UNIVERSITY OF WITWATERSRAND

ELEN4020

DATA INTENSIVE COMPUTING IN DATA SCIENCE

Laboratory Exercise 1

Authors:
KISHAN NAROTAM - 717 931
JESAL CHANA - 603 177

Authors:
SYED HUSSAIN - 600 524
YUSUF ALLY - 604 973

23rd February 2018

I. PROBLEM DESCRIPTION

A K-dimensional integer array, represented as $A[N_0][N_1]...[N_{K-1}]$, must be created for different values of K. The integer array is bounded by $N_0, N_1, ..., N_{K-1}$, and once created, the array must undergo three procedures which are as follows:

1) Initialize all elements in the array to be zeroes
2) Set 10% of all elements in the array uniformly to 1's
3) In a random fashion across all dimensions in the array:

    a) Choose 5% of the elements
    b) Print the coordinate indices of the elements
    c) Print the value of the element at the chosen cell.

Four integer arrays must be created, each with various dimension size as well as array size. For each of these arrays, the three procedures must be called, however it must be completed as if the number of dimensions in the array are not static.

II. BACKGROUND

An array is defined as a data structure that contains a group of elements, in the case of this problem the elements are integers [1]. An array can have multiple dimensions, for example an array with one dimension is regarded as a row, whereas an array with two dimensions is regarded as a matrix with a row and a column. However as the dimensions of the matrix reach three or more, the array has a new coordinate property known as depth/page [2], [3]. Figure 1 in Appendix A is a representation of a 3-dimensional array with the coordinate properties. As the dimensions increase it becomes more difficult to represent these arrays in a graphical representation.

A. Memory Allocation

The memory and layout of a K-dimensional array could be allocated in multiple ways. Two of the most common ways of this memory layout are row-major order or column-major order.

In a 2D array, the row-major order layout would put the first row in contiguous memory, the second row would follow, and so on. Similarly, for column-major order layout, the first column would be in contiguous memory, the second column would follow and so on [2].

In a 3D array, using the dimensions of row, column and depth/page, Figure 2 in Appendix A shows how a 3-dimensional array memory is allocated and how the layout is presented in row-major order [2].

In order to calculate the memory location of an element from its respective index in the array, the following formulae can be used for row-major order and column-major order respectively for a K-dimensional array.

$$offset = n_K + N_K \cdot (n_{K-1} + N_{K-1} \cdot (n_{K-2} + N_{K-2} \cdot$$
$$(... + N_2 n_1)...))$$
$$= \sum_{i=1}^{K} \left(\prod_{j=i+1}^{K} N_j \right) n_i \tag{1}$$

$$offset = n_1 + N_1 \cdot (n_2 + N_2 \cdot (n_3 + N_3 \cdot (... + N_{K-1} n_K)...))$$
$$= \sum_{i=1}^{K} \left(\prod_{j=1}^{i-1} N_j \right) n_i \tag{2}$$

III. FUNCTION DESCRIPTIONS

A. allocateArray

This function has two main responsibilities : to dynamically allocate the desired array, and to then call the three procedures that execute on the array. This function takes as inputs the initializer array (`iniArray[]`), and the number of dimensions.

1) Dynamic memory allocation: The array is generated using `malloc`, which allocates a contiguous memory block. The size of the block is determined by the number of elements in the array and the number of bytes per element.

2) Calling procedures: Once the array has been generated, the three procedures are called in succession.

B. procedureOne

This procedure sets every element in the array to zero. It takes as inputs the array, and the number of elements in the array (in order to know how many elements it should step through).

C. procedureTwo

This procedure sets the first 10% of elements of the array to one. Like `procedureOne` it takes as inputs the array and the number of elements in the array. Variable `tenPercentElements` is the value of 10% of elements of the array, and is used to determine whether an element should be set to one.

D. procedureThree

This procedure randomly chooses 5% of elements in the array and prints the corresponding value and coordinate indices of these elements.

1) Random coordinate generation: The initialized array is in the form `int iniArray[] = ` $\{N_0, N_1, ..., N_{K-1}\}$, where K is the array dimension and $N_0, N_1, ...$ are the sizes of each dimension. For example, `int iniArray[] = ` $\{3, 3\}$ creates a 2D $(3X3)$ array.

