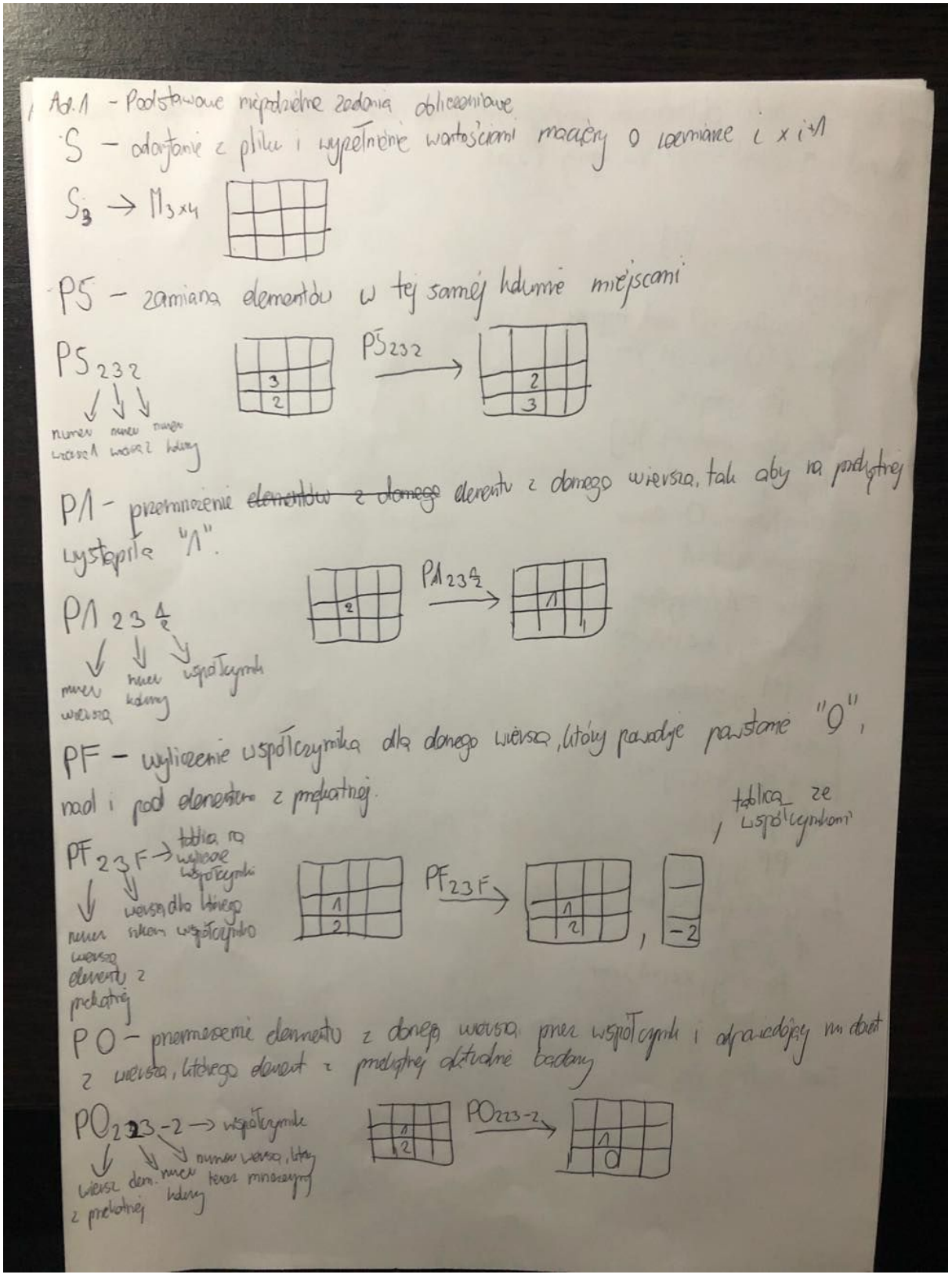
Sprawozdanie z zadania domowego z laboratorium 5. - Dawid Witek 14:40

Pytania teoretyczne 1-6:



