Data Structures

Array Representation

Andres Mendez-Vazquez

August 25, 2016

Outline

Outline

Introduction

Observation

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

Example

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

We could use a name variable for each data

```
double score0; double score1; double score2;
double score3; double score4; double score5;
```

Introduction

Observation

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

Example

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

We could use a name variable for each data

```
double score0; double score1; double score2;
double score3; double score4; double score5;
```

Introduction

Observation

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

Example

When a program manipulates many variables that contain "similar" forms of data, organizational problems quickly arise.

We could use a name variable for each data

double score0; double score1; double score2;
double score3; double score4; double score5;

Making your life miserable

Because you can ask

Which is the highest score?

Look at the code

Making your life miserable

Because you can ask

Which is the highest score?

Look at the code

```
double high_score = score0;
if ( score1 > high_score ) { high_score = score1; }
if ( score2 > high_score ) { high_score = score2; }
if ( score3 > high_score ) { high_score = score3; }
if ( score4 > high_score ) { high_score = score4; }
if ( score5 > high_score ) { high_score = score5; }
System.out.println(high_score);
```

How do we solve this?

Thus

Java, C and C++ use array variables to put together this collection of equal elements.

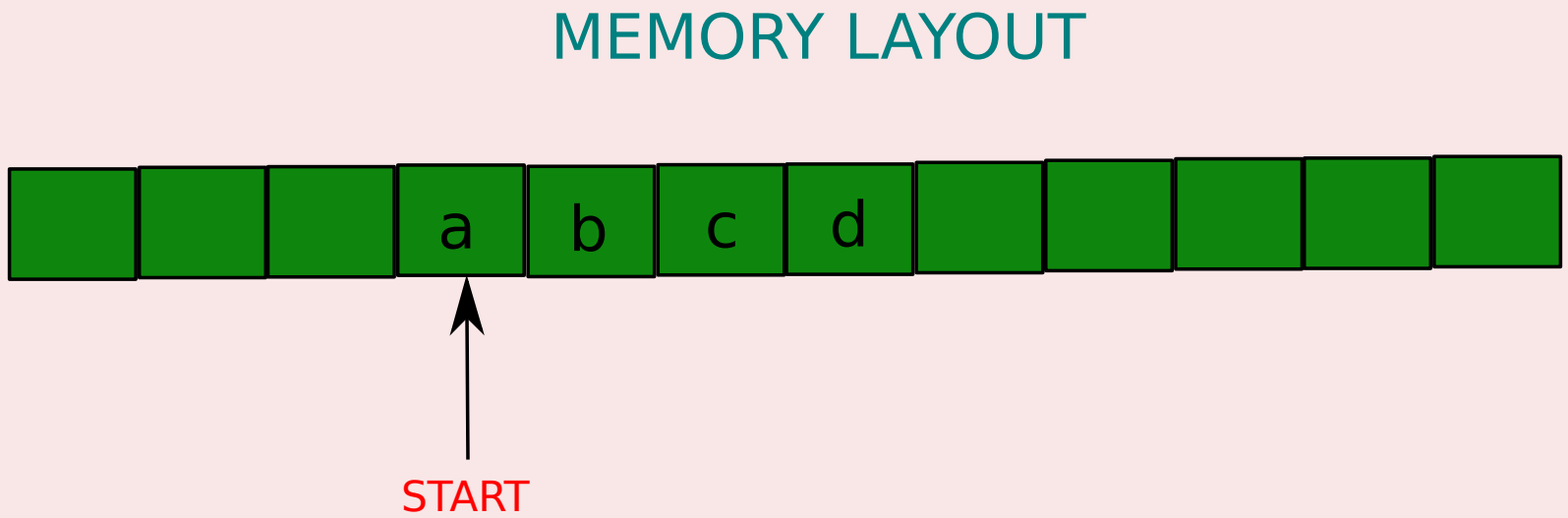
Memory Layout

How do we solve this?

Thus

Java, C and C++ use array variables to put together this collection of equal elements.

Memory Layout



Outline

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
   3. Space for the elements, for example 4 bytes for n elements

   Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

1D Array Representation In Java, C, and C++

Notes

For 1-dimensional array:

- The elements are mapped into contiguous memory locations
- They are accessed through the use of $location\,(x\,[i]) = start + i$

How much memory is used for representing an array?

1. Space Overhead
   1. Storing the start address: 4 bytes
   2. Storing x.length: 4 bytes
2. Space for the elements, for example 4 bytes for n elements

Total: $4 + 4 + 4n = 8 + 4n$ bytes

Outline

Now, 2D Arrays

We declare 2-dimensional arrays as follow (Java, C)

int[][] A = new int[3][4]

It can be shown as a table

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

Now, 2D Arrays

We declare 2-dimensional arrays as follow (Java, C)

int[][] A = new int[3][4]

It can be shown as a table

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

Rows and Columns in a 2-D Array

Rows

A[0][0] A[0][1] A[0][2] A[0][3] ➤
A[1][0] A[1][1] A[1][2] A[1][3] ➤
A[2][0] A[2][1] A[2][2] A[2][3] ➤

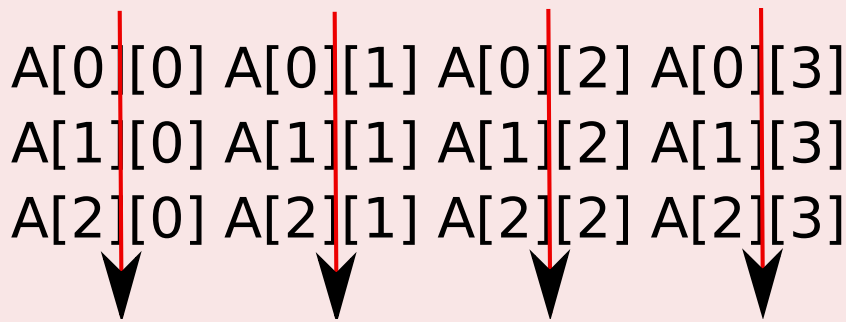
Columns

Rows and Columns in a 2-D Array

Rows

A[0][0] A[0][1] A[0][2] A[0][3]
A[1][0] A[1][1] A[1][2] A[1][3]
A[2][0] A[2][1] A[2][2] A[2][3]

Columns



Outline

2D Array Representation In Java, C, and C++

Given the following 2-dimensional array x

$$\mathrm{x} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

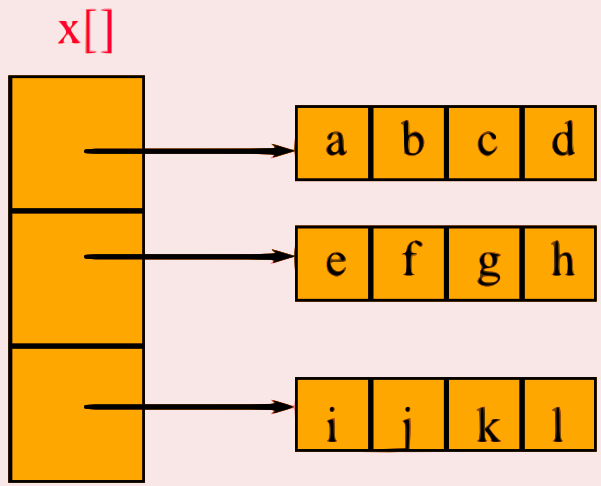
This information is stored as 1D arrays with a reference to them

2D Array Representation In Java, C, and C++

Given the following 2-dimensional array x

$$x = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

This information is stored as 1D arrays with a reference to them



Some Properties

If we call the lengths of each of them

- x.length ⇒ 3
- x[0].length === x[1].length === x[2].length ⇒ 3

Some Properties

If we call the lengths of each of them

- x.length $\Rightarrow$ 3
- x[0].length == x[1].length == x[2].length $\Rightarrow$ 3

Space Overhead

Remember, it is 8 bytes per 1D array

1. Thus, we have $4 \times 8 = 32$ bytes
2. Or we can see this as (number of rows $+ 1) \times 8$ bytes

Some Properties

If we call the lengths of each of them

- x.length $\Rightarrow$ 3
- x[0].length == x[1].length == x[2].length $\Rightarrow$ 3

Space Overhead

Remember, it is 8 bytes per 1D array

1. Thus, we have $4 \times 8 = 32$ bytes
2. Or we can see this as (number of rows $+ 1) \times 8$ bytes

Some Properties

If we call the lengths of each of them

- x.length $\Rightarrow$ 3
- x[0].length $==$ x[1].length $==$ x[2].length $\Rightarrow$ 3

Space Overhead

Remember, it is 8 bytes per 1D array

1. Thus, we have $4 \times 8 = 32$ bytes
2. Or we can see this as $(\text{number of rows} + 1) \times 8$ bytes

Some Properties

If we call the lengths of each of them

- x.length $\Rightarrow$ 3
- x[0].length $==$ x[1].length $==$ x[2].length $\Rightarrow$ 3

Space Overhead

Remember, it is 8 bytes per 1D array

1. Thus, we have $4 \times 8 = 32$ bytes
2. Or we can see this as $(\text{number of rows } + 1) \times 8$ bytes

More Properties

First

This representation is called the **array-of-arrays** representation.

Second

It requires contiguous memory of size 3 bytes, 4 bytes, 4 bytes, and 4 bytes for the 4 1D-arrays.

Third

One memory block of size **number of rows** and **number of rows** blocks of size **number of columns**.

More Properties

First

This representation is called the **array-of-arrays** representation.

