Name: Khushi Nitinkumar Patel
PRN: 2020BTECS00037
Batch: T5

Design and analysis of algorithm Lab

Week 4 Assignment

Part 2: Divide and conquer strategy

Strassen's Matrix Multiplication

A) Implement **Naive Method** multiply two matrices. and justify Complexity is $O(n^3)$
B) Implement **Divide and Conquer** multiply tow matrices . and justify Complexity is $O(n^3)$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

A B C

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

C) Implement **Strassen's Matrix Multiplication** and justify Complexity is $O(n^{2.8})$

$$p1 = a(f - h) \qquad p2 = (a + b)h$$
$$p3 = (c + d)e \qquad p4 = d(g - e)$$
$$p5 = (a + d)(e + h) \qquad p6 = (b - d)(g + h)$$
$$p7 = (a - c)(e + f)$$

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

A) Naive Method:

Algorithm:

Algorithm matrix_multiply(a,b,c) // a and b are input matrices of size n x n and c is the output matrix of size n x n

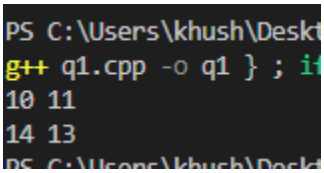
```
for i = 0 to n-1 do
   for j = 0 to n-1 do
      for k = 0 to n-1 do
         c[i][j]= c[i][j] + a[i][k]*[k][j]
      end
   end
```

end

Code:

```cpp
#include<iostream>
using namespace std;
#include<vector>
void matrix_multiply(int a[][2],int b[][2],int c[][2])
{
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            for(int k=0;k<2;k++)
            {
                c[i][j]+=a[i][k]*b[k][j];
            }
        }
    }
}
int main()
{
    int a[2][2]={{1,2},{3,2}},b[2][2]={{2,1},{4,5}},c[2][2]={0};
    matrix_multiply(a,b,c);
    for(int i=0;i<2;i++)
    {
        for(int j=0;j<2;j++)
        {
            cout<<c[i][j]<<" ";
        }
        cout<<endl;
    }
    return 0;
}
```

Output:



Complexity Analysis:

Time complexity: $O(n^3)$ where n is size of matrix

Space complexity: $O(n^2)$ As extra 2D array is used to store result

B) Divide and Conquer Method:

Algorithm:

Algorithm multiply_matrix(A, B, n) //A and B are input matrices and n is size of matrices

if(n<=2)

c11= (a11*b11) + (a12*b21)

c12= (a11*b12) + (a12*b22)

c21= (a21*b11) + (a22*b21)

c22= (a21*b12) + (a22*b22)

else

multiply_matrix(A11, B11, n/2) + (A12, B21, n/2)

multiply_matrix(A11, B12, n/2) + (A12, B22, n/2)

multiply_matrix(A21, B11, n/2) + (A22, B21, n/2)

multiply_matrix(A21, B12, n/2) + (A22, B22, n/2)

