Q.5. STRASSENS's Matrix multiplication

Practical implementation of Strassens's matrix multiplication algorithm usually switch to the brute force method after matrix sizes become smaller than some crossover point. Run an experiment to determine such crossover point on your computer system.

NORMAL MULTIPLICATION

```
function [ ct ] = multiply( a,b,n )
%UNTITLED4 Summary of this function goes here
%   Detailed explanation goes here
ct=0;
c=zeros(n,n);
for i=1:n
    for j=1:n
       for k=1:n
          c(i,j)=c(i,j)+a(i,k)*b(k,j);
          ct=ct+1;
      end
    end
end
%disp(ct);
end
```

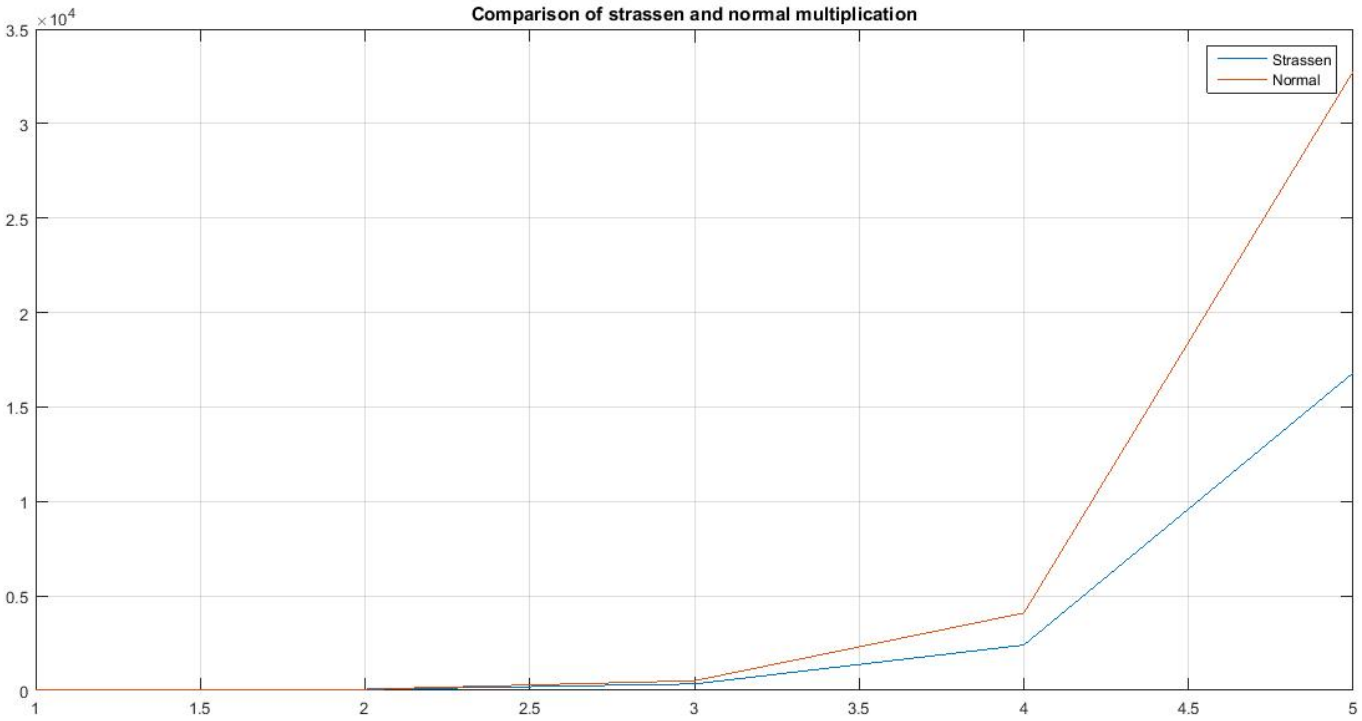
STRASSEN MULTIPLICATION

```
function [ C ct] = strassen( A, B, ct )
    n = size(A, 1);
%disp(n);
    if (n == 1)
        C = A * B;
        ct = ct + 1;

    else
        A11 = A(1: n/2 , 1: n/2);
        A12 = A(1: n/2 , n/2+1: n);
        A21 = A(n/2+1: n , 1: n/2);
        A22 = A(n/2+1: n , n/2+1: n);
        B11 = B(1: n/2 , 1: n/2);
        B12 = B(1: n/2 , n/2+1: n);
        B21 = B(n/2+1: n , 1: n/2);
        B22 = B(n/2+1: n , n/2+1: n);

        [M1 ct] = strassen(A11 + A22, B11 + B22, ct);
        [M2 ct] = strassen(A21 + A22, B11, ct);
        [M3 ct] = strassen(A11, B12 - B22, ct);
        [M4 ct] = strassen(A22 , B21 - B11, ct);
        [M5 ct] = strassen(A11 + A12, B22, ct);
        [M6 ct] = strassen(A21 - A11, B11 + B12, ct);
        [M7 ct] = strassen(A12 - A22, B21 + B22, ct);

        C(1: n/2 , 1: n/2) = M1 + M4 - M5 + M7;
        C(1: n/2 , n/2+1: n) = M3 + M5;
```

```
        C(n/2+1: n , 1: n/2) = M2 + M4;
        C(n/2+1: n , n/2+1: n) = M1 + M3 - M2 + M6;


    end
end
```

