Research Report Summer 2021

Raman Bhandari

August 31, 2021

Contents

Part I

Calculating the permanent of a $N \times N$ matrix

1.1 *Recursive Algorithm*

Let's take a general 3 x 3 matrix :

$$A_{3,3} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

In the basic form, permanent of this matrix can be calculated as :

per(A) = $a_1 \left(\begin{smallmatrix} b_2 & b_3 \\ c_2 & c_3 \end{smallmatrix}\right) a_2 \left(\begin{smallmatrix} b_1 & b_3 \\ c_1 & c_3 \end{smallmatrix}\right) a_3 \left(\begin{smallmatrix} b_1 & b_2 \\ c_1 & c_2 \end{smallmatrix}\right)$

Based on this, an algorithm can be constructed which will break down any N x N matrix into smaller matrices unless a 2 x 2 matrix is formed, for which value is calculated using recursion. The algorithm I constructed for this is given in Appendix section **Brute Force Algorithm to calculate permanent of a matrix**

In this algorithm, a matrix is input as a parameter which is a 2-D vector. The algorithm check the first two edge cases,

- If size of the matrix is 1 x 1, permanent of matrix is the only value itself.
- If size of the matrix is 2 x 2, permanent of matrix is,
  $(A_{1,1}\ A_{2,2})\ (A_{1,2}\ A_{2,1})$
  then, it goes in the third condition where it creates a temporary matrix of size less than 1 for the input matrix. It creates the submatrix and stores it in variable **temp** and calls the function caclPermanent() recursively and adds the value in variable **per**.

  Analyzing the efficiency, this algorithm works well for very small matrices but for bigger matrices it takes forever to find the value of the permanent.

  Alternatively, this algorithm can be run for **double** data type, by changing required **int** to **double** and also changing the function return type to double.

1.2 *Ryser's Formula*

Ryser's Formula is one of the current efficient method available to calculate permanent of a matrix. It has different ways to calculate the end product

with slight variations.
It works by calculating the sum of all the rows and then the product of those sums with extra work of removing unwanted terms.

Let's take a general 3 x 3 matrix :

$$A_{3,3} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

Ryser's Formula : $\sum\limits_{k=0}^{n-1} (-1)^k \sum k$

For the matrix terms : $per(A) = (-1)^n \sum_{S \subseteq 1,..,n} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{ij}$

Explaination :

- Calculate the sum of the rows of the matrix. For the general matrix above, sum of rows will be,
  $(a_1 + a_2 + a_3)$, $(b_1 + b_2 + b_3)$ and $(c_1 + c_2 + c_3)$

- Calculate the product of the sums of rows of the matrix found. For the general matrix above, product of sums of rows will be,
  $(a_1b_1c_1 + a_1b_1c_2 + a_1b_1c_3 + a_1b_2c_1 + a_1b_2c_2 + a_1b_2c_3 + a_1b_3c_1 + a_1b_3c_2 + a_1b_3c_3$ $+ a_2b_1c_1 + a_2b_1c_2 + a_2b_1c_3 + a_2b_2c_1 + a_2b_2c_2 + a_2b_2c_3 + a_2b_3c_1 + a_2b_3c_2 + a_2b_3c_3$ $+ a_3b_1c_1 + a_3b_1c_2 + a_3b_1c_3 + a_3b_2c_1 + a_3b_2c_2 + a_3b_2c_3 + a_3b_3c_1 + a_3b_3c_2 +$ $a_3b_3c_3)$,
  taking in account what is shown by the *recursive algorithm*, we just need 6 of these terms to actually calculate the permanent of the matrix. So we find the possible combinations of the columns by subtracting k columns and adding/subtracting the product of sums from matrix formed by deleting k columns depending on $(-1)^k$ where k is number of columns deleted.

- Calculating possible sub-matrices after deleting $k \in 1,..,n-1$ columns from n x n matrix, in this case we take possible sub-matrices with n x 2 and n x 1. Possible 3 x 2 matrices are $\left(\begin{smallmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} a_1 & a_3 \\ b_1 & b_3 \\ c_1 & c_3 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} a_2 & a_3 \\ b_2 & b_3 \\ c_2 & c_3 \end{smallmatrix}\right)$.
  Possible 1 x 1 matrices are $\left(\begin{smallmatrix} a_1 \\ b_1 \\ c_1 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} a_2 \\ b_2 \\ c_2 \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} a_3 \\ b_3 \\ c_3 \end{smallmatrix}\right)$. Getting their product of sums follows same procedure as done for 3 x 3 matrix.

- Summing up all the parts calculated we get,

  $per(A)$ = $(-1)^0$(product of sums for 3 x 3) $+(-1)^1$(product of sums for 2 x 2) $+(-1)^2$(product of sums for 1 x 1)

  where in $(-1)^k$, k is the number of columns deleted.

- $per(A) = a_1b_2c_3 + a_1b_3c_2 + a_2b_1c_3 + a_2b_3c_1 + a_3b_1c_2 + a_3b_2c_1$

1.3 Running both algorithms for I_{100} and I_{200} with 1's only on main diagonal

Brute Force Algorithm

Starting with smaller matrix,
$I_{(10,10)}$:
Time noted : 4.090000 sec
$I_{(20,20)}$:
Time noted : $> 10minutes$
$I_{(30,30)}$:
Time noted : $> 10minutes$

- Running $perI_{100}$ using brute force algorithm .
  $I_{100,100}$ = **Time noted :** $> 10minutes$

- Similarly running the algorithm for $I_{200,200}$
  Time noted : > 10 minutes

Ryser's Formula

Starting with smaller matrix,
$I_{10,10}$:
Time noted : 0.00000 sec
$I_{20,20}$:
Time noted : 5.82000 sec
$I_{30,30}$:
Time noted : $> 10minutes$

- Running $perI_{100}$ using Ryser's Formula :

  Time noted :$> 10minutes$

- Similarly running Ryser's Formula for I_{200}
  Time noted : $> 10 minutes$

1.4 Running both algorithms for 10 random 50 X 50 matrices with 0/1 entries

(1) First Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(2) Second Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(3) Third Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(4) Fourth Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(5) Fifth Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(6) Sixth Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(7) Seventh Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(8) Eighth Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(9) Ninth Observation :
Brute Force Algorithm, **Time noted :** $>$ 10 minutes
Ryser's Formula, **Time noted :** $>$ 10 minutes

(10) Tenth Observation :
Brute Force Algorithm, **Time noted :** > 10 minutes
Ryser's Formula, **Time noted :** > 10 minutes

1.5 Program to find matrices with largest and smallest permanents among all $0/1$ matrices of size $N \times N$

For a given N, this algorithm finds all matrices possible with 0/1 entries and amongst them finds the matrices with largest and smallest permanents. Program is included in appendix. **O**bservations :

(1) The matrices with largest permanents for a $N \times N$ with 0/1 entries is the matrix with all 1's in it.