Ad. 2 - Ciąg zadań obliczeniowych wykonywanych przez algorytm sekwencyjny w formie pseudokodu

matrix = Silnad.file() - macierz na wymiary i x i+1

```
for y=0; y<i; y++:
    diagElem = matrix[i][i]
    tmp = y+1
    while diagElem==0 and tmp<i:
        for x=0; x<i+1; x++:
            PS y tmp x
        diagElem = matrix[i][i]
        tmp++;
    if diagElem == 0: error
    if diagElem != 1:
        factor = 1/diagElem
        for x=y; x<i+1; x++:
            P1 y x factor
    factors = [i]
    for y'=0; y'<i; y'++:
        if y' != y:
            PF y y' factors
    for y'=0; y'<i; y'++:
        if y' != y:
            for x=y; x<i+1; x++:
                PO y x y' factors[y']
Save result to file;
```

Ad. 3 - Identyfikacja alfabetu w sensie teorii śladów

~~Słowo~~ N $S = \{S_i \mid i \in N\}$

N - wymiar macierzy, po S_i pastoje macierz $M_{N \times N+1}$

~~Regu~~ $PS = \{PS_{ijk} \mid i \in [1, N-1],\ j \in [i+1, N],\ k \in [1, N+1],\ i,j,k \in N\}$

$P1 = \{P1_{ijf} \mid i \in [1, N],\ j \in [1, N+1],\ f \in R,\ i,j \in N\}$

$PF = \{PF_{ij} \mid i,j \in [1, N],\ i \neq j,\ i,j \in N\}$

$PO = \{PO^{ki}_{ijf} \mid i, k \in [1, N],\ i \neq k,\ j \in [1, N+1],\ f \in R,\ i,j,k \in N\}$

$A = S \cup PS \cup P1 \cup PF \cup PO$

Ad. 4 - Identyfikacja relacji zależności

$D_1 = \{(Si', PS_{ijk}), [illegible] \mid i \in [1, N-1],\ j \in [i+1, N],\ k \in [1, N+1]$

$N = i',\ i, i', j, k \in N\}$

$D_2 = \{(Si', P1_{ijf}), (Si', PF_{ij}) \mid i \in [1, N],\ j \in [1, N+1],\ f \in R,\ N = i',\ i, i', j \in N\}$

$D_3 = \{(PS_{ijk}, PS_{(j+1)k}) \mid i \in [1, N-2],\ j \in [i+1, N-1],\ k \in [1, N+1],\ i,j,k \in N\}$

$D_4 = \{(PS_{i*jk}, PS_{(i+1)jk}) \mid i \in [1, N-2],\ j \in [i+1, N],\ k \in [1, N+1],\ i,j,k \in N\}$

$D_5 = \{(PS_{ijk}, P1_{ikf}) \mid i \in [1, N-1],\ j \in [i+1, N],\ k \in [1, N+1],\ f \in R,\ i,j,k \in N\}$

$D_6 = \{(P1_{ijf}, P1_{(i+1)jf}) \mid i \in [1, N-1],\ j \in [1, N+1],\ f \in R,\ i,j,k \in N\}$

$D_7 = \{(P1_{ijf}, PF_{ik}) \mid i \in [1, N],\ j \in [1, N+1],\ k \in [1, N],\ i \neq k,\ f \in R,\ i,j,k \in N\}$

$D_8 = \{(PF_{ij}, PF_{(i+1)j}) \mid i \in [1, N+1],\ j \in [1, N],\ i \neq j,\ i,j \in N\}$

$D_9 = \{(PF_{ij}, PO_{ikjf}) \mid i,j \in [1, N],\ i \neq j,\ k \in [1, N+1],\ f \in R,\ i,j,k \in N\}$

$D_{10} = \{(PO_{ikjf}, PO_{(i+1)kjf}) \mid i \in [1, N+1],\ j \in [1, N],\ i \neq j,\ k \in [1, N+1],\ f \in R,\ i,j,k \in N\}$

$D = D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5 \cup D_6 \cup D_7 \cup D_8 \cup D_9 \cup D_{10}$

