```
vector<vector<int>> grassmanCalcul(vector<vector<int>> S, int k){
    vector<vector<int>> SI;
    int n = S.size();
    vector<pairs> row1;

    vector<pairs>  row2;
    vector<string> combinaisons = comb(n, k);



    for(int h=0; h<combinaisons.size(); h++){

        vector<pairs> result;
        for (int a = 0; a<S[0].size(); a++){
            result.push_back(makePairs(S[stringToInt(combinaisons[h], 0)][a],std::to_string(a) ));
        }



        for(int i = 1; i<combinaisons[h].size(); i++){

            row1 = result;
            result.clear();
            row2.clear();


            for (int a = 0; a<S[0].size(); a++){
                row2.push_back(makePairs(S[stringToInt(combinaisons[h], i)][a],std::to_string(a) )
            }



            result = multipli2Vects(row1, row2);


            vector<string> combTemp = comb(maxIndex(result)+1, result[0].second.size());

            int size = result.size();
            for(int j = 0; j<size; ){
                if(checkExactConcordance(result[j].second  , combTemp) == -1){
                    int combinaisonNumber=checkIndexConcordance(result[j].second, combTemp);
                    if(combinaisonNumber!= -1){

                        int sign = parity(result[j].second, combTemp[combinaisonNumber] );
                                                displayPairVector(result);
                        result = useSym(result,j,  sign*result[j].first, combTemp[combinaisonNumbe
                        size--;
                    }
                    else{
                        j++;
                    }
                }
                else{
                    j++;
                }
            }
        }

        vector<int> vectorTemp;
        vectorTemp.clear();
        for(int k = 0; k<result.size(); k++){
            vectorTemp.push_back(result[k].first);
        }
        SI.push_back(vectorTemp);
    }
    return SI;
}
```

nouveau 1

```
GrassmanCalulcul(matrix S, int K){
    matrix S*;
    int n = size of S;
    Vector<pair> row1, row2;
    Vector<string> combinations = comb(n,k);


    For h From 0 To size of combinations, h<-h+1{
        Vector<pair> result;

        result <- the row of S corresponding to the first element of combinations[h];



        For i From 0 To k, i<-i+1{

            row1 <- result;
            clear result and row2;






            row2 <- the row of S corresponding to the iᵗʰ of combinations[h];


            result = row1*row2;


            Vector<string> comTemp = comb (max index of result +1, size of second of result elements;

            For all elements j of result{
                if result[j].second != any elements of combTemp{


                    if result[j].second correspond to the combinationNumberᵗʰ element of combTemp but with a element ≠
                order{
                        int sign = parityOfPermutation(result[j].second, combtemp(combinationNumber);

                        add  sign*result[j].first to the element of result who has the same combination but in the
                    rigth order;
                        delete the line j of result;
                    }
                    else{
                        j <- j+1;
                    }
                }
                else{
                    j <- j+1;
                }
            }




    Add the first column of result to S*;
    }

    Retun S*;
}
```

Vector<pair> :
Int first, string second

```
5, 0
1, 1
4, 2
```

5***e0**+1***e1**+4***e2**

```
   row1:
5, 0
1, 1
4, 2
```

*

```
   row2:
-2, 0
3, 1
1, 2
```

=

```
15, 01
5, 02
-2, 10
1, 12
-8, 20
12, 21
```

Max index of result:2

Size of scond of result : 2
because combination of
 2 vectors

```
before sym:
15, 01
5, 02
-2, 10
1, 12
-8, 20
12, 21
remplacing -2 with index 10 by adding 2 to index 01
after sym:
17, 01
5, 02
1, 12
-8, 20
12, 21
```

Calcul of the parity of the permutation by counting cycles
If even=> parity ← 1
If odd => parity ← -1

```
end of loop
17, 01
13, 02
-11, 12
```

```
end of loop
10, 01
35, 02
-1, 12
```

```
end of loop
-4, 01
-14, 02
19, 12
```

17	13	-11
10	35	-1
-4	-14	19

<u>Complexity in fucntion of n, k and s= number of elements of comb(n,k) :</u>
We have 3 main nested loops of lengths s, k and s again.

Consequently, the complexity of this algorithm is $T(n,k,s) \in O(s^2k)$

Choice of the matrix :

Generation of a random generation of a on-invertible n*n matrix

Generation of a n*(n-1) matrix, the last column is the sum of all other collums:

Choice of n and k :

n k s s^2 s^2/k (with s=number of combination(n, k))

```
4 2 6   36   18
5 2 10   100   50
5 3 10   100   33.3333
6 2 15   225   112.5
6 3 20   400   133.333
6 4 15   225   56.25
7 2 21   441   220.5
7 3 35   1225   408.333
7 4 35   1225   306.25
7 5 21   441   88.2
8 2 28   784   392
8 3 56   3136   1045.33
8 4 70   4900   1225
8 5 56   3136   627.2
8 6 28   784   130.667
9 2 36   1296   648
9 3 84   7056   2352
9 4 126   15876   3969
9 5 126   15876   3175.2
9 6 84   7056   1176
9 7 36   1296   185.143
10 2 45   2025   1012.5
10 3 120   14400   4800
10 4 210   44100   11025
10 5 252   63504   12700.8
10 6 210   44100   7350
10 7 120   14400   2057.14
10 8 45   2025   253.125
11 2 55   3025   1512.5
11 3 165   27225   9075
11 4 330   108900   27225
11 5 462   213444   42688.8
11 6 462   213444   35574
11 7 330   108900   15557.1
11 8 165   27225   3403.12
11 9 55   3025   336.111
12 2 66   4356   2178
12 3 220   48400   16133.3
12 4 495   245025   61256.2
12 5 792   627264   125453
12 6 924   853776   142296
12 7 792   627264   89609.1
12 8 495   245025   30628.1
12 9 220   48400   5377.78
12 10 66   4356   435.6
13 2 78   6084   3042
13 3 286   81796   27265.3
13 4 715   511225   127806
```


Choose