A random coordinate for all K indexes is generated using $rand()\%N_K$, where $\%N_K$ bounds each coordinate index. This bounding ensures a valid coordinate is created every time. Each coordinate is then stored in `coordinate_array`.

2) Coordinate mapping: The randomly generated coordinate is now mapped to its 1D linear array index. This is done using the row major order indexing equation (lines 150-156). The corresponding value is obtained using this index. The coordinate and value is then printed, for 5% of elements in the array.

IV. PSEUDO CODE

A. `main` Function

Algorithm 1 `main` Function

$srand(time(NULL))$
$Dimension$

$arrayInitializer1[\,] \leftarrow (100, 100)$
$Dimension \leftarrow sizeof(arrayInitializer1)/sizeof(int)$
$allocateArray(arrayInitializer1, Dimension)$
$arrayInitializer2[\,] \leftarrow (100, 100, 100)$
$Dimension \leftarrow sizeof(arrayInitializer2)/sizeof(int)$
$allocateArray(arrayInitializer2, Dimension)$
$arrayInitializer3[\,] \leftarrow (50, 50, 50, 50)$
$Dimension \leftarrow sizeof(arrayInitializer3)/sizeof(int)$
$allocateArray(arrayInitializer3, Dimension)$
$arrayInitializer4[\,] \leftarrow (20, 20, 20, 20, 20)$
$Dimension \leftarrow sizeof(arrayInitializer4)/sizeof(int)$
$allocateArray(arrayInitializer4, Dimension)$

B. `allocateArray` Function

Algorithm 2 `allocateArray` Function

Input: $iniArray[\,]$ and $dimension$

$num\_elements \leftarrow 1$
for $i = 0$ to $dimension$ **do**
    $num\_elements \leftarrow num\_elements * iniArray[i]$
end for
refToInteger $array$
$array \leftarrow$ refToInteger $malloc(sizeof($refToInteger $num\_elements))$
Display "Dimensions $dimensions$, Elements $num\_elements$"
if $array \leftarrow NULL$ **then**
    Display "malloc failed"
    $exit(1)$
end if

for $i = 0$ to $num\_elements$ **do**
    $array[i] \leftarrow i$
end for
$procedureOne(array, num\_elements)$
$procedureTwo(array, num\_elements)$
$procedureThree(array, iniArray, num\_elements, dimension)$

C. `procedureOne` Function

Algorithm 3 `procedureOne` Function

Input: $arr[\,]$ and $numElements$

Display "Procedure 1: "
for $i = 0$ to $numElements$ **do**
    $arr[i] \leftarrow 0$
end for
Display "...Executed procedure 1..."

D. `procedureTwo` Function

Algorithm 4 `procedureTwo` Function

Input: $arr[\,]$ and $numElements$

Display "Procedure 2: "
$tenPercentElements \leftarrow numElements/10$
for $i = 0$ to $numElements$ **do**
    if $i < tenPercentElements$ **then**
        $arr[i] \leftarrow 1$
    end if
end for
Display "...Executed procedure 2..."

E. `procedureThree` Function

Algorithm 5 `procedureThree` Function

Input: $arr[\,]$, $arr2[\,]$, $numElements$, $dimensions$

V. FINAL CODE AND OUTPUT

VI. CONCLUSION

The conclusion goes here.