Ad. 5 - Wyprowadzenie grafu zależności Dickerta

Si N=i

↓

jeśli na przekątnej jest 0

P5 12 j , j∈[1, N+1] , j∈N

↓

jeśli dalej jest 0 na przekątnej

P5 13 j , j∈[1, N+1] , j∈N

⋮ ani będzie coś innego niż 0 na przekątnej,
⋮ jeśli nie znaleźliśmy to error

↓

jeśli na przekątnej nie ma 1

P1 1 jf , j∈[1, N+1] , f∈R , j∈N

↓

PF 1 jF , j∈[1, N] , j≠1 , j∈N

↓

PO 1 i jF[j] , i∈[1, N+1] , j∈[1, N] , j != 1 , F[j]∈R , i, j∈N

⋮ dla każdego wiersza od 1 do N to
⋮ samo, z uwzględnieniem warunków

↓

jeśli na przekątnej nie ma 1

P1 N jf , j∈[1, N+1] , f∈R , j∈N

↓

PF N jF , j∈[1, N] , j≠N , j∈N

↓

PO N i jF[j] , i∈[1, N+1] , j∈[1, N] , j != N , F[j]∈R , i, j∈N

↓

save results

Adr. 6 – Obliczanie klas Footy, wartki są takie same jak w grafie Dicksta

$F_0 = [Si]$ $N=i$

$F_1 = [PS_{1\,2}j]$, $i, j \in [1, N+1]$, $j \in \mathbb{N}$

⋮ – aż na przekątnej nie będzie 0

$F_a = [P1_{1\,j}f]$, $j \in [1, N+1]$, $f \in \mathbb{R}$, $j \in \mathbb{N}$

$F_{a+1} = [PF_{1\,j}F]$, $j \in [1, N]$, $j != 1$, $j \in \mathbb{N}$

$F_{a+2} = [PO_{1\,i\,j}F[j]]$, $i \in [1, N+1]$, $j \in [1, N]$, $j != 1$, $F[j] \in \mathbb{R}$, $i, j \in \mathbb{N}$

⋮ – dla każdego wiersza od 1 do N to

samo, z uwzględnieniem warunków

$F_x = [P1_{N\,j}f]$, $j \in [1, N+1]$, $f \in \mathbb{R}$, $j \in \mathbb{N}$

$F_{x+1} = [PF_{N\,j}F]$, $j \in [1, N]$, $j != N$, $j \in \mathbb{N}$

$F_{x+3} = [PO_{N\,i\,j}F[j]]$, $i \in [1, N+1]$, $j \in [1, N]$, $j != N$, $F[j] \in \mathbb{R}$, $i, j \in \mathbb{N}$

$F_{x+4} = [$save results$]$

Cześć praktyczne 7-9:
Implementacja podstawowych operacji macierzowych z wykorzystaniem schedulera z laboratorium trzeciego.
Wymiania produkcji na odpowiednie dla tego problemu:

```
public class PS extends AbstractProduction<Matrix> {
    private int oldY;
    private int newY;
    private int x;

    public PS(Matrix _matrix, int oldY, int newY, int x){
        super(_matrix);
        this.oldY = oldY;
        this.newY = newY;
        this.x = x;
    }

    @Override
    public Matrix apply(Matrix matrix){
        double tmp = matrix.get(oldY, x);
        matrix.set(oldY, x, matrix.get(newY, x));
        matrix.set(newY, x, tmp);
        return matrix;
    }
}
```

```
public class P1 extends AbstractProduction<Matrix> {
    private int y;
    private int x;
    private double factor;

    public P1(Matrix _matrix, int y, int x, double factor){
        super(_matrix);
        this.y = y;
        this.x = x;
        this.factor = factor;
    }

    @Override
    public Matrix apply(Matrix matrix){
        double newValue = matrix.get(y, x) * factor;
```

```
        matrix.set(y, x, newValue);
        return matrix;
    }
}
```

```
public class PF extends AbstractProduction<Matrix> {
    private int y;
    private int curY;
    private double[] factors;

    public PF(Matrix _matrix, int y, int curY, double[] factors){
        super(_matrix);
        this.y = y;
        this.curY = curY;
        this.factors = factors;
    }

    @Override
    public Matrix apply(Matrix matrix){
        double factor = -1 * matrix.get(curY, y) / matrix.get(y,
y);
        factors[curY] = factor;
        return matrix;
    }
}
```

```
public class P0 extends AbstractProduction<Matrix> {
    private int y;
    private int x;
    private int curY;
    private double factor;

    public P0(Matrix _matrix, int y, int x, int curY, double
factor){
        super(_matrix);
        this.y = y;
        this.x = x;
        this.curY = curY;
        this.factor = factor;
    }
```

```
    @Override
    public Matrix apply(Matrix matrix){
        double newValue = matrix.get(curY, x) + matrix.get(y, x) *
factor;
        matrix.set(curY, x, newValue);
        return matrix;
    }
}
```