(2) The matrices with smallest permanent for a $N \times N$ with 0/1 entries are all the matrices that have permanent value of 0.

1.6 Calculating permanents of some matrices for both algorithms

(a)

$$per \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Brute Force Algorithm
Permanent = 450
Time Taken = 0.000027 sec

Ryser's Formula
Permanent = 450
Time Taken = 0.000046 sec

Comparing time taken, Ryser's formula is a bit slower for really small matrices like given 3 X 3 matrix in comparison with Brute Force Algorithm Program.

(b)

$$per\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Brute Force Algorithm
Permanent = 55456
Time Taken = 0.000095 sec

Ryser's Formula
Permanent = 55456
Time Taken = 0.00006999 sec

Comparing time taken, Ryser's formula is faster than Brute Force Algorithm Program for this 4 X 4 matrix.

(c)

$$per\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Brute Force Algorithm
Permanent = 2
Time Taken = 0.000357 sec

Ryser's Formula
Permanent = 2
Time Taken = 0.000115 sec

Comparing time taken, Ryser's formula is faster than Brute Force Algorithm Program for this 5 X 5 matrix.

1.7 Program to calculate subsets of size k of a given set of size N

Program is included in Appendix

(a) Test 1, $N = 5$ $k = 3$
Program output is as follows :

```
1) N = 5, k = 3
2) Number of subsets = 10
3) Subsets are as follows :
[3 4 5 ] [2 4 5 ] [2 3 5 ] [2 3 4 ] [1 4 5 ] [1 3 5 ] [1 3 4 ] [1 2 5 ]
[1 2 4 ] [1 2 3 ]
```

(b) Test 2, $N = 6$ $k = 3$
Program output is as follows :

```
1) N = 6, k = 3
2) Number of subsets = 20
3) Subsets are as follows :
[4 5 6 ] [3 5 6 ] [3 4 6 ] [3 4 5 ] [2 5 6 ] [2 4 6 ] [2 4 5 ] [2 3 6 ]
[2 3 5 ] [2 3 4 ] [1 5 6 ] [1 4 6 ] [1 4 5 ] [1 3 6 ] [1 3 5 ] [1 3 4 ]
[1 2 6 ] [1 2 5 ] [1 2 4 ] [1 2 3 ]
```

(c) Test 3, $N = 7$ $k = 3$
Program output is as follows :

```
1) N = 7, k = 3
2) Number of subsets = 35
3) Subsets are as follows :
[5 6 7 ] [4 6 7 ] [4 5 7 ] [4 5 6 ] [3 6 7 ] [3 5 7 ] [3 5 6 ] [3 4 7 ]
[3 4 6 ] [3 4 5 ] [2 6 7 ] [2 5 7 ] [2 5 6 ] [2 4 7 ] [2 4 6 ] [2 4 5 ]
[2 3 7 ] [2 3 6 ] [2 3 5 ] [2 3 4 ] [1 6 7 ] [1 5 7 ] [1 5 6 ] [1 4 7 ]
[1 4 6 ] [1 4 5 ] [1 3 7 ] [1 3 6 ] [1 3 5 ] [1 3 4 ] [1 2 7 ] [1 2 6 ]
[1 2 5 ] [1 2 4 ] [1 2 3 ]
```

(d) Test 4, $N = 8$ $k = 3$
Program output is as follows :

```
1) N = 8, k = 3
2) Number of subsets = 56
3) Subsets are as follows :
[6 7 8 ] [5 7 8 ] [5 6 8 ] [5 6 7 ] [4 7 8 ] [4 6 8 ] [4 6 7 ] [4 5 8 ]
[4 5 7 ] [4 5 6 ] [3 7 8 ] [3 6 8 ] [3 6 7 ] [3 5 8 ] [3 5 7 ] [3 5 6 ]
[3 4 8 ] [3 4 7 ] [3 4 6 ] [3 4 5 ] [2 7 8 ] [2 6 8 ] [2 6 7 ] [2 5 8 ]
[2 5 7 ] [2 5 6 ] [2 4 8 ] [2 4 7 ] [2 4 6 ] [2 4 5 ] [2 3 8 ] [2 3 7 ]
[2 3 6 ] [2 3 5 ] [2 3 4 ] [1 7 8 ] [1 6 8 ] [1 6 7 ] [1 5 8 ] [1 5 7 ]
[1 5 6 ] [1 4 8 ] [1 4 7 ] [1 4 6 ] [1 4 5 ] [1 3 8 ] [1 3 7 ] [1 3 6 ]
[1 3 5 ] [1 3 4 ] [1 2 8 ] [1 2 7 ] [1 2 6 ] [1 2 5 ] [1 2 4 ] [1 2 3 ]
```

1.8 Program to calculate sub-matrices of size k x k of a given matrix of size N x N

Program is included in Appendix

(a) Case 1 where $N = 4$ $k = 2$
Result of the program is as follows:

```
1) N = 4, k = 2
Matrix 4 x 4
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16
2) Number of sub-matrices of size 2 x 2 = 36
3) Sub-matrices are as follows :
11 12
15 16

10 12
14 16

10 11
14 15

 9 12
13 16

 9 11
13 15

 9 10
13 14

 7  8
15 16

 6  8
14 16

 6  7
```

```
14 15

 5  8
13 16

 5  7
13 15

 5  6
13 14

 7  8
11 12

 6  8
10 12

 6  7
10 11

 5  8
 9 12

 5  7
 9 11

 5  6
 9 10

 3  4
15 16

 2  4
14 16

 2  3
14 15

 1  4
13 16
```

```
 1  3
13 15

 1  2
13 14

 3  4
11 12

 2  4
10 12

 2  3
10 11

 1  4
 9 12

 1  3
 9 11

 1  2
 9 10

 3  4
 7  8

 2  4
 6  8

 2  3
 6  7

 1  4
 5  8

 1  3
 5  7

 1  2
 5  6
```

