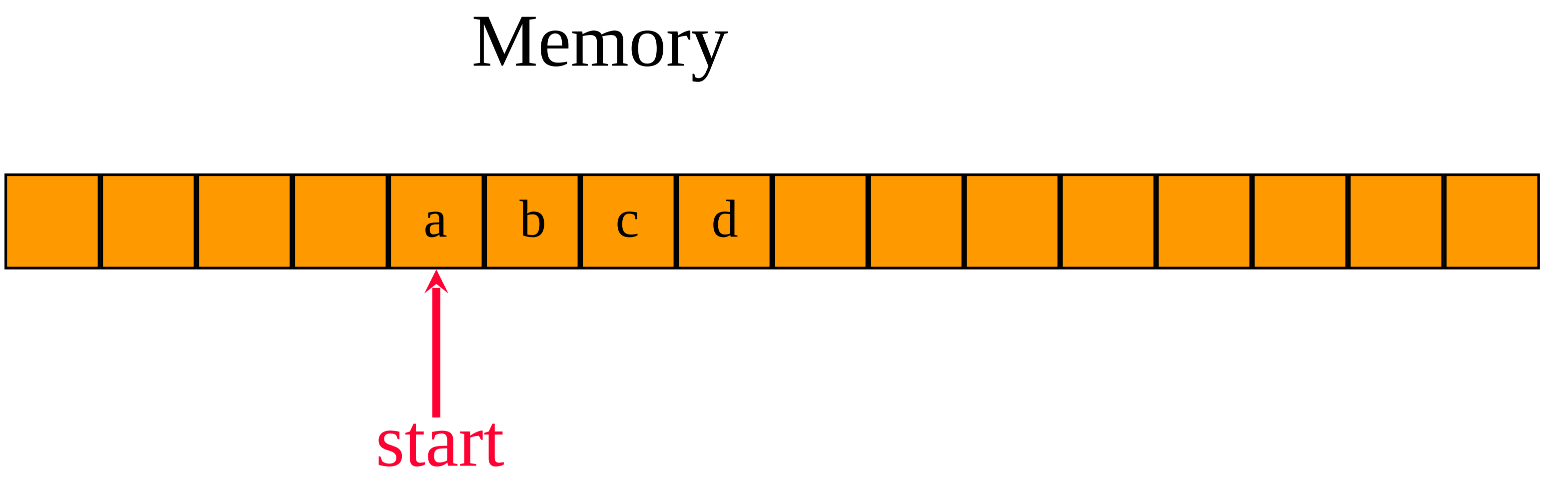
Arrays and Matrices

Data structures

Fall 2018

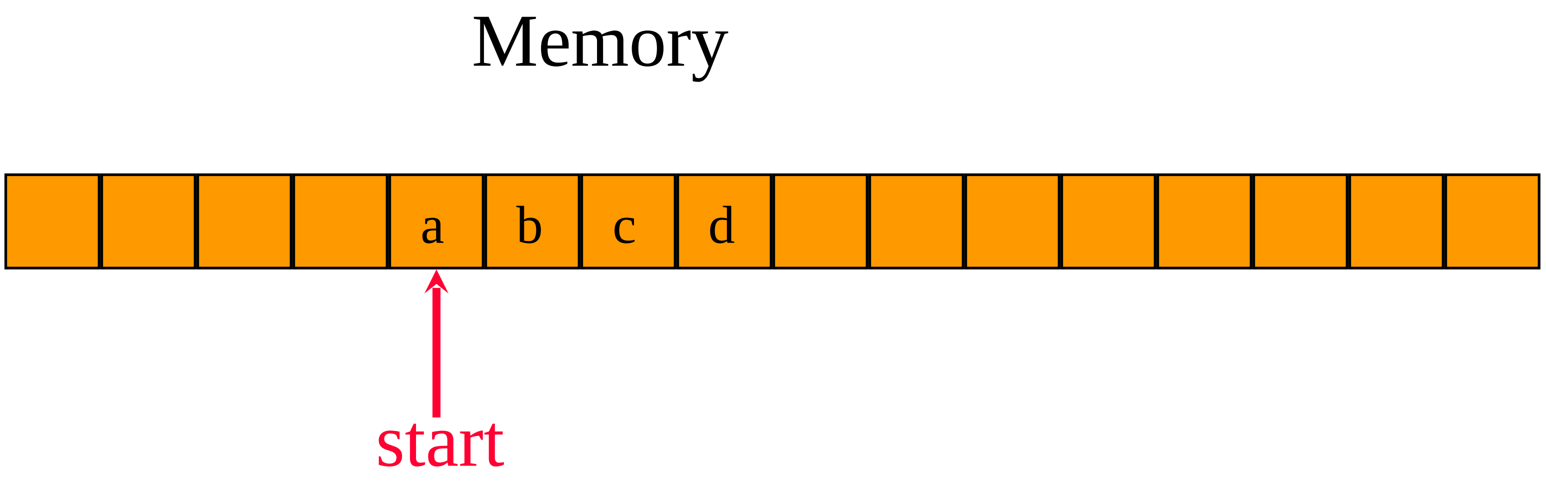
1D Array Representation In Java, C, and C++