Wczytywanie danej macierzy z pliku:

```
int size = 0;
double[][] array = new double[0][0];
try{
    BufferedReader reader = new BufferedReader(new
FileReader(args[0]));
    size =
Integer.parseInt(reader.readLine().replaceAll("\\s",""));
    array = new double[size][size+1];
    String line;
    String[] splittedLine;

    for(int i=0; i<size; i++){
        line = reader.readLine();
        splittedLine = line.split(" ");
        for(int j=0; j<size; j++){
            array[i][j] = Double.parseDouble(splittedLine[j]);
        }
    }
    line = reader.readLine();
    splittedLine = line.split(" ");
    if(splittedLine.length != size){
        System.out.println("Wrong arguments in last line of file
:(");
        System.exit(1);
    }
    for(int i=0; i<size; i++){
        array[i][size] = Double.parseDouble(splittedLine[i]);
    }
    reader.close();
```

```
} catch (Exception e){
   e.printStackTrace();
   System.exit(1);
}

if(size == 0){
   System.out.println("Matrix must exist to solve it !");
   System.exit(1);
}
```

Zapis obliczonej macierzy do pliku:

```
array = matrix.getArray();

try{
   PrintStream ps = new PrintStream(new FileOutputStream(new
File(args[1])));
   System.setOut(ps);
   System.out.println(size);
   for(int y=0; y<size; y++){
       for(int x=0; x<size; x++){
           System.out.print(array[y][x] + " ");
       }
       System.out.println();
   }
   for(int y=0; y<size; y++){
       System.out.print(array[y][size] + " ");
   }
   ps.close();
} catch(Exception e){
   e.printStackTrace();
   System.exit(1);
}
```

Implementacja schedulera dla nowych klas Foaty, opisujących algorytm eliminacji Gaussa:

```
Matrix matrix = new Matrix(array);

for(int y=0; y<size; y++){
   double diagElem = matrix.get(y, y);
   int tmp = y+1;
   while(diagElem == 0 && tmp<size){
       for(int x=0; x<=size; x++){
           PS ps = new PS(matrix, y, tmp, x);
           this.runner.addThread(ps);
       }
       this.runner.startAll();
       diagElem = matrix.get(y, y);
       tmp++;
   }

   if (diagElem == 0){
       System.out.println("Matrix is incorrect :(");
       System.exit(1);
   }

   if (diagElem != 1){
       double factor = 1/diagElem;
       for(int x=y; x<=size; x++){
           P1 p1 = new P1(matrix, y, x, factor);
           this.runner.addThread(p1);
       }
       this.runner.startAll();
   }

   double[] factors = new double[size];
   for(int _y=0; _y<size; _y++){
       if(_y != y){
           PF pf = new PF(matrix, y, _y, factors);
           this.runner.addThread(pf);
       }
   }
   this.runner.startAll();
```

```
    for(int _y=0; _y<size; _y++){
        if(_y != y){
            for(int x=y; x<=size; x++){
                P0 p0 = new P0(matrix, y, x, _y, factors[_y]);
                this.runner.addThread(p0);
            }
        }
    }
    this.runner.startAll();
}
```

Przykładowe wykonanie programu.
Plik wejściowy:

```
5
2.0 0.0 0.0 0.0 0.0
1.0 2.0 0.0 0.0 0.0
0.0 0.0 2.0 0.0 0.0
0.0 0.0 0.0 2.0 0.0
0.0 0.0 0.0 0.0 2.0
4.0 8.0 10.0 12.0 4.0
```

Plik wyjściowy:

```
5
1.0 0.0 0.0 0.0 0.0
0.0 1.0 0.0 0.0 0.0
0.0 0.0 1.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 0.0 1.0
2.0 3.0 5.0 6.0 2.0
```

Do sprawozdania dołączam folder z kodem źródłowym schedulera.