MAIN FUNCTION

```
r = [1:1:5];
s = zeros(length(r), 1);
n = zeros(length(r), 1);
for i = 1:length(r)
    x(i)=2^r(i);
        a = randi(100, x(i));
        b = randi(100, x(i));
        [dummy,s(i)] = strassen(a,b, 0);
        n(i) = multiply(a,b,x(i));
end
plot(r, s,r, n);
grid on;
title('Comparison of strassen and normal multiplication');
legend('Strassen', 'Normal');
```



Q.6. QUICK SELECT

Use the **QUICK SELECT** algorithm to find **3rd largest element** in an array of n integers. Analyze the performance of **QUICK SELECT** algorithm for the different instance of size 50 to 500 element. Record your observation with the *number of comparison made* vs. *instance*.

//PARTITION

```
function [ arr, index, n ] = partition_quickSelect( arr, left, right, n)

    pivot = arr(1);
    i = left;
    j = right;

    while(i < j)
        while(pivot >= arr(i) && i < right)
            i = i + 1;
            n = n + 1;
        end

        while(pivot < arr(j) && j > left )
            j = j - 1;
            n = n + 1;
        end

        if(i < j)
            temp = arr(i);
            arr(i) = arr(j);
            arr(j) = temp;
        end
    end

        arr(1) = arr(j);
        arr(j) = pivot;
        index = j;
end
```

