ESC101: Introduction to Computing

Multi dimensional Arrays

Why Multidimensional Arrays?

- Marks of 800 students in 5 subjects each.
- Distance between cities
- Sudoku
- All the above require 2D arrays
- Properties of points in space (Temperature, Pressure etc.)
- Mathematical Plots
- > 2D arrays

Multidimensional Arrays

Multidimensional arrays are defined like this:

```
double  mat[5][6];
```
OR
```
int mat[5][6];
```
OR
```
float mat[5][6];
```
etc.

The definition states that mat is a 5 X 6 matrix of doubles (or ints or floats). It has 5 rows, each row has 6 columns, each entry is of type double.

mat

2.1	1.0	-0.11	-0.87	31.5	11.4
-3.2	-2.5	1.678	4.5	0.001	1.89
7.889	3.333	0.667	1.1	1.0	-1.0
-4.56	-21.5	1.0e7	-1.0e-9	1.0e-15	-5.78
45.7	26.9	-0.001	1000.09	1.0e15	1.0

Accessing matrix elements-I

1. The (i,j) th member of mat is accessed as mat[i][j]. Note the slight difference from the matrix notation in maths.
2. The row and column numbering each start at 0 (not 1).
3. The following program prints the input matrix.

```
void print_matrix(double mat[5][6]) {
    int i,j;
    for (i=0; i < 5; i=i+1) {      /* prints the ith row i = 0..4. */
      for (j=0; j < 6; j = j+1) {
         printf("%f ", mat[i][j]);
      }
      printf("\n");   /* prints a newline after each row */
    }
}
```

/* In each row, prints each of the six columns j=0..5 */

Accessing matrix elements-II

1. Code for reading the matrix from the terminal.
2. The address of the i,j th matrix element is &mat[i][j].
3. This works without parentheses since the array indexing operator [] has higher precedence than &.

```
void read_matrix(double mat[5][6]) {
    int i,j;
    for (i=0; i < 5; i=i+1) {
        for (j=0; j < 6; j = j+1) {
          scanf("%f ", &mat[i][j]);
        }
    }
}
```

/* read the ith row i = 0..4. */

/* In each row, read each of the six columns j=0..5 */

scanf with %f option will skip over whitespace.

So it really doesn't matter whether the entire input is given in 5 rows of 6 doubles in a row or all 30 doubles in a single line, etc..

Initializing 2 dimensional arrays

We want a[4][3] to be this 4 X 3 int matrix.

1	2	3
4	5	6
7	8	9
0	1	2

Initialize as

```
int  a[][3] = {
                {1,2,3},
                 {4,5,6},
                {7,8,9},
                 {0,1,2}
      };
```

Initialization rules:

1. Most important: values are given row-wise, first row, then second row, so on.
2. Number of columns must be specified.
3. Values in each row are enclosed in braces {…}.
4. Number of values in a row may be less than the number of columns specified. Remaining col values set to 0 (or 0.0 for double, '\0' for char, etc.)

```
int a[][3] = { {1}, {2,3}, {3,4,5} };
```

gives this matrix for a:

1	0	0
2	3	0
3	4	5

Accessing matrix elements

```
void read_matrix(double mat[5][6]) {
    int i,j;
    for (i=0; i < 5; i=i+1) {
        for (j=0; j < 6; j = j+1) {
          scanf("%f ", &mat[i][j]);
        }
    }
}
```

/* read the ith row i = 0..4. */

/* In each row, read each of the six columns j=0..5 */

Could I change the formal parameter to mat[6][5]? Would it mean the same? Or mat[10][3]?

That would not be correct. It would change the way elements of mat are addressed. We will discuss this in details later.

Coin Collection: Practice Problem

You have an $n \times n$ grid with a certain number of coins in each cell of the grid. The grid cells are indexed by (i, j) where $0 \leq i, j \leq n - 1$.

For example, here is a 3x3 grid of coins:

	0	1	2
0	5	8	2
1	3	6	9
2	10	15	2

Coin Collection: Problem Statement

- You have to go from cell (0, 0) to (n-1, n-1).
- Whenever you pass through a cell, you collect all the coins in that cell.
- You can only move right or down from your current cell.

Goal: Collect the maximum number of coins.

