router.forward_packet()
# param_3 = Router.get_count(destination, start_time, end_time)

# Router.init_ will be called before every test case, in which you can do
some necessary initializations.

```

## Erlang:

```

-spec router_init_(MemoryLimit :: integer()) -> any().
router_init_(MemoryLimit) ->
.

-spec router_add_packet(Source :: integer(), Destination :: integer(),
Timestamp :: integer()) -> boolean().
router_add_packet(Source, Destination, Timestamp) ->
.

-spec router_forward_packet() -> [integer()].
router_forward_packet() ->
.

-spec router_get_count(Destination :: integer(), StartTime :: integer(),
EndTime :: integer()) -> integer().
router_get_count(Destination, StartTime, EndTime) ->
.

%% Your functions will be called as such:
%% router_init_(MemoryLimit),

```

```

%% Param_1 = router_add_packet(Source, Destination, Timestamp),
%% Param_2 = router_forward_packet(),
%% Param_3 = router_get_count(Destination, StartTime, EndTime),

%% router_init_ will be called before every test case, in which you can do
some necessary initializations.

```

### Racket:

```

(define router%
  (class object%
    (super-new)

    ; memory-limit : exact-integer?
    (init-field
      memory-limit)

    ; add-packet : exact-integer? exact-integer? exact-integer? -> boolean?
    (define/public (add-packet source destination timestamp)
    )

    ; forward-packet : -> (listof exact-integer?)
    (define/public (forward-packet)
    )

    ; get-count : exact-integer? exact-integer? exact-integer? -> exact-integer?
    (define/public (get-count destination start-time end-time)
    )))

;; Your router% object will be instantiated and called as such:
;; (define obj (new router% [memory-limit memory-limit]))
;; (define param_1 (send obj add-packet source destination timestamp))
;; (define param_2 (send obj forward-packet))
;; (define param_3 (send obj get-count destination start-time end-time))

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Implement Router
 * Difficulty: Medium

```

```

* Tags: array, dp, hash, search, queue
*
* 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 Router {
public:
Router(int memoryLimit) {

}

bool addPacket(int source, int destination, int timestamp) {

}

vector<int> forwardPacket() {

}

int getCount(int destination, int startTime, int endTime) {

}
};

/***
* Your Router object will be instantiated and called as such:
* Router* obj = new Router(memoryLimit);
* bool param_1 = obj->addPacket(source,destination,timestamp);
* vector<int> param_2 = obj->forwardPacket();
* int param_3 = obj->getCount(destination,startTime,endTime);
*/

```

## Java Solution:

```

/**
* Problem: Implement Router
* Difficulty: Medium
* Tags: array, dp, hash, search, queue
*

```

```

* 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 Router {

    public Router(int memoryLimit) {

    }

    public boolean addPacket(int source, int destination, int timestamp) {

    }

    public int[] forwardPacket() {

    }

    public int getCount(int destination, int startTime, int endTime) {

    }
}

```

```

/**
 * Your Router object will be instantiated and called as such:
 * Router obj = new Router(memoryLimit);
 * boolean param_1 = obj.addPacket(source,destination,timestamp);
 * int[] param_2 = obj.forwardPacket();
 * int param_3 = obj.getCount(destination,startTime,endTime);
 */

```

### Python3 Solution:

```

"""
Problem: Implement Router
Difficulty: Medium
Tags: array, dp, hash, search, queue

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 Router:

def __init__(self, memoryLimit: int):

    def addPacket(self, source: int, destination: int, timestamp: int) -> bool:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Router(object):

def __init__(self, memoryLimit):
    """
    :type memoryLimit: int
    """

    def addPacket(self, source, destination, timestamp):
        """
        :type source: int
        :type destination: int
        :type timestamp: int
        :rtype: bool
        """

    def forwardPacket(self):
        """
        :rtype: List[int]
        """

    def getCount(self, destination, startTime, endTime):
        """
        :type destination: int
        :type startTime: int

```