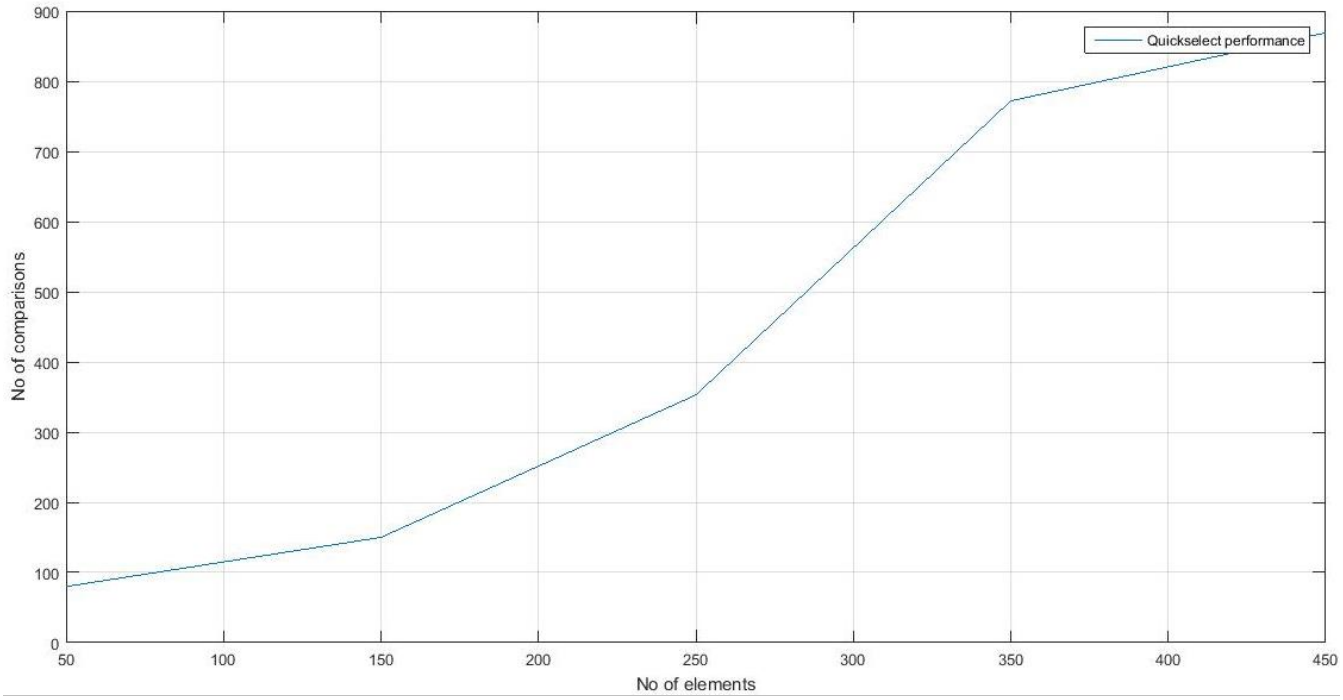
//QUICKSELECT

```
function [ m, n ] = quickSelect(a, k, n)

    l = 1;
    s = length(a);
    [a, pivot, n] = partition_quickSelect(a, l, s, n);
    if(pivot == k)
        m = a(pivot);
    elseif (pivot > k)
        s = pivot-1;
        a = a(l:s);
        [m, n] = quickSelect(a, k, n);
```

```
    else
        l = pivot + 1;
        k = k - pivot;
        a = a(l:s);
        [m, n] =  quickSelect(a, k, n);
    end
end
```

//MAIN FUNCTION

```
x =  [50:25:500];
y = zeros(1,length(x));
fprintf('\nSize of array\t3rd largest element\n');
for i = 1:length(x)
        a = round(rand(x(i))*1000);
        %disp(a);
        [m, y(i)] = quickSelect(a, 3, 0);
        fprintf('\t%d\t\t\t\t%d\n',x(i),m);
end
plot(x,y);
xlabel('No of elements');
ylabel('No of comparisons');
legend('Quickselect performance');
grid on;
```



Size of array	3rd largest element
50	50
75	136
100	7
125	26
150	13
175	11
200	8
225	2
250	9
275	17
300	9
325	6
350	11
375	3
400	10
425	9
450	8
475	4
500	4

Q.7. Matrix Chain Multiplication Comparision

Given a matrix chain $A_1 \ldots A_n$ with the dimension of each of the matrices given by the vector **p** = <12,21,65,18,24,93,121,16,41,31,47,5,47,29,76,18,72,15>. (n=17) Write and run both the dynamic programming and memorized versions of this algorithm to determine the minimum number of multiplications that are needed (use type ***longint***) and the factorization that produces this best case number of multiplications. Run each of the two programs over an appropriately large number of times (put each in a loop to run repeatedly x times) and obtain the times at the beginning and end of the run. Use these times to determine the comparative runtimes of the two algorithms.

//MATRIX CHAIN DYNAMIC PROGRAMMING

```
function [ c ] = MatrixChainOrder_DP( p,n )
%UNTITLED6 Summary of this function goes here
%   Detailed explanation goes here
   m = zeros(n,n);
   for L = 2:n
       for i = 1:n-L+1
```

```
            j = i+L-1;
            m(i,j) = intmax();
            for k = i:j-1
                q = m(i,k) + m(k+1,j) + p(i)*p(k+1)*p(j+1);
                if q < m(i,j)
                    m(i,j) = q;
                end
            end
        end
    end
    c = m(1,n);
    %display(m);

end
```

