Submission for Project-I

Name: Amish Gupta

ID: 2019B5AA1386H

Course Title: Applied Stochastic Processes

Course Code: Math F424

Q7(i) Write a program that calculates different powers of a square matrix of any given order.

Answer: To solve the given problem, we will try various methods and look at their time complexities.

Method 1: Brute Force or the basic approach that we usually do, here we multiply the matrices as many times the power mentions us to do.

The code for this is divided into Four fragments:
1)A function to multiply any given matrices.
2)A function to find inverse of a matrix.
3)A iterative function to find the mth power of the given matrix.
4)Taking the matrix input.

Worst Case Time complexity analysis to perform the operation:
Multiplication of any two given square matrix in brute force manner takes O(n^3), this can be understood as there are three nested loops used in the multiply function. or another way to think about it is that the resultant matrix has n^2 elements and computing each element takes n operations as any (i,j) element in resultant matrix requires i-th row elements to be multiplied by each element of j-th column in a fashion such that:

result[i][j]= A[i][0]* B[0][j]+ A[i][1]* B[1][j] +.....+ A[i][n] * B[n][j]

Note that O(number of operations)=O(number of multiplication + number of addition)= O((n) + (n-1)) = O(n).
This is now done m times to achieve a final time complexity of O(n^3*m).

Now, to find inverse (in order to handle negative powers) I here have used a library, as the brute force time complexity of taking inverse of a matrix is O(n^3) and best being O(n^2) which gets engulfed in O(n^3*m) as O(n^3*m + n^3) =O(n^3 * m). Finding inverse and it's optimisation is whole another set of problems which I think is irrelevant to the discussion here as big-Oh notation engulfs this set of operations no matter what algorithm we use. However, the following links talks in detail about the same:

https://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations#Matrix_algebra

Hence, the time final time-complexity turns out to be O(n^3 * m). Now ofcourse this is assuming the fact that multiplication/addition of any two numbers occur in O(1) time-complexity. This simplication is assummed because large and repeated multiplications are also problems of their own.

In [67]:
```
#Code for Method 1:
def multiply(A,B):
    ans = [[0 for i in range(len(B))] for j in range(len(A[0]))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                ans[i][j] += A[i][k] * B[k][j]
    return ans

def inverse(matrix):
    import numpy as np
    try:
        i=np.linalg.inv(matrix)
        return i
    except:
        print("Inverse of the given matrix not possible")

def exponentOfMatrix(matrix,m):
    if m<0:
        answer=[[0 if i!=j else 1 for i in range(len(matrix))] for j in range(ler
        for i in range(abs(m)):
            answer=multiply(answer,matrix)
        return inverse(matrix)
    else:
        answer=[[0 if i!=j else 1 for i in range(len(matrix))] for j in range(ler
        for i in range(abs(m)):
            answer=multiply(answer,matrix)
        return answer



try:
    n=int(input("Enter the order of matrix: "))
    matrix=([ print("enter " +str(j)+"th row elements (use spaces between two ele
             or [int(i) for i in input().split() ]  for j in range(n)])
    m=int(input("Enter the power "))
    answer=exponentOfMatrix(matrix,m)
    print(answer)
except:
    print("Invalid input given, please try again")
```

```
Enter the order of matrix:  3
enter 0th row elements (use spaces between two elements)
12 7 3
enter 1th row elements (use spaces between two elements)
4 5 6
enter 2th row elements (use spaces between two elements)
7 8 9
Enter the power 3
[[3623, 2906, 2382], [2396, 2043, 1800], [3842, 3258, 2853]]
```

```
Method 2: Recursive approach to above problem.

Reccurence relation used here is:
A^m = A^(m/2) * A^(m/2)

The code for this is  again divided into Four fragments:
1)A function to multiply any given matrices.
2)A function to find inverse of a matrix.
3)A iterative function to find the mth power of the given matrix.
4)Taking the matrix input.

Time complexity= O(n^3 * (number of matrix multiplications)).

O(number of matrix multiplications)= O(1 + 2 + 4 + 8 + .... log(m)) = O(m).

Therefore overall time complexity is O(n^3 * m).

Here, since we make log(m) recursive calls we also make use of auxilliary
space.
```