- 1-dimensional array x = [a, b, c, d]
- map into contiguous memory locations

- location(x[i]) = start + i

Space Overhead



space overhead = 4 bytes for start

+ 4 bytes for x.length

= 8 bytes

(excludes space needed for the elements of x)

2D Arrays

The elements of a 2-dimensional array a declared as:

int [][]a = new int[3][4];

may 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 Of A 2D Array

a[0][0]	a[0][1]	a[0][2]	a[0][3]	row 0
a[1][0]	a[1][1]	a[1][2]	a[1][3]	row 1
a[2][0]	a[2][1]	a[2][2]	a[2][3]	row 2

Columns Of A 2D Array

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]
column 0	column 1	column 2	column 3

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

2-dimensional array x

a, b, c, d

e, f, g, h

i, j, k, l

view 2D array as a 1D array of rows

x = [row0, row1, row 2]

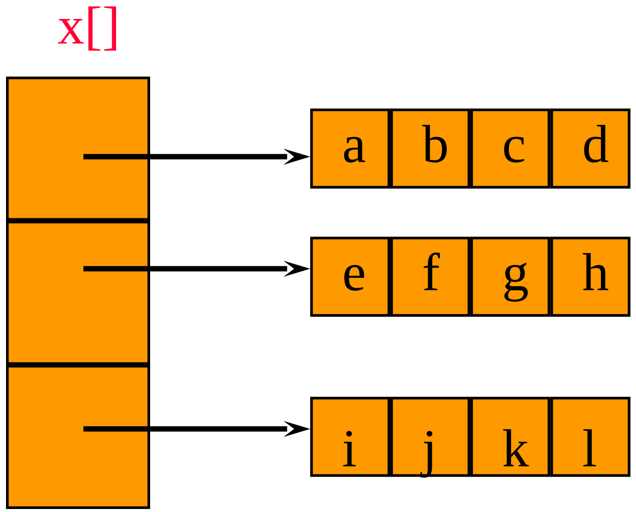
row 0 = [a,b, c, d]

row 1 = [e, f, g, h]

row 2 = [i, j, k, l]

and store as 4 1D arrays

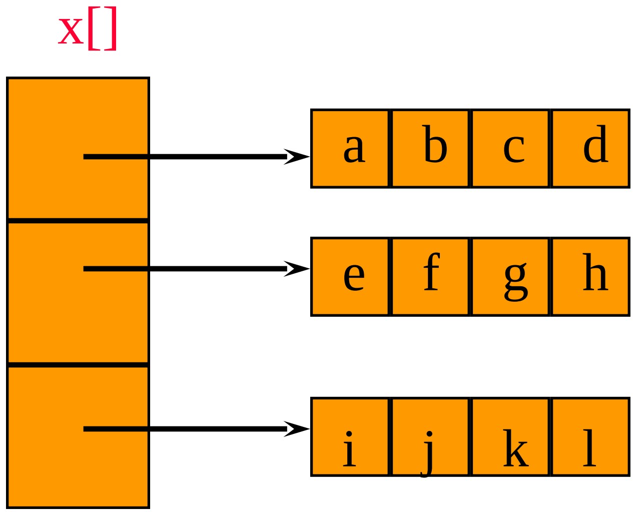
2D Array Representation In Java, C, and C++



x.length = 3

x[0].length = x[1].length = x[2].length = 4

Space Overhead



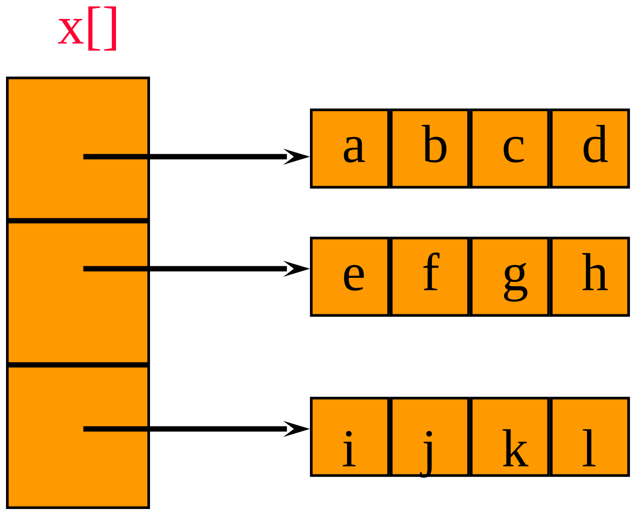
space overhead = overhead for 4 1D arrays

