

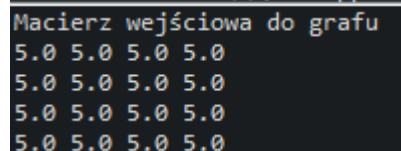
1. Rozkład LU macierzy metodą Gaussa.

Zastosowano poniższy kod:

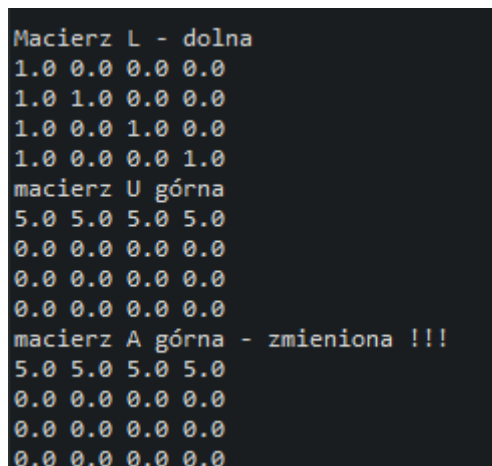
```
for (i = 1; i < N; i++) {
    for(j = i+1; j <=N; j++) {
        for(k = i+1; k <=N; k++) {
            if(A[i][i] != 0) {
                L[j][i] = A[j][i] / A[i][i];
            }else
                L[j][i] = 0;
        }
    }
    for(j = i+1; j <= N; j++) {
        for(k = i; k <= N ; k++) {
            A[j][k] = A[j][k] - (L[j][i]*A[i][k]);
        }
    }
}
```

2. Sprawdzenie algorytmu liczącego rozkład LU sprawdzono za pomocą kalkulatora zewnętrznego.

Po dokonaniu operacji otrzymano rozkładu LU dwie macierze: macierz dolna L oraz macierz górna U.



```
Macierz wejściowa do grafu
5.0 5.0 5.0 5.0
5.0 5.0 5.0 5.0
5.0 5.0 5.0 5.0
5.0 5.0 5.0 5.0
```



```
Macierz L - dolna
1.0 0.0 0.0 0.0
1.0 1.0 0.0 0.0
1.0 0.0 1.0 0.0
1.0 0.0 0.0 1.0
macierz U górna
5.0 5.0 5.0 5.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
macierz A górna - zmieniona !!!
5.0 5.0 5.0 5.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
```

3. Uzyskanie informacji o wierzchołkach i łukach.

W celu otrzymania informacji o wierzchołkach i łukach została stworzona tabela danych wyjściowych za pomocą poniższego kodu:

```
for (i = 1; i < N; i++) {
    for(j = i+1; j <=N; j++) {
        for(k = i+1; k <=N; k++) {           // dane kolejne gniazdo
            nr++;
            W1[nr] = i;
            W2[nr] = j;
            W3[nr] = k;
            Ia1i2[nr]= j;
            Ia1i3[nr] = k;
```

```

Ia2i1[nr] = i;
Ia2i3[nr] = k;
Imi2[nr] = j;
Imi1[nr] = i; // Konstruowanie grafu algorytmu
kolumna = 0;
wiersz = nr;
tabela[wiersz][kolumna] = i; // Tworzenie tabeli
kolumna++;
tabela[wiersz][kolumna] = j;
kolumna++;
tabela[wiersz][kolumna] = k;
kolumna++;
tabela[wiersz][kolumna] = j;
kolumna++;
tabela[wiersz][kolumna] = i;
kolumna++;
tabela[wiersz][kolumna] = i;
kolumna++;
tabela[wiersz][kolumna] = k;
kolumna++;
tabela[wiersz][kolumna] = j;
kolumna++;
tabela[wiersz][kolumna] = k;
// Wyświetlanie tabeli pierwsze gniazdo
System.out.println(nr + " -> " + W1[nr] + " " + W2[nr] + " " + W3[nr]
+ " " + Imi2[nr] + " " + Imi1[nr] + " " + Ia2i1[nr] + " " + Ia2i3[nr] + " " + Ia1i2[nr]
+ " " + Ia1i3[nr]);
}

if(A[i][i] != 0) {
    L[j][i] = A[j][i] / A[i][i];
}
else
    L[j][i] = 0;
}
wiersz++;
System.out.println("Drugie gniazdo");
for(j = i+1; j <= N; j++) {
    for(k = i; k <= N ; k++) {
        A[j][k] = A[j][k] - (L[j][i]*A[i][k]);
        nr++;
        W1[nr] = i;
        W2[nr] = j;
        W3[nr] = k;
        Ia1i2[nr] = j;
        Ia1i3[nr] = k;
        Ia2i1[nr] = i;
        Ia2i3[nr] = k;
        Imi2[nr] = j;
        Imi1[nr] = i; // Konstruowanie grafu algorytmu
        kolumna = 0;
        wiersz = nr;
        tabela[wiersz][kolumna] = i; // Tworzenie tabeli
        kolumna++;
        tabela[wiersz][kolumna] = j;
        kolumna++;
        tabela[wiersz][kolumna] = k;
        kolumna++;
        tabela[wiersz][kolumna] = j;
        kolumna++;
        tabela[wiersz][kolumna] = i;
        kolumna++;
    }
}