//MAIN FUNCTION

```
p = [12,21,65,18,24,93,121,16,41,31,47,5,47,29,76,18,72,15];

%count1 = MatrixChainOrder_Memorised(p,2,18);
count2 = MatrixChainOrder_DP(p,17);
%display(count1);
fprintf('No. of operations required are : %d\n',count2);
%display(count2);
```

```
>> Ques_7

m =

  Columns 1 through 10

         0       16380       29106       34290       61074      196110      219342      227214      242466      259950
         0           0       24570       33642       80514      313038      253290      267066      284042      312730
         0           0           0       28080      148986      384300      241392      282450      288206      333152
         0           0           0           0       40176      242730      222672      234480      251936      278162
         0           0           0           0           0      270072      215760      231504      248000      277456
         0           0           0           0           0           0      180048      241056      246512      293632
         0           0           0           0           0           0           0       79376       80352      134640
         0           0           0           0           0           0           0           0       20336       43648
         0           0           0           0           0           0           0           0           0       59737
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
         0           0           0           0           0           0           0           0           0           0
```

```
  Columns 11 through 17

    110120    112940    118675    132515    135875    145595    147575
    108860    113795    118720    134675    135425    147575    146990
    102035    117310    118275    144570    132560    156590    143465
     96185    100415    105610    120860    122480    133820    134090
     94025     99665    104320    120980    120860    133820    132380
     82865    104720    103165    136040    115910    147500    126395
     26600     55035     50960     90415     62165    101315     72230
     16920     20680     26055     40835     43035     53835     54675
     13640     23275     26400     47055     42005     59555     53270
      7285     14570     18595     36900     34750     49600     46165
         0     11045     13630     35695     28905     48075     40080
         0         0      6815     17835     24675     31155     36555
         0         0         0    103588     64206    125118     87387
         0         0         0         0     39672     77256     66942
         0         0         0         0         0     98496     39960
         0         0         0         0         0         0     19440
         0         0         0         0         0         0         0

No. of operations required are : 147575
```

Q.8. BINOMIAL COEFFICIENTS

Computing the **binomial coefficients** $C(n, k)$ defined by the following recursive formula:

$$C(n,k) = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n; \\ C(n-1,k) + C(n-1,k-1), & \text{if } 0 < k < n; \\ 0, & \text{otherwise.} \end{cases}$$

Write the program with three different algorithm to compute **binomial coefficients** $C(n, k)$ and compare them?

//FUNCTION

```
function [ ans ] = Binomial( n,k )
%UNTITLED5 Summary of this function goes here
%   Detailed explanation goes here
    c=zeros(n+1,k+1);
    for i=1:n+1
        for j=1:min(i+1,k+1)
            %co=co+1;
            if j==1 || j==i+1
                c(i,j)=1;
            else
                c(i,j)=c(i-1,j-1)+c(i-1,j);
            end
        end
```

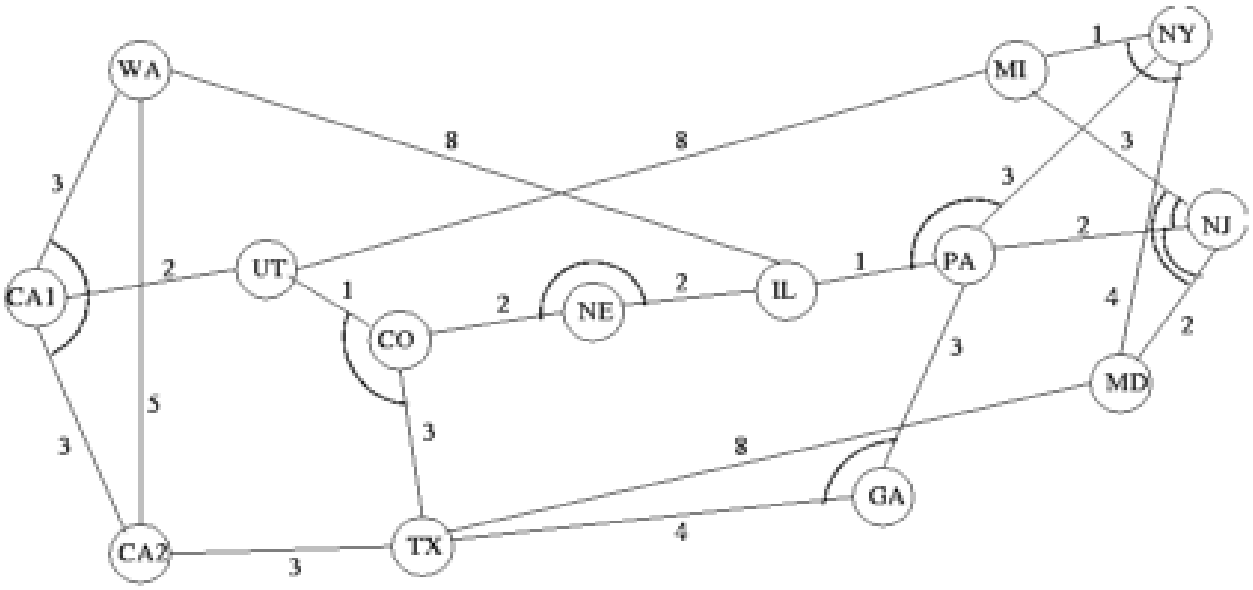
```
    end
    ans=c(n,k+1);
    disp(c);
end
```