Consider the example grid

5	8	2
3	6	9
10	15	2

There are many ways to go from (0,0) to (n-1,n-1)

5	8	2
3	6	9
10	15	2

Total = 35

5	8	2
3	6	9
10	15	2

Total = 25

5	8	2
3	6	9
10	15	2

Total = 31

5	8	2
3	6	9
10	15	2

Total = 30

5	8	2
3	6	9
10	15	2

Total = 23

5	8	2
3	6	9
10	15	2

Total = 36

Max = 36

Building a Solution

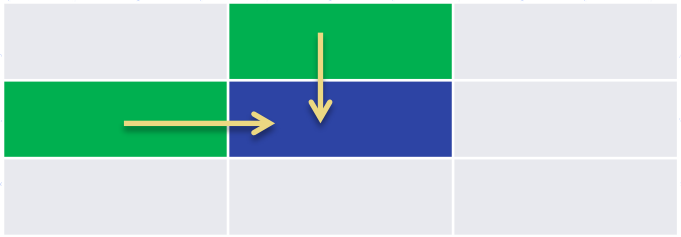
- We cannot afford to check every possible path and find the maximum.
  - Why?

In an $n \times n$ grid, how many such paths are possible?

- Instead we will iteratively try to build a solution.

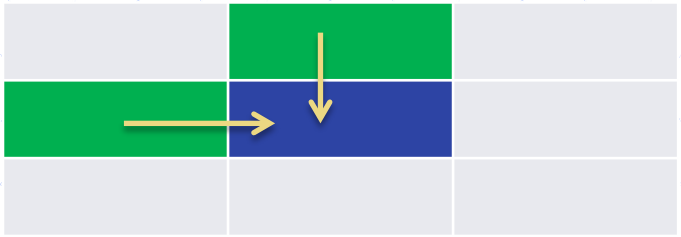
Solution Idea

- Consider a portion of some matrix



- What is the maximum number of coins that I can collect when I reach the blue cell?
  - This number depends only on the maximum number of coins that I can collect when I reach the two green cells!
  - Why? Because I can only come to the blue cell via one of the two green cells.

Solution Idea (dynamic programming)



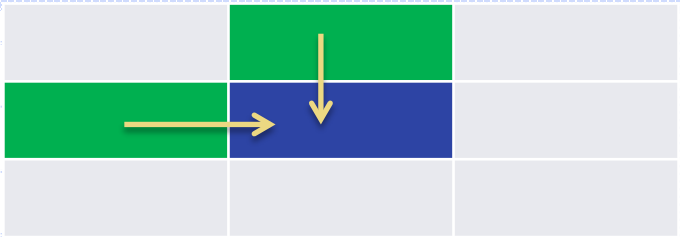
Max-coins (bluecell) =

max(Max-coins (greencell-1),

Max-coins (greencell-2))

+ No. of coins (bluecell))

Solution Idea



- Let a(i,j) be the number of coins in cell(i,j)
- Let coin(i,j) be the maximum number of coins collected when travelling from (0,0) to (i,j).
- Then,

coin(i,j) = max(coin(i,j-1), coin(i-1,j)) + a(i,j))

Implementation

- Use an additional two dimensional array, whose (i,j)-th cell will store the maximum number of coins collected when travelling from (0,0) to (i,j).
- Fill this array one row at a time, from left to right.
- When the array is completely filled, return the (n-1, n-1)-th element.

Implementation: Boundary Cases

- To fill a cell of this array, we need to know the information of the cell above and to the left of the cell.
- What about elements in the top most row and left most column?
  - Cell in top row: no cell above
  - Cell in leftmost column: no cell on left
- Before starting with the other elements, we will fill these first.

```
int coin_collect(int a[][100], int n){
  int i,j, coins[100][100];

  coins[0][0] = a[0][0]; //initial cell

  for (i=1; i<n; i++) //first row
    coins[0][i] = coins[0][i-1] + a[0][i];

  for (i=1; i<n; i++) //first column
    coins[i][0] = coins[i-1][0] + a[i][0];

  for (i=1; i<n; i++) //filling up the rest of the array
    for (j=1; j<n; j++)
      coins[i][j] = max(coins[i-1][j], coins[i][j-1])
                    + a[i][j];

  return coins[n-1][n-1]; //value of last cell
}
```

```
int max(int a, int b){
  if (a>b) return a;
  else return b;
}

int main(){
  int m[100][100],i,j,n;

  scanf("%d", &n);
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      scanf("%d", &m[i][j]);

  printf("%d\n", coin_collect(m,n));
  return 0;
}
```