Second

It requires contiguous memory of size 3 bytes, 4 bytes, 4 bytes, and 4 bytes for the 4 1D-arrays.

Third

One memory block of size **number of rows** and **number of rows** blocks of size **number of columns**.

More Properties

First

This representation is called the **array-of-arrays** representation.

Second

It requires contiguous memory of size 3 bytes, 4 bytes, 4 bytes, and 4 bytes for the 4 1D-arrays.

Third

One memory block of size **number of rows** and **number of rows** blocks of size **number of columns.**

Outline

Row-Major Mapping

Again

$$\mathsf{x} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

For this

We will convert it into 1D-array y by collecting elements by rows.

- Rows are collected from top to bottom.

Row-Major Mapping

Again

$$\mathbf{x} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

For this

We will convert it into 1D-array **y** by collecting elements by rows.

Thus

- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.

Row-Major Mapping

Again

$$\mathbf{x} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

For this

We will convert it into 1D-array **y** by collecting elements by rows.

Thus

- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.

We get

An array

$\mathbf{y[] = \{a, b, c, d, e, f, g, h, i, j, k, l\}}$

Memory Map

We get

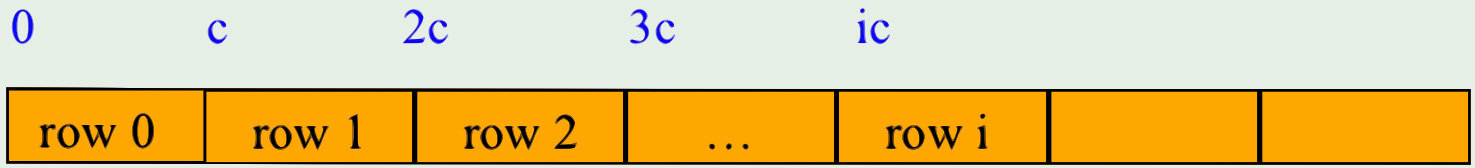
An array

y[] = {a, b, c, d, e, f, g, h, i, j, k, l}

Memory Map

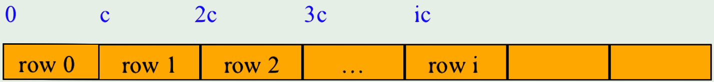
How do we locate an element?

Look At This



How do we locate an element?

Look At This

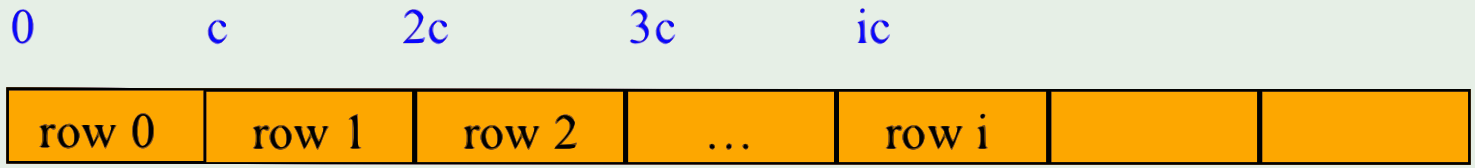


Then

- Assume **x** has **r** rows and **c** columns.
- Each row has c elements.
- i rows to the left of row i.

How do we locate an element?

Look At This



Then

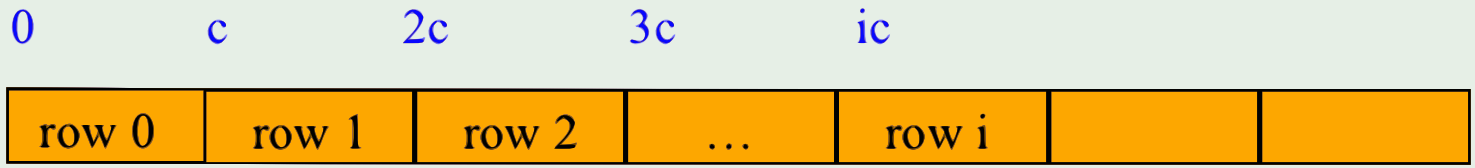
- Assume **x** has **r** rows and **c** columns.
- Each row has **c** elements.
- **i** rows to the left of row **i**.

Thus

- We have ic elements to the left of x[i][0].
- Then, x[i][j] is mapped to position ic + j of 1D array.

How do we locate an element?

Look At This



Then

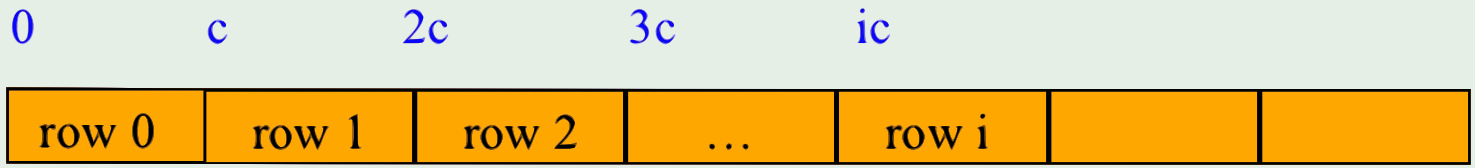
- Assume **x** has **r** rows and **c** columns.
- Each row has **c** elements.
- **i** rows to the left of row **i.**

Thus

- We have ic elements to the left of x[i][0].
- Then, x[i][j] is mapped to position ic + j of 1D array.

How do we locate an element?

Look At This



Then

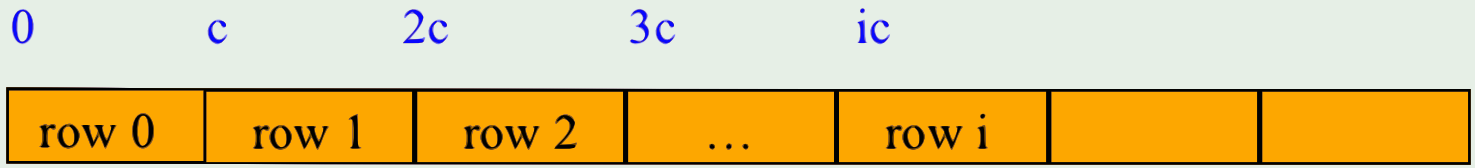
- Assume **x** has **r** rows and **c** columns.
- Each row has **c** elements.
- **i** rows to the left of row **i.**

Thus

- We have **ic** elements to the left of **x[i][0].**
- Then, x[i][j] is mapped to position ic + j of 1D array.

How do we locate an element?

Look At This



Then

- Assume **x** has **r** rows and **c** columns.
- Each row has **c** elements.
- **i** rows to the left of row **i.**

Thus

- We have **ic** elements to the left of **x[i][0].**
- Then, **x[i][j]** is mapped to position **ic + j** of 1D array.

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for **c** (number of columns)

Total: 12 bytes

Note: The number of rows = **length**/**c**

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for c (number of columns)

Total: 12 bytes

Note: The number of rows = length/c

Still

Disadvantage: Need contiguous memory of size rc.

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for **c** (number of columns)

Total: 12 bytes

Note: The number of rows = length/c

Still

Disadvantage: Need contiguous memory of size rc.

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for **c** (number of columns)

Total: 12 bytes

Note: The number of rows = length/c

Still

Disadvantage: Need contiguous memory of size rc.

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for **c** (number of columns)

Total: 12 bytes

Note: The number of rows = **length/c**

Still

Disadvantage: Need contiguous memory of size rc.

Space Overhead

We have

- 4 bytes for **start** of 1D array.
- 4 bytes for **length** of 1D array.
- 4 bytes for **c** (number of columns)

Total: 12 bytes

Note: The number of rows = **length/c**

Still

Disadvantage: Need contiguous memory of size rc.

Column-Major Mapping

Similar Setup

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.

Column-Major Mapping

Similar Setup

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.

Thus

$$x = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

Column-Major Mapping

Similar Setup

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.

Thus

$$x = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

Column-Major Mapping

Similar Setup

- Convert into 1D array y by collecting elements by columns.
- Within a column elements are collected from top to bottom.
- Columns are collected from left to right.

Thus

$$\mathsf{x} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{bmatrix}$$

We get y = {a, e, i, b, f, j, c, g, k, d, h, l}

Outline

Matrix

Something Notable

- Table of values.
- It has rows and columns, but numbering begins at 1 rather than 0.

Matrix

Something Notable

- Table of values.
- It has rows and columns, but numbering begins at 1 rather than 0.

We use the following notation

Use notation x(i,j) rather than x[i][j].

Matrix

Something Notable

- Table of values.
- It has rows and columns, but numbering begins at 1 rather than 0.

We use the following notation

Use notation x(i,j) rather than x[i][j].

Note

It may use a 2D array to represent a matrix.

Drawbacks of using a 2D Array

First

Indexes are off by 1.

Second

Java arrays do not support matrix operations such as add, transpose, multiply, and so on.

However

You can develop a class Matrix for object-oriented support of all matrix operations.

Drawbacks of using a 2D Array

First

Indexes are off by 1.

Second

Java arrays do not support matrix operations such as add, transpose, multiply, and so on.

However

You can develop a class Matrix for object-oriented support of all matrix operations.

Drawbacks of using a 2D Array

First

Indexes are off by 1.

Second

Java arrays do not support matrix operations such as add, transpose, multiply, and so on.

However

You can develop a class Matrix for object-oriented support of all matrix operations.

Outline

Some Basic Matrices: Diagonal Matrix

Diagonal Matrix

An $n \times n$ matrix in which all nonzero terms are on the diagonal.

