Problem 497: Random Point in Non-overlapping Rectangles

Problem Information

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

Problem Description

You are given an array of non-overlapping axis-aligned rectangles

rects

where

rects[i] = [a

i

, b

i

, x

i

, y

i

]

indicates that

(a_i, b_i) is the bottom-left corner point of the ith rectangle and (x_i, y_i) is the top-right corner point of the ith rectangle. Design an algorithm to pick a random integer point inside the space covered by one of the given rectangles. A point on the perimeter of a rectangle is included in the space covered by the rectangle.

Any integer point inside the space covered by one of the given rectangles should be equally likely to be returned.

Note

that an integer point is a point that has integer coordinates.

Implement the

Solution

class:

Solution(int[][] rects)

Initializes the object with the given rectangles

rects

.

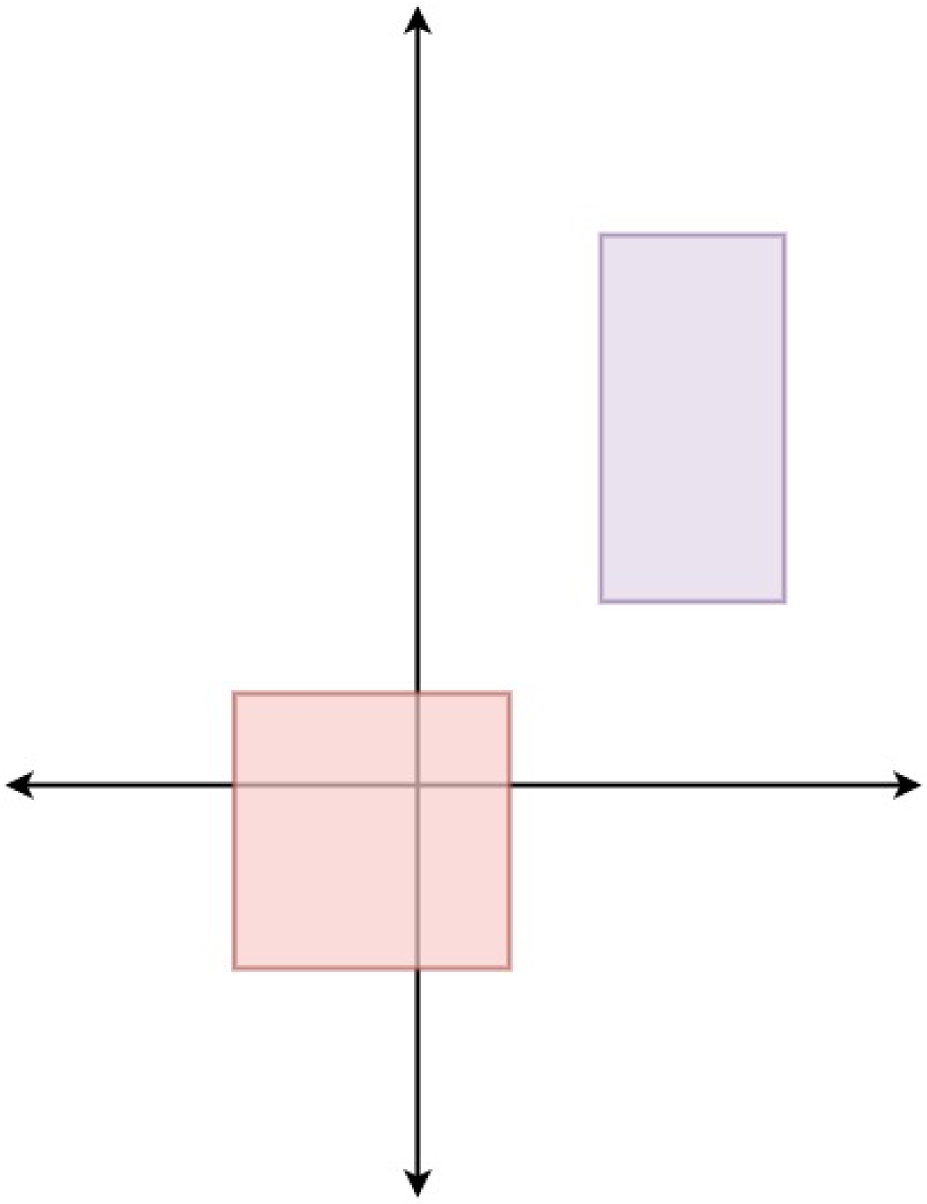
int[] pick()

Returns a random integer point

[u, v]

inside the space covered by one of the given rectangles.