//MAIN FUNCTION

```
n=10;
k=5;
disp('Binomial Coefficients Table');
a=Binomial(n,k);
fprintf ('Value of C(%d, %d) is %d \n', n, k, a );
```

```
>> Ques_8
Binomial Coefficients Table
     1     1     0     0     0     0
     1     2     1     0     0     0
     1     3     3     1     0     0
     1     4     6     4     1     0
     1     5    10    10     5     1
     1     6    15    20    15     6
     1     7    21    35    35    21
     1     8    28    56    70    56
     1     9    36    84   126   126
     1    10    45   120   210   252

Value of C(10, 5) is 252
```

Q.9. SPANNING TREE



Write a program obtain minimum cost spanning tree the above NSF network using Prim's algorithm, Kruskal's algorithm and Boruvka's algorithm..

Q.10. 0-1 KNAPSACK PROBLEM

Write a program that computes optimal solution to the 0–1 Knapsack Problem using dynamic programming? You may test your program with the following example:

There are n = 5 objects with integer weights w[1..5] = {1,2,5,6,7}, and values v[1..5] = {1,6,18,22,28}. Assuming a knapsack capacity of 11).

//FUNCTION

```
function [Max ,Seq] = knapsack_algo(Wts, V, Capacity)
    A = zeros(length(Wts)+1,Capacity+1);
    for j = 1:length(Wts)
        for X = 1:Capacity
            if Wts(j) > X
                A(j+1,X+1) = A(j,X+1);
            else
                A(j+1,X+1) = max( A(j,X+1), V(j) + A(j,X-Wts(j)+1));
            end
        end
    end
   Max = A(end,end);

   %Finding the optimal sequence
   Seq = zeros(length(Wts),1);
   a = Max;
   j = length(Wts);
   X = Capacity;
   while a > 0
       while A(j+1,X+1) == a
           j = j - 1;
       end
       j = j + 1;
       Seq(j) = 1;
       X = X - Wts(j);
       j = j - 1;
       a = A(j+1,X+1);
   end
end
```

//MAIN FUNCTION

```
n = 5;
Wts = [1, 2, 5, 6, 7]
V = [1, 6, 18, 22, 28]
Capacity = 11;
```

```
[max, Seq] = knapsack_algo(Wts, V, Capacity);
fprintf ('Max values is %d\n',max);
items = find(Seq);
fprintf('Items are \n');
disp(items);
```

```
>> Ques_10

Wts =

     1     2     5     6     7


V =

     1     6    18    22    28

Max values is 40
Items are
     3
     4
```