Properties

- x(i,j) is on diagonal iff i = j

Example

$$x = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix}$$

Some Basic Matrices: Diagonal Matrix

Diagonal Matrix

An $n \times n$ matrix in which all nonzero terms are on the diagonal.

Properties

- **x(i,j)** is on diagonal iff **i = j**

Example

$$x = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix}$$

Some Basic Matrices: Diagonal Matrix

Diagonal Matrix

An $n \times n$ matrix in which all nonzero terms are on the diagonal.

Properties

- **x(i,j)** is on diagonal iff **i = j**

Example

$$\mathsf{x} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & d \end{bmatrix}$$

Some Basic Matrices: Diagonal Matrix

Properties

x(i,j) is on diagonal iff $\mathbf{i = j}$

DRAWBACK

For a $n \times n$ you are required to have:

- For example using integers: $4n^2$ bytes...

What to do?

Store diagonal only vs n^2 whole

Some Basic Matrices: Diagonal Matrix

Properties

x(i,j) is on diagonal iff **i = j**

DRAWBACK

For a $n \times n$ you are required to have:

- For example using integers: $4n^2$ bytes...

What to do?

Store diagonal only vs n^2 whole

Some Basic Matrices: Diagonal Matrix

Properties

x(i,j) is on diagonal iff **i = j**

DRAWBACK

For a $n \times n$ you are required to have:

- For example using integers: $4n^2$ bytes...

What to do?

Store diagonal only vs n^2 whole

Some Basic Matrices: Lower Triangular Matrix

Definition

An $n \times n$ matrix in which all nonzero terms are either on or below the diagonal.

Example

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

Some Basic Matrices: Lower Triangular Matrix

Definition

An $n \times n$ matrix in which all nonzero terms are either on or below the diagonal.

Example

$$\mathsf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

Some Basic Matrices: Lower Triangular Matrix

Properties

x(i,j) is part of lower triangle iff i >= j.

Number of elements in lower triangle is

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \quad (1)$$

Thus

You can store only the lower triangle

Some Basic Matrices: Lower Triangular Matrix

Properties

x(i,j) is part of lower triangle iff i >= j.

Number of elements in lower triangle is

$$1 + 2 + 3 + ... + n = \frac{n\,(n+1)}{2} \tag{1}$$

Thus

You can store only the lower triangle

Some Basic Matrices: Lower Triangular Matrix

Properties

x(i,j) is part of lower triangle iff i >= j.

Number of elements in lower triangle is

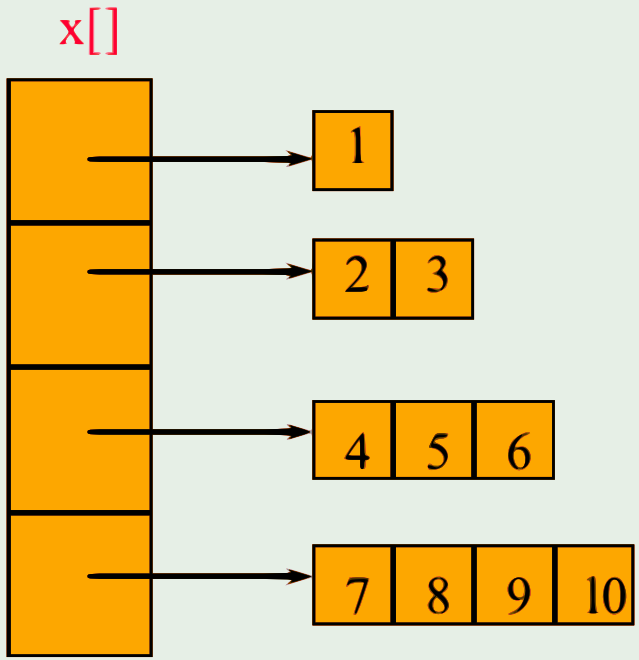
$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \quad (1)$$

Thus

You can store only the lower triangle

Array Of Arrays Representation

We can use this

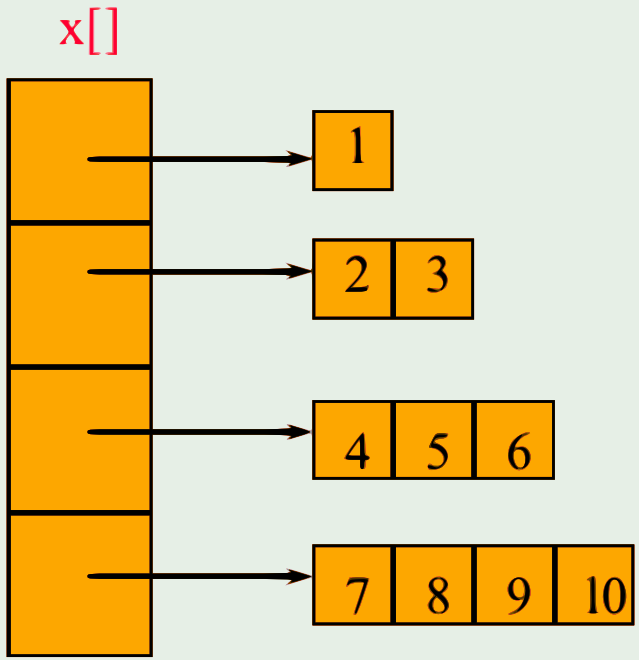


Something Notable

You can use an irregular 2-D array ... length of rows is not required to be the same.

Array Of Arrays Representation

We can use this



Something Notable

You can use an irregular 2-D array ... length of rows is not required to be the same.

Code

Code for irregular array

```
// declare a two-dimensional array variable
// and allocate the desired number of rows
int [][] irregularArray = new int [numberOfRows][];

// now allocate space for the elements in each row
for (int i = 0; i < numberOfRows; i++)
        irregularArray[i] = new int [size[i]];

// use the array like any regular array
irregularArray[2][3] = 5;
irregularArray[4][6] = irregularArray[2][3] + 2;
irregularArray[1][1] += 3;
```

However, You can do better

Map Lower Triangular Array Into A 1D Array

You can use a row-major order, but omit terms that are not part of the lower triangle.

For example

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

We get

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

However, You can do better

Map Lower Triangular Array Into A 1D Array

You can use a row-major order, but omit terms that are not part of the lower triangle.

For example

$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

We get

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

However, You can do better

Map Lower Triangular Array Into A 1D Array

You can use a row-major order, but omit terms that are not part of the lower triangle.

For example

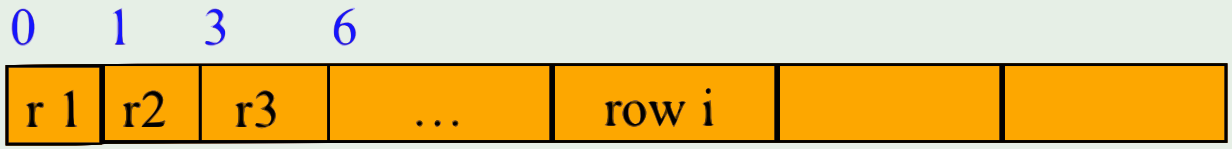
$$x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 10 \end{bmatrix}$$

We get

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

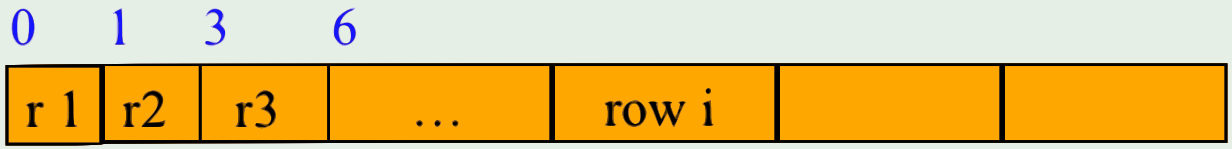
The final mapping

We want



The final mapping

We want

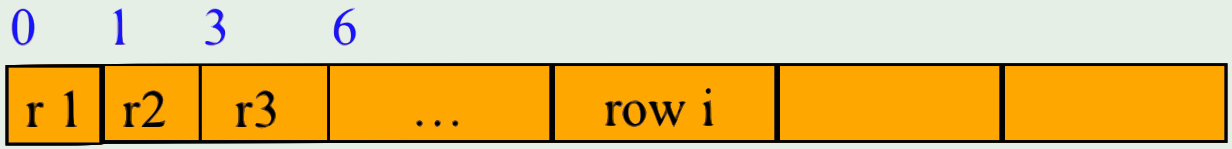


We have that

1. Order is: row 1, row 2, row 3, ...
2. Row i is preceded by rows 1, 2, ..., i-1
3. Size of row i is i.

The final mapping

We want

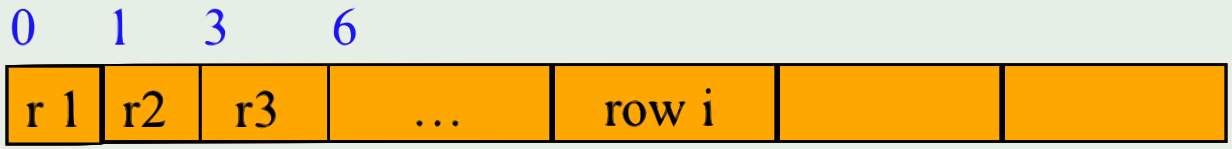


We have that

1. Order is: row 1, row 2, row 3, ...
2. Row **i** is preceded by rows **1**, **2**, ..., **i-1**
3. Size of row **i** is **i**.