Example 1:



Input

["Solution", "pick", "pick", "pick", "pick", "pick"] [[[[-2, -2, 1, 1], [2, 2, 4, 6]]], [], [], [], [], []]

Output

[null, [1, -2], [1, -1], [-1, -2], [-2, -2], [0, 0]]

Explanation

Solution solution = new Solution([[-2, -2, 1, 1], [2, 2, 4, 6]]); solution.pick(); // return [1, -2] solution.pick(); // return [1, -1] solution.pick(); // return [-1, -2] solution.pick(); // return [-2, -2] solution.pick(); // return [0, 0]

Constraints:

1 <= rects.length <= 100

rects[i].length == 4

-10

9

<= a

i

< x

i

<= 10

9

-10

9

<= b

i

< y

i

<= 10

9

x

i

- a

i

<= 2000

y

i

- b

i

<= 2000

All the rectangles do not overlap.

At most

10

4

calls will be made to

pick

.

Code Snippets

C++:

```
class Solution {
public:
Solution(vector<vector<int>>& rects) {

}

vector<int> pick() {

}
};

/**
* Your Solution object will be instantiated and called as such:
* Solution* obj = new Solution(rects);
* vector<int> param_1 = obj->pick();
*/
```

Java:

```
class Solution {

public Solution(int[][] rects) {

}

public int[] pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(rects);
* int[] param_1 = obj.pick();
*/
```

Python3:

```
class Solution:

def __init__(self, rects: List[List[int]]):
```

```
def pick(self) -> List[int]:



# Your Solution object will be instantiated and called as such:
# obj = Solution(rects)
# param_1 = obj.pick()
```

Python:

```
class Solution(object):

def __init__(self, rects):
"""
:type rects: List[List[int]]
"""



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




# Your Solution object will be instantiated and called as such:
# obj = Solution(rects)
# param_1 = obj.pick()
```

JavaScript:

```
/**
* @param {number[][]} rects
*/
var Solution = function(rects) {

};

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

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(rects)
* var param_1 = obj.pick()
*/
```

TypeScript:

```
class Solution {
constructor(rects: number[][]) {

}

pick(): number[] {

}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(rects)
* var param_1 = obj.pick()
*/
```

C#:

```
public class Solution {

public Solution(int[][] rects) {

}

public int[] Pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
```

```
* Solution obj = new Solution(rects);
* int[] param_1 = obj.Pick();
*/
```

C:

```



typedef struct {

} Solution;


Solution* solutionCreate(int** rects, int rectsSize, int* rectsColSize) {

}

int* solutionPick(Solution* obj, int* retSize) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(rects, rectsSize, rectsColSize);
* int* param_1 = solutionPick(obj, retSize);

* solutionFree(obj);
*/
```

Go:

```
type Solution struct {

}


func Constructor(rects [][]int) Solution {
}


func (this *Solution) Pick() []int {

}


/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(rects);
* param_1 := obj.Pick();
*/
```

Kotlin:

```
class Solution(rects: Array<IntArray>) {

fun pick(): IntArray {

}

}

/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(rects)
* var param_1 = obj.pick()
*/
```

Swift:

```

class Solution {

init(_ rects: [[Int]]) {

}

func pick() -> [Int] {
}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution(rects)
* let ret_1: [Int] = obj.pick()
*/
```

Rust:

```
struct Solution {

}



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

fn new(rects: Vec<Vec<i32>>) -> Self {

}

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

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(rects);
* let ret_1: Vec<i32> = obj.pick();
*/
```

Ruby:

```
class Solution

=begin
```

```
:type rects: Integer[][]
=end
def initialize(rects)

end



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

end



end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(rects)
# param_1 = obj.pick()
```

PHP:

```
class Solution {
/**
* @param Integer[][] $rects
*/
function __construct($rects) {

}

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

}
}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($rects);
* $ret_1 = $obj->pick();
*/
```

Dart:

```
class Solution {

Solution(List<List<int>> rects) {

}

List<int> pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(rects);
* List<int> param1 = obj.pick();
*/
```

Scala:

```
class Solution(_rects: Array[Array[Int]]) {

def pick(): Array[Int] = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(rects)
* val param_1 = obj.pick()
*/
```

Elixir:

```
defmodule Solution do
@spec init_(rects :: [[integer]]) :: any
```

```
def init_(rects) do

end

@spec pick() :: [integer]
def pick() do

end
end

# Your functions will be called as such:
# Solution.init_(rects)
# param_1 = Solution.pick()

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

Erlang:

```
-spec solution_init_(Rects :: [[integer()]]) -> any().
solution_init_(Rects) ->
.

