Arrays
1D Array Representation In C++

Memory

- 1-dimensional array $x = [a, b, c, d]$
- map into contiguous memory locations
- $\text{location}(x[i]) = \text{start} + i$
Space Overhead

Memory

space overhead = 4 bytes for start

(excludes space needed for the elements of x)
2D Arrays

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

```c
int a[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]  
```
Rows Of A 2D Array

\[
\begin{array}{cccc}
\text{a[0][0]} & \text{a[0][1]} & \text{a[0][2]} & \text{a[0][3]} \\
\text{a[1][0]} & \text{a[1][1]} & \text{a[1][2]} & \text{a[1][3]} \\
\text{a[2][0]} & \text{a[2][1]} & \text{a[2][2]} & \text{a[2][3]} \\
\end{array}
\]
### Columns Of A 2D Array

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
</tr>
<tr>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
</tr>
</tbody>
</table>

column 0  column 1  column 2  column 3
2D Array Representation In C++

2-dimensional array \( x \)

\[
\begin{align*}
& a, b, c, d \\
& e, f, g, h \\
& i, j, k, l
\end{align*}
\]

view 2D array as a 1D array of rows

\[
\begin{align*}
x &= [\text{row0}, \text{row1}, \text{row2}] \\
\text{row 0} &= [a,b, c, d] \\
\text{row 1} &= [e, f, g, h] \\
\text{row 2} &= [i, j, k, l]
\end{align*}
\]

and store as 4 1D arrays
2D Array Representation In C++

4 separate 1-dimensional arrays
Space Overhead

Space overhead = overhead for 4 1D arrays
= 4 * 4 bytes
= 16 bytes
= (number of rows + 1) x 4 bytes
• 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 $y[] = \{a, b, c, d, e, f, g, h, i, j, k, l\}$
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 **c** (number of columns)
= 8 bytes
Disadvantage

Need contiguous memory of size $rc$. 
Column-Major Mapping

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
</tr>
</tbody>
</table>

• 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.

\begin{array}{cccc}
  a & b & c & d \\
  e & f & g & h \\
  i & j & k & l \\
\end{array} \quad \text{row 1}

\begin{array}{cccc}
  & & & \\
  e & f & g & h \\
  & & & \\
  i & j & k & l \\
\end{array} \quad \text{row 2}

\begin{array}{cccc}
  & & & \\
  & & & \\
  i & j & k & l \\
\end{array} \quad \text{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.
• C++ 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 C++.
• Develop a class matrix for object-oriented support of all matrix operations. See text.
Diagonal Matrix

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

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

- $x(i,j)$ is part of lower triangle iff $i \geq j$.
- 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 elements in each row
for (int i = 0; i < numberOfRows; i++)
    irregularArray[i] = new int [length[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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 \\
4 & 5 & 6 & 0 \\
7 & 8 & 9 & 10
\end{pmatrix}
\]

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
  \[1 + 2 + 3 + \ldots + i-1 = \frac{i(i-1)}{2}\]
- So element (i,j) is at position \(\frac{i(i-1)}{2} + j - 1\) 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.
Airline flight matrix.

- airports are numbered 1 through \( n \)
- \( \text{flight}(i,j) = \) list of nonstop flights from airport \( i \) to airport \( j \)
- \( n = 1000 \) (say)
- \( n \times n \) array of list references \( \Rightarrow 4 \) million bytes
- total number of flights \( = 20,000 \) (say)
- need at most 20,000 list references \( \Rightarrow \) 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
Web Page Matrix

- $n = 2$ billion (and growing by 1 million a day)
- $n \times n$ array of ints $\Rightarrow 16 \times 10^{18}$ bytes ($16 \times 10^9$ 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

list =

row  column value
0 0 3 0 4 1 1 2 2 4 4
0 0 5 7 0 3 5 3 4 2 3
0 0 0 0 0 3 4 5 7 2 6
0 2 6 0 0
### Array Linear List Representation

<table>
<thead>
<tr>
<th>row</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>value</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

| element | 0 | 1 | 2 | 3 | 4 | 5 |
| row | 1 | 1 | 2 | 2 | 4 | 4 |
| column | 3 | 5 | 3 | 4 | 2 | 3 |
| value | 3 | 4 | 5 | 7 | 2 | 6 |
Chain Representation

Node structure.

<table>
<thead>
<tr>
<th>row</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>next</td>
</tr>
</tbody>
</table>
Single Chain

row

| 1 | 1 | 2 | 2 | 4 | 4 |

list = column

| 3 | 5 | 3 | 4 | 2 | 3 |

value

| 3 | 4 | 5 | 7 | 2 | 6 |

firstNode
One Linear List Per Row

\[
\begin{array}{cccccc}
0 & 0 & 3 & 0 & 4 & \\
0 & 0 & 5 & 7 & 0 & \\
0 & 0 & 0 & 0 & 0 & \\
0 & 2 & 6 & 0 & 0 & \\
\end{array}
\]

\[
\text{row1} = [(3, 3), (5, 4)]
\]

\[
\text{row2} = [(3, 5), (4, 7)]
\]

\[
\text{row3} = [
\]

\[
\text{row4} = [(2, 2), (3, 6)]
\]
Array Of Row Chains

Node structure.

- next
- col
- value
Array Of Row Chains

row[]
Orthogonal List Representation

Both row and column lists.

Node structure.
Row Lists
Column Lists

0 0 3 0 4
0 0 5 7 0
0 0 0 0 0
0 2 6 0 0
Orthogonal Lists

row[]
Variations

May use circular lists instead of chains.
Approximate Memory Requirements

500 x 500 matrix with 1994 nonzero elements

2D array \[ 500 \times 500 \times 4 = \text{1 million bytes} \]

Single Array List \[ 3 \times 1994 \times 4 = 23,928 \text{ bytes} \]

One Chain Per Row \[ 23,928 + 500 \times 4 = 25,928 \]
Runtime Performance

Matrix Transpose

500 x 500 matrix with 1994 nonzero elements

2D array 1.97 ms
Single Array List 0.09 ms
One Chain Per Row 1.57 ms
Performance

Matrix Addition.

500 x 500 matrices with 1994 and 999 nonzero elements

<table>
<thead>
<tr>
<th>Representation</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D array</td>
<td>2.69</td>
</tr>
<tr>
<td>Single Array List</td>
<td>0.13</td>
</tr>
<tr>
<td>One Chain Per Row</td>
<td>Not Measured</td>
</tr>
</tbody>
</table>