The final mapping

We want



We have that

1. Order is: row 1, row 2, row 3, ...
2. Row **i** is preceded by rows **1**, **2**, ..., **i-1**
3. Size of row **i** is **i**.

More

Find the total number of elements before row i

$$1 + 2 + 3 + ... + i - 1 = \frac{i\,(i-1)}{2} \tag{2}$$

Thus

So element (i,j) is at position i(i-1)/2 + j -1 of the 1D array.

More

Find the total number of elements before row i

$$1 + 2 + 3 + ... + i - 1 = \frac{i\,(i-1)}{2} \tag{2}$$

Thus

So element **(i,j)** is at position **i(i-1)/2 + j -1** of the 1D array.

Now the Interface Matrix

You will need this for your homework

```
public interface Matrix<Item>{
    public Item get(int row, int column);
    public void add(int row, int column,Item myobj);
    public Item remove(int row, int column);
    public void output();
}
```

Outline

Sparse Matrices

Definition

A sparse array is simply an array most of whose entries are zero (or null, or some other default value).

For Example

Suppose you wanted a 2-dimensional array of course grades, whose rows are students and whose columns are courses.

- There are about 5000 courses.
- This array would have about 4500 zeroes.

Sparse Matrices

Definition

A sparse array is simply an array most of whose entries are zero (or null, or some other default value).

For Example

Suppose you wanted a 2-dimensional array of course grades, whose rows are students and whose columns are courses.

Properties

- There are about 90,173 students
- There are about 5000 courses
- This array would have about 450,865,000 entries

Sparse Matrices

Definition

A sparse array is simply an array most of whose entries are zero (or null, or some other default value).

For Example

Suppose you wanted a 2-dimensional array of course grades, whose rows are students and whose columns are courses.

Properties

- There are about 90,173 students
- There are about 5000 courses
- This array would have about 450,865,000 entries

Sparse Matrices

Definition

A sparse array is simply an array most of whose entries are zero (or null, or some other default value).

For Example

Suppose you wanted a 2-dimensional array of course grades, whose rows are students and whose columns are courses.

Properties

- There are about 90,173 students
- There are about 5000 courses
- This array would have about 450,865,000 entries

Thus

Something Notable

Since most students take fewer than 5000 courses, there will be a lot of empty spaces in this array.

Something Notable

This is a big array, even by modern standards.

Thus

Something Notable

Since most students take fewer than 5000 courses, there will be a lot of empty spaces in this array.

Something Notable

This is a big array, even by modern standards.

Example: Airline Flights

Example

Airline flight matrix.

Example: Airline Flights

Example

Airline flight matrix.

Properties

- Airports are numbered 1 through n
- flight(i,j) = list of nonstop flights from airport i to airport j

Example: Airline Flights

Example

Airline flight matrix.

Properties

- Airports are numbered 1 through n
- **flight(i,j)** = list of nonstop flights from airport **i** to airport **j**

Assume $n = 1000$ and you use an integer for representation

$n \times n$ array of list references $\Longrightarrow$ 4 million bytes when all the airports are connected

Example: Airline Flights

Example

Airline flight matrix.

Properties

- Airports are numbered 1 through n
- **flight(i,j)** = list of nonstop flights from airport **i** to airport **j**

Assume $n = 1000$ and you use an integer for representation

$n \times n$ array of list references $\Longrightarrow$ 4 million bytes when all the airports are connected

Example: Airline Flights

However

The total number of flights is way lesser $\Longrightarrow$ 20,000

Needed Storage

We need at most 20,000 list references $\Longrightarrow$ Thus, we need at most 80,000 bytes

Example: Airline Flights

However

The total number of flights is way lesser $\Longrightarrow$ 20,000

Needed Storage

We need at most 20,000 list references $\Longrightarrow$ Thus, we need at most 80,000 bytes

Web Page Matrix

Web page matrix

- Web pages are numbered 1 through **n**.
- web(i,j) = number of links from page i to page j.

Web Page Matrix

Web page matrix

- Web pages are numbered 1 through **n**.
- **web(i,j)** = number of links from page **i** to page **j**.

Web Analysis

- Authority page ... page that has many links to it.
- Hub page ... links to many authority pages.

Web Page Matrix

Web page matrix

- Web pages are numbered 1 through **n**.
- **web(i,j)** = number of links from page **i** to page **j**.

Web Analysis

- Authority page ... page that has many links to it.
- Hub page ... links to many authority pages.

Web Page Matrix

Web page matrix

- Web pages are numbered 1 through **n**.
- **web(i,j)** = number of links from page **i** to page **j**.

Web Analysis

- Authority page ... page that has many links to it.
- Hub page ... links to many authority pages.

Web Page Matrix

Something Notable

$n=$ 2,000,000,000 (and growing by 1 million a day)

Web Page Matrix

Something Notable

$n=$ 2,000,000,000 (and growing by 1 million a day)

If we used integers for representation

$n \times n$ array of integers $\Longrightarrow 16 * 10^{18}$ bytes ($16 * 10^9$ GB)

Properties

- Each page links to 10 (say) other pages on average.
- On average, there are 10 nonzero entries per row.
- Space needed for nonzero entries: approximately 80 billion bytes (80 GB)

Web Page Matrix

Something Notable

$n=$ 2,000,000,000 (and growing by 1 million a day)

If we used integers for representation

$n \times n$ array of integers $\Longrightarrow 16 * 10^{18}$ bytes $(16 * 10^{9}$ GB)

Properties

- Each page links to 10 (say) other pages on average.
- On average there are 10 nonzero entries per row.
- Space needed for non-zero elements is approximately 20,000,000,000 $\times$ 4 bytes $=$ 80,000,000,000 bytes (80 GB)

Web Page Matrix

Something Notable

$n=$ 2,000,000,000 (and growing by 1 million a day)

If we used integers for representation

$n \times n$ array of integers $\Longrightarrow 16 * 10^{18}$ bytes $(16 * 10^{9}$ GB)

Properties

- Each page links to 10 (say) other pages on average.
- On average there are 10 nonzero entries per row.
- Space needed for non-zero elements is approximately 20,000,000,000 $\times$ 4 bytes = 80,000,000,000 bytes (80 GB)

Web Page Matrix

Something Notable

$n=$ 2,000,000,000 (and growing by 1 million a day)

If we used integers for representation

$n \times n$ array of integers $\Longrightarrow 16 * 10^{18}$ bytes $(16 * 10^{9}$ GB)

Properties

- Each page links to 10 (say) other pages on average.
- On average there are 10 nonzero entries per row.
- Space needed for non-zero elements is approximately 20,000,000,000 x 4 bytes = 80,000,000,000 bytes (80 GB)

What we need...

We need...

There are ways to represent sparse arrays efficiently

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Examples of structured sparse matrices

- Diagonal
- Tridiagonal
- Lower triangular (Actually in the Middle of Sparse and Dense)

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Examples of structured sparse matrices

- Diagonal
- Tridiagonal
- Lower triangular (Actually in the Middle of Sparse and Dense)

Actually

They may be mapped into a 1D array so that a mapping function can be used to locate an element.

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Examples of structured sparse matrices

- Diagonal
- Tridiagonal
- Lower triangular (Actually in the Middle of Sparse and Dense)

Actually

They may be mapped into a 1D array so that a mapping function can be used to locate an element.

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Examples of structured sparse matrices

- Diagonal
- Tridiagonal
- Lower triangular (Actually in the Middle of Sparse and Dense)

Actually

They may be mapped into a 1D array so that a mapping function can be used to locate an element.

Sparse Matrices

Definition

- Sparse ... many elements are zero
- Dense ... few elements are zero

Examples of structured sparse matrices

- Diagonal
- Tridiagonal
- Lower triangular (Actually in the Middle of Sparse and Dense)

Actually

They may be mapped into a 1D array so that a mapping function can be used to locate an element.

Outline

Representation Of Unstructured Sparse Matrices

We can use a

Single linear list in row-major order.

Representation Of Unstructured Sparse Matrices

We can use a

Single linear list in row-major order.

Thus

- We scan the non-zero elements of the sparse matrix in row-major order.
- Each nonzero element is represented by a triple (**row, column, value**).
- The list of triples may be an array list or a linked list (chain).

Representation Of Unstructured Sparse Matrices

We can use a

Single linear list in row-major order.

Thus

- We scan the non-zero elements of the sparse matrix in row-major order.
- Each nonzero element is represented by a triple **(row, column, value)**.
- The list of triples may be an array list or a linked list (chain).

Representation Of Unstructured Sparse Matrices

We can use a

Single linear list in row-major order.

Thus

- We scan the non-zero elements of the sparse matrix in row-major order.
- Each nonzero element is represented by a triple **(row, column, value)**.
- The list of triples may be an array list or a linked list (chain).

