Problem 3391: Design a 3D Binary Matrix with Efficient Layer Tracking

Problem Information

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

Problem Description

You are given a

n x n x n

binary

3D array

matrix

.

Implement the

Matrix3D

class:

Matrix3D(int n)

Initializes the object with the 3D binary array

matrix

, where

all

elements are initially set to 0.

void setCell(int x, int y, int z)

Sets the value at

matrix[x][y][z]

to 1.

void unsetCell(int x, int y, int z)

Sets the value at

matrix[x][y][z]

to 0.

int largestMatrix()

Returns the index

x

where

matrix[x]

contains the most number of 1's. If there are multiple such indices, return the

largest

x

.

Example 1:

Input:

["Matrix3D", "setCell", "largestMatrix", "setCell", "largestMatrix", "setCell", "largestMatrix"]

[[3], [0, 0, 0], [], [1, 1, 2], [], [0, 0, 1], []]

Output:

[null, null, 0, null, 1, null, 0]

Explanation

Matrix3D matrix3D = new Matrix3D(3); // Initializes a

3 x 3 x 3

3D array

matrix

, filled with all 0's.

matrix3D.setCell(0, 0, 0); // Sets

matrix[0][0][0]

to 1.

matrix3D.largestMatrix(); // Returns 0.

matrix[0]

has the most number of 1's.

matrix3D.setCell(1, 1, 2); // Sets

matrix[1][1][2]

to 1.

matrix3D.largestMatrix(); // Returns 1.

matrix[0]

and

matrix[1]

tie with the most number of 1's, but index 1 is bigger.

matrix3D.setCell(0, 0, 1); // Sets

matrix[0][0][1]

to 1.

matrix3D.largestMatrix(); // Returns 0.

matrix[0]

has the most number of 1's.

Example 2:

Input:

["Matrix3D", "setCell", "largestMatrix", "unsetCell", "largestMatrix"]

[[4], [2, 1, 1], [], [2, 1, 1], []]

Output:

[null, null, 2, null, 3]

Explanation

Matrix3D matrix3D = new Matrix3D(4); // Initializes a

4 x 4 x 4

3D array

matrix

, filled with all 0's.

matrix3D.setCell(2, 1, 1); // Sets

matrix[2][1][1]

to 1.

matrix3D.largestMatrix(); // Returns 2.

matrix[2]

has the most number of 1's.

matrix3D.unsetCell(2, 1, 1); // Sets

matrix[2][1][1]

to 0.

matrix3D.largestMatrix(); // Returns 3. All indices from 0 to 3 tie with the same number of 1's, but index 3 is the biggest.

Constraints:

1 <= n <= 100

0 <= x, y, z < n

At most

10

5

calls are made in total to

setCell

and

unsetCell

.

At most

10

4

calls are made to

largestMatrix

.

Code Snippets

C++:

```
class Matrix3D {
public:
Matrix3D(int n) {

}

void setCell(int x, int y, int z) {

}
void unsetCell(int x, int y, int z) {

}

int largestMatrix() {

}
};

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D* obj = new Matrix3D(n);
* obj->setCell(x,y,z);
* obj->unsetCell(x,y,z);
* int param_3 = obj->largestMatrix();
*/
```

Java:

```
class Matrix3D {

public Matrix3D(int n) {

}

public void setCell(int x, int y, int z) {

}

public void unsetCell(int x, int y, int z) {

}

public int largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = new Matrix3D(n);
* obj.setCell(x,y,z);
* obj.unsetCell(x,y,z);
* int param_3 = obj.largestMatrix();
*/
```

Python3:

```
class Matrix3D:

def __init__(self, n: int):


def setCell(self, x: int, y: int, z: int) -> None:


def unsetCell(self, x: int, y: int, z: int) -> None:


def largestMatrix(self) -> int:



# Your Matrix3D object will be instantiated and called as such:
# obj = Matrix3D(n)
# obj.setCell(x,y,z)
# obj.unsetCell(x,y,z)
# param_3 = obj.largestMatrix()
```

Python:

```
class Matrix3D(object):

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


def setCell(self, x, y, z):
"""
:type x: int
:type y: int
```

```
:type z: int
:rtype: None
"""


def unsetCell(self, x, y, z):
"""
:type x: int
:type y: int
:type z: int
:rtype: None
"""


def largestMatrix(self):
"""
:rtype: int
"""



# Your Matrix3D object will be instantiated and called as such:
# obj = Matrix3D(n)
# obj.setCell(x,y,z)
# obj.unsetCell(x,y,z)
# param_3 = obj.largestMatrix()
```