REFERENCES

[1] Christensson, P; *Array Definition*; https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays; Last Accessed: 21/02/2018
[2] Bendersky, E; *Memory layout of multi-dimensional arrays - Eli Bendersky's website*; https://eli.thegreenplace.net/2015/memory-layout-of-multi-dimensional-arrays; Last Accessed: 21/02/2018
[3] Mathworks; *Multidimensional Arrays - MATLAB & Simulink*; https://www.mathworks.com/help/matlab/math/multidimensional-arrays.html; Last Accessed: 21/02/2018
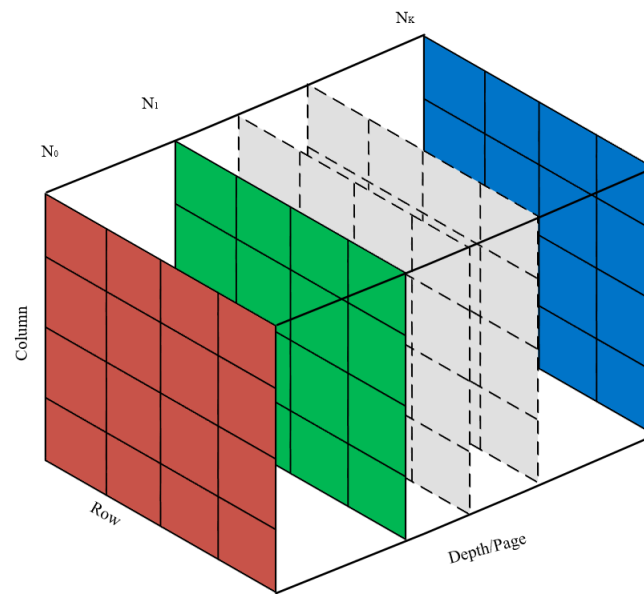
APPENDIX A
DIAGRAMS

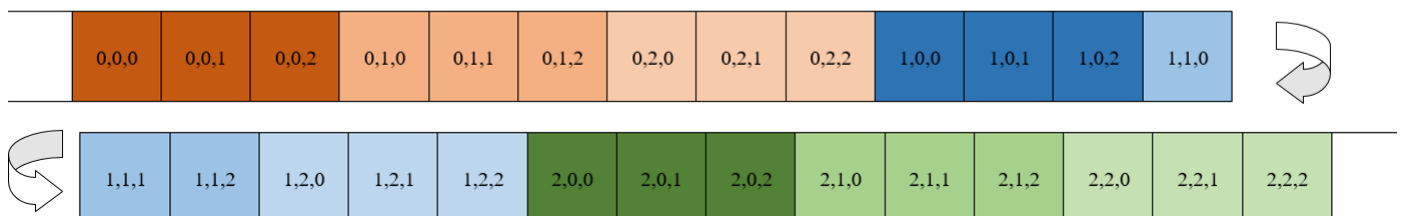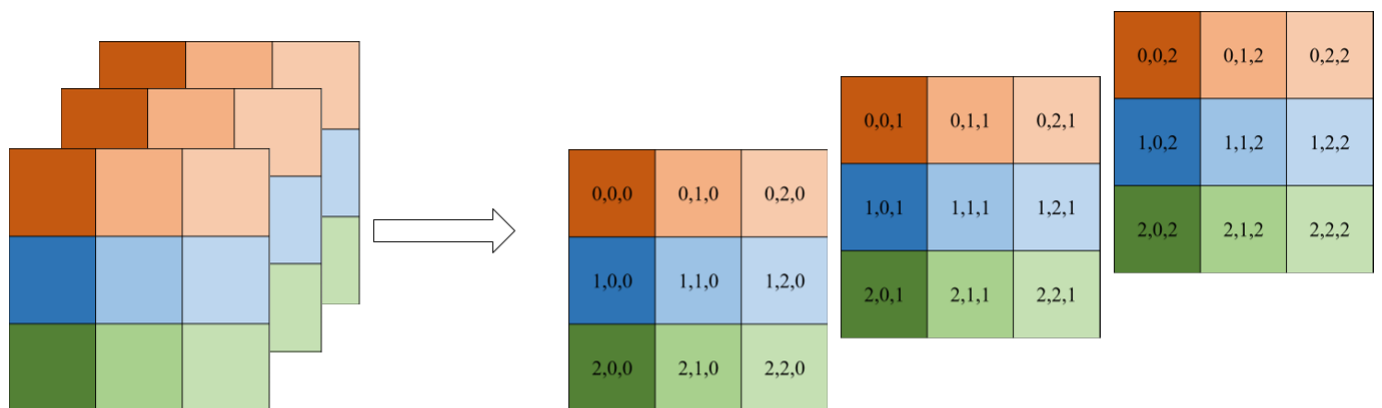Fig. 1. Representation of a *3*-dimensional array

Fig. 2. Memory allocation of a 3-dimensional in row-major order [2]

APPENDIX B
CODING