Introduction to Programming (in C++)

Multi-dimensional vectors

Jordi Cortadella, Ricard Gavaldà, Fernando Orejas
Dept. of Computer Science, UPC
Matrices

• A matrix can be considered a two-dimensional vector, i.e. a vector of vectors.

my_matrix:  

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

// Declaration of a matrix with 3 rows and 4 columns
vector< vector<int> > my_matrix(3,vector<int>(4));

// A more elegant declaration
typedef vector<int> Row;   // One row of the matrix
typedef vector<Row> Matrix; // Matrix: a vector of rows

Matrix my_matrix(3,Row(4));   // The same matrix as above
Matrices

- A matrix can be considered as a 2-dimensional vector, i.e., a vector of vectors.

```
my_matrix:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>
```

```
my_matrix[0][2]

my_matrix[1][3]

my_matrix[2]
```
\( n \)-dimensional vectors

- Vectors with any number of dimensions can be declared:

```cpp
typedef vector<int> Dim1;
typedef vector<Dim1> Dim2;
typedef vector<Dim2> Dim3;
typedef vector<Dim3> Matrix4D;

Matrix4D my_matrix(5, Dim3(i+1, Dim2(n, Dim1(9))));
```
• Design a function that calculates the sum of two \( n \times m \) matrices.

\[
\begin{bmatrix}
2 & -1 \\
0 & 1 \\
1 & 3 \\
\end{bmatrix} + \begin{bmatrix}
1 & 1 \\
2 & -1 \\
0 & -2 \\
\end{bmatrix} = \begin{bmatrix}
3 & 0 \\
2 & 0 \\
1 & 1 \\
\end{bmatrix}
\]

typedef vector< vector<int> > Matrix;

Matrix matrix_sum(const Matrix& a, const Matrix& b);
How are the elements of a matrix visited?

• By rows
  For every row \( i \)
  For every column \( j \)
  Visit \( \text{Matrix}[i][j] \)

• By columns
  For every column \( j \)
  For every row \( i \)
  Visit \( \text{Matrix}[i][j] \)

For every row \( i \)
  For every column \( j \)
  Visit \( \text{Matrix}[i][j] \)

For every column \( j \)
  For every row \( i \)
  Visit \( \text{Matrix}[i][j] \)
**Sum of matrices (by rows)**

```cpp
typedef vector<vector<int>> Matrix;

// Pre: a and b are non-empty matrices and have the same size. // Returns a+b (sum of matrices).
Matrix matrix_sum(const Matrix& a, const Matrix& b) {
    int nrows = a.size();
    int ncols = a[0].size();
    Matrix c(nrows, vector<int>(ncols));

    for (int i = 0; i < nrows; ++i) {
        for (int j = 0; j < ncols; ++j) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
```

Introduction to Programming © Dept. CS, UPC
typedef vector< vector<int> > Matrix;

// Pre: a and b are non-empty matrices and have the same size. // Returns a+b (sum of matrices).

Matrix matrix_sum(const Matrix& a, const Matrix& b) {
    int nrows = a.size();
    int ncols = a[0].size();
    Matrix c(nrows, vector<int>(ncols));

    for (int j = 0; j < ncols; ++j) {
        for (int i = 0; i < nrows; ++i) {
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    return c;
}
Transpose a matrix

• Design a procedure that transposes a square matrix in place:

```c
void Transpose (Matrix& m);
```

```
3 8 1
0 6 2
4 5 9
```

```
3 0 4
8 6 5
1 2 9
```

• Observation: we need to swap the upper with the lower triangular matrix. The diagonal remains intact.
Transpose a matrix

// Interchanges two values
void swap(int& a, int& b) {
    int c = a;
    a = b;
    b = c;
}

// Pre: m is a square matrix
// Post: m contains the transpose of the input matrix
void Transpose(Matrix& m) {
    int n = m.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j < n; ++j) {
            swap(m[i][j], m[j][i]);
        }
    }
}

Introduction to Programming © Dept. CS, UPC
Is a matrix symmetric?

• Design a procedure that indicates whether a matrix is symmetric:

```cpp
bool is_symmetric(const Matrix& m);
```

• Observation: we only need to compare the upper with the lower triangular matrix.
Is a matrix symmetric?