JavaScript:

```
/**
* @param {number} n
*/
var Matrix3D = function(n) {

};

/**
* @param {number} x
* @param {number} y
* @param {number} z
* @return {void}
*/
```

```
Matrix3D.prototype.setCell = function(x, y, z) {

};

/**
* @param {number} x
* @param {number} y
* @param {number} z
* @return {void}
*/
Matrix3D.prototype.unsetCell = function(x, y, z) {

};

/**
* @return {number}
*/
Matrix3D.prototype.largestMatrix = function() {

};

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

TypeScript:

```
class Matrix3D {
constructor(n: number) {

}

setCell(x: number, y: number, z: number): void {

}

unsetCell(x: number, y: number, z: number): void {
}

largestMatrix(): number {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

C#:

```
public class Matrix3D {

public Matrix3D(int n) {

}

public void SetCell(int x, int y, int z) {

}

public void UnsetCell(int x, int y, int z) {

}

public int LargestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = new Matrix3D(n);
* obj.SetCell(x,y,z);
* obj.UnsetCell(x,y,z);
* int param_3 = obj.LargestMatrix();
```

```
*/
```

C:

```
typedef struct {

} Matrix3D;


Matrix3D* matrix3DCreate(int n) {

}

void matrix3DSetCell(Matrix3D* obj, int x, int y, int z) {

}

void matrix3DUnsetCell(Matrix3D* obj, int x, int y, int z) {

}

int matrix3DLargestMatrix(Matrix3D* obj) {

}

void matrix3DFree(Matrix3D* obj) {

}

/**
* Your Matrix3D struct will be instantiated and called as such:
* Matrix3D* obj = matrix3DCreate(n);
* matrix3DSetCell(obj, x, y, z);

* matrix3DUnsetCell(obj, x, y, z);

* int param_3 = matrix3DLargestMatrix(obj);

* matrix3DFree(obj);
```

```
*/
```

Go:

```
type Matrix3D struct {

}


func Constructor(n int) Matrix3D {

}


func (this *Matrix3D) SetCell(x int, y int, z int) {

}


func (this *Matrix3D) UnsetCell(x int, y int, z int) {

}


func (this *Matrix3D) LargestMatrix() int {

}


/**
* Your Matrix3D object will be instantiated and called as such:
* obj := Constructor(n);
* obj.SetCell(x,y,z);
* obj.UnsetCell(x,y,z);
* param_3 := obj.LargestMatrix();
*/
```

Kotlin:

```
class Matrix3D(n: Int) {

fun setCell(x: Int, y: Int, z: Int) {
}

fun unsetCell(x: Int, y: Int, z: Int) {

}

fun largestMatrix(): Int {

}

}

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

Swift:

```
class Matrix3D {

init(_ n: Int) {

}

func setCell(_ x: Int, _ y: Int, _ z: Int) {

}

func unsetCell(_ x: Int, _ y: Int, _ z: Int) {

}

func largestMatrix() -> Int {

}
}
```

```
/**
* Your Matrix3D object will be instantiated and called as such:
* let obj = Matrix3D(n)
* obj.setCell(x, y, z)
* obj.unsetCell(x, y, z)
* let ret_3: Int = obj.largestMatrix()
*/
```

Rust:

```
struct Matrix3D {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Matrix3D {

fn new(n: i32) -> Self {

}

fn set_cell(&self, x: i32, y: i32, z: i32) {

}

fn unset_cell(&self, x: i32, y: i32, z: i32) {

}

fn largest_matrix(&self) -> i32 {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* let obj = Matrix3D::new(n);
* obj.set_cell(x, y, z);
* obj.unset_cell(x, y, z);
* let ret_3: i32 = obj.largest_matrix();
*/
```

Ruby:

```
class Matrix3D

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

end



=begin
:type x: Integer
:type y: Integer
:type z: Integer
:rtype: Void
=end
def set_cell(x, y, z)

end



=begin
:type x: Integer
:type y: Integer
:type z: Integer
:rtype: Void
=end
def unset_cell(x, y, z)

end



=begin
:rtype: Integer
=end
```

```
def largest_matrix()

end


end

# Your Matrix3D object will be instantiated and called as such:
# obj = Matrix3D.new(n)
# obj.set_cell(x, y, z)
# obj.unset_cell(x, y, z)
# param_3 = obj.largest_matrix()
```

PHP:

```
class Matrix3D {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return NULL
*/
function setCell($x, $y, $z) {

}

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return NULL
*/
function unsetCell($x, $y, $z) {

}
/**
* @return Integer
*/
function largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* $obj = Matrix3D($n);
* $obj->setCell($x, $y, $z);
* $obj->unsetCell($x, $y, $z);
* $ret_3 = $obj->largestMatrix();
*/
```

Dart:

```
class Matrix3D {

Matrix3D(int n) {

}

void setCell(int x, int y, int z) {

}

void unsetCell(int x, int y, int z) {

}

int largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = Matrix3D(n);
* obj.setCell(x,y,z);
* obj.unsetCell(x,y,z);
* int param3 = obj.largestMatrix();
*/
```

Scala:

```
class Matrix3D(_n: Int) {

def setCell(x: Int, y: Int, z: Int): Unit = {

}

def unsetCell(x: Int, y: Int, z: Int): Unit = {

}

def largestMatrix(): Int = {

}

}

/**
* Your Matrix3D object will be instantiated and called as such:
* val obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* val param_3 = obj.largestMatrix()
*/
```

Elixir:

```
defmodule Matrix3D do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec set_cell(x :: integer, y :: integer, z :: integer) :: any
def set_cell(x, y, z) do

end
```

```
@spec unset_cell(x :: integer, y :: integer, z :: integer) :: any
def unset_cell(x, y, z) do

end

@spec largest_matrix() :: integer
def largest_matrix() do

end
end

# Your functions will be called as such:
# Matrix3D.init_(n)
# Matrix3D.set_cell(x, y, z)
# Matrix3D.unset_cell(x, y, z)
# param_3 = Matrix3D.largest_matrix()

# Matrix3D.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang:

```
-spec matrix3_d_init_(N :: integer()) -> any().
matrix3_d_init_(N) ->
.

-spec matrix3_d_set_cell(X :: integer(), Y :: integer(), Z :: integer()) ->
any().
matrix3_d_set_cell(X, Y, Z) ->
.

-spec matrix3_d_unset_cell(X :: integer(), Y :: integer(), Z :: integer()) ->
any().
matrix3_d_unset_cell(X, Y, Z) ->
.

-spec matrix3_d_largest_matrix() -> integer().
matrix3_d_largest_matrix() ->
.
```

```
%% Your functions will be called as such:
%% matrix3_d_init_(N),
%% matrix3_d_set_cell(X, Y, Z),
%% matrix3_d_unset_cell(X, Y, Z),
%% Param_3 = matrix3_d_largest_matrix(),

%% matrix3_d_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket:

```
(define matrix3-d%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; set-cell : exact-integer? exact-integer? exact-integer? -> void?
(define/public (set-cell x y z)
)
; unset-cell : exact-integer? exact-integer? exact-integer? -> void?
(define/public (unset-cell x y z)
)
; largest-matrix : -> exact-integer?
(define/public (largest-matrix)
)))

;; Your matrix3-d% object will be instantiated and called as such:
;; (define obj (new matrix3-d% [n n]))
;; (send obj set-cell x y z)
;; (send obj unset-cell x y z)
;; (define param_3 (send obj largest-matrix))
```

Solutions

C++ Solution:

```
/*
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Matrix3D {
public:
Matrix3D(int n) {

}

void setCell(int x, int y, int z) {

}

void unsetCell(int x, int y, int z) {

}

int largestMatrix() {

}
};

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D* obj = new Matrix3D(n);
* obj->setCell(x,y,z);
* obj->unsetCell(x,y,z);
* int param_3 = obj->largestMatrix();
*/
```

Java Solution:

```
/**
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Matrix3D {

public Matrix3D(int n) {

}

public void setCell(int x, int y, int z) {

}

public void unsetCell(int x, int y, int z) {

}

public int largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = new Matrix3D(n);
* obj.setCell(x,y,z);
* obj.unsetCell(x,y,z);
* int param_3 = obj.largestMatrix();
*/
```

Python3 Solution:

```
"""
Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
Difficulty: Medium
Tags: array, hash, queue, heap
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Matrix3D:

def __init__(self, n: int):



def setCell(self, x: int, y: int, z: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Matrix3D(object):