Outline

Example with an Sparse Array

First

We will start with sparse one-dimensional arrays, which are simpler

Example

	0	1	2	3	4	5	6	7	8	9	10	11
Array	0	0	0	0	17	0	0	23	14	0	0	0

Example with an Sparse Array

First

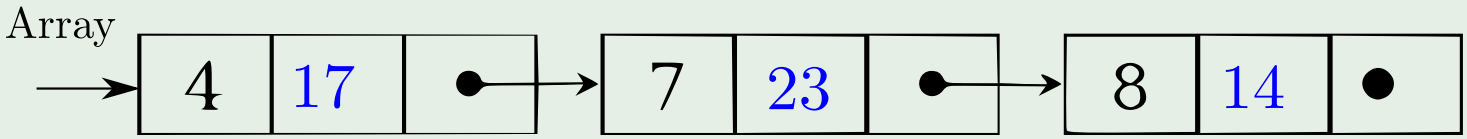
We will start with sparse one-dimensional arrays, which are simpler

Example

	0	1	2	3	4	5	6	7	8	9	10	11
Array	0	0	0	0	**17**	0	0	**23**	**14**	0	0	0

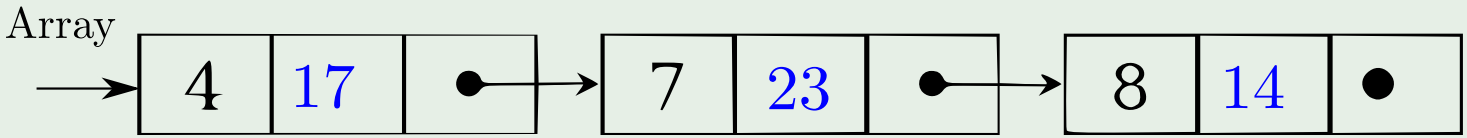
Representation

we can have the following representation



Representation

we can have the following representation

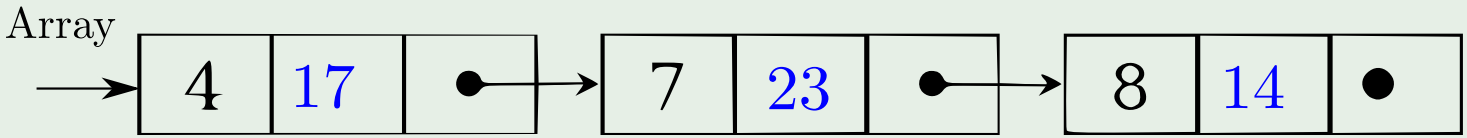


Where

1. The front element is the index.
2. The second element is the value at cell index.
3. A pointer to the next element

Representation

we can have the following representation

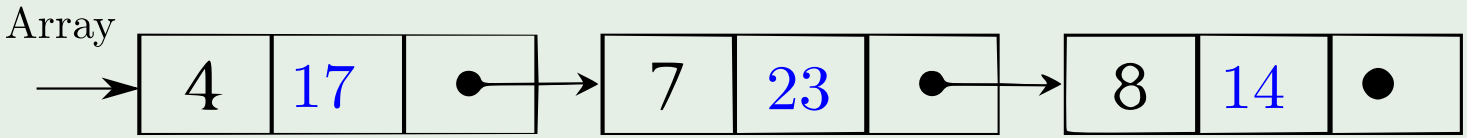


Where

1. The front element is the index.
2. The second element is the value at cell index.
3. A pointer to the next element

Representation

we can have the following representation



Where

1. The front element is the index.
2. The second element is the value at cell index.
3. A pointer to the next element

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - SparseArray(int length)
- A way to get values from the array:
  - Object get(int index)
- A way to store values in the array:
  - void add(int index, Item value)

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - void add(int index, Item value)

In addition, it is OK to ask for a value from an empty array position.

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position.

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Sparse Array ADT

For a one-dimensional array of Item

- A constructor:
  - **SparseArray(int length)**
- A way to get values from the array:
  - **Object get(int index)**
- A way to store values in the array:
  - **void add(int index, Item value)**

In addition, it is OK to ask for a value from an empty array position

1. For number, it should return ZERO.
2. For Item, it should return NULL.

Other Operations

What about?

1. **int length()**:
   - return the size of the original array
2. **int elementCount()** :
   - how many non-null values are in the array

Other Operations

What about?

1. **int length()**:
   - return the size of the original array
2. **int elementCount()** :
   - how many non-null values are in the array

Question

Do we forget something?

Other Operations

What about?

1. **int length()**:
   - return the size of the original array
2. **int elementCount()** :
   - how many non-null values are in the array

Question

Do we forget something?

Other Operations

What about?

1. **int length()**:
   - return the size of the original array
2. **int elementCount()** :
   - how many non-null values are in the array

Question

Do we forget something?

Other Operations

What about?

1. **int length()**:
   - return the size of the original array
2. **int elementCount()** :
   - how many non-null values are in the array

Question

Do we forget something?

Example of Get

Code

```
public Item get(int index) {
    ChainNode current = this.firstNode;
        do {
            if (index == current.index) {
                // found correct location
                return current.value;
            }
            current = current.next;
        } while (index < current.index && next != null);
    return null;
}
```

Time Analysis

First

- We must search a linked list for a given index
- We can keep the elements in order by index

Time Analysis

First

- We must search a linked list for a given index
- We can keep the elements in order by index

Expected Time for both get and add

If we have n elements, we have the following complexity $O(n)$.

Time Analysis

First

- We must search a linked list for a given index
- We can keep the elements in order by index

Expected Time for both get and add

If we have n elements, we have the following complexity $O(n)$.

For the other methods

length, elementCount $\Longrightarrow O(1)$

Problem with this analysis

True fact

In an ordinary array, indexing to find an element is the only operation we really need!!!

True fact

In a sparse array, we can do indexing reasonably quickly

False Conclusion

In a sparse array, indexing to find an element is the only operation we really need

- The problem is that in designing the ADT, we did not think enough about how it would be used !!!

Problem with this analysis

True fact

In an ordinary array, indexing to find an element is the only operation we really need!!!

True fact

In a sparse array, we can do indexing reasonably quickly

False Conclusion

In a sparse array, indexing to find an element is the only operation we really need

- The problem is that in designing the ADT, we did not think enough about how it would be used !!!

Problem with this analysis

True fact

In an ordinary array, indexing to find an element is the only operation we really need!!!

True fact

In a sparse array, we can do indexing reasonably quickly

False Conclusion

In a sparse array, indexing to find an element is the only operation we really need

- The problem is that in designing the ADT, we did not think enough about how it would be used !!!

Look at this code

To find the maximum element in a normal array:

```
double max = array[0];
for (int i = 0; i < array.length; i++)
{
    if (array[i] > max) max = array[i];
}
```

Look at this code

To find the maximum element in a sparse array:

```
Double max = (Double) array.get(0);
for (int i = 0; i < array.length(); i++)
{
      Double temp = (Double) array.get(i);
      if (temp.compareTo(max) > 0) {
            max = temp;
      }
}
```

What is the problem?

First

A lot of wrapping and casting because using generics.

Second

More importantly, in a normal array, every element is relevant

Third

- If a sparse array is 1% full, 99% of its elements will be zero.
  - This is 100 times as many elements as we should need to examine

What is the problem?

First

A lot of wrapping and casting because using generics.

Second

More importantly, in a normal array, every element is relevant

Third

- If a sparse array is 1% full, 99% of its elements will be zero.
  - This is 100 times as many elements as we should need to examine

What is the problem?

First

A lot of wrapping and casting because using generics.

Second

More importantly, in a normal array, every element is relevant

Third

- If a sparse array is 1% full, 99% of its elements will be zero.
  - This is 100 times as many elements as we should need to examine

What is the problem?

Fourth

Our search time is based on the size of the sparse array, not on the number of elements that are actually in it

Question

What to do?

What is the problem?

Fourth

Our search time is based on the size of the sparse array, not on the number of elements that are actually in it

Question

What to do?

Fixing the ADT

Something Notable

Although "stepping through an array" is not a fundamental operation on an array, it is one we do frequently.

Fixing the ADT

Something Notable

Although "stepping through an array" is not a fundamental operation on an array, it is one we do frequently.

However

- This is a very expensive thing to do with a sparse array
- This should not be so expensive:
  - We have a list, and all we need to do is step through it

Fixing the ADT

Something Notable

Although "stepping through an array" is not a fundamental operation on an array, it is one we do frequently.

However

- This is a very expensive thing to do with a sparse array
- This should not be so expensive:
  - We have a list, and all we need to do is step through it

Fixing the ADT

Something Notable

Although "stepping through an array" is not a fundamental operation on an array, it is one we do frequently.

However

- This is a very expensive thing to do with a sparse array
- This should not be so expensive:
  - We have a list, and all we need to do is step through it

Fixing the ADT

Poor Solution

- Let the user step through the list.
- The user should not need to know anything about implementation.
- We cannot trust the user not to screw up the sparse array.
- These arguments are valid even if the user is also the implementer!

Fixing the ADT

Poor Solution

- Let the user step through the list.
- The user should not need to know anything about implementation.
- We cannot trust the user not to screw up the sparse array.
- These arguments are valid even if the user is also the implementer!

Correct Solution

- Use an Inner iterator!!!

Fixing the ADT

Poor Solution

- Let the user step through the list.
- The user should not need to know anything about implementation.
- We cannot trust the user not to screw up the sparse array.
- These arguments are valid even if the user is also the implementer!

Correct Solution

- Use an Inner iterator!!!

Fixing the ADT

Poor Solution

- Let the user step through the list.
- The user should not need to know anything about implementation.
- We cannot trust the user not to screw up the sparse array.
- These arguments are valid even if the user is also the implementer!

Correct Solution

- Use an Inner iterator!!!

Fixing the ADT

Poor Solution

- Let the user step through the list.
- The user should not need to know anything about implementation.
- We cannot trust the user not to screw up the sparse array.
- These arguments are valid even if the user is also the implementer!

Correct Solution

- Use an Inner iterator!!!