= 4 * 8 bytes

= 32 bytes

= (number of rows + 1) x 8 bytes

Array Representation In Java, C, and C++



- This representation is called the array-of-arrays representation.
- Requires contiguous memory of size 3, 4, 4, and 4 for the 4 1D arrays.
- 1 memory block of size number of rows and number of rows blocks of size number of columns

Row-Major Mapping

- Example 3 x 4 array:

  a b c d

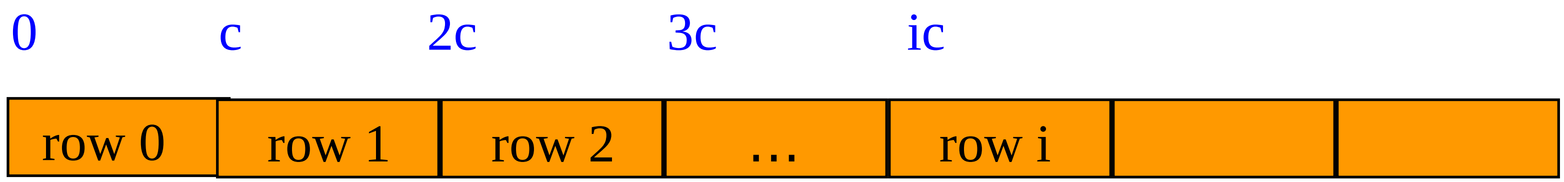
  e f g h

  i j k l

- Convert into 1D array y by collecting elements by rows.
- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.
- We get y[] = {a, b, c, d, e, f, g, h, i, j, k, l}

row 0	row 1	row 2	…	row i		

Locating Element x[i][j]



- assume x has r rows and c columns
- each row has c elements
- i rows to the left of row i
- so ic elements to the left of x[i][0]
- so x[i][j] is mapped to position

  ic + j of the 1D array

Space Overhead

row 0	row 1	row 2	…	row i		

4 bytes for start of 1D array +

4 bytes for length of 1D array +

4 bytes for c (number of columns)

= 12 bytes

(number of rows = length /c)

Disadvantage

Need contiguous memory of size rc.

Column-Major Mapping

a b c d

e f g h

i j k l

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

Matrix

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

a b c d	row 1
e f g h	row 2
i j k l	row 3

- Use notation x(i,j) rather than x[i][j].
- May use a 2D array to represent a matrix.

Shortcomings Of Using A 2D Array For A Matrix

- Indexes are off by 1.
- Java arrays do not support matrix operations such as add, transpose, multiply, and so on.
  - Suppose that x and y are 2D arrays. Can't do x + y, x –y, x * y, etc. in Java.
- Develop a class Matrix for object-oriented support of all matrix operations. See text.

Diagonal Matrix

An n x n matrix in which all nonzero terms are on the diagonal.

Diagonal Matrix

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{matrix}$$

- $x(i,j)$ is on diagonal iff $i = j$
- number of diagonal elements in an $n \times n$ matrix is n
- non-diagonal elements are zero
- store diagonal only vs n^2 whole

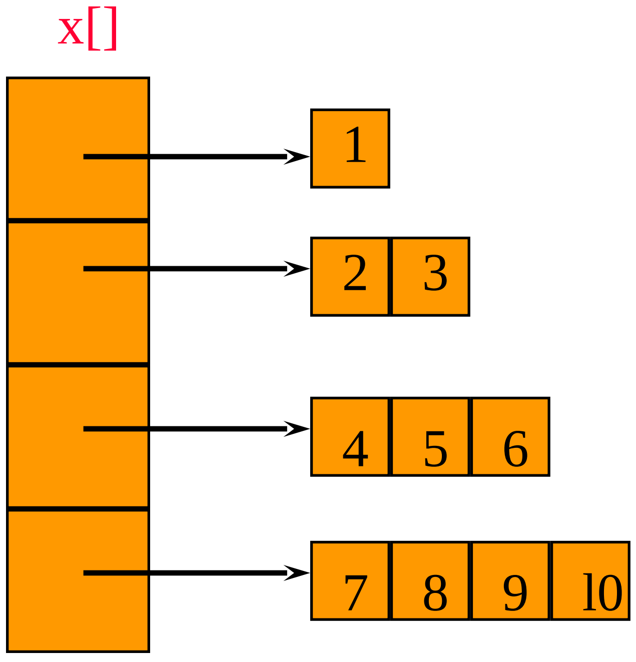
Lower Triangular Matrix

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

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

- x(i,j) is part of lower triangle iff $i >= j$.
- number of elements in lower triangle is $1 + 2 + \ldots + n = n(n+1)/2$.
- store only the lower triangle

Array Of Arrays Representation



Use an irregular 2-D array … length of rows is not required to be the same.