end

Code:

```
#include <iostream>
using namespace std;
#include<iomanip>
#include<vector>
void add_matrix(vector<vector<int>> A,vector<vector<int>> B,vector<vector<int>>&
C,int split_index)
{
    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++)
            C[i][j]= A[i][j] + B[i][j];
}
vector<vector<int>>multiply_matrix(vector<vector<int>> A,vector<vector<int>> B)
{
    int col_1 = A[0].size();
    int row_1 = A.size();
    int col_2 = B[0].size();
    int row_2 = B.size();
    if (col_1 != row_2) {
        cout << "\nError: The number of columns in Matrix "
                "A must be equal to the number of rows in "
                "Matrix B\n";
        return {};
    }
    vector<int> row(col_2, 0);
    vector<vector<int>> ans(row_1,row);
    if (col_1 == 1) ans[0][0]= A[0][0] * B[0][0];
    else
    {
        int split_index = col_1 / 2;
        vector<int> row_vector(split_index, 0);
        vector<vector<int>> ans_00(split_index,row_vector);
        vector<vector<int>> ans_01(split_index,row_vector);
        vector<vector<int>> ans_10(split_index,row_vector);
        vector<vector<int>> ans_11(split_index,row_vector);
        vector<vector<int>> a00(split_index, row_vector);
        vector<vector<int>> a01(split_index, row_vector);
        vector<vector<int>> a10(split_index, row_vector);
        vector<vector<int>> a11(split_index, row_vector);
        vector<vector<int>> b00(split_index, row_vector);
        vector<vector<int>> b01(split_index, row_vector);
        vector<vector<int>> b10(split_index, row_vector);
        vector<vector<int>> b11(split_index, row_vector);
        for (auto i = 0; i < split_index; i++)
```

```
            for (auto j = 0; j < split_index; j++) {
                a00[i][j] = A[i][j];
                a01[i][j] = A[i][j + split_index];
                a10[i][j] = A[split_index + i][j];
                a11[i][j] = A[i + split_index][j + split_index];
                b00[i][j] = B[i][j];
                b01[i][j] = B[i][j + split_index];
                b10[i][j] = B[split_index + i][j];
                b11[i][j] = B[i + split_index][j + split_index];
            }
        add_matrix(multiply_matrix(a00, b00),multiply_matrix(a01, b10),ans_00,
split_index);
        add_matrix(multiply_matrix(a00, b01),multiply_matrix(a01, b11),ans_01,
split_index);
        add_matrix(multiply_matrix(a10, b00),multiply_matrix(a11, b10),ans_10,
split_index);
        add_matrix(multiply_matrix(a10, b01),multiply_matrix(a11, b11),ans_11,
split_index);
        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++) {
                ans[i][j]= ans_00[i][j];
                ans[i][j + split_index]= ans_01[i][j];
                ans[split_index + i][j]= ans_10[i][j];
                ans[i + split_index][j + split_index]= ans_11[i][j];
            }
        ans_00.clear();
        ans_01.clear();
        ans_10.clear();
        ans_11.clear();
        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
    }
    return ans;
}
int main()
{
    vector<vector<int> > A = {{ 1, 1, 0, 1 },
                              { 2, 5, 2, 6 },
                              { 3, 3, 4, 3 },
```

```
                            { 2, 0, 1, 4 }};
    vector<vector<int> > B = {{ 0, 1, 1, 1 },
                            { 2, 6, 2, 3 },
                            { 3, 2, 3, 1 },
                            { 2, 1, 0, 2 }};
    vector<vector<int> > ans(multiply_matrix(A, B));
    cout<<"Multiplication :"<<endl;
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
        {
            cout<<setw(4)<<ans[i][j];
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

```
PS C:\Users\khush\Desktop\acad
g++ q2.cpp -o q2 } ; if ($?) {
Multiplication :
   4   8   3   6
  28  42  18  31
  24  32  21  22
  11   8   5  11
```

Complexity Analysis:

Time complexity: $O(n^3)$

$T(n) = 8T(n/2) + O(n^2)$

$T(n)= O(n^3)$ …using Master's theorem

Space complexity: $O(n^2)$

C) Strassen's Matrix Multiplication:

Algorithm:

Calculate

P=(A11 + A22)*(B11+B22)

Q=(A21 + A22)* B11

R= A11*(B12 – B22)

S= A22*(B21 – B11)

T=(A11 + A12)*B22

U=(A21 - A11)*(B11 + B12)

V=(A12 – A22)*(B21 + B22)

Then,

C11= P + S – T + V

C12= R + T

C21= Q + S

C22= P + R – Q + U

Code:

```
#include<iostream>
using namespace std;
#include<iomanip>
#include<vector>
vector<vector<int>>add_matrix(vector<vector<int>> A,vector<vector<int>> B, int
split_index,int multiplier = 1)
{
    for (auto i = 0; i < split_index; i++)
        for (auto j = 0; j < split_index; j++)
            A[i][j]= A[i][j]+ (multiplier * B[i][j]);
    return A;
}
vector<vector<int>>multiply_matrix(vector<vector<int>> A,vector<vector<int>> B)
```