```

:type endTime: int
:rtype: int
"""

# Your Router object will be instantiated and called as such:
# obj = Router(memoryLimit)
# param_1 = obj.addPacket(source,destination,timestamp)
# param_2 = obj.forwardPacket()
# param_3 = obj.getCount(destination,startTime,endTime)

```

### JavaScript Solution:

```

/**
 * Problem: Implement Router
 * Difficulty: Medium
 * Tags: array, dp, hash, search, queue
 *
 * 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} memoryLimit
 */
var Router = function(memoryLimit) {

};

/**
 * @param {number} source
 * @param {number} destination
 * @param {number} timestamp
 * @return {boolean}
 */
Router.prototype.addPacket = function(source, destination, timestamp) {

};

```

```

    /**
 * @return {number[]}
 */
Router.prototype.forwardPacket = function() {
};

/**
 * @param {number} destination
 * @param {number} startTime
 * @param {number} endTime
 * @return {number}
 */
Router.prototype.getCount = function(destination, startTime, endTime) {

};

/**
 * Your Router object will be instantiated and called as such:
 * var obj = new Router(memoryLimit)
 * var param_1 = obj.addPacket(source,destination,timestamp)
 * var param_2 = obj.forwardPacket()
 * var param_3 = obj.getCount(destination,startTime,endTime)
 */

```

### TypeScript Solution:

```

    /**
 * Problem: Implement Router
 * Difficulty: Medium
 * Tags: array, dp, hash, search, queue
 *
 * 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 Router {
constructor(memoryLimit: number) {

}

```

```

    addPacket(source: number, destination: number, timestamp: number): boolean {
        }

    forwardPacket(): number[] {
        }

    getCount(destination: number, startTime: number, endTime: number): number {
        }
    }

    /**
     * Your Router object will be instantiated and called as such:
     * var obj = new Router(memoryLimit)
     * var param_1 = obj.addPacket(source,destination,timestamp)
     * var param_2 = obj.forwardPacket()
     * var param_3 = obj.getCount(destination,startTime,endTime)
     */
}

```

### C# Solution:

```

/*
 * Problem: Implement Router
 * Difficulty: Medium
 * Tags: array, dp, hash, search, queue
 *
 * 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 Router {

    public Router(int memoryLimit) {

    }

    public bool AddPacket(int source, int destination, int timestamp) {

```

```

}

public int[] ForwardPacket() {

}

public int GetCount(int destination, int startTime, int endTime) {

}

/**
 * Your Router object will be instantiated and called as such:
 * Router obj = new Router(memoryLimit);
 * bool param_1 = obj.AddPacket(source,destination,timestamp);
 * int[] param_2 = obj.ForwardPacket();
 * int param_3 = obj.GetCount(destination,startTime,endTime);
 */

```

## C Solution:

```

/*
* Problem: Implement Router
* Difficulty: Medium
* Tags: array, dp, hash, search, queue
*
* 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
*/

```

```

typedef struct {

} Router;

Router* routerCreate(int memoryLimit) {

```

```

}

bool routerAddPacket(Router* obj, int source, int destination, int timestamp)
{
}

int* routerForwardPacket(Router* obj, int* retSize) {

}

int routerGetCount(Router* obj, int destination, int startTime, int endTime)
{
}

void routerFree(Router* obj) {

}

/**
 * Your Router struct will be instantiated and called as such:
 * Router* obj = routerCreate(memoryLimit);
 * bool param_1 = routerAddPacket(obj, source, destination, timestamp);
 *
 * int* param_2 = routerForwardPacket(obj, retSize);
 *
 * int param_3 = routerGetCount(obj, destination, startTime, endTime);
 *
 * routerFree(obj);
 */

```

## Go Solution:

```

// Problem: Implement Router
// Difficulty: Medium
// Tags: array, dp, hash, search, queue
//
// 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

type Router struct {

}

func Constructor(memoryLimit int) Router {
}

func (this *Router) AddPacket(source int, destination int, timestamp int)
bool {

}

func (this *Router) ForwardPacket() []int {

}

func (this *Router) GetCount(destination int, startTime int, endTime int) int
{
}

