Ahmed Shafique, Haiqa Qaiser 2021-EE-313, 2022-EE-255

Data Structures and Algorithms

Lab 6: Matrices-II

Q1): Write the program that:

A function that initializes $Xm \times n$ and $Yl \times k$, where {m, n, l, k} are user defined and also its contents.

Code:

```
void initialize_matrix(int row, int col, int *ptr)
{
    int k=0;
    for(int i=0;i<row;i++)
    {
        for (int j = 0; j < col; j++)
        {
            cin>>k;
            *(ptr + (i*col)+ j) = k;
        }
    }
    return;
}
```

A function that evaluates $Z = X \times Y$.

Code:

```
element matrices_multiplication(int r1, int c1, int r2, int c2, int
*p1, int *p2)
{
    element matrix;
    if (c1 == r2)
    {
        matrix.c = c2;
        matrix.r = r1;
        matrix.p = new int [r1*c2];
        for (int i = 0; i < r1; i++)
        {
            for (int j = 0; j < c2; j++)
            {
                long long sum = 0;
                for (int k = 0; k < r2; k++)
                {
                    sum += *(p1 + i*r2 + k) * *(p2 + k * c2 + j);
                }
                *(matrix.p + i *c2 + j) =  sum;
                sum = 0;
            }

        }

    }
    else
    {
        cout << "Can't perform multiplication" << endl;
        matrix.c = 0;
        matrix.r = 0;
        matrix.p = NULL;
    }
    return matrix;
}
```

A function that evaluates $Z = \det(X)$.

Code:

```
double determinant(int m, int *p)
{
    double ans = 0;
    double inner_determinant, inner_sol;
    int a, b, c, d;
    if ((m == 1) || (m = 2))
    // stopping criteria
    {
        if (m == 1)
        {
            ans = *p;
        }
        else
        {
            a = *p;
            b = *(p+1);
            c = *(p+2);
            d = *(p+3);
            ans = (a*d) - (b*c);
        }
    }
    else
    {
        int n, l, sign, basic, element;
        n = 0;
        sign = +1;
        int *q;
        q = new int [(m-1)*(m-1)];
        for (int i = 0; i < m; i++)
        {
            l = 0;
            n = 0;
            basic = *(p+i);
            for (int j = 0; j < m; j++)
            {
                for (int k = 0; k < m; k++)
                {
                    element = *(p+l);
                    cout<<element<<" ";
                    if ((j==0) || (i==k));
                    else
                    {
                        *(q + n) = element;
                        n = n + 1;
                    }
                    l = l + 1;
                }
            }

            cout<<endl<<basic<<"x"<<endl;
            printing((m-1),(m-1), q);
            inner_determinant = determinant(m-1, q);
            inner_sol = sign * basic * inner_determinant;
            cout<<" sign "<<sign<<" x basic "<<basic<<" x Determinant
"<<inner_determinant<<" = "<<inner_sol<<endl;
            ans = ans + inner_sol;
            sign = sign * -1;
        }
        delete [] q;
    }
    return ans;
```

```
    }
```

A function called ‘print’ that simply prints the answer.

Code:

```
void printing(int row, int col, int *ptr)
{
    for (int i = 0; i < row; i++)
    {
        cout << "[";
        for (int j = 0; j < col; j++)
        {
            cout << " " << *ptr; // Corrected dereferencing here
            ptr = ptr + 1;
        }
        cout << " ]" << endl;
    }
    return;
}
```

Output:

```
D:\6th Semester\DSA\Lab1\DSA_lab>matrix_det_and_multi.exe matrix_det_and_multi.cpp array_data.txt
```

Fig: 01 CMD Input

```
Resultant Matrix:
[ 6525 5778 3844 ]
[ 2388 2187 1373 ]
[ 4090 3708 4990 ]
Determinant: -6.62893e+006
```

Fig: 02 CMD Output

DSA_lab > array_data.txt

```
1  54 89 54 12 31 45 87 45 55 10 9 45 63 54 8 7
```

Fig: 03 Input text file

Methodology:

This code implements matrix multiplication and determinant calculation. It begins by prompting the user to input the dimensions of two matrices. Memory space is then allocated for both matrices, and the user is asked to input their respective elements. The code then performs matrix multiplication if the number of columns in the first matrix matches the number of rows in the second matrix.

During multiplication, it dynamically allocates memory for the resultant matrix and computes each element using nested loops. The function for computing the determinant is also defined, recursively calculating the determinant of a matrix using cofactor expansion until it reaches the base case of a 1x1 or 2x2 matrix.

The determinant function handles matrix sizes recursively, calculating the determinant by recursively calling itself for submatrices. It prints intermediate steps, including individual determinants and their corresponding signs. Finally, it prints the resultant matrix and the calculated determinant.

Memory allocated for the matrices is properly deallocated at the end to prevent memory leaks. Overall, the code facilitates matrix multiplication and determinant calculation with clear user prompts and informative output messages throughout the process.