// Pre: m is a square matrix
// Returns true if m is symmetric, and false otherwise

bool is_symmetric(const Matrix& m) {
    int n = m.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (m[i][j] != m[j][i]) return false;
        }
    }
    return true;
}
Search in a matrix

• Design a procedure that finds a value in a matrix. If the value belongs to the matrix, the procedure will return the location \((i, j)\) at which the value has been found.

```cpp
// Pre: m is a non-empty matrix
// Post: i and j define the location of a cell
//       that contains the value x in m.
//       In case x is not in m, then i = j = -1.

void search(const Matrix& m, int x, int& i, int& j);
```
// Pre: m is a non-empty matrix
// Post: i and j define the location of a cell
//       that contains the value x in M.
//       In case x is not in m, then i = j = -1

void search(const Matrix& m, int x, int& i, int& j) {
  int nrows = m.size();
  int ncols = m[0].size();
  for (i = 0; i < nrows; ++i) {
    for (j = 0; j < ncols; ++j) {
      if (m[i][j] == x) return;
    }
  }
  i = -1;
  j = -1;
}
Search in a sorted matrix

• A sorted matrix $m$ is one in which

\[
\begin{align*}
m[i][j] & \leq m[i][j+1] \\
m[i][j] & \leq m[i+1][j]\end{align*}
\]
Search in a sorted matrix

• Example: let us find 10 in the matrix. We look at the lower left corner of the matrix.

• Since 13 > 10, the value cannot be found in the last row.
Search in a sorted matrix

- We look again at the lower left corner of the remaining matrix.
- Since $11 > 10$, the value cannot be found in the row.
Search in a sorted matrix

- Since $9 < 10$, the value cannot be found in the column.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>13</td>
<td>14</td>
<td>17</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>23</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>26</td>
</tr>
</tbody>
</table>
Search in a sorted matrix

• Since 11 > 10, the value cannot be found in the row.
Search in a sorted matrix

- Since $7 < 10$, the value cannot be found in the column.
Search in a sorted matrix

• The element has been found!
Search in a sorted matrix

• *Invariant*: if the element is in the matrix, then it is located in the sub-matrix \( [0...i, j...\text{ncols}-1] \)
Search in a sorted matrix

// Pre: m is non-empty and sorted by rows and columns
//      in ascending order.
// Post: i and j define the location of a cell that contains the value
//       x in m. In case x is not in m, then i=j=-1.

void search(const Matrix& m, int x, int& i, int& j) {
    int nrows = m.size();
    int ncols = m[0].size();
    
    i = nrows - 1;
    j = 0;

    // Invariant: x can only be found in M[0..i,j..ncols-1]
    while (i >= 0 and j < ncols) {
        if (m[i][j] < x) j = j + 1;
        else if (m[i][j] > x) i = i - 1;
        else return;
    }

    i = -1;
    j = -1;
}
Search in a sorted matrix

• What is the largest number of iterations of a search algorithm in a matrix?

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted matrix</td>
<td>$n_{rows} \times n_{cols}$</td>
</tr>
<tr>
<td>Sorted matrix</td>
<td>$n_{rows} + n_{cols}$</td>
</tr>
</tbody>
</table>

• The search algorithm in a sorted matrix cannot start in all of the corners of the matrix. Which corners are suitable?
Matrix multiplication

• Design a function that returns the multiplication of two matrices.

\[
\begin{bmatrix}
2 & -1 & 0 & 1 \\
1 & 3 & 2 & 0
\end{bmatrix}
\times
\begin{bmatrix}
1 & 2 & -1 \\
3 & 0 & 2 \\
-1 & 1 & 3 \\
2 & -1 & 4
\end{bmatrix}
= 
\begin{bmatrix}
1 & 3 & 0 \\
8 & 4 & 11
\end{bmatrix}
\]

// Pre: a is a non-empty n×m matrix, 
// b is a non-empty m×p matrix 
// Returns a×b (an n×p matrix) 
Matrix multiply(const Matrix& a, const Matrix& b);
Matrix multiplication

// Pre: a is a non-empty n×m matrix, b is a non-empty m×p matrix.
// Returns a×b (an n×p matrix).

Matrix multiply(const Matrix& a, const Matrix& b) {
    int n = a.size();
    int m = a[0].size();
    int p = b[0].size();
    Matrix c(n, vector<int>(p));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < p; ++j) {
            int sum = 0;
            for (int k = 0; k < m; ++k) {
                sum = sum + a[i][k] * b[k][j];
            }
            c[i][j] = sum;
        }
    }
    return c;
}
Matrix multiplication

// Pre: a is a non-empty n×m matrix, b is a non-empty m×p matrix.
// Returns a×b (an n×p matrix).

Matrix multiply(const Matrix& a, const Matrix& b) {
    int n = a.size();
    int m = a[0].size();
    int p = b[0].size();
    Matrix c(n, vector<int>(p, 0));

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < p; ++j) {
            for (int k = 0; k < m; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    return c;
}
Matrix multiplication

// Pre: a is a non-empty n×m matrix, b is a non-empty m×p matrix.
// Returns a×b (an n×p matrix).

Matrix multiply(const Matrix& a, const Matrix& b) {
    int n = a.size();
    int m = a[0].size();
    int p = b[0].size();
    Matrix c(n, vector<int>(p, 0));

    for (int j = 0; j < p; ++j) {
        for (int k = 0; k < m; ++k) {
            for (int i = 0; i < n; ++i) {
                c[i][j] += a[i][k]*b[k][j];
            }
        }
    }

    return c;
}