Example

Code

```
public class SparseArrayIterator<Item> implements Iterator<Item> {
        // any data you need to keep track of goes here
        SparseArrayIterator() { ...You do need one... }

        public boolean hasNext ( ) { ...you write this code... }

        public Item next ( ) { ...you write this code... }

        public Item remove ( ) { ...you write this code... }
}
```

So, we have that

Code for Max

```
SparseArrayIterator iterator = new SparseArrayIterator(array);
Double max = (Double) array.get(0);

while (iterator.hasNext()) {
      temp = (Double) iterator.next();
      if (temp.compareTo(max) > 0) {
            max = temp;
      }
}
```

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell

- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index
- We could deal with this by overloading get to take an Item argument

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell.

Problem

- Somewhat more awkward to use, since we would need array.get(iterator.next()) instead of just iterator.next()
- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index.
- We could deal with this by overloading get to take an Item argument.

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell

Problem

- Somewhat more awkward to use, since we would need array.get(iterator.next()) instead of just iterator.next()
- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index.
- We could deal with this by overloading get to take an Item argument.

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell

Problem

- Somewhat more awkward to use, since we would need array.get(iterator.next()) instead of just iterator.next().
- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index.
- We could deal with this by overloading get to take an Item argument.

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell

Problem

- Somewhat more awkward to use, since we would need array.get(iterator.next()) instead of just iterator.next().
- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index.
- We could deal with this by overloading get to take an Item argument

However

Problem

- Our SparseArrayIterator is fine for stepping through the elements of an array, but...
  - It does not tell us what index they were at.
  - For some problems, we may need this information

First Solution

Revise our iterator to tell us, not the value in each list cell, but the index in each list cell

Problem

- Somewhat more awkward to use, since we would need array.get(iterator.next()) instead of just iterator.next().
- But it's worse than that, because next is defined to return an Item, so we would have to wrap the index.
- We could deal with this by overloading get to take an Item argument.

Instead of that use an IndexIterator Too

Code

```
public class IndexIterator
{
    private ChainNode current;
    IndexIterator() { // constructor
            current = firstNode;
      }
   public boolean hasNext()
       { // just like before }

    public int next()  {
        int index = current.index;
        current = current.next;
        return index;
    }
}
```

Now, Sparse Matrices

First

Something Simple!!

Using Array Linear List for Matrices

Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Thus

list =

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	3	4	5	6	2	6

Using Array Linear List for Matrices

Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Thus

list =

row	1	1	2	2	4	4
column	3	5	3	4	2	3
value	**3**	**4**	**5**	**6**	**2**	**6**

Now, How do we represent this list using arrays?

Why?

We want compact representations...

Use your friend element from arrays

Thus you declare an array element with the Node information

```
Node element[] = new Node[1000];
```

Now, How do we represent this list using arrays?

Why?

We want compact representations...

Use your friend element from arrays

```
package dataStructures;
//Using Generic in Java
public class Node<Item>{
    //elements
    private int row;
    private int column
    private Item value;
    // constructors come here
}
```

Thus you declare an array element with the Node information

```
Node element[] = new Node[1000];
```

Now, How do we represent this list using arrays?

Why?

We want compact representations...

Use your friend element from arrays

```
package dataStructures;
//Using Generic in Java
public class Node<Item>{
    //elements
    private int row;
    private int column
    private Item value;
    // constructors come here
}
```

Thus you declare an array element with the Node information

Node element[] = new Node[1000];

What about other representations?

We can use chains too

We need to modify our node

To something like this

Graphically

What about other representations?

We can use chains too

We need to modify our node

To something like this

```
package dataStructures;
//Using Generic in Java
public class Node<Item>{
    //elements
    private int row;
    private int column;
    private Item value;
    private Node next;
    // constructors come here
}
```

Graphically

What about other representations?

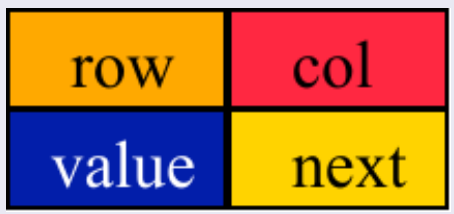
We can use chains too

We need to modify our node

To something like this

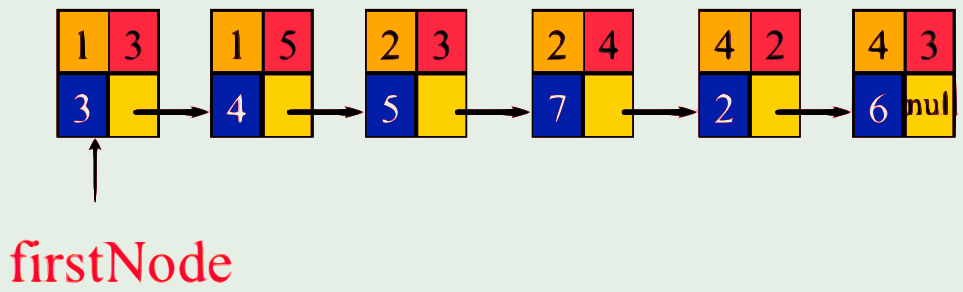
```
package dataStructures;
//Using Generic in Java
public class Node<Item>{
    //elements
    private int row;
    private int column;
    private Item value;
    private Node next;
    // constructors come here
}
```

Graphically



We have then

Example



One Linear List Per Row

Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Thus

row1 = [(3,3) (5,4)]
row2 = [(3,5) (4,7)]
row3 = []
row4 = [(2,2) (3,6)]

Node Structure

One Linear List Per Row

Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Thus

$$\begin{array}{rc} \text{row1} = & [(3,3) \quad (5,4)] \\ \text{row2} = & [(3,5) \quad (4,7)] \\ \text{row3} = & [] \\ \text{row4} = & [(2,2) \quad (3,6)] \end{array}$$

Node Structure

One Linear List Per Row

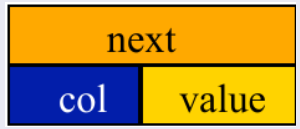
Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Thus

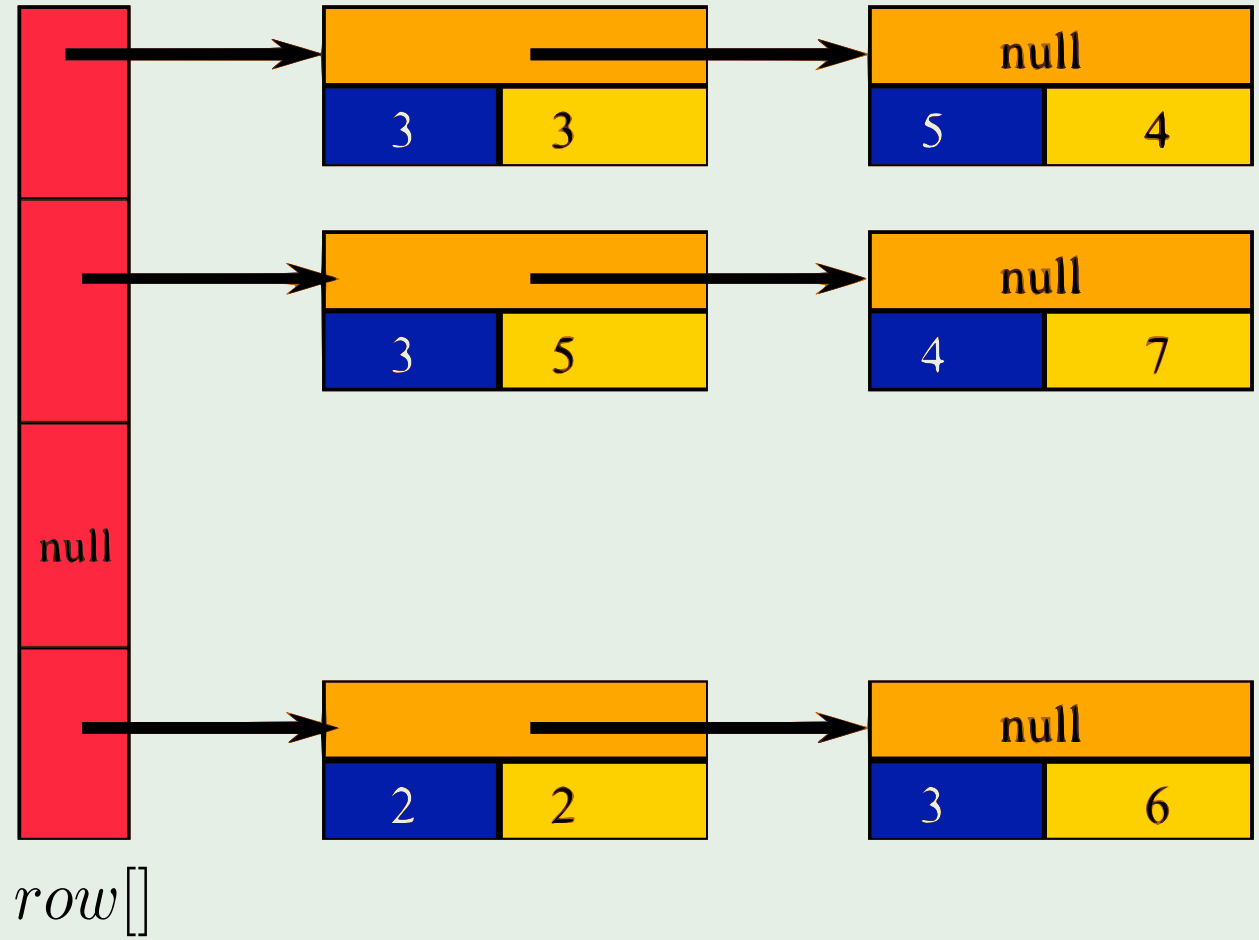
$$\begin{array}{rc} \text{row1} = & [(3,3) \quad (5,4)] \\ \text{row2} = & [(3,5) \quad (4,7)] \\ \text{row3} = & [] \\ \text{row4} = & [(2,2) \quad (3,6)] \end{array}$$

Node Structure



Array Of Row Chains

Example



We have a problem here

With 99% of the matrix as zeros or nulls

We have the problem that many row pointers are null...