/**
 * Your Router object will be instantiated and called as such:
 * obj := Constructor(memoryLimit);
 * param_1 := obj.AddPacket(source,destination,timestamp);
 * param_2 := obj.ForwardPacket();
 * param_3 := obj.GetCount(destination,startTime,endTime);
 */

```

## Kotlin Solution:

```

class Router(memoryLimit: Int) {

```

```

fun addPacket(source: Int, destination: Int, timestamp: Int): Boolean {
}

fun forwardPacket(): IntArray {

}

fun getCount(destination: Int, startTime: Int, endTime: Int): Int {

}

/**
 * Your Router object will be instantiated and called as such:
 * var obj = Router(memoryLimit)
 * var param_1 = obj.addPacket(source,destination,timestamp)
 * var param_2 = obj.forwardPacket()
 * var param_3 = obj.getCount(destination,startTime,endTime)
 */

```

### Swift Solution:

```

class Router {

init(_ memoryLimit: Int) {

}

func addPacket(_ source: Int, _ destination: Int, _ timestamp: Int) -> Bool {

}

func forwardPacket() -> [Int] {

}

func getCount(_ destination: Int, _ startTime: Int, _ endTime: Int) -> Int {
}

```

```

    }

}

/***
* Your Router object will be instantiated and called as such:
* let obj = Router(memoryLimit)
* let ret_1: Bool = obj.addPacket(source, destination, timestamp)
* let ret_2: [Int] = obj.forwardPacket()
* let ret_3: Int = obj.getCount(destination, startTime, endTime)
*/

```

### Rust Solution:

```

// Problem: Implement Router
// Difficulty: Medium
// Tags: array, dp, hash, search, queue
//
// 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

struct Router {

}

/***
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Router {

fn new(memoryLimit: i32) -> Self {

}

fn add_packet(&self, source: i32, destination: i32, timestamp: i32) -> bool {

}

fn forward_packet(&self) -> Vec<i32> {

```

```

}

fn get_count(&self, destination: i32, start_time: i32, end_time: i32) -> i32
{
}

}

}

/***
* Your Router object will be instantiated and called as such:
* let obj = Router::new(memoryLimit);
* let ret_1: bool = obj.add_packet(source, destination, timestamp);
* let ret_2: Vec<i32> = obj.forward_packet();
* let ret_3: i32 = obj.get_count(destination, startTime, endTime);
*/

```

## Ruby Solution:

```

class Router

=begin
:type memory_limit: Integer
=end
def initialize(memory_limit)

end

=begin
:type source: Integer
:type destination: Integer
:type timestamp: Integer
:rtype: Boolean
=end
def add_packet(source, destination, timestamp)

end

=begin

```

```

:rtype: Integer[]
=end

def forward_packet()

end

=begin

:type destination: Integer
:type start_time: Integer
:type end_time: Integer
:rtype: Integer
=end

def get_count(destination, start_time, end_time)

end

end

# Your Router object will be instantiated and called as such:
# obj = Router.new(memory_limit)
# param_1 = obj.add_packet(source, destination, timestamp)
# param_2 = obj.forward_packet()
# param_3 = obj.get_count(destination, start_time, end_time)

```

## PHP Solution:

```

class Router {

    /**
     * @param Integer $memoryLimit
     */

    function __construct($memoryLimit) {

    }

    /**
     * @param Integer $source
     * @param Integer $destination
     * @param Integer $timestamp
     * @return Boolean
     */
}

```

```

/*
function addPacket($source, $destination, $timestamp) {

}

/***
* @return Integer[]
*/
function forwardPacket() {

}

/***
* @param Integer $destination
* @param Integer $startTime
* @param Integer $endTime
* @return Integer
*/
function getCount($destination, $startTime, $endTime) {