Creating And Using An 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;
```

Map Lower Triangular Array Into A 1D Array

Use row-major order, but omit terms that are not part of the lower triangle.

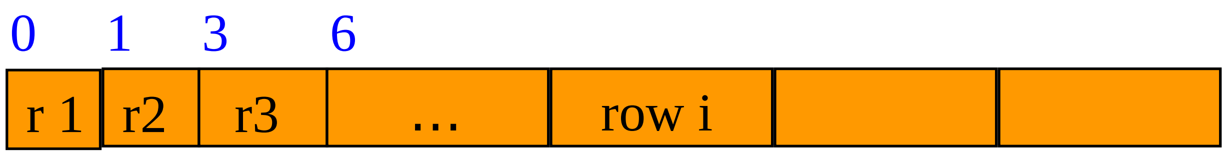
For the matrix

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

we get

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

Index Of Element [i][j]



- Order is: row 1, row 2, row 3, …
- Row i is preceded by rows 1, 2, …, i-1
- Size of row i is i.
- Number of elements that precede row i is

  $1 + 2 + 3 + \ldots + i-1 = i(i-1)/2$
- So element (i,j) is at position $i(i-1)/2 + j - 1$ of the 1D array.

Sparse Matrices

sparse … many elements are zero

dense … few elements are zero

Example Of Sparse Matrices

diagonal

tridiagonal

lower triangular (?)

These are structured sparse matrices.

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