What can you do?

Ideas

We have a problem here

With 99% of the matrix as zeros or nulls

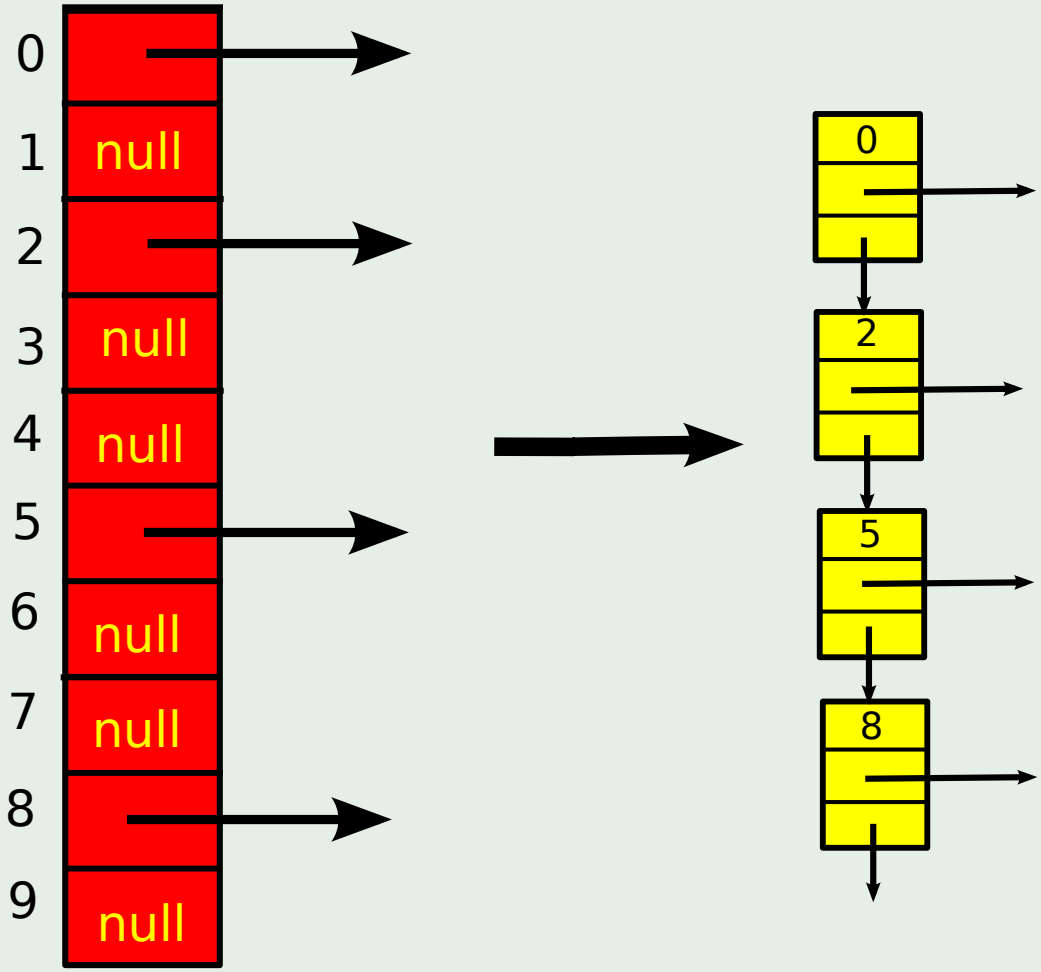
We have the problem that many row pointers are null...

What can you do?

Ideas

Compress the row

Use a sparse array!!!



However

Problems, problems Will Robinson!!!

Ideas?

Yes, access!! You still need to do an scanning!!!

We have a problem there!!

It is

Not so fast!!!! SOLUTIONS???

However

Problems, problems Will Robinson!!!

Ideas?

Yes, access!! You still need to do an scanning!!!

We have a problem there!!

It is

Not so fast!!!! SOLUTIONS???

However

Problems, problems Will Robinson!!!

Ideas?

Yes, access!! You still need to do an scanning!!!

We have a problem there!!

It is

Not so fast!!!! SOLUTIONS???

Orthogonal List Representation

More complexity

Both row and column lists.

Node Structure

Example

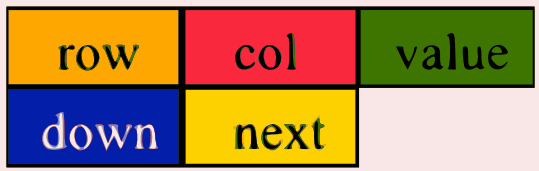
$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Orthogonal List Representation

More complexity

Both row and column lists.

Node Structure



Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Orthogonal List Representation

More complexity

Both row and column lists.