(b) Case 2 where $N = 5$ $k = 3$
Result of the program is as follows:

```
1) N = 5, k = 3
Matrix 5 x 5
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
2) Number of sub-matrices of size 3 x 3 = 100
```

(c) Case 3 where $N = 6$ $k = 3$
Result of the program is as follows:

```
1) N = 6, k = 3
Matrix 6 x 6
  1   2  3   4  5  6
  7   8   9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
2) Number of sub-matrices of size 3 x 3 = 400
```

(d) Case 4 where $N = 7$ $k = 3$
Result of the program is as follows:

```
1) N = 7, k = 3
Matrix 7 x 7
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49
2) Number of sub-matrices of size 3 x 3 = 1225
```

(e) Case 5 where $N = 8$ $k = 3$
Result of the program is as follows:

```
1) N = 8, k = 3
Matrix 8 x 8
 1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
2) Number of sub-matrices of size 3 x 3 = 3136
```

1.9 Program to calculate matrices among all $N \times N$ matrices from the class $\Lambda_1 \cup \Lambda_2 \cup \cdots \cup \Lambda_N$ finds all matrices with the minimum and maximum permanent

Program included in Appendix

(a) For $N = 2$, Time taken : 0.001123 sec

(b) For $N = 3$, Time taken : 0.001707 sec

(c) For $N = 4$, Time taken : 0.078591 sec

(d) For $N = 5$, Time taken : 19.08000 sec

(e) For $N = 6$, Time taken : >2 hours

Observations :

(1) The matrices with largest permanents for a $N \times N$ belong to class Λ_1

(2) The matrices with smallest permanent for a $N \times N$ belong to class Λ_N.

About the program :
The program has to calculate smallest and largest permanents for a fixed N from all matrices that lie in Λ_1 for a specific k. To find all the possible matrices that belong to a particular class, we use the *nextpermutation(array,*

array+number of elements) package in C++. We start by creating an array of size NxN elements. We set all the elements to 0 using *memset* package in C++. We add k number of 1's at the last k indexes of the array using a basic for loop that starts from $k*N$to $N*N$.
nextpermutation(array, array+number of elements) package works with a do-while loop. It starts with the last element in the array and then it keeps on finding different permutations that can be formed, when it forms the most superior permutations of the array, this means it went from bottom to up then the condition in while loop evaluates to false meaning all possible permutations of the array has been formed.
For ease to calculate permanents for each of the matrix formed, we make a temporary vector matrix and fill it using the array formed by *nextpermutation(array, array+number of elements)* package.

1.10 Given an $N \times N$ matrix A, write a program to calculate $\sigma_{N-1}(A)$ which is the sum of all subpermanents of order $N-1$

Program included in Appendix. This program finds the sum of sub-permanents of order N-1 that are fromed from Matrix of size NxN. There are N^2 possible matrices of order $N-1$.

Following are some results from randomly generated matrices with 0/1 values.

(a) For $N = 3$, Time taken : 0.000538 sec. Program output:

```
1) Matrix 3 x 3
 0  1  0
 1  1  0
 0  0  1
2) Number of sub-matrices of order 2 = 9
3) Sub-matrices are as follows :
 1  0
 0  1

 1  0
 0  1
 1  1
 0  0

 1  0
 0  1

 0  0
 0  1

 0  1
 0  0

 1  0
 1  0

 0  0
 1  0

 0  1
 1  1

4) Sum of sub-permanents of all the matrix of order N-1 = 6
```

(b) For $N = 4$, Time taken : 0.001508 sec. Program output:

```
1) Matrix 4 x 4
 0  0  0  0
 1  1  0  0
 0  0  1  0
 1  1  1  0
2) Number of sub-matrices of order 3 = 16
3) Sub-matrices are as follows :
 1  0  0
 0  1  0
 1  1  0

 1  0  0
 0  1  0
 1  1  0
```

```
1  1  0
0  0  0
1  1  0

1  1  0
0  0  1
1  1  1

0  0  0
0  1  0
1  1  0

0  0  0
0  1  0
1  1  0

0  0  0
0  0  0
1  1  0

0  0  0
0  0  1
1  1  1

0  0  0
1  0  0
1  1  0

0  0  0
1  0  0
1  1  0

0  0  0
1  1  0
1  1  0

0  0  0
1  1  0
1  1  1

0  0  0
```

```
 1  0  0
 0  1  0

 0  0  0
 1  0  0
 0  1  0

 0  0  0
 1  1  0
 0  0  0

 0  0  0
 1  1  0
 0  0  1

4) Sum of sub-permanents of all the matrix of order N-1 = 10
```

(c) For $N = 5$, Time taken : 0.002650 sec. Program output:

```
1) Matrix 5 x 5
 1  0  0  0  0
 1  1  0  1  0
 1  1  0  0  0
 1  0  1  1  0
 0  0  1  1  1
2) Number of sub-matrices of order 4 = 25
4) Sum of sub-permanents of all the matrix of order N-1 = 281
```

(d) For $N = 10$, Time taken : 0.091463 sec. Program output:

```
1) Matrix 10 x 10
 1  0  0  1  0  0  0  0  0  0
 1  1  0  0  1  0  1  0  1  0
 0  1  1  1  0  1  1  0  0  0
 0  0  0  1  0  0  1  0  0  0
 1  1  1  1  0  0  1  0  0  0
 0  0  1  1  0  1  1  0  1  0
 0  0  1  1  0  0  0  1  0  1
 1  1  0  1  0  0  1  1  0  1
 1  0  1  0  0  0  0  1  0  0
 0  1  0  1  1  1  0  0  0  0
2) Number of sub-matrices of order 9 = 100
3) Sum of sub-permanents of all the matrix of order N-1 = 8242700
```