Unstructured Sparse Matrices

Airline flight matrix.

- airports are numbered 1 through n
- flight(i,j) = list of nonstop flights from airport i to airport j
- n = 1000 (say)
- n x n array of list references => 4 million bytes
- total number of flights = 20,000 (say)
- need at most 20,000 list references => at most 80,000 bytes

Unstructured Sparse Matrices

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
- PageRank … “importance” of a page based on links to and from the page

Web Page Matrix

- n = 2 billion (and growing by 1 million a day)
- n x n array of ints => $16 * 10^{18}$ bytes ($16 * 10^{9}$ GB)
- each page links to 10 (say) other pages on average
- on average there are 10 nonzero entries per row
- space needed for nonzero elements is approximately 20 billion x 4 bytes = 80 billion bytes (80 GB)

Representation Of Unstructured Sparse Matrices

- Single linear list in row-major order.
  - scan the nonzero 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)

Single Linear List Example

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

list =

$$\begin{array}{l} \text{row} \\ \text{column} \\ \text{value} \end{array} \begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 3 & 5 & 3 & 4 & 2 & 3 \\ 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix}$$

Array Linear List Representation

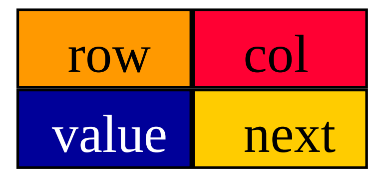
$$\text{list} = \begin{array}{r} \text{row} \\ \text{column} \\ \text{value} \end{array} \begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 3 & 5 & 3 & 4 & 2 & 3 \\ 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix}$$

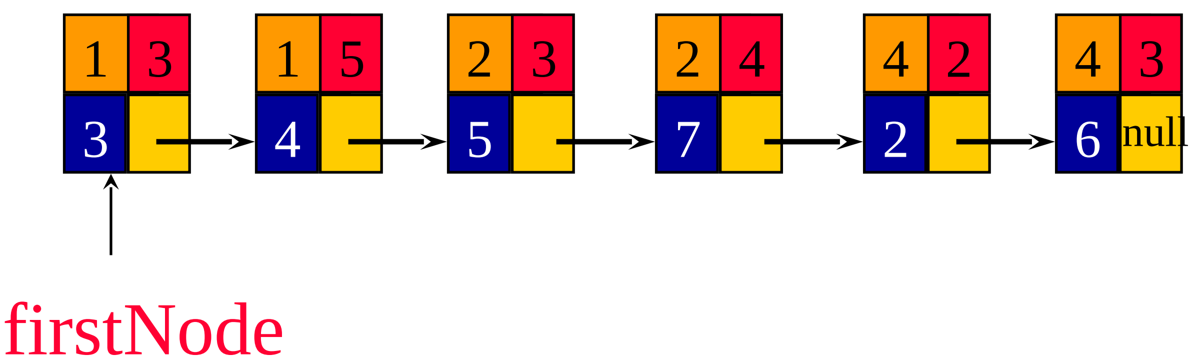
