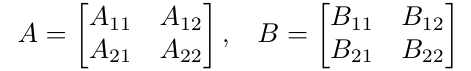
strassen’s matrix multiplication

**Strassen in 1969 which gives an overview that how we can find the multiplication of two 2\*2 dimension matrix by the brute-force algorithm. But by using divide and conquer technique the overall complexity for multiplication two matrices is reduced. This happens by decreasing the total number if multiplication performed at the expenses of a slight increase in the number of addition.**

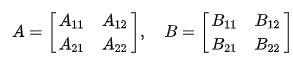
**For multiplying the two 2\*2 dimension matrices Strassen's used some formulas in which there are seven multiplication and eighteen addition, subtraction, and in brute force algorithm, there is eight multiplication and four addition. The utility of Strassen's formula is shown by its asymptotic superiority when order n of matrix reaches infinity. Let us consider two matrices A and B, n\*n dimension, where n is a power of two. It can be observed that we can contain four n/2\*n/2 submatrices from A, B and their product C. C is the resultant matrix of A and B.**

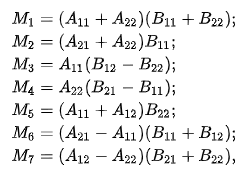
**For example, consider two 4 x 4 matrices A and B that we need to multiply. A 4 x 4 can be divided into four 2 x 2 matrices.**

****

**Here, Aᵢⱼ and Bᵢⱼ are 2 x 2 matrices.**

**Now, we can calculate the product of A and B (matrix C) with the following formulas:**

****

****

****

**Strassen's algorithm for matrix multiplication just gives a marginal improvement over the conventional O(N^3) algorithm. It has higher constant factors and is much harder to implement.**

Applications

* **Although this algorithm seems to be more close to pure mathematics than to computer practically everywhere we use NxN arrays we can benefit from matrix multiplication.**
* **In the other hand the algorithm of Strassen is not much faster than the general n^3 matrix multiplication algorithm. That’s very important because for small n (usually n < 45) the general algorithm is practically a better choice. However as you can see from the chart above for n > 100 the difference can be very big.**
* **In the same time typically NxN arrays are used always when we talk about adjacency matrix of graphs |V| = n and some graph algorithms practically depend on matrix multiplication.**

**Generally, Strassen’s Method is not preferred for practical applications for the following reasons.**

* **The constants used in Strassen’s method are high and for a typical application Naive method works better.**
* **For Sparse matrices, there are better methods especially designed for them.**
* **The submatrices in recursion take extra space.**
* **Because of the limited precision of computer arithmetic on noninteger values, larger errors accumulate in Strassen’s algorithm than in Naive Method**

ADVANTAGES: **Strassen's Matrix Multiplication, SMM, is used to multiply two matrices, and it is better than Native matrix multiplication. due to the fact that SMM's has a complexity of around n^{2.81}n2.81 whereas usual multiplication's complexity is n^{3}n3 .**

**The reason for this is because the number of operations required in SMM is less than in usual multiplication.**

**While usual multiplication requires 8 multiplications and 4 additions SMM requires 7 multiplications and 18 additions.**

DISADVANTAGES:

* **Recursion stack consumes more memory.**
* **The recursive calls add latency.**

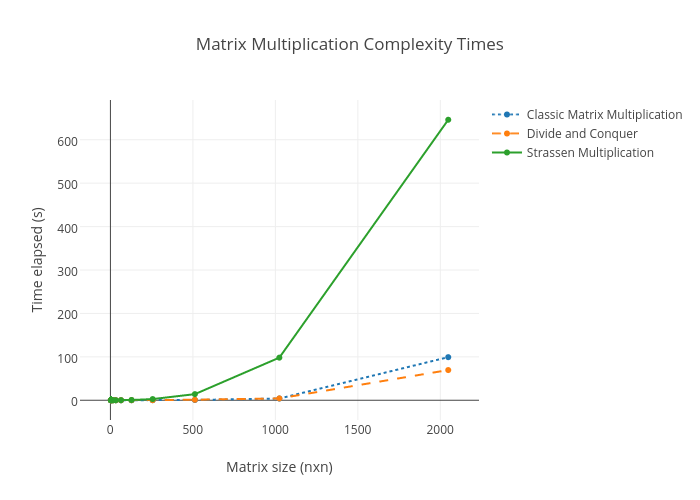
**Due to these reasons, naive algorithm is a better option for smaller inputs which can be determined from the graph too.**