(e) For $N = 15$, Time taken : 5.742147 sec. Program output:

```
1) Matrix 15 x 15
 1  1  1  0  1  0  1  1  0  1  0  0  1  0  0
 0  1  1  0  0  1  0  1  1  0  1  1  0  1  1
 1  1  0  1  1  1  0  1  1  1  0  0  1  0  0
 1  0  1  0  1  0  1  1  1  1  0  0  1  1  0
 0  1  0  1  0  1  1  0  1  1  0  0  1  0  0
 0  1  0  1  1  1  1  0  0  1  1  1  1  0  1
 0  0  0  1  0  1  0  0  0  0  0  1  1  0  0
 1  0  1  0  0  1  0  0  1  0  0  0  0  1  0
 1  1  1  1  1  0  0  0  0  0  1  0  1  0  1
 1  1  0  1  1  1  1  0  0  1  1  1  1  0  0
 0  0  0  1  1  1  1  1  0  1  1  0  0  0  0
 1  1  1  0  0  0  1  1  0  1  1  1  1  1  0
 1  1  0  0  0  0  0  0  0  1  0  0  0  0  1
 0  1  1  1  1  0  1  0  1  1  1  1  1  0  0
 1  0  0  0  0  1  0  0  0  0  1  0  0  1  1
2) Number of sub-matrices of order 14 = 225
4) Sum of sub-permanents of all the matrix of order N-1 = 111870621122304
```

1.11 Tasks (May 3-August 20, 2021 = 16 weeks)

Task 1: **Deadline: May 10.**
Given an $N \times N$ matrix with real entries, write a program to calculate the exact value of the permanent of this matrix, i.e., if the entries are integers, the answer should be the exact integer. You may want to think about the interface as well, i.e., suppose I will want to calculate the permanent of a 25×25 matrix with 0/1 entries and 1's only in very specific positions. What would be an efficient way to specify this matrix and input it into your program for calculations.

Task 2: **Deadline: May 20.**
Write a program to use Ryser's formula to calculate the permanent of an $N \times N$ matrix for any integer $N \geq 1$.

- ☐ Let I_n be an $n \times n$ matrix with 1's on its main diagonal and 0 elsewhere. Run your algorithms (brute force and using Ryser's formula) to calculate $perI_{100}$, $perI_{200}$, etc. The answer should clearly be 1, but the execution times will be different. Record execution times for both of your algorithms. Stop when it takes longer than 10 minutes for your program to finish calculations.
- ☐ Generate a random 50×50 matrix with 0/1 entries and calculate its permanent using both algorithms, record execution times. Compare the results (the permanents should be the same). Do this for 10 matrices.

Task 3: **Deadline: May 24.**
Write a program that, among all 0/1 matrices of size $N \times N$ finds the ones with the largest and the smallest permanents by (i) searching over all possible matrices, and then (ii) by reducing the search space as much as possible in order to decrease the computation time (since, for example, there are exactly 2^{100} 10×10 matrices with entries 0 and 1). Come up with ideas about how the search space can be reduced.

Task 4: **Deadline: June 3.**
Let's check that whatever you are calculating is correct. So, do the following:

- Calculate the following (exact answers, different algorithms, compare execution times):

$$per\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}, \quad per\begin{bmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\\13&14&15&16\end{bmatrix}, \quad per\begin{bmatrix}1&1&0&0&0\\0&1&1&0&0\\0&0&1&1&0\\0&0&0&1&1\\1&0&0&0&1\end{bmatrix}$$

- Write a program that finds all subsets of k elements of the set $\{1, 2, 3, \ldots, N\}$, where $1 \leq k \leq N$. The output of this program should be: 1) Values of k and N, 2) the number of all such subsets, and 3) the list of all these subsets.
  Run this program and produce output for: i) $N = 5$, $k = 3$; ii) $N = 6$, $k = 3$; iii) $N = 7$, $k = 3$; iv) $N = 8$, $k = 3$. Make sure that your output is in the readable form.
- Write an algorithm that finds all $k \times k$ submatrices of an $N \times N$ matrix A, where $1 \leq k \leq N$. The output of this program should

be: 1) Values of k and N, and matrix A, 2) the number of all such submatrices, and 3) the list of all these submatrices.
Run this program and produce output for: $N = 4$, $k = 2$, $A = (a_{ij})_{1 \leq i,j \leq N}$.
Suppressing listing of all matrices (i.e., without outputting 3) and only outputting 1) and 2)), run this program for: i) $N = 5$, $k = 3$; ii) $N = 6$, $k = 3$; iii) $N = 7$, $k = 3$; iv) $N = 8$, $k = 3$.

Task 5: **Deadline: June 17.**
From now on, we will be working with things related to doubly stochastic (DS) matrices (i.e., nonnegative matrices such that the sums of the elements in each row and each column are all equal to 1).

Before we go to general DS matrices, we will consider a particular subclass Λ_k that we will generate as follows: given an $N \times N$ matrix, let k be such that $1 \leq k \leq N$. Now, we say that a matrix A is in the class Λ_k if kA is a 0/1 matrix with exactly k 1's in each row and each column. If Ω_N denotes the space of all DS matrices of size $N \times N$, then it is clear that $\Lambda_k \subset \Omega_N$. Note also that, if A is an $N \times N$ matrix, then $per(\alpha A) = \alpha^N per(A)$.

Task: write a program that, among all $N \times N$ matrices from the class $\Lambda_1 \cup \Lambda_2 \cup \cdots \cup \Lambda_N$ finds all matrices with the minimum and maximum permanent. Run this program and output the results and the execution time for $N = 2, 3, 4, 5, \ldots$ (until your program runs for longer than, say, 2 hours).

Task 6: **Deadline: July 7.**
Given an $N \times N$ matrix A, write a program to calculate $\sigma_{N-1}(A)$ which is the sum of all subpermanents of order $N-1$. In other words, $\sigma_{N-1}(A)$ is the sum of permanents of all matrices that you get by deleting 1 row and 1 column in A (there are N^2 matrices like that).

Task 7: **Deadline: July 14.**
Let N and k be natural numbers with $1 \leq k \leq N$. Write a program that, among all $N \times N$ matrices A from the class Λ_k finds all matrices that minimize and maximize the following quantity:

$$per(A) - \frac{1}{N^2}\sigma_{N-1}(A).$$

Then calculate the maximum and minimum of this quantity over all $1 \leq k \leq N$ for each N. Run this program for $N = 2, 3, 4, 5, 6$ and output all your results in an easy to read form.

(Recall that while working with a matrix from Λ_k, you can assume that all of your $1/k$'s in the first row are in the first k positions, i.e., if $A = (a_{ij})$, then $a_{11} = a_{12} = \cdots = a_{1k} = 1/k$ and $a_{1,k+1} = \cdots = a_{1N} = 0$. You can also assume that $a_{21} = 1/k$.)