In [83]:
```
#Code for Method 2:

def multiply(A,B):
    if tuple([A,B]) in hm:
        return hm[tuple([A,B])]
    ans = [[0 for i in range(len(B))] for j in range(len(A[0]))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                ans[i][j] += A[i][k] * B[k][j]
    hm[tuple([A,B])]=ans
    return ans

def inverse(matrix):
    import numpy as np
    try:
        i=np.linalg.inv(matrix)
        return i
    except:
        print("Inverse of the given matrix not possible")


def exponentOfMatrix(matrix,m):

    if m<0:
        m=-1*m
        if abs(m)==1:
            return inverse(matrix)
        if abs(m)%2==0:
            return inverse(multiply(exponentOfMatrix(matrix,m//2),exponentOfMatri
        if abs(m)%2!=0:
            return inverse(multiply(exponentOfMatrix(matrix,(m-1)/2),exponentOfMa
    else:
        if m==0:
            return [[1 for i in range(len(matrix))] for j in range(matrix[0])]
        if m==1:
            return matrix
        if m%2==0:
            return multiply(exponentOfMatrix(matrix,m//2),exponentOfMatrix(matrix
        if m%2!=0:
            return multiply(exponentOfMatrix(matrix,(m-1)/2),exponentOfMatrix(mat



try:
    n=int(input("Enter the order of matrix: "))
    matrix=([ print("enter " +str(j)+"th row elements (use spaces between two ele
             or [int(i) for i in input().split() ]  for j in range(n)])
    m=int(input("Enter the power "))
    answer=exponentOfMatrix(matrix,m)
    print(answer)
except:
    print("Invalid input given, please try again")
```

```
Enter the order of matrix: 3
```

```
enter 0th row elements (use spaces between two elements)
12 7 3
enter 1th row elements (use spaces between two elements)
4 5 6
enter 2th row elements (use spaces between two elements)
7 8 9
Enter the power 3
[[3623, 2906, 2382], [2396, 2043, 1800], [3842, 3258, 2853]]
```

Method 3: With a very small trick applied to method two, we can easily reduce the time complexity to O(n^3* log(m)).

All that needs to be done is to store the matrix multiplication matrices
The code for this is the same as above with an additional dictionary storing the key value pairs where keys are the matrix to be multiplied and values are the answers. We basically eliminate repeated matrix multiplication.
So when recursive calls are made, at each layer of the tree, only one matrix multiplication is made, previous the number of multiplications were 2^x, where x ranged from 1 to log(m)
There's auxilliary space being used up due to the recursive calls and normal space being used up due to the caching. Auxilliary space can be reduced by converting it to an iterative solution

In [113]:
```
#Code for Method 3:

hm={}
def multiply(A,B):
    ans = [[0 for i in range(len(B))] for j in range(len(A[0]))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                ans[i][j] += A[i][k] * B[k][j]
    return ans

def inverse(matrix):
    import numpy as np
    try:
        i=np.linalg.inv(matrix)
        return i
    except:
        print("Inverse of the given matrix not possible")


def exponentOfMatrix(matrix,m):
    if m<0:
        m=-1*m
        if abs(m) in hm:
            return inverse(hm[1*m])
        if abs(m)==1:
            return inverse(matrix)
        if abs(m)%2==0:
            hm[m]=(multiply(exponentOfMatrix(matrix,m//2),exponentOfMatrix(matri
            return inverse(hm[m])
        if abs(m)%2!=0:
            hm[m]=multiply(exponentOfMatrix(matrix,(m-1)/2),exponentOfMatrix(matr
            return inverse(hm[m])
    else:
        if abs(m) in hm:
            return hm[m]
        if m==0:
            return [[1 for i in range(len(matrix))] for j in range(matrix[0])]
        if m==1:
            return matrix
        if m%2==0:
            hm[m]=multiply(exponentOfMatrix(matrix,m//2),exponentOfMatrix(matrix,
            return hm[m]
        if m%2!=0:
            hm[m]=multiply(exponentOfMatrix(matrix,(m-1)/2),exponentOfMatrix(matr
            return hm[m]



try:
    n=int(input("Enter the order of matrix: "))
    matrix=([ print("enter " +str(j)+"th row elements (use spaces between two ele
             or [int(i) for i in input().split() ]  for j in range(n)])
    m=int(input("Enter the power "))
    answer=exponentOfMatrix(matrix,m)
    print(answer)
```

```
except:
    print("Invalid input given, please try again")
```

```
Enter the order of matrix: 3
enter 0th row elements (use spaces between two elements)
12 7 3
enter 1th row elements (use spaces between two elements)
4 5 6
enter 2th row elements (use spaces between two elements)
7 8 9
Enter the power -10
[[ -307.80428919 -4569.22576498  3139.78133209]
 [  687.73328403 10209.11257904 -7015.27627254]
 [ -370.85557447 -5505.19568869  3782.94081799]]
```