Q.11. STRING MATCHING ALGORITHM

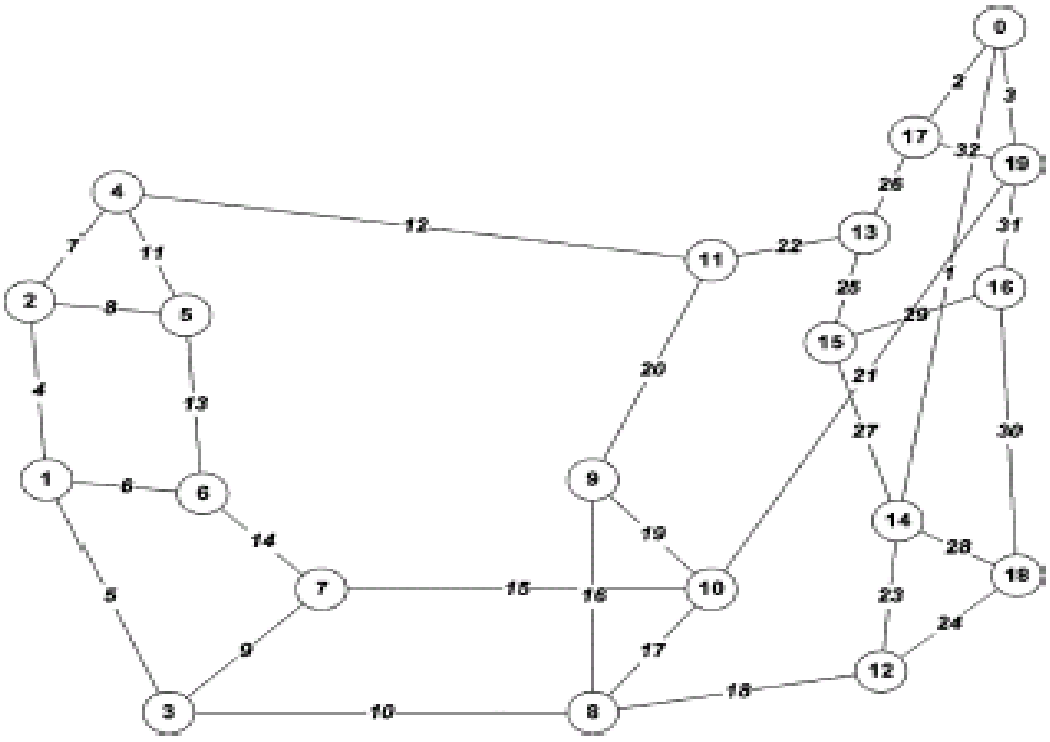
Given two strings P and T over the same alphabet set Σ, determine whether P occurs as a substring in T (or find in which position(s) P occurs as a substring in T). The strings P and T are called pattern and text respectively. Compare the efficiency of three string matching algorithms (Brute-Force Algorithm, Knuth-Morris-Pratt and Boyer-Moore Algorithm) by varying pattern length [1-15] for n=5000.

Q.12. MATRIX CHAIN MULTIPLICATION

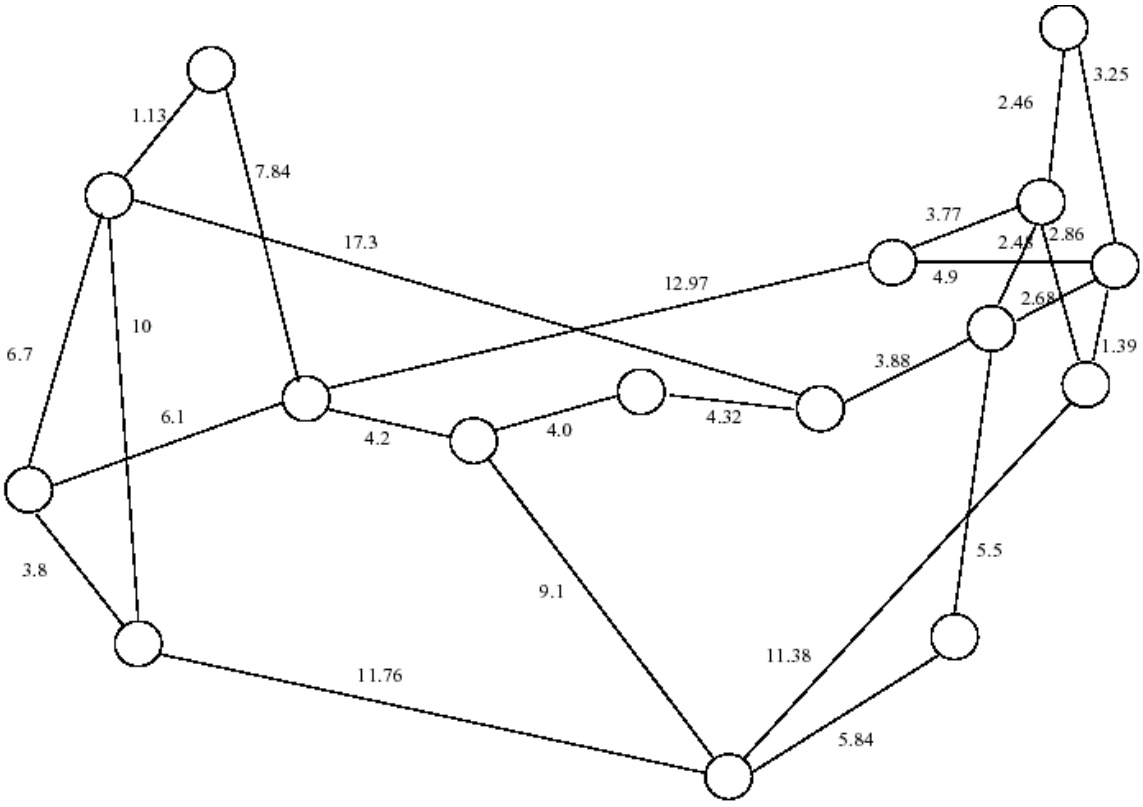
Write a program to compute the best ordering of matrix multiplication. Include the routine to print the actual ordering.