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



%% Your functions will be called as such:
%% solution_init_(Rects),
%% Param_1 = solution_pick(),

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

Racket:

```
(define solution%
(class object%
(super-new)

; rects : (listof (listof exact-integer?))
(init-field
rects)

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

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [rects rects]))
;; (define param_1 (send obj pick))
```

Solutions

C++ Solution:

```
/*
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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:
Solution(vector<vector<int>>& rects) {

}

vector<int> pick() {

}
};

/**
* Your Solution object will be instantiated and called as such:
* Solution* obj = new Solution(rects);
* vector<int> param_1 = obj->pick();
```

```
*/
```

Java Solution:

```
/**
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution(int[][] rects) {

}

public int[] pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(rects);
* int[] param_1 = obj.pick();
*/
```

Python3 Solution:

```
"""
Problem: Random Point in Non-overlapping Rectangles
Difficulty: Medium
Tags: array, math, search

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 __init__(self, rects: List[List[int]]):



def pick(self) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):

def __init__(self, rects):
"""
:type rects: List[List[int]]
"""



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




# Your Solution object will be instantiated and called as such:
# obj = Solution(rects)
# param_1 = obj.pick()
```

JavaScript Solution:

```
/**
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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[][]} rects
*/
var Solution = function(rects) {

};

/**
* @return {number[]}
*/
Solution.prototype.pick = function() {

};

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(rects)
* var param_1 = obj.pick()
*/
```

TypeScript Solution:

```
/**
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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 {
constructor(rects: number[][]) {

}

pick(): number[] {
}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(rects)
* var param_1 = obj.pick()
*/
```

C# Solution:

```
/*
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution(int[][] rects) {

}

public int[] Pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(rects);
* int[] param_1 = obj.Pick();
*/
```

C Solution:

```
/*
* Problem: Random Point in Non-overlapping Rectangles
* Difficulty: Medium
* Tags: array, math, search
*
* 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
*/




typedef struct {

} Solution;


Solution* solutionCreate(int** rects, int rectsSize, int* rectsColSize) {

}

int* solutionPick(Solution* obj, int* retSize) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(rects, rectsSize, rectsColSize);
* int* param_1 = solutionPick(obj, retSize);

* solutionFree(obj);
*/
```

Go Solution:

```
// Problem: Random Point in Non-overlapping Rectangles
// Difficulty: Medium
// Tags: array, math, search
//
// 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

type Solution struct {

}


func Constructor(rects [][]int) Solution {

}


func (this *Solution) Pick() []int {

}


/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(rects);
* param_1 := obj.Pick();
*/
```

Kotlin Solution:

```
class Solution(rects: Array<IntArray>) {

fun pick(): IntArray {

}

}

/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(rects)
* var param_1 = obj.pick()
```

```
*/
```

Swift Solution:

```
class Solution {

init(_ rects: [[Int]]) {

}

func pick() -> [Int] {

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution(rects)
* let ret_1: [Int] = obj.pick()
*/
```

Rust Solution:

```
// Problem: Random Point in Non-overlapping Rectangles
// Difficulty: Medium
// Tags: array, math, search
//
// 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

struct Solution {

}



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

```
impl Solution {

fn new(rects: Vec<Vec<i32>>) -> Self {

}

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

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(rects);
* let ret_1: Vec<i32> = obj.pick();
*/
```

Ruby Solution:

```
class Solution

=begin
:type rects: Integer[][]
=end
def initialize(rects)

end



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

end



end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(rects)
# param_1 = obj.pick()
```

PHP Solution:

```
class Solution {
/**
* @param Integer[][] $rects
*/
function __construct($rects) {

}

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

}
}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($rects);
* $ret_1 = $obj->pick();
*/
```

Dart Solution:

```
class Solution {

Solution(List<List<int>> rects) {

}

List<int> pick() {

}
}

/**
* Your Solution object will be instantiated and called as such:
```

```
* Solution obj = Solution(rects);
* List<int> param1 = obj.pick();
*/
```

Scala Solution:

```
class Solution(_rects: Array[Array[Int]]) {

def pick(): Array[Int] = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(rects)
* val param_1 = obj.pick()
*/
```

Elixir Solution:

```
defmodule Solution do
@spec init_(rects :: [[integer]]) :: any
def init_(rects) do

end

@spec pick() :: [integer]
def pick() do

end
end

# Your functions will be called as such:
# Solution.init_(rects)
# param_1 = Solution.pick()

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

Erlang Solution:

```
-spec solution_init_(Rects :: [[integer()]]) -> any().
solution_init_(Rects) ->
.

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



%% Your functions will be called as such:
%% solution_init_(Rects),
%% Param_1 = solution_pick(),

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

Racket Solution:

```
(define solution%
(class object%
(super-new)

; rects : (listof (listof exact-integer?))
(init-field
rects)

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

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [rects rects]))
;; (define param_1 (send obj pick))
```