```
{
    int col_1 = A[0].size();
    int row_1 = A.size();
    int col_2 = B[0].size();
    int row_2 = B.size();
    if (col_1 != row_2) {
        cout << "\nError: The number of columns in Matrix "
                "A must be equal to the number of rows in "
                "Matrix B\n";
        return {};
    }
    vector<int> result_matrix_row(col_2, 0);
    vector<vector<int> > ans(row_1,result_matrix_row);
    if (col_1 == 1)
        ans[0][0]= A[0][0] * B[0][0];
    else
    {
        int split_index = col_1 / 2;
        vector<int> row_vector(split_index, 0);
        vector<vector<int> > a00(split_index, row_vector);
        vector<vector<int> > a01(split_index, row_vector);
        vector<vector<int> > a10(split_index, row_vector);
        vector<vector<int> > a11(split_index, row_vector);
        vector<vector<int> > b00(split_index, row_vector);
        vector<vector<int> > b01(split_index, row_vector);
        vector<vector<int> > b10(split_index, row_vector);
        vector<vector<int> > b11(split_index, row_vector);
        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++) {
                a00[i][j] = A[i][j];
                a01[i][j] = A[i][j + split_index];
                a10[i][j] = A[split_index + i][j];
                a11[i][j] = A[i + split_index][j + split_index];
                b00[i][j] = B[i][j];
                b01[i][j] = B[i][j + split_index];
                b10[i][j] = B[split_index + i][j];
                b11[i][j] = B[i + split_index][j + split_index];
            }
        vector<vector<int> > p(multiply_matrix(a00, add_matrix(b01, b11,
split_index, -1)));
        vector<vector<int> > q(multiply_matrix(add_matrix(a00, a01, split_index),
b11));
        vector<vector<int> > r(multiply_matrix(add_matrix(a10, a11, split_index),
b00));
```

```
        vector<vector<int> > s(multiply_matrix(a11, add_matrix(b10, b00,
split_index, -1)));
        vector<vector<int> > t(multiply_matrix(add_matrix(a00, a11,
split_index),add_matrix(b00, b11, split_index)));
        vector<vector<int> > u(multiply_matrix(add_matrix(a01, a11, split_index,
-1),add_matrix(b10, b11, split_index)));
        vector<vector<int> > v(multiply_matrix(add_matrix(a00, a10, split_index,
-1),add_matrix(b00, b01, split_index)));
        vector<vector<int> > result_matrix_00(add_matrix(add_matrix(add_matrix(t,
s, split_index), u,split_index),q, split_index, -1));
        vector<vector<int> > result_matrix_01(add_matrix(p, q, split_index));
        vector<vector<int> > result_matrix_10(add_matrix(r, s, split_index));
        vector<vector<int> > result_matrix_11(add_matrix(add_matrix(add_matrix(t,
p, split_index), r,split_index, -1),v, split_index, -1));

        for (auto i = 0; i < split_index; i++)
            for (auto j = 0; j < split_index; j++) {
                ans[i][j]= result_matrix_00[i][j];
                ans[i][j + split_index]= result_matrix_01[i][j];
                ans[split_index + i][j]= result_matrix_10[i][j];
                ans[i + split_index][j + split_index]= result_matrix_11[i][j];
            }
        a00.clear();
        a01.clear();
        a10.clear();
        a11.clear();
        b00.clear();
        b01.clear();
        b10.clear();
        b11.clear();
        p.clear();
        q.clear();
        r.clear();
        s.clear();
        t.clear();
        u.clear();
        v.clear();
        result_matrix_00.clear();
        result_matrix_01.clear();
        result_matrix_10.clear();
        result_matrix_11.clear();
    }
    return ans;
}
```

```cpp
int main()
{
    vector<vector<int> > A = {{ 1, 1, 0, 1 },
                              { 2, 5, 2, 6 },
                              { 3, 4, 4, 3 },
                              { 2, 0, 1, 4 }};
    vector<vector<int> > B = {{ 0, 1, 1, 1 },
                              { 2, 6, 2, 3 },
                              { 3, 2, 3, 1 },
                              { 2, 1, 0, 2 }};
    vector<vector<int> > ans(multiply_matrix(A, B));
    cout<<"Multiplication :"<<endl;
    for(int i=0;i<4;i++)
    {
        for(int j=0;j<4;j++)
        {
            cout<<setw(4)<<ans[i][j];
        }
        cout<<endl;
    }
    return 0;
}
```

Output:

```
PS C:\Users\khush\Desktop
g++ q3.cpp -o q3 } ; if (
Multiplication :
   4   8   3   6
  28  42  18  31
  26  38  23  25
  11   8   5  11
```

Complexity Analysis:

Time complexity: $O(n^{2.81})$

$T(n) = 7T(n/2)+O(n^2)$

$T(n)=O(n^{\log 7})$ …using Master's theorem

Space complexity: $O(n^2)$