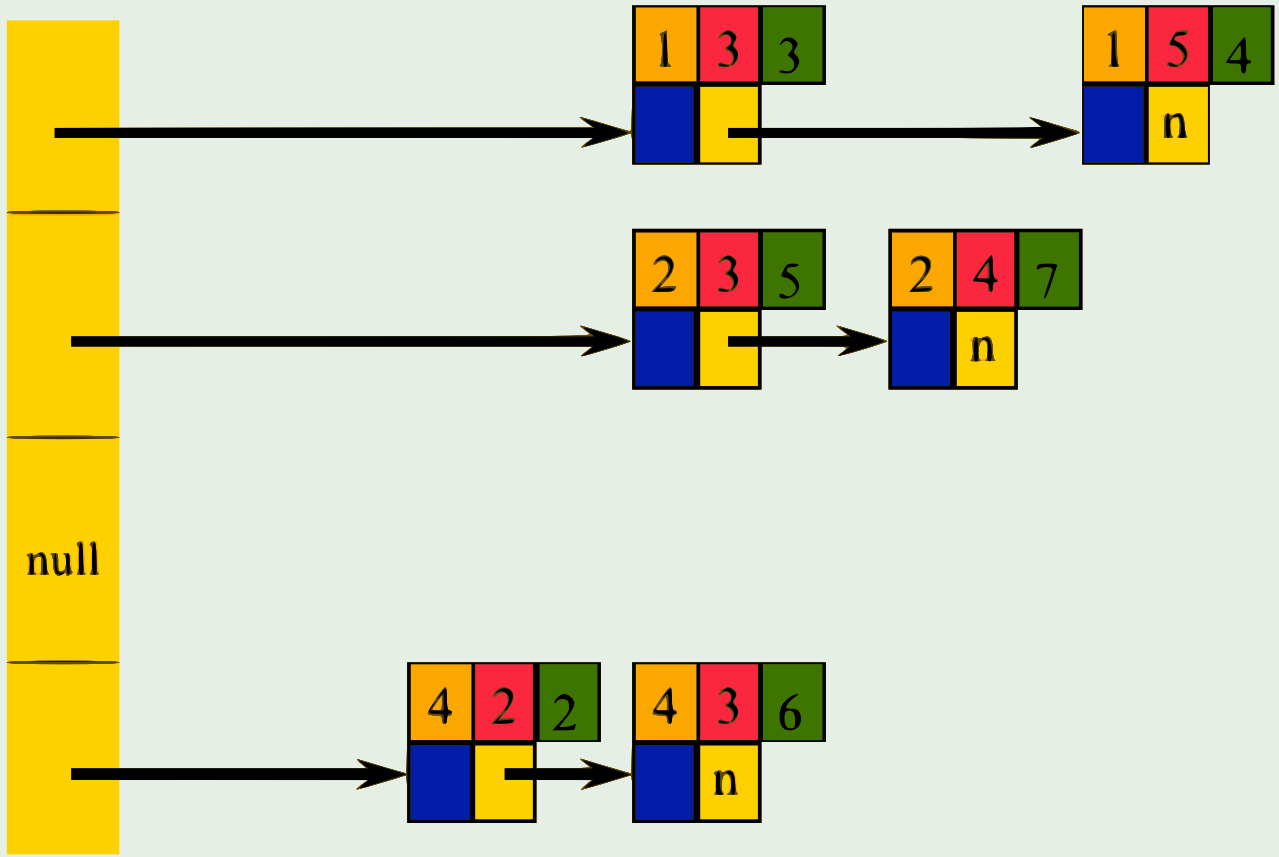
Node Structure

Example

$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

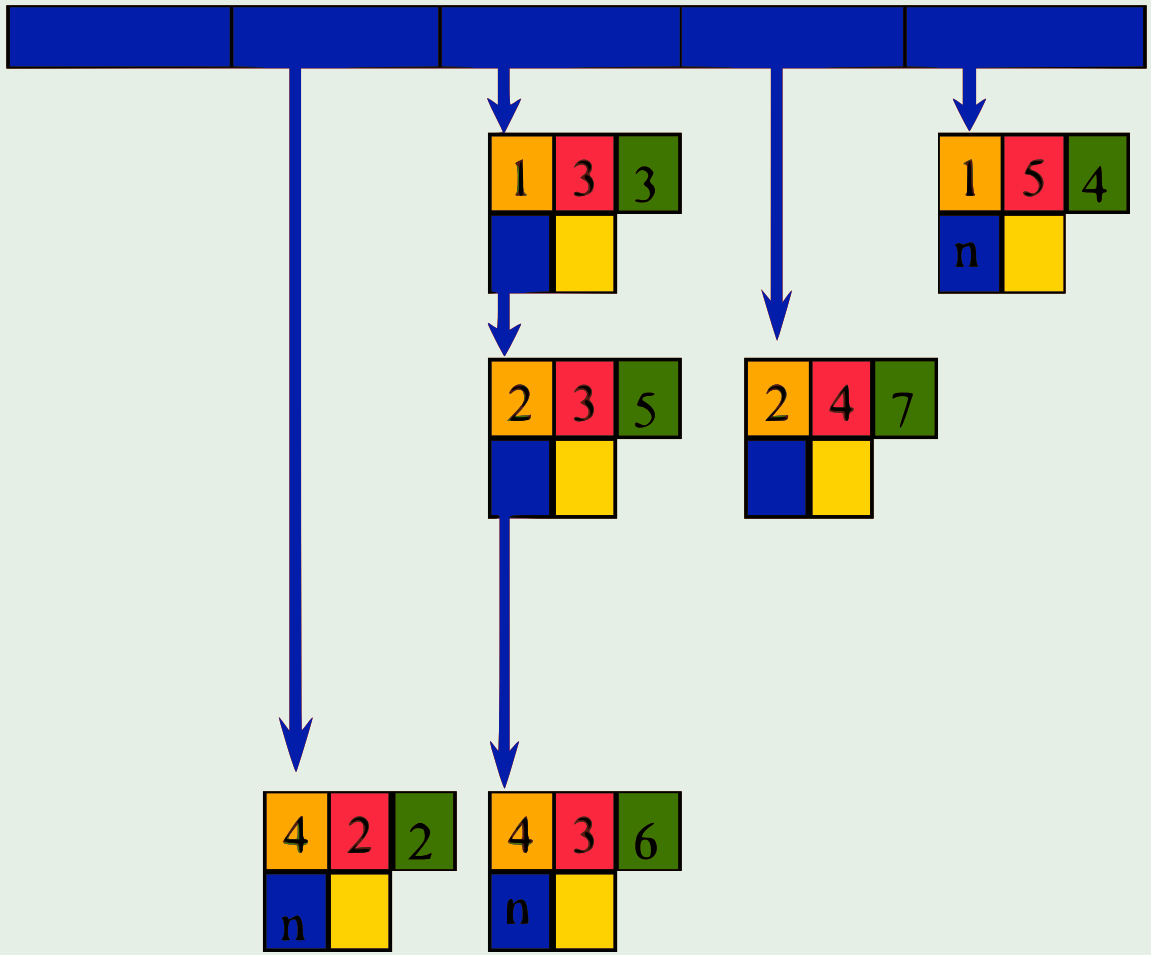
Row Lists

Row direction



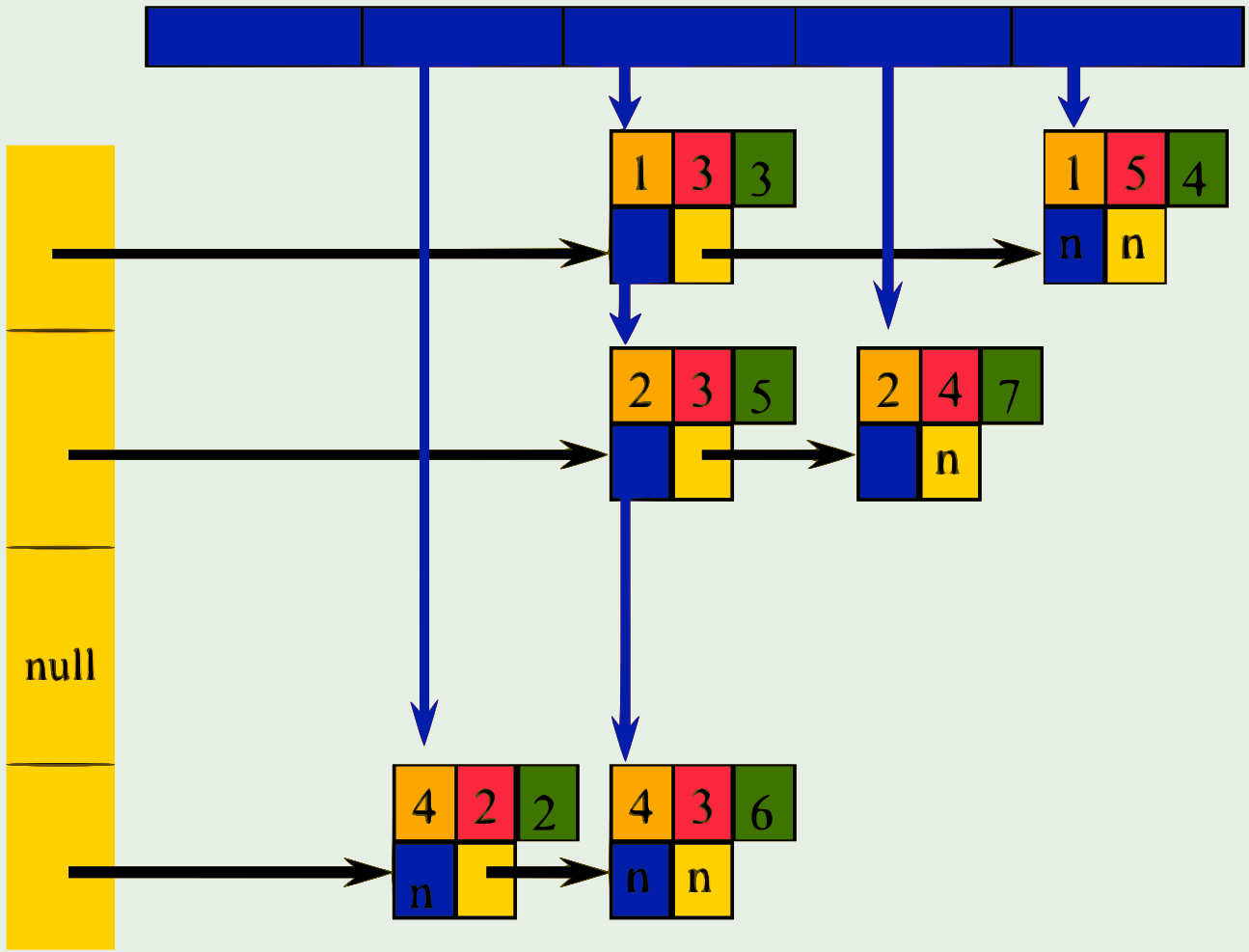
Column Lists

Column Direction



Orthogonal Lists

All Together



Go One Step Further

Question

We have another problem the sparse arrays in the column and row pointers!!!

Thus

How it will look like for this (You can try!!!)

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{3}$$

Go One Step Further

Question

We have another problem the sparse arrays in the column and row pointers!!!

Thus

How it will look like for this (You can try!!!)

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{3}$$

Do not worry!!!

You have MANY VARIATIONS!!!

Approximate Memory Requirements

Example

500 x 500 matrix with 1994 nonzero elements

- 2D array $500 \times 500 \times 4 = 1$ million bytes
- Single Array List $3 \times 1994 \times 4 = 23{,}928$ bytes
- One Chain Per Row $23928 + 500 \times 4 = 25{,}928$
- What about the sparse version even in row and column pointer?

Approximate Memory Requirements

Example

500 x 500 matrix with 1994 nonzero elements

- 2D array **500 x 500 x 4 = 1 million bytes**
- Single Array List 3 x 1994 x 4 = 23,928 bytes
- One Chain Per Row 23928 + 500 x 4 = 25,928
- What about the sparse version even in row and column pointer?

Approximate Memory Requirements

Example

500 x 500 matrix with 1994 nonzero elements

- 2D array **500 x 500 x 4 = 1 million bytes**
- Single Array List **3 x 1994 x 4** = 23,928 bytes
- One Chain Per Row 23928 + 500 x 4 = 25,928
- What about the sparse version even in row and column pointer?

Approximate Memory Requirements

Example

500 x 500 matrix with 1994 nonzero elements

- 2D array **500 x 500 x 4 = 1 million bytes**
- Single Array List **3 x 1994 x 4** = 23,928 bytes
- One Chain Per Row **23928 + 500 x 4** = 25,928
- What about the sparse version even in row and column pointer?

Approximate Memory Requirements

Example

500 x 500 matrix with 1994 nonzero elements

- 2D array **500 x 500 x 4 = 1 million bytes**
- Single Array List **3 x 1994 x 4** = 23,928 bytes
- One Chain Per Row **23928 + 500 x 4** = 25,928
- What about the sparse version even in row and column pointer?

Run-time Performance

Example Matrix Transpose

500 x 500 matrix with 1994 nonzero elements:

Method	Time
2D Array	210 ms
Single Array List	6 ms
One Chain per Row	12 ms

Example Matrix Addition

500 x 500 matrix with 1994 nonzero elements:

Method	Time
2D Array	880 ms
Single Array List	18 ms
One Chain per Row	29 ms

Run-time Performance

Example Matrix Transpose

500 x 500 matrix with 1994 nonzero elements:

Method	Time
2D Array	210 ms
Single Array List	6 ms
One Chain per Row	12 ms

Example Matrix Addition

500 x 500 matrix with 1994 nonzero elements:

Method	Time
2D Array	880 ms
Single Array List	18 ms
One Chain per Row	29 ms