In [91]:
```
hm #to access other powers lower than m for the  matrix, we can see the given dic
```

Out[91]:
```
{2.0: [[214, 167, 168], [110, 101, 108], [179, 161, 171]],
 3.0: [[4412, 3677, 3798], [2480, 2139, 2238], [3989, 3426, 3579]],
 5: [[2028480, 1719659, 1787790],
  [1166612, 990517, 1030350],
  [1871147, 1588408, 1652169]],
 10: [[9466123830838, 8031385300343, 8352078126360],
  [5449934439614, 4623914935397, 4808548787580],
  [8740091168099, 7415398362761, 7711496482491]]}
```

```
Method 4:
M = V * D * V^-1
Where V is the eigenvector matrix and D is a diagonal matrix. To raise this to
the Nth power, you get something like:
M^n = (V * D * V^-1) * (V * D * V^-1) * ... * (V * D * V^-1) = V * D^n * V^-1.
Time to find eigenvector is O(n^3)
This reduces the time complexity further down from O(n^3 * log(m)) to O(n^3).


To find eigenvector matrix, inverse and diagnol matrix, I have used libraries.
Method inspired and taken from: https://stackoverflow.com/a/12312009/16179688
```

In [145]:
```
# Code for Method 4:
import numpy as np

def inverse(matrix):
    import numpy as np
    try:
        i=np.linalg.inv(matrix)
        return i
    except:
        print("Inverse of the given matrix not possible")

def multiply(A,B):
    ans = [[0 for i in range(len(B))] for j in range(len(A[0]))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                ans[i][j] += A[i][k] * B[k][j]
    return ans

def eigenvectorMatrix(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    return eigenvectors

def diagnolMatrix(matrix,m):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    for i in range(len(eigenvalues)):
        eigenvalues[i]=eigenvalues[i]**m
    D=[[0 if i!=j else eigenvalues[j] for i in range(len(matrix))] for j in range
    return D

def exponentOfMatrix(matrix,m):
    V=eigenvectorMatrix(matrix)
    D=diagnolMatrix(matrix,m)
    Vinv=inverse(V)
    answer=multiply(V,multiply(D,Vinv))
    return answer

n=int(input("Enter the order of matrix: "))
matrix=([ print("enter " +str(j)+"th row elements (use spaces between two element
         or [int(i) for i in input().split() ]  for j in range(n)])
m=int(input("Enter the power "))
# matrix = np.asarray(matrix, dtype=np.float32)
answer=exponentOfMatrix(matrix,m)
print(answer)
```

```
Enter the order of matrix: 3
enter 0th row elements (use spaces between two elements)
12 7 3
enter 1th row elements (use spaces between two elements)
4 5 6
enter 2th row elements (use spaces between two elements)
7 8 9
Enter the power 3
[[3623.0000000000005, 2905.999999999998, 2381.9999999999986], [2395.99999999999
```

```
86, 2042.999999999997, 1799.999999999998], [3841.9999999999977, 3257.9999999999
96, 2852.9999999999973]]
```