CS 101: Computer Programming and Utilization

11-Matrices
Multidimensional Arrays

Instructor: Sridhar Iyer
IIT Bombay
Activity – Load balancing

There are two trucks, A and B, each having packages with different weights. We wish to balance the load in both the trucks by swapping exactly one pair of packages between them.

Write a program to determine if such load balancing is possible for a given set of input weights, and if yes, identify which packages to swap. Assume that all weights are integers (e.g. 85 Kg, 23 Kg, 7 Kg, etc.).

Discuss: pseudo-code; Run: demo11-balance.cpp
Load balancing trucks – basic idea

If $x$ is element of $A$, and $y$ element of $B$, we need to find $x$ and $y$ such that, if $x$ and $y$ are exchanged, the resulting sums match.

$$\text{sum}_A + y - x = \text{sum}_B - y + x \quad \text{(after exchanging } x \text{ and } y)$$

or

$$y = \frac{\text{sum}_B - \text{sum}_A}{2} + x$$

For some $i$, if we are looking at $A[i]$; then for some $j$, we must have $B[j] = \frac{\text{sum}_B - \text{sum}_A}{2} + A[i]$
Multidimensional Arrays

We can use arrays with more than one dimension

```c
int A[50][40];
```

- Declares a 2D array with 50 rows and 40 columns
- Each element is accessed by a reference requiring two index expressions, e.g., `A[i][j] = 37;`
  - Row index ‘i’, can have a value from 0 to 49,
  - Column index ‘j’ can have a value from 0 to 39
- Violations may lead to silent garbage
- All rules for index expression apply to index for each dimension
Storage: row major

- Internally, no distinction between [3][5] and [15]
- cellNumber(rx, cx) = rx * cols + cx
- Base address of imat[rx][cx] is base address of imat plus 4 * cellNumber(rx, cx)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Cell 0, Byte 0</td>
<td>1, 4</td>
<td>2, 8</td>
<td>3, 12</td>
<td>4, 16</td>
</tr>
<tr>
<td>1</td>
<td>5, 20</td>
<td>6, 24</td>
<td>7, 28</td>
<td>8, 32</td>
<td>9, 36</td>
</tr>
<tr>
<td>2</td>
<td>10, 40</td>
<td>11, 44</td>
<td>12, 48</td>
<td>13, 52</td>
<td>14, 56...59</td>
</tr>
</tbody>
</table>
Matrix manipulations

2D arrays are commonly used to represent Matrices and are found in programs such as:

- **Matrix multiplication**
  - Given: m x n matrix A; n x p matrix B, the matrix product C = AB will be m x p matrix.
  - \( C[i][j] = \sum_{k=0}^{n-1} A[i][k] \times B[k][j] \)

- **Solving linear equations by Gaussian elimination**
Basic operations

• Initialize square matrix to identity

for (int rx = 0; rx < rows; ++rx) {
    for (int cx = 0; cx < cols; ++cx) {
        dmat[rx][cx] = (rx == cx)? 1 : 0;
    }
}

• Better code uses two loops (why?)
  • In first rx,cx loop set all elements to zero

for (int rx=0; rx<rows; ++rx) {
    for (int cx=0; cx<cols; ++cx) { dmat[rx][cx]=0; }
}

  • In second loop set diagonal elements to one

for (int dx=0; dx<rows; ++dx) {
    dmat[dx][dx]=1;
}
**Think-Pair-Share**: Write a program to transpose a square matrix

- **Think (Individually)**: Write the pseudo-code in your notebook; See example on next slide.

- **Pair (with your neighbour)**: Write the C++ code.

- **Share (with class)**: Compare your solution with others.
Transpose a square matrix – example
Write program using **Think-Pair-Share**
Transpose a square matrix - code

- Don’t double transpose back to square one!

```c
for (int rx = 0; rx < rows; ++rx) {
    for (int cx = rx+1; cx < cols; ++cx) {
        float tmp = fmat[rx][cx];
        fmat[rx][cx] = fmat[cx][rx];
        fmat[cx][rx] = tmp;
    }
}
```

No need to transpose diagonal or below

Swap \([rx][cx]\) with \([cx][rx]\)
Matrix vector multiplication – example
Write program using **Think-Pair-Share**

![Matrix and Vector Multiplication](image.png)

- **Think**: Write the pseudo-code in your notebook.
- **Pair**: Write the C++ code.
- **Share**: Compare your solution with others.
Matrix-vector multiplication - code

```c
float A[rows][cols];
float x[cols], y[rows];
// fill up A and x
for (int rx=0; rx < rows; ++rx) {
    y[rx] = 0;
    for (int cx=0; cx < cols; ++cx) {
        y[rx] += A[rx][cx] * x[cx];
    }
}
```
Matrix-matrix multiplication

float amat[lsize][msize], bmat[msize][nsize],
    cmat[lsize][nsize];
for (int crx=0; crx<lsize; ++crx) {
    for (int ccx=0; ccx<nsize; ++ccx) {
        cmat[crx][ccx] = 0;
        for (int abx=0; abx<msize; ++abx) {
            cmat[crx][ccx] +=
                amat[crx][abx] * bmat[abx][ccx];
        }
    }
}

- Like matrix-vector multiplication
- But x has many columns
- Triple nested loop
- Time required is lsize*msize*nsize
Gaussian elimination method

Consider two equations: $2x + 4y = 8$ and $4x + 5y = 1$

They can be represented as:

\[
\begin{bmatrix} 2 & 4 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \end{bmatrix}
\]

Gauss elimination method involves transformations (by using multiplication and linear combinations) to reduce the coefficient matrix to upper triangular form:

\[
\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}
\]

Then solving by back-substitution to get $y = 5$, $x = -6$. 
Gaussian elimination representation

The previous coefficient matrix can be stored as:

A general system of $n$ equations can be written as

$$a[][] \times[] = b[]$$

With the coefficient matrix in upper triangular form, we have the following system, using which, back substitution can be applied

$$x[0] + a[0][1] \times[1] + \ldots + a[0][n-1] \times[n-1] = b[0]$$
$$x[1] + \ldots + a[1][n-1] \times[n-1] = b[1]$$
$$\ldots$$
$$x[n-1] = b[n-1]$$
Gaussian elimination pseudo-code

- Read matrices $A[][][]$ and $B[]$
- For each row
  - Divide the row by the coeff on the diagonal
  - Recalculate all the coeffs in that row
  - Normalize (recalculate) the RHS of that row
- Replace subsequent rows, by subtracting the appropriate portion of the $i$th equation from it
- Do back-substitution starting from the last row
  - Sum up the $i$th row using the values of $x[]$ already determined
- Output the results – values of $x[i]$
Gaussian elimination program

- Work with your neighbour to write the C++ code
- Compare with: demo11-gauss.cpp
Handling large inputs

- Tedious to type the input values one at a time!
- Solution: I/O Redirection …

- When we execute our program, OS assigns it three standard files - stdin, stdout, stderr
- By default, OS ‘connects’ these to devices:
  - stdin to keyboard, stdout and stderr to monitor

- `./a.out < input.txt` → Redirects stdin from a file
- `./a.out > output.txt` → Redirects stdout to a file