$$\begin{array}{rc} \text{element} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{r} \text{row} \\ \text{column} \\ \text{value} \end{array} & \begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 3 & 5 & 3 & 4 & 2 & 3 \\ 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix} \end{array}$$

Single Chain

$$
list = \begin{array}{r} \text{row} \\ \text{column} \\ \text{value} \end{array} \begin{bmatrix} 1 & 1 & 2 & 2 & 4 & 4 \\ 3 & 5 & 3 & 4 & 2 & 3 \\ 3 & 4 & 5 & 7 & 2 & 6 \end{bmatrix}
$$

Node structure





One Linear List Per Row

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

row1 = [(3, 3), (5,4)]

row2 = [(3,5), (4,7)]

row3 = []

row4 = [(2,2), (3,6)]

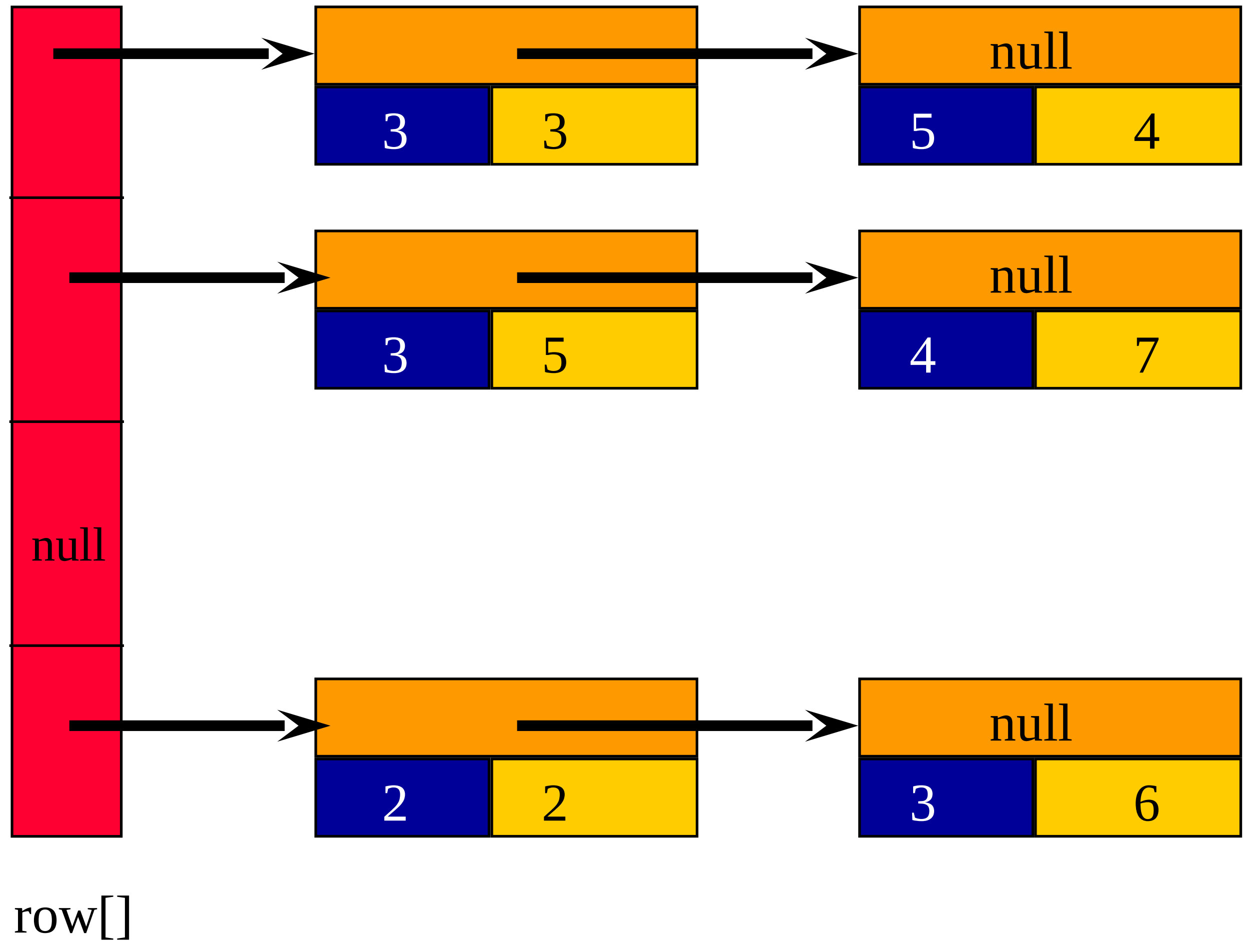
Array Of Row Chains

0 0 3 0 4

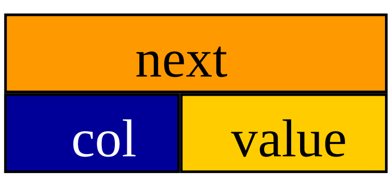
0 0 5 7 0

0 0 0 0 0

0 2 6 0 0



Node structure



Orthogonal List Representation

Both row and column lists.

Node structure.

row	col	value
down	next	

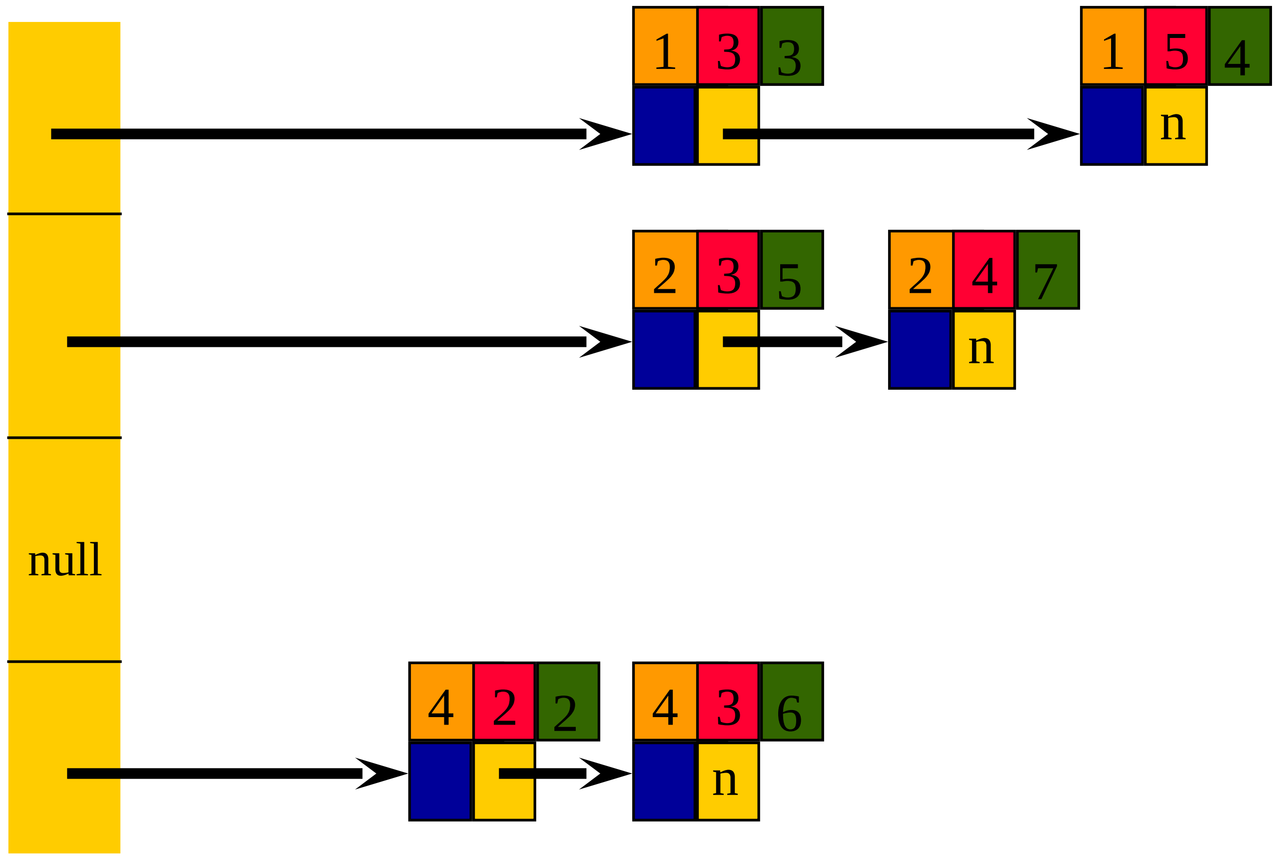