Task 8: **Deadline: July 21.**
Recall that a matrix is called doubly stochastic (DS) if all of its elements are nonnegative and the sum of all elements in each row and each column is equal to 1. We will denote the set of all DS matrices of size $N \times N$ by Ω_N. Given a natural number N, write a program to generate **random** matrices from Ω_N.
Hint: Birkhoff–von Neumann theorem, see e.g.

`https://en.wikipedia.org/wiki/Doubly_stochastic_matrix`

Describe your algorithm.

Task 9: **Deadline: July 28.**
For each $N = 2, 3, 4, 5, 6, 7\ldots$ randomly generating matrices from Ω_N, try to minimize

$$per(A) - \frac{1}{N^2}\sigma_{N-1}(A).$$

Output N, your 'best' matrices minimizing this quantity, the computed value, the number of matrices considered, and the execution time.

1.12 Appendix

- **Brute Force algorithm to calculate permanent of a matrix**

```
int Project1::recPermanent(std::vector<std::vector<int>> matrix){
   int per = 0;

   if(matrix.size() == 1){
       per = matrix[0][0];
   }
   else if(matrix.size()==2){
       per = matrix[0][0]*matrix[1][1] + matrix[0][1]*matrix[1][0];
   }
   else{
       std::vector<std::vector<int>> temp;
```

```
        temp.resize(matrix.size()-1, std::vector<int>(matrix.size()-1, 0));

        for(int i=0;i<matrix.size();i++){
            int row = 0;
            //we start from second row to calculate permanent
            for(int j=1;j<matrix.size();j++){
                int col = 0;
                for(int k=0;k<matrix.size();k++){
                    if(k != i){
                        temp[row][col] = matrix[j][k];
                        col++;
                    }
                }
                row++;
            }
            per += matrix[0][i] * recPermanent(temp);
        }
    }
    return per;
}
```

- **Ryser's Formula to calculate permanent of a matrix**

```
int* Project1::store(unsigned long long int index, int size){
    int* storeBin = (int*)calloc(size+1, sizeof(int));
    int end = size-1;

    while(index>0){
        storeBin[end] = index%2;
        storeBin[size] += storeBin[end];
        index = index / 2;
        end--;
    }
    return storeBin;
}

int Project1::ryserPermanent(std::vector<std::vector<int>> matrix){
    int per = 0;

    int size = (int)matrix.size();
    cout<<"Check the size of the matrix is correct : "<<size<<endl;
    unsigned long long int subMatrices = (long)pow((int)2, size);
    cout<<"Check the possible combinations of the matrix is correct : "<<su

    int* arr = new int[size+1];
    int product = 1;
    int sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (int)pow((int)-1, size-arr[size]) * product;
    }
    cout<<"Permanent of the matrix is : "<< per <<endl;
    return per;
}
}

void Project1::combinationsR(std::vector<std::vector<int>> store[],
std::vector<std::vector<int>> matrix, std::vector<std::vector<int>> temp,
int start, int end, int cols, int index){

    if(index == cols){
        store[cols-1] = temp;
    }

    for(int i = start; i<=end && end-i+1 >= cols-index;i++){

    }
}
```

- **Program to calculate matrices with largest and minimum permanents among all N x N matrices with 0/1 values**

```
void Project1::smallLarge(int size, int n){
    vector<std::vector<std::vector<int>>> storeMatrix;
    vector<int> storePer;

    int smallestPer = -1;
    int largestPer = -1;
    bool check = true;
    //how many 1's in the matrix?
    for(int i = 1; i <= n; i++){
        int temp[n];
        //creating a size 'n' array for size*size elements
        for(int k = 0;k<n;k++){
            temp[k] = 0;
        }
        //insert 'i' number of 1's in the array
        for(int j = 0;j<i;j++){
            temp[j] = 1;
        }
        sort(temp, temp+n);
        std::vector<std::vector<int>> matrix;
        do{
            matrix.resize(size, std::vector<int>(size, 0));
            int count = 0;
            for(int p =0;p<size;p++){
                for(int q =0;q<size;q++){
                    matrix[p][q] = temp[count];
                    count++;
                }
            }
            storeMatrix.push_back(matrix);

            /*
             For this particular example, to reduce search space
             whenever we have a matrix with just 1 we will not calculate
             the permanent and simply add '0' as the value as N >= 2 always
             */
            int per;
            if(i!=1){
                per = ryserPermanentSmallLarge(matrix);
            }
            else{
                per = 0;
            }
            storePer.push_back(per);

            if(check){
                largestPer = per;
                smallestPer = per;
                check = false;
            }
            if(per>=largestPer){
                largestPer = per;
            }
            if(per<=smallestPer){
                smallestPer = per;
            }
            matrix.clear();
        }while(next_permutation(temp, temp+n));
    }
    cout<<"Smallest permanent found = "<<smallestPer<<endl;
    cout<<"Matrices with smallest permanents are as follows : "<<endl;
    for(int i=0;i<storePer.size();i++){
        if(storePer[i] == smallestPer){
            cout<<"Matrix : "<<endl;
            printM(storeMatrix[i]);
        }
    }
    cout<<"\nLargestpermanent found = "<<largestPer<<endl;
    cout<<"Matrices with largest permanents are as follows : "<<endl;
    for(int i=0;i<storePer.size();i++){
        if(storePer[i] == largestPer){
            cout<<"Matrix : "<<endl;
            printM(storeMatrix[i]);
        }
    }
    cout<<"Total matrices found "<<storeMatrix.size()<<endl;
}

int Project1::ryserPermanentSmallLarge
(std::vector<std::vector<int>> matrix){
    int per = 0;

    int size = (int)matrix.size();

    unsigned long long int subMatrices = (long)pow((int)2, size);

    int* arr = new int[size+1];
    int product = 1;
    int sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (int)pow((int)-1, size-arr[size]) * product;
    }
    return per;
}
}
```

- **Program to calculate subsets of size k of a given set of size N**

```
void Project1::subsetsProgram(int set[], int n, int k){
    cout<<"1) N = "<< n<<", k = "<< k<<endl;

    int count = 0;
    int subsets = (int)pow(2, n);

    vector<int*> answer;
    for(int i = 1; i<subsets;i++){
        int* value =(int*)calloc(n, sizeof(int));
        int end = n-1;
        int temp = i;
        int check = 0;
        //converting to binary
        while(temp > 0){
            int r = temp%2;
            if(r == 1){
                check++;
            }
            value[end] = r;
            temp = temp/2;
            end--;
        }
        if(end>-1){
            while(end>-1){
                value[end] = 0;
                end--;
            }
        }
        if(check == k){
            count++;
            answer.push_back(value);

        }
    }
    cout<<"2) Number of subsets = "<<count<<endl;
    cout<<"3) Subsets are as follows : "<<endl;
    for(int p = 0; p<answer.size();p++){
        int* c = answer[p];
        cout<<"\t";
        for(int j = 0;j<n;j++){
            if(c[j] == 1){
                cout<<set[j]<<" ";
            }
        }
        cout<<endl;
    }
}
```