### Pseudocode of Strassen’s multiplication

* **Divide matrix A and matrix B in 4 sub-matrices of size N/2 x N/2 as shown in the above diagram.**
* **Calculate the 7 matrix multiplications recursively.**
* **Compute the submatrices of C.**
* **Combine these submatricies into our new matrix C**

### Complexity

* **Worst case time complexity: Θ(n^2.8074)**
* **Best case time complexity: Θ(1)**
* **Space complexity: Θ(logn)**

****

Algorithm for Strassen’s matrix multiplication

**Algorithm Strassen(n, a, b, d)**

**begin**

**If n = threshold then compute**

**C = a \* b is a conventional matrix.**

**Else**

**Partition a into four sub matrices a11, a12, a21, a22.**

**Partition b into four sub matrices b11, b12, b21, b22.**

**Strassen ( n/2, a11 + a22, b11 + b22, d1)**

**Strassen ( n/2, a21 + a22, b11, d2)**

**Strassen ( n/2, a11, b12 – b22, d3)**

**Strassen ( n/2, a22, b21 – b11, d4)**

**Strassen ( n/2, a11 + a12, b22, d5)**

**Strassen (n/2, a21 – a11, b11 + b22, d6)**

**Strassen (n/2, a12 – a22, b21 + b22, d7)**

**C = d1+d4-d5+d7 d3+d5**

**d2+d4 d1+d3-d2-d6**

**end if**

**return (C)**

**end.**

### Code for Strassen matrix multiplication

**#include <stdio.h>**

**int main()**

**{**

**printf("\*\*\*ATHARVA PURANIK\*\*\*\n");**

**printf("\*\*\*0827CS201049\*\*\*\n");**

**int a[2][2],b[2][2],c[2][2],i,j;**

**int m1,m2,m3,m4,m5,m6,m7;**

**printf("Enter the 4 elements of first matrix: \n");**

**for(i=0;i<2;i++)**

**for(j=0;j<2;j++)**

**scanf("%d",&a[i][j]);**

**printf("Enter the 4 elements of second matrix: \n");**

**for(i=0;i<2;i++)**

**for(j=0;j<2;j++)**

**scanf("%d",&b[i][j]);**

**printf("\nThe first matrix is\n");**

**for(i=0;i<2;i++)**

**{**

**printf("\n");**

**for(j=0;j<2;j++)**

**printf("%d\t",a[i][j]);**

**}**

**printf("\nThe second matrix is\n");**

**for(i=0;i<2;i++)**

**{**

**printf("\n");**

**for(j=0;j<2;j++)**

**printf("%d\t",b[i][j]);**

**}**

**m1= (a[0][0] + a[1][1])\*(b[0][0]+b[1][1]);**

**m2= (a[1][0]+a[1][1])\*b[0][0];**

**m3= a[0][0]\*(b[0][1]-b[1][1]);**

**m4= a[1][1]\*(b[1][0]-b[0][0]);**

**m5= (a[0][0]+a[0][1])\*b[1][1];**

**m6= (a[1][0]-a[0][0])\*(b[0][0]+b[0][1]);**

**m7= (a[0][1]-a[1][1])\*(b[1][0]+b[1][1]);**

**c[0][0]=m1+m4-m5+m7;**

**c[0][1]=m3+m5;**

**c[1][0]=m2+m4;**

**c[1][1]=m1-m2+m3+m6;**

**printf("\nAfter multiplication \n");**

**for(i=0;i<2;i++)**

**{**

**printf("\n");**

**for(j=0;j<2;j++)**

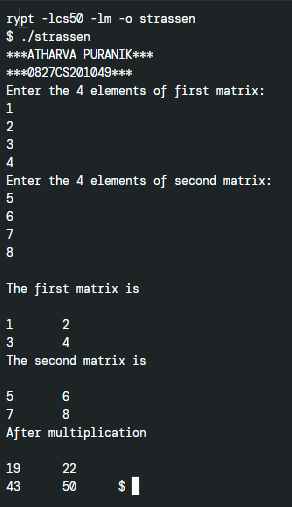
**printf("%d\t",c[i][j]);**

**}**

**return 0;**

**}**

OUTPUT:

****