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



def setCell(self, x, y, z):
"""
:type x: int
:type y: int
:type z: int
:rtype: None
"""



def unsetCell(self, x, y, z):
"""
:type x: int
:type y: int
:type z: int
:rtype: None
"""
```

```
def largestMatrix(self):
"""
:rtype: int
"""



# Your Matrix3D object will be instantiated and called as such:
# obj = Matrix3D(n)
# obj.setCell(x,y,z)
# obj.unsetCell(x,y,z)
# param_3 = obj.largestMatrix()
```

JavaScript Solution:

```
/**
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
*/
var Matrix3D = function(n) {

};

/**
* @param {number} x
* @param {number} y
* @param {number} z
* @return {void}
*/
Matrix3D.prototype.setCell = function(x, y, z) {
};

/**
* @param {number} x
* @param {number} y
* @param {number} z
* @return {void}
*/
Matrix3D.prototype.unsetCell = function(x, y, z) {

};

/**
* @return {number}
*/
Matrix3D.prototype.largestMatrix = function() {

};

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

TypeScript Solution:

```
/**
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Matrix3D {
constructor(n: number) {
}

setCell(x: number, y: number, z: number): void {

}

unsetCell(x: number, y: number, z: number): void {

}

largestMatrix(): number {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

C# Solution:

```
/*
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Matrix3D {

public Matrix3D(int n) {

}
public void SetCell(int x, int y, int z) {

}

public void UnsetCell(int x, int y, int z) {

}

public int LargestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = new Matrix3D(n);
* obj.SetCell(x,y,z);
* obj.UnsetCell(x,y,z);
* int param_3 = obj.LargestMatrix();
*/
```

C Solution:

```
/*
* Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
* Difficulty: Medium
* Tags: array, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} Matrix3D;
```

```
Matrix3D* matrix3DCreate(int n) {

}

void matrix3DSetCell(Matrix3D* obj, int x, int y, int z) {

}

void matrix3DUnsetCell(Matrix3D* obj, int x, int y, int z) {

}

int matrix3DLargestMatrix(Matrix3D* obj) {

}

void matrix3DFree(Matrix3D* obj) {

}

/**
* Your Matrix3D struct will be instantiated and called as such:
* Matrix3D* obj = matrix3DCreate(n);
* matrix3DSetCell(obj, x, y, z);

* matrix3DUnsetCell(obj, x, y, z);

* int param_3 = matrix3DLargestMatrix(obj);

* matrix3DFree(obj);
*/
```

Go Solution:

```
// Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type Matrix3D struct {

}


func Constructor(n int) Matrix3D {

}


func (this *Matrix3D) SetCell(x int, y int, z int) {

}


func (this *Matrix3D) UnsetCell(x int, y int, z int) {

}


func (this *Matrix3D) LargestMatrix() int {

}


/**
* Your Matrix3D object will be instantiated and called as such:
* obj := Constructor(n);
* obj.SetCell(x,y,z);
* obj.UnsetCell(x,y,z);
* param_3 := obj.LargestMatrix();
*/
```

Kotlin Solution:

```
class Matrix3D(n: Int) {

fun setCell(x: Int, y: Int, z: Int) {
}

fun unsetCell(x: Int, y: Int, z: Int) {

}

fun largestMatrix(): Int {

}

}

/**
* Your Matrix3D object will be instantiated and called as such:
* var obj = Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* var param_3 = obj.largestMatrix()
*/
```

Swift Solution:

```
class Matrix3D {

init(_ n: Int) {

}

func setCell(_ x: Int, _ y: Int, _ z: Int) {

}

func unsetCell(_ x: Int, _ y: Int, _ z: Int) {

}

func largestMatrix() -> Int {

}
}
```

```
/**
* Your Matrix3D object will be instantiated and called as such:
* let obj = Matrix3D(n)
* obj.setCell(x, y, z)
* obj.unsetCell(x, y, z)
* let ret_3: Int = obj.largestMatrix()
*/
```

Rust Solution:

```
// Problem: Design a 3D Binary Matrix with Efficient Layer Tracking
// Difficulty: Medium
// Tags: array, hash, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct Matrix3D {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl Matrix3D {

fn new(n: i32) -> Self {

}

fn set_cell(&self, x: i32, y: i32, z: i32) {

}

fn unset_cell(&self, x: i32, y: i32, z: i32) {

}
fn largest_matrix(&self) -> i32 {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* let obj = Matrix3D::new(n);
* obj.set_cell(x, y, z);
* obj.unset_cell(x, y, z);
* let ret_3: i32 = obj.largest_matrix();
*/
```

Ruby Solution:

```
class Matrix3D

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

end



=begin
:type x: Integer
:type y: Integer
:type z: Integer
:rtype: Void
=end
def set_cell(x, y, z)

end



=begin
:type x: Integer
:type y: Integer
:type z: Integer
```

```
:rtype: Void
=end
def unset_cell(x, y, z)

end



=begin
:rtype: Integer
=end
def largest_matrix()

end



end

# Your Matrix3D object will be instantiated and called as such:
# obj = Matrix3D.new(n)
# obj.set_cell(x, y, z)
# obj.unset_cell(x, y, z)
# param_3 = obj.largest_matrix()
```

PHP Solution:

```
class Matrix3D {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return NULL
*/
function setCell($x, $y, $z) {

}

/**
* @param Integer $x
* @param Integer $y
* @param Integer $z
* @return NULL
*/
function unsetCell($x, $y, $z) {

}

/**
* @return Integer
*/
function largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* $obj = Matrix3D($n);
* $obj->setCell($x, $y, $z);
* $obj->unsetCell($x, $y, $z);
* $ret_3 = $obj->largestMatrix();
*/
```

Dart Solution:

```
class Matrix3D {

Matrix3D(int n) {

}

void setCell(int x, int y, int z) {

}

void unsetCell(int x, int y, int z) {
}

int largestMatrix() {

}
}

/**
* Your Matrix3D object will be instantiated and called as such:
* Matrix3D obj = Matrix3D(n);
* obj.setCell(x,y,z);
* obj.unsetCell(x,y,z);
* int param3 = obj.largestMatrix();
*/
```

Scala Solution:

```
class Matrix3D(_n: Int) {

def setCell(x: Int, y: Int, z: Int): Unit = {

}

def unsetCell(x: Int, y: Int, z: Int): Unit = {

}

def largestMatrix(): Int = {

}

}

/**
* Your Matrix3D object will be instantiated and called as such:
* val obj = new Matrix3D(n)
* obj.setCell(x,y,z)
* obj.unsetCell(x,y,z)
* val param_3 = obj.largestMatrix()
*/
```

Elixir Solution:

```
defmodule Matrix3D do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec set_cell(x :: integer, y :: integer, z :: integer) :: any
def set_cell(x, y, z) do

end

@spec unset_cell(x :: integer, y :: integer, z :: integer) :: any
def unset_cell(x, y, z) do

end

@spec largest_matrix() :: integer
def largest_matrix() do

end
end

# Your functions will be called as such:
# Matrix3D.init_(n)
# Matrix3D.set_cell(x, y, z)
# Matrix3D.unset_cell(x, y, z)
# param_3 = Matrix3D.largest_matrix()

# Matrix3D.init_ will be called before every test case, in which you can do
some necessary initializations.
```

Erlang Solution:

```
-spec matrix3_d_init_(N :: integer()) -> any().
matrix3_d_init_(N) ->
.

-spec matrix3_d_set_cell(X :: integer(), Y :: integer(), Z :: integer()) ->
any().
matrix3_d_set_cell(X, Y, Z) ->
```

```
.

-spec matrix3_d_unset_cell(X :: integer(), Y :: integer(), Z :: integer()) ->
any().
matrix3_d_unset_cell(X, Y, Z) ->
.

-spec matrix3_d_largest_matrix() -> integer().
matrix3_d_largest_matrix() ->
.



%% Your functions will be called as such:
%% matrix3_d_init_(N),
%% matrix3_d_set_cell(X, Y, Z),
%% matrix3_d_unset_cell(X, Y, Z),
%% Param_3 = matrix3_d_largest_matrix(),

%% matrix3_d_init_ will be called before every test case, in which you can do
some necessary initializations.
```

Racket Solution:

```
(define matrix3-d%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; set-cell : exact-integer? exact-integer? exact-integer? -> void?
(define/public (set-cell x y z)
)
; unset-cell : exact-integer? exact-integer? exact-integer? -> void?
(define/public (unset-cell x y z)
)
; largest-matrix : -> exact-integer?
(define/public (largest-matrix)
)))
```

```
;; Your matrix3-d% object will be instantiated and called as such:
;; (define obj (new matrix3-d% [n n]))
;; (send obj set-cell x y z)
;; (send obj unset-cell x y z)
;; (define param_3 (send obj largest-matrix))
```