- **Program to calculate sub-matrices of size k x k of a given matrix of size N x N**

```
void Project1::submatricesProgram3(std::vector<std::vector<int>> matrix, in
    cout<<"1) N = "<< n<<", k = "<< k<<endl;
    cout<<"Matrix "<<n<<" x "<< n <<endl;
    print();

    int set[n];
    for(int i =0; i<n;i++){
        set[i] = i;
    }
    vector<int*> subsets = subMatrices3Help(set, n, k);

    vector<std::vector<std::vector<int>>> storeBin;

    for(int i = 0;i<subsets.size();i++){
        int* r = subsets[i];
        for(int j =0;j<subsets.size();j++){
            int* c = subsets[j];
            std::vector<std::vector<int>> temp;
            temp.resize(k, std::vector<int>(k, 0));

            for(int t = 0;t<k;t++){
                for(int p =0;p<k;p++){
                    temp[t][p] = matrix[r[t]][c[p]];
                }
            }
            storeBin.push_back(temp);
            temp.clear();
        }
    }

    cout<<"2) Number of sub-matrices of size "<<k<<" x "
             <<k<<" = "<<(int)storeBin.size()<<endl;
    cout<<"3) Sub-matrices are as follows : "<<endl;
    for(int i = 0;i<storeBin.size();i++){
        printM(storeBin[i]);
    }
}

vector<int*> Project1::subMatrices3Help(int set[], int n, int k){

    vector<int*> ans;
```

```
int count = 0;
int subsets = (int)pow(2, n);

vector<int*> answer;
for(int i = 1; i<subsets;i++){
    int* value =(int*)calloc(n, sizeof(int));
    int end = n-1;
    int temp = i;
    int check = 0;
    //converting to binary
    while(temp > 0){
        int r = temp%2;
        if(r == 1){
            check++;
        }
        value[end] = r;
        temp = temp/2;
        end--;
    }
    if(end>-1){
        while(end>-1){
            value[end] = 0;
            end--;
        }
    }
    if(check == k){
        count++;
        answer.push_back(value);
    }
}

for(int p = 0; p<answer.size();p++){
    int* c = answer[p];
    int* toReturn = (int*)calloc(k, sizeof(int));;
    int index = 0;
    for(int j = 0;j<n;j++){
        if(c[j] == 1){
            toReturn[index] = set[j];
            index++;
        }
        }
        ans.push_back(toReturn);
    }
    return ans;
}
```

- **Program to calculate matrices among all $N \times N$ matrices from the class $\Lambda_1 \cup \Lambda_2 \cup \cdots \cup \Lambda_N$ finds all matrices with the minimum and maximum permanent**

```
void Project1::subclassMaxMin(int N, int k){
    /*
     INPUT Variables :
     1. N - Size of the Matrix
     2. k - To determine a matrix in class \Lambda_k
     */
    //Total elements in N x N matrix are N*N
    int elements = N*N;
    vector<std::vector<std::vector<long double>>> storeMatrix;
    vector<long double> storePer;

    int temp[elements];
    //initialize the array with 0's
    memset(temp, 0, sizeof(temp));
    /*
     add 1's to tha last k*N indexes of the array
     to use next_Permutation()
     */
    for(int i = k*N;i<elements;i++){
        temp[i] = 1;
    }
    //make a temp matrix to push in vector
    std::vector<std::vector<long double>> matrix;
    do{
        matrix.resize(N, std::vector<long double>(N, 0));
        //coverting array to vector matrix
        int count = 0;
        bool test = true;
        /*
         Create 2-D vector matrix and calculate if
         this matrix is considered in \Lambda_k
     by calling row and column sums if they
     equal 'k', if not get next permutation
     */
    for(int p =0;p<N;p++){
        //Varible to check each row sum
        int rowSum = 0;
        //Variables to check each column sum
        int colSum = 0;
        //To find column sum in 1-D array, track row number
        int colTrace = p;

        for(int q =0;q<N;q++){
            matrix[p][q] = temp[count];
            rowSum += matrix[p][q];

            colSum += temp[colTrace];
            //To traverse the 1-D array
            count++;
            //get rowNumber + N to get the next column element
            colTrace += N;
        }
        if(rowSum != k || colSum != k){
            test = false;
        }
    }

    if(test){
        storeMatrix.push_back(matrix);
        //per(kA) = k^N per(A)
        long double per = ryserPermanentDouble(matrix)/
         ((long double)pow(k, N));
        storePer.push_back(per);
    }
    matrix.clear();

    //Using in-built next_permutation in C++ which
    //gives all possible permutations of an array
}while(next_permutation(temp, temp+elements));


cout<<"Number of matrices considered are : "<<
```

```
     storePer.size()<<endl;
    cout<<"Printing all matrices considered in Lambda_"<<k
     <<" where N = "<<N<<endl;
    cout<<endl;
    for(int i =0;i<storeMatrix.size();i++){
        cout<<"("<<i+1<<") "<<"Permanent of the matrix below is : "
         <<storePer[i]<<endl;
        printM(storeMatrix[i]);
    }
}

long double Project1::ryserPermanentDouble
(std::vector<std::vector<long double>> matrix){
    long double per = 0.0;

    int size = (int)matrix.size();

    unsigned long long int subMatrices = (long)pow((int)2, size);

    int* arr = new int[size+1];
    double product = 1;
    long double sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (long double)pow((int)-1, size-arr[size]) * product;
    }
    return per;
}
```

