Matrices

Jordi Cortadella
Department of Computer Science
Matrices

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

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

// 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.

```python
my_matrix:
[[3, 8, 1, 0],
 [5, 0, 6, 3],
 [7, 2, 9, 4]]

```
$n$-dimensional vectors

- Vectors with any number of dimensions can be declared:

```
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))));
```
Sum of matrices

• 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 $\text{Matrix}[i][j]$
typedef vector< vector<int> > Matrix;

// Pre: A and B are non-empty matrices with 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;
}
typedef vector<vector<int>> Matrix;

// Pre: A and B are non-empty matrices with 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:

```cpp
void Transpose (Matrix& A);
```

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

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

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

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

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

bool is_symmetric(const Matrix& A) {
    int n = A.size();
    for (int i = 0; i < n - 1; ++i) {
        for (int j = i + 1; j < n; ++j) {
            if (A[i][j] != A[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. Otherwise, it will return \((-1, -1)\).

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

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

void search(const Matrix& A, int x, int& i, int& j) {

    int nrows = A.size();
    int ncols = A[0].size();

    for (i = 0; i < nrows; ++i) {
        for (j = 0; j < ncols; ++j) {
            if (A[i][j] == x) return;
        }
    }

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

• A sorted matrix $A$ is one in which

\[
A[i][j] \leq A[i][j+1] \\
A[i][j] \leq A[i+1][j]
\]

\[
\begin{array}{cccccc}
1 & 4 & 5 & 7 & 10 & 12 \\
2 & 5 & 8 & 9 & 10 & 13 \\
6 & 7 & 10 & 11 & 12 & 15 \\
9 & 11 & 13 & 14 & 17 & 20 \\
11 & 12 & 19 & 20 & 21 & 23 \\
13 & 14 & 20 & 22 & 25 & 26 \\
\end{array}
\]
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.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>13</td>
<td>14</td>
<td>17</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>20</td>
<td>22</td>
<td>25</td>
<td>26</td>
<td></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!

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>12</th>
</tr>
</thead>
<tbody>
<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

- **Invariant**: if the element is in the matrix, then it is located in the sub-matrix $[0...i, j...ncols-1]$
Search in a sorted matrix

// Pre:  A 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 A. In case x is not in A, then i=j=-1

void search(const Matrix& A, int x, int& i, int& j) {

    int nrows = A.size();
    int ncols = A[0].size();

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

    i = -1;
    j = -1;
}

Introduction to Programming © Dept. CS, UPC
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>nrows × ncols</td>
</tr>
<tr>
<td>Sorted matrix</td>
<td>nrows + ncols</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;
}
Summary

• Matrices can be represented as vectors of vectors. N-dimensional matrices can be represented as vectors of vectors of vectors ...

• Recommendations:
  – Use indices i, j, k, ..., consistently to refer to rows and columns of the matrices.
  – Use const reference parameters (const Matrix&) whenever possible to avoid costly copies of large matrices.