}
}

/***
* Your Router object will be instantiated and called as such:
* $obj = Router($memoryLimit);
* $ret_1 = $obj->addPacket($source, $destination, $timestamp);
* $ret_2 = $obj->forwardPacket();
* $ret_3 = $obj->getCount($destination, $startTime, $endTime);
*/

```

## Dart Solution:

```

class Router {

Router(int memoryLimit) {

}

bool addPacket(int source, int destination, int timestamp) {

```

```

}

List<int> forwardPacket() {

}

int getCount(int destination, int startTime, int endTime) {

}

}

/***
* Your Router object will be instantiated and called as such:
* Router obj = Router(memoryLimit);
* bool param1 = obj.addPacket(source,destination,timestamp);
* List<int> param2 = obj.forwardPacket();
* int param3 = obj.getCount(destination,startTime,endTime);
*/

```

### Scala Solution:

```

class Router(_memoryLimit: Int) {

def addPacket(source: Int, destination: Int, timestamp: Int): Boolean = {

}

def forwardPacket(): Array[Int] = {

}

def getCount(destination: Int, startTime: Int, endTime: Int): Int = {

}

}

/***
* Your Router object will be instantiated and called as such:
* val obj = new Router(memoryLimit)
* val param_1 = obj.addPacket(source,destination,timestamp)

```

```
* val param_2 = obj.forwardPacket()
* val param_3 = obj.getCount(destination,startTime,endTime)
*/
```

### Elixir Solution:

```
defmodule Router do
  @spec init_(memory_limit :: integer) :: any
  def init_(memory_limit) do
    end

    @spec add_packet(source :: integer, destination :: integer, timestamp :: integer) :: boolean
    def add_packet(source, destination, timestamp) do
      end

      @spec forward_packet() :: [integer]
      def forward_packet() do
        end

        @spec get_count(destination :: integer, start_time :: integer, end_time :: integer) :: integer
        def get_count(destination, start_time, end_time) do
          end
        end

        # Your functions will be called as such:
        # Router.init_(memory_limit)
        # param_1 = Router.add_packet(source, destination, timestamp)
        # param_2 = Router.forward_packet()
        # param_3 = Router.get_count(destination, start_time, end_time)

        # Router.init_ will be called before every test case, in which you can do
        some necessary initializations.
```

### Erlang Solution:

```

-spec router_init_(MemoryLimit :: integer()) -> any().
router_init_(MemoryLimit) ->
.

-spec router_add_packet(Source :: integer(), Destination :: integer(),
Timestamp :: integer()) -> boolean().
router_add_packet(Source, Destination, Timestamp) ->
.

-spec router_forward_packet() -> [integer()].
router_forward_packet() ->
.

-spec router_get_count(Destination :: integer(), StartTime :: integer(),
EndTime :: integer()) -> integer().
router_get_count(Destination, StartTime, EndTime) ->
.

%% Your functions will be called as such:
%% router_init_(MemoryLimit),
%% Param_1 = router_add_packet(Source, Destination, Timestamp),
%% Param_2 = router_forward_packet(),
%% Param_3 = router_get_count(Destination, StartTime, EndTime),

%% router_init_ will be called before every test case, in which you can do
some necessary initializations.

```

## Racket Solution:

```

(define router%
  (class object%
    (super-new)

    ; memory-limit : exact-integer?
    (init-field
      memory-limit)

    ; add-packet : exact-integer? exact-integer? exact-integer? -> boolean?
    (define/public (add-packet source destination timestamp)
    )
    ; forward-packet : -> (listof exact-integer?)

```

```
(define/public (forward-packet)
)
; get-count : exact-integer? exact-integer? exact-integer? -> exact-integer?
(define/public (get-count destination start-time end-time)
))

;; Your router% object will be instantiated and called as such:
;; (define obj (new router% [memory-limit memory-limit]))
;; (define param_1 (send obj add-packet source destination timestamp))
;; (define param_2 (send obj forward-packet))
;; (define param_3 (send obj get-count destination start-time end-time))
```