- **Program to calculate sum of all subpermanents of order $n-1$ which is $\sigma_{N-1}(A)$**

```
unsigned long long int Project1::subPermanents
(std::vector<std::vector<int>> matrix, int n){
    unsigned long long int sum = 0;
    cout<<"1) Matrix "<<n<<" x "<< n <<endl;
    print();
    /*Create a set of ’n’ elements which is the order of the matrix*/
    int set[n];
    for(int i =0; i<n;i++){
        set[i] = i;
    }
    vector<int*> subsets = subPermanentsHelp(set, n);

    vector<std::vector<std::vector<int>>> storeBin;

    for(int i = 0;i<subsets.size();i++){
        int* r = subsets[i];
        for(int j =0;j<subsets.size();j++){
            int* c = subsets[j];
            std::vector<std::vector<int>> temp;
            temp.resize(n-1, std::vector<int>(n-1, 0));

            for(int t = 0;t<n-1;t++){
                for(int p =0;p<n-1;p++){
                    temp[t][p] = matrix[r[t]][c[p]];
                }
            }
            sum += ryserPermanent(temp);
            storeBin.push_back(temp);
            temp.clear();
        }
    }
    cout<<"2) Number of sub-matrices of order "<<n-1<<" = "<<
     (int)storeBin.size()<<endl;
    cout<<"3) Sub-matrices are as follows : "<<endl;
    for(int i = 0;i<storeBin.size();i++){
        printM(storeBin[i]);
    }
    cout<<"4) Sum of sub-permanents of all the matrix of order N-1 = "<<
     sum<<endl;
    return sum;
}
```

```
/*This method finds the possible permutations of n-1 elements*/
vector<int*> Project1::subPermanentsHelp(int set[], int n){
    vector<int*> ans;

    int count = 0;
    int subsets = (int)pow(2, n);

    vector<int*> answer;
    for(int i = 1; i<subsets;i++){
        int* value =(int*)calloc(n, sizeof(int));
        int end = n-1;
        int temp = i;
        int check = 0;

        while(temp > 0){
            int r = temp%2;
            //Count number of 1's
            if(r == 1){
                check++;
            }
            value[end] = r;
            temp = temp/2;
            end--;
        }
        if(end>-1){
            while(end>-1){
                value[end] = 0;
                end--;
            }
        }
        /*Check if we have number of 1's = n-1,
         we push it as we need N-1 order matrices possible*/
        if(check == n-1){
            count++;
            answer.push_back(value);
        }
    }

    //We create a vector<int*> with the possible permutations
    /*we store all the possible permutations of size n-1 in
```

```
     int* and store each in vector and return it*/
    for(int p = 0; p<answer.size();p++){
        int* c = answer[p];
        int* toReturn = (int*)calloc(n-1, sizeof(int));;
        int index = 0;
        for(int j = 0;j<n;j++){
            if(c[j] == 1){
                toReturn[index] = set[j];
                index++;
            }
        }
        ans.push_back(toReturn);
    }
    return ans;
}

unsigned long long int Project1::ryserPermanent
(std::vector<std::vector<int>> matrix){
    unsigned long long int per = 0;

    int size = (int)matrix.size();
    unsigned long long int subMatrices = (long)pow((int)2, size);

    int* arr = new int[size+1];
    unsigned long long int product = 1;
    unsigned long long int sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (unsigned long long int)pow
         ((unsigned long long int)-1, size-arr[size]) * product;
    }
    return per;
}
```

- **Write a program that, among all $N \times N$ matrices A from the class Λ_k finds all matrices that minimize and maximize the following quantity:**