Row Lists

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0



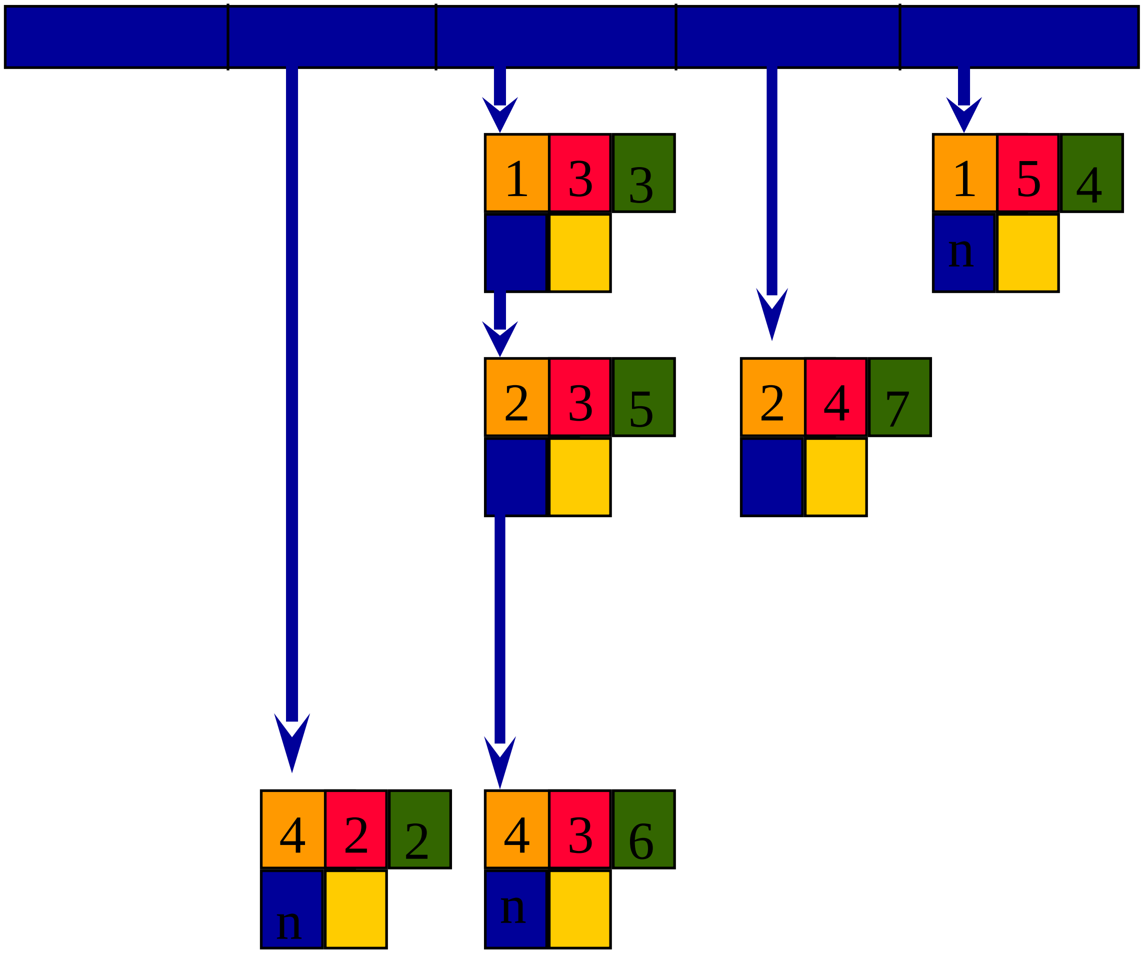
Column Lists

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0



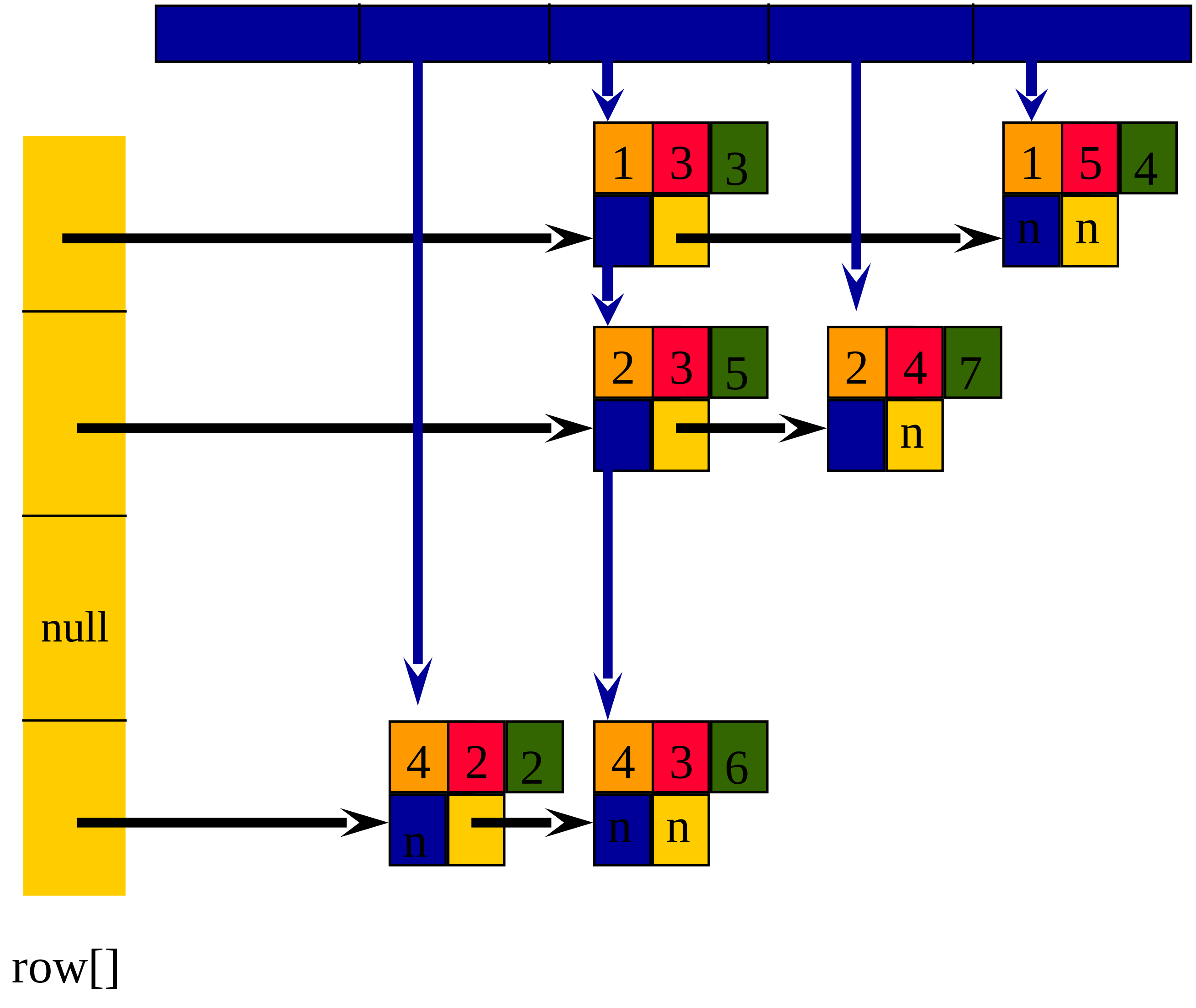
Orthogonal Lists

0 0 3 0 4

0 0 5 7 0

0 0 0 0 0

0 2 6 0 0



Approximate Memory Requirements

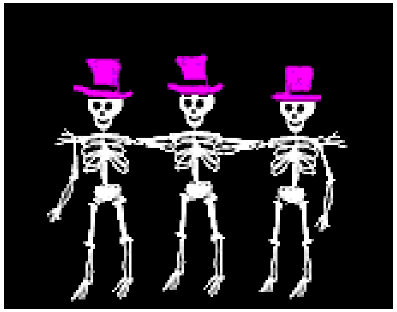
500 x 500 matrix with 2000 nonzero elements

2D array 500 x 500 x 4 = 1million bytes

Single Array List 3 x 2000 x 4 = 24,000 bytes

One Chain Per Row 24000 + 500 x 4 = 26,000

Runtime Performance

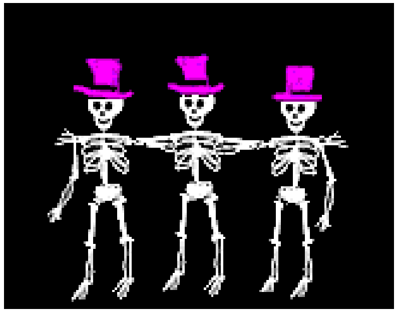


Matrix Transpose

500 x 500 matrix with 2000 nonzero elements

2D array	210 ms
Single Array List	6 ms
One Chain Per Row	12 ms

Performance



Matrix Addition.

500 x 500 matrices with 2000 and 1000 nonzero elements

2D array	880 ms
Single Array List	18 ms
One Chain Per Row	29 ms