Q.13. PATHING ALGORITHM

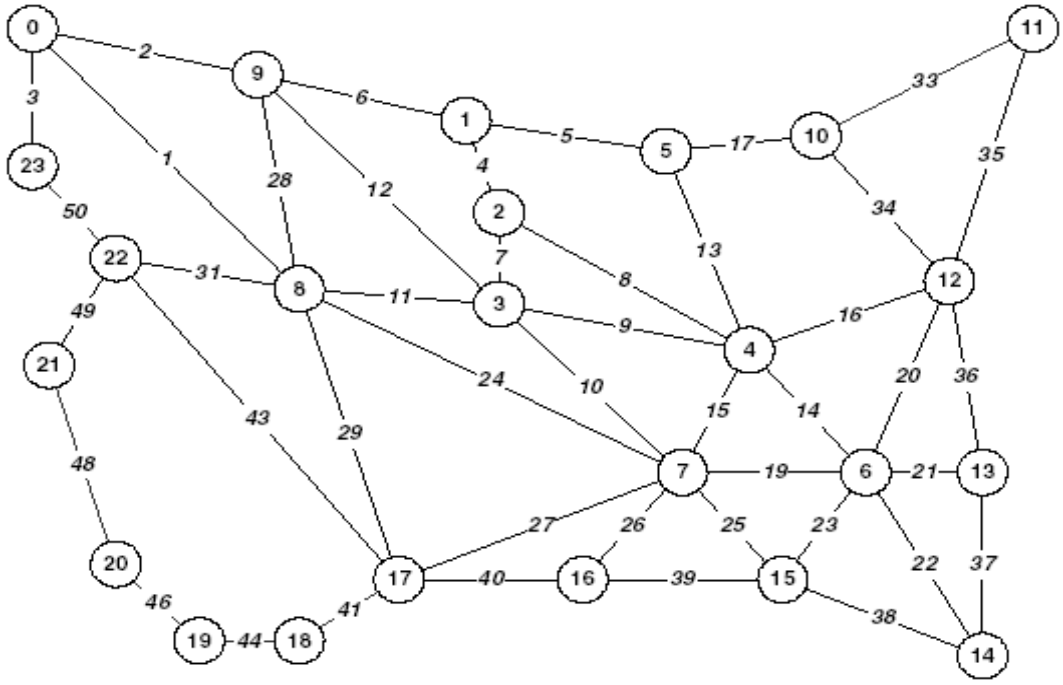
Use Floyd–Warshall algorithm (also known as Floyd's algorithm) to compute all pair shortest path for any one of the following standard network



ARPA Network



NSF Network



NATIONAL NETWORK