$$per(A) - \frac{1}{N^2}\sigma_{N-1}(A).$$

```
//We have a fixed N
//We have a variable 'k' such that 1<= k <= N
/*
 We use a part of the similar code from Task 5 to calculate
 all the matrices in a particular \Lambda_k for 1,= k <= N,
 the only difference here is that we calculate the functional
 value for each of the matrix rather than just calculating
 the permanent of the matrix.
 */
/* Functional value to be calculated:
               per(A) - ((1/N^2) * sigma(N-1)(A))
 */
void Project1::task7(int N){
    cout<<"N = "<<N<<endl;
    cout<<"Functional value to minimise and maximise : per(A) - ((1/N^2)"
    <<" * sigma(N-1)(A))\n"<<endl;
    bool check = true;
    long double largestFunc = 0;
    long double smallestFunc = 0;
    vector<std::vector<std::vector<int>>> storeMatrix;
    vector<double> storeFunc;

    for(int k=1;k<=N;k++){
        cout<<"CASE "<<k<<" : "<<"k = "<<k<<endl;
        cout<<"All the F(A) values found for matrices in Lambda_"
        <<k<<" are : "<<endl;
        long double largeFuncK = 0;
        long double smallFuncK = 0;
        bool thisCheck = true;

        int elements = N*N;
        int temp[elements];
        //initialize the array with 0's
        memset(temp, 0, sizeof(temp));
        /*
         add 1's to tha last k*N indexes of the array
         to use next_Permutation()
         */
        for(int i = elements-(k*N);i<elements;i++){
            temp[i] = 1;
        }
        //make a temp matrix to push in vector
        std::vector<std::vector<int>> matrix;
        do{
            matrix.resize(N, std::vector<int>(N, 0));
            //coverting array to vector matrix
            int count = 0;
            bool test = true;
            /*
             Create 2-D vector matrix and calculate if
             this matrix is considered in \Lambda_k
             by calculating row and column sums if they
             equal 'k', if not get next permutation
             */
            for(int p =0;p<N;p++){
                //Varible to check each row sum
                int rowSum = 0;
                //Variables to check each column sum
                int colSum = 0;
                //To find column sum in 1-D array, track row number
                int colTrace = p;

                for(int q =0;q<N;q++){
                    matrix[p][q] = temp[count];
                    rowSum += matrix[p][q];

                    colSum += temp[colTrace];
                    //To traverse the 1-D array
                    count++;
                    //get rowNumber + N to get the next column element
                    colTrace += N;
                }
                if(rowSum != k || colSum != k){
                    test = false;
                }
            }
            //If matrix is in \Lambda_k then find the functional value
            //and then store the matrix and the value
           if(test){
                //store the matrix
                storeMatrix.push_back(matrix);
                //FIND functional value
                //per(A) - ((1/N^2) * sigma(N-1)(A))
                cout<<"Matrix :"<< endl;
                printM(matrix);
                double permanent =
                (double)((task7Help1(matrix)/((double)pow(k, N))));
                cout<<"Permanent of the matrix A, per(A) = "
                <<setprecision(8)
                 <<permanent<<endl;

                double oneByNsq = 1/pow(N, 2);
                cout<<"1/N^2 value = "<<oneByNsq<<endl;

                cout<<"Calculating sigma(A)N-1 now, "
                    <<"step by step : "<<endl;
                double perSumN = (double)(task7Help2(matrix, k));
                cout<<"sigma(A)N-1 value = "<<perSumN<<endl;


                double func =
                (double)(((permanent*permanent) - (oneByNsq * perSumN)*
                          (oneByNsq * perSumN))/(permanent+
                          (oneByNsq * perSumN)));
                cout<<"per(A) - ((1/N^2) * sigma(N-1)(A)) =
                 "<<setprecision(10)<<func<<endl;

                //store functional value
                storeFunc.push_back(func);
                //To find minimum and maximum value of the function
                //amongst all the matrices considered
                if(check){
                    largestFunc = func;
```

```
            smallestFunc = func;
            check = false;
        }
        if(func>=largestFunc){
            largestFunc = func;
        }
        if(func<=smallestFunc){
            smallestFunc = func;
        }
        //Check for this particular 'k'
        if(thisCheck){
            largeFuncK = func;
            smallFuncK = func;
            thisCheck = false;
        }
        if(func>=largeFuncK){
            largeFuncK = func;
        }
        if(func<=smallFuncK){
            smallFuncK = func;
        }
    }
    matrix.clear();

    //Using in-built next_permutation in C++ which
    //gives all possible permutations of an array
}while(next_permutation(temp, temp+elements));

cout<<"\n\nAmong all "<<N<<"x"<<N<<" matrices from Lambda_"
<<k<<" the matrix/matrices that MINIMIZES "<<
"F(A)= per(A)-1/"<<N<<"^2"<<" * "<<"sigma_"<<N-1<<"(A)"
<<" are as following with MINIMUM F(A) = "<<smallFuncK<<endl;

for(int r = 0;r<storeFunc.size();r++){
    if(storeFunc[r] == smallFuncK){
        cout<<"A = "<<endl;
        printM(storeMatrix[r]);
    }
}

cout<<"Among all "<<N<<"x"<<N<<" matrices from Lambda_"
```

```
            <<k<<" the matrix/matrices that MAXIMZES "<<
            "F(A)= per(A)-1/"<<N<<"^2"<<" * "<<"sigma_"<<N-1<<
            "(A)"<<" are as following with MAXIMUM F(A) = "<<largeFuncK<<endl;

            for(int r = 0;r<storeFunc.size();r++){
                if(storeFunc[r] == largeFuncK){
                    cout<<"A = "<<endl;
                    printM(storeMatrix[r]);
                }
            }
        }
        cout<<"Final CASE for k = 1 to k = "<<N<<endl;
        cout<<"Number of matrices considered are "<<
        "(belonging to Lambda_1...Lambda_N in total) : "
            <<storeMatrix.size()<<endl;

        cout<<"\nAmong all "<<N<<"x"<<N<<" matrices from Lambda_1...Lambda_"
        <<N<<" the matrices that MINIMIZES "<<
        "F(A)= per(A)-1/"<<N<<"^2"<<" * "<<"sigma_"<<N-1<<"(A)"
        <<" are as following with minimum value F(A) = "<<smallestFunc<<endl;
        for(int i=0;i<storeFunc.size();i++){
            if(storeFunc[i] == smallestFunc){
                cout<<"A = "<<endl;
                printM(storeMatrix[i]);
            }
        }
        cout<<"Among all "<<N<<"x"<<N<<" matrices from Lambda_1...Lambda_"
            <<N<<" the matrices that MAXIMIZES "<<
        "F(A)= per(A)-1/"<<N<<"^2"<<" * "<<"sigma_"<<N-1<<"(A)"
        <<" are as following with maximum value F(A) = "<<largestFunc<<endl;
        for(int i=0;i<storeFunc.size();i++){
            if(storeFunc[i] == largestFunc){
                cout<<"A = "<<endl;
                printM(storeMatrix[i]);
            }
        }
    }
/*RYSER's Formula, we can simply use the general ryserPermanent
 method but I made a diffent method for this task to stay in
 flow and better understand this program and what this is using */
int Project1::task7Help1(std::vector<std::vector<int>> matrix){
    int per = 0;

    int size = (int)matrix.size();

    unsigned long long int subMatrices = (long)pow((int)2, size);

    int* arr = new int[size+1];
    int product = 1;
    int sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (int)pow((int)-1, size-arr[size]) * product;
    }
    return per;
}

//GET sum  of all sub-permanents of order N-1 for the input NxN matrix
/*Same code as Task 6, but made a different method because we need not
print the results for this method, we just need to use it to find our
funtional value*/
/*This still uses the helper methods from Task 6 but they don't print
 any results so we can keep them as it is*/
double Project1::task7Help2
            (std::vector<std::vector<int>> matrix, int k){
    int n = (int)matrix.size();
    double sum = 0;
    /*Create a set of 'n' elements which is the order of the matrix*/
    int set[n];
    for(int i =0; i<n;i++){
        set[i] = i;
    }
    vector<int*> subsets = task6Help1(set, n);

    vector<std::vector<std::vector<double>>> storeBin;

    for(int i = 0;i<subsets.size();i++){
        int* r = subsets[i];
        for(int j =0;j<subsets.size();j++){
            int* c = subsets[j];
            std::vector<std::vector<double>> temp;
            temp.resize(n-1, std::vector<double>(n-1, 0));

            for(int t = 0;t<n-1;t++){
                for(int p =0;p<n-1;p++){
                    temp[t][p] = (double)((matrix[r[t]][c[p]])/(double)k);
                }
            }
            sum += task7Help3(temp);
            storeBin.push_back(temp);
            temp.clear();
        }
    }
    cout<<"Sub-matrices of N-1 order are as follows : "<<endl;
    for(int i = 0;i<storeBin.size();i++){
        printM(storeBin[i]);
    }
    return sum;
}

double Project1::task7Help3(std::vector<std::vector<double>> matrix){
    double per = 0;

    int size = (int)matrix.size();
    unsigned long long int subMatrices = (long)pow((int)2, size);

    int* arr = new int[size+1];
    double product = 1;
    double sum = 0;

    for(long i = 1;i<subMatrices;i++){
        product = 1;
        arr = store(i, size);
        for(int j =0; j< size;j++){
            sum = 0;

            for(int k = 0; k<size;k++){
                sum += arr[k] * matrix[j][k];
            }
            product *= sum;
        }
        per += (double)((int)pow(-1, size-arr[size]) * (double)product);
    }
    return per;
}
```