ELEN4020 Lab One Report

Arlo Eardley (1108472), Carel Ross (1106684) and Ryan Verpoort (1136745)
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg 2050, South Africa

I. Overview

Three procedures are written in the C programming language. The main function has inputs for the K-dimension and corresponding N dimension. A one dimensional integer pointer array A, of N length is then declared. Each index in array A points to a corresponding array of E/N length as well, where E is the total number of elements in the total array. A dynamically sized array is then achievable. This allows for an array of K-by-N-dimensions. This concept is demonstrated by the example below:

K = 3
N = 4
Since K is three, the array A is of size K, where each value is a pointer to an array of size N:

This can be represented as follows:
A[N][N][N]

The parameters for all procedures are the same. The parameters are as follows:

- Pointer array of integers, denoted as A
- The number of dimensions, denoted as K
- The size of the array attached to each K value, denoted as N
- The total number of elements, denoted as E

By making A an array of pointers, it allows for larger amounts of data to be processed. This is due to multiple arrays being distributed across the available memory since an array of pointers is used. This means that each subsequent N array can be distributed throughout memory. These pointers all point to their specified N arrays. Since new blocks of memory are being allocated to each index of A, these blocks need to be deleted to avoid memory leaks. The method used in this system can be recursively performed to improve the aforementioned phenomenon. In the code demonstrated this process is only completed once to show proof of concept, since further development was unnecessary.

An alternative approach would be to make an extremely large one dimensional array A that contains all subsequent N arrays within itself, it would severely limit the amount of data that can be processed since one large contiguous block of memory would be required to store this information. The aforementioned approach was therefore taken. For example if one had a 10-dimensional array where N = 100, a block of memory with a size of 10^{20} would be required. This is undesirable, therefore the aforementioned method was developed.

II. First Procedure

This procedure initializes all the elements of the array to zero. This procedure is responsible for allocating memory addresses for N number of arrays, each with a size of E/N to each index of A. After a block of memory of size E/N is allocated to a specific index of A, all values within this new array are initialized to zero. The construction of this array can be improved by recursively allocating memory by having pointers, point to each corresponding block of memory.

III. Second Procedure

First the total size of the array is calculated by taking N to the K^{th} power. This value is then multiplied by 0.1 and assigned to an integer value, this means that the decimal values are truncated. This particular design choice is flexible. The procedure will then step through the full array at index 1 then the full array at index 2 until E number of elements have been stepped through. During this iterative process some values will be altered to 1. The changing of values ensures that 10% of

the elements of the overall array are uniformly set to 1's. This uniform distribution of 1's is calculated by the following equation:

$$\frac{\text{TotalElements} - \text{TenPercentOfElements}}{\text{TenPercentOfElements} + 1}$$

After 10% of the elements have been set to 1, the loop will break. This is to ensure only the specified number of 1's exist within the elements. This is due to some division procedures resulting in decimal values that could potentially cause an excess of 1's within the elements.

IV. Third Procedure

First the total size of the array is calculated by taking N to the K^{th} power. This value is then multiplied by 0.05 and assigned to an integer value, this means that the decimal values are truncated. This particular design choice is flexible. An array of size X is then made, where X is 5% of all elements of A. This is regarded as a cache and its purpose is to ensure that the same value in array A is not chosen twice. All values in the cache are initialized to -1. This ensures an index cannot be equal to a negative number. Random values are chosen for the searching index (the index created to access the indices of A) as a value between and inclusive of 0 and E-1 and by manipulating the searching index, the indices of A can be determined. Each searching index is stored within the cache. The resulting random value indicates the coordinate indices of the elements through some manipulation. The modulus of the random value and N is the first index. This value is then subtracted from the initial random value. The proceeding index values are calculated by determining how many times the random value has been divided by N, until the value is zero. After each division iteration, if the value is still greater than zero, the latest index number is the modulus of the current division of the random value and N. The coordinate indices and the corresponding value of the array element are all printed to console.

V. Pseudo-code

Procedure 1

```
for N values
        set A[i] to new malloc array of size E/N
        for new malloc array values to 0
end for loop
```

Procedure 2

```
find size of 10% of elements
for all elements of array
        step through various arrays
        set 10% of array uniformly to 1
        end loop when 10% are set to 1
end for loop
```

Procedure 3

```
find size of 5% of elements
create cache of size 5% of elements
for 5% of elements
        random value (Searching index) = 0 to E-1
        save random value in cache
        first array index = Searching index modulus N
        while random value > 0
                random value = random value/N
                if random value >= 0 next index = random value modulus N
                else next index = N
        end while loop
        print dimensions, corresponding indices and values
end for loop
```

VI. Results

The main function declares an array A that is defined by the bounds N. Memory is allocated for the various arrays and all three procedures operate on them, with the corresponding K and N values, sequentially. Since the procedures accommodate for dynamically sized arrays, the same procedures can be used for various sizes of K and N. The left most array dimension is regarded as the least significant value, whereas the right most array dimension is regarded as the most significant value.

Example output:

For an array with dimensions [2][2][2][2][2]

[0][0][1][0][1] = 20

1. Value 0