```

```

        tabela[wiersz][kolumna] = i;
        kolumna++;
        tabela[wiersz][kolumna] = k;
        kolumna++;
        tabela[wiersz][kolumna] = j;
        kolumna++;
        tabela[wiersz][kolumna] = k;
        // Wyświetlanie tabeli - drugie gniazdo
        System.out.println(nr + " -> " + W1[nr] + " " + W2[nr] + " " + W3[nr]
+ " " + Imi2[nr] + " " + Imi1[nr] + " " + Ia2i1[nr] + " " + Ia2i3[nr] + " " + Ia1i2[nr]
+ " " + Ia1i3[nr]));
    }
}

```

Tabela wynikowa

```

Tabela Temp - oba gniazda, bez sortu
1 2 2 2 1 1 2 2 2
1 2 3 2 1 1 3 2 3
1 2 4 2 1 1 4 2 4
1 3 2 3 1 1 2 3 2
1 3 3 3 1 1 3 3 3
1 3 4 3 1 1 4 3 4
1 4 2 4 1 1 2 4 2
1 4 3 4 1 1 3 4 3
1 4 4 4 1 1 4 4 4
1 2 1 2 1 1 1 2 1
1 3 1 3 1 1 1 3 1
1 4 1 4 1 1 1 4 1
2 3 3 3 2 2 3 3 3
2 3 4 3 2 2 4 3 4
2 4 3 4 2 2 3 4 3
2 4 4 4 2 2 4 4 4
2 3 2 3 2 2 2 3 2
2 4 2 4 2 2 2 4 2
3 4 4 4 3 3 4 4 4
3 4 3 4 3 3 4 3

```

Dokonano sortowania danych po pierwszych trzech indeksach w celu uzyskania właściwego odwzorowania połączeń między węzłami.

```

Tabela Graf
1 2 1 2 1 1 1 2 1
1 2 2 2 1 1 2 2 2
1 2 3 2 1 1 3 2 3
1 2 4 2 1 1 4 2 4
1 3 1 3 1 1 1 3 1
1 3 2 3 1 1 2 3 2
1 3 3 3 1 1 3 3 3
1 3 4 3 1 1 4 3 4
1 4 1 4 1 1 1 4 1
1 4 2 4 1 1 2 4 2
1 4 3 4 1 1 3 4 3
1 4 4 4 1 1 4 4 4
2 3 2 3 2 2 2 3 2
2 3 3 3 2 2 3 3 3
2 3 4 3 2 2 4 3 4
2 4 2 4 2 2 2 4 2
2 4 3 4 2 2 3 4 3
2 4 4 4 2 2 4 4 4
3 4 3 4 3 3 4 3
3 4 4 4 3 3 4 4

```

4. Uzyskanie luków między węzłami:

```
for(i = 0 ; i < iloscBezDuplikatow ; i++) {
    for(j = i +1 ; j < iloscBezDuplikatow ; j++) {
        // Pierwsza kolumna -> Pierwsza kolumna
        if(tabelaGraf[i][3] == tabelaGraf[j][3] && tabelaGraf[i][4] ==
tabelaGraf[j][4] ) {
            if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
                ;
            }
            else {
                System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
            }
        }
        // Pierwsza kolumna -> Druga kolumna
        if(tabelaGraf[i][3] == tabelaGraf[j][5] && tabelaGraf[i][4] ==
tabelaGraf[j][6] ) {
            if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
                ;
            }
            else {
                System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
            }
        }
        // Pierwsza kolumna -> Trzecia kolumna
        if(tabelaGraf[i][3] == tabelaGraf[j][5] && tabelaGraf[i][4] ==
tabelaGraf[j][6] ) {
            if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
                ;
            }
            else {
                System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
            }
        }
        // Druga kolumna -> Druga kolumna
        if(tabelaGraf[i][5] == tabelaGraf[j][5] && tabelaGraf[i][6] ==
tabelaGraf[j][6] ) {
            if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
                ;
            }
            else {
                System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
            }
        }
        // Druga kolumna -> Trzecia kolumna
        if(tabelaGraf[i][5] == tabelaGraf[j][7] && tabelaGraf[i][6] ==
tabelaGraf[j][8] ) {
            if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
                ;
            }
            else {
                System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
            }
        }
        // Trzecia kolumna -> Trzecia kolumna
        if(tabelaGraf[i][7] == tabelaGraf[j][7] && tabelaGraf[i][8] ==
tabelaGraf[j][8] ) {
```

```

        if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
            ;
        }
        else {
            System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
        }
    }
    // Trzecia kolumna -> Druga kolumna
    if(tabelaGraf[i][7] == tabelaGraf[j][5] && tabelaGraf[i][8] ==
tabelaGraf[j][6] ) {
        if((tabelaGraf[i][0] + 1 ) < tabelaGraf[j][0] || (tabelaGraf[i][1] +
1 ) < tabelaGraf[j][1] || (tabelaGraf[i][2] + 1 ) < tabelaGraf[j][2]) {
            ;
        }
        else {
            System.out.println("Węzełek " + tabelaGraf[i][0] +
tabelaGraf[i][1] + tabelaGraf[i][2] + " - > " + + tabelaGraf[j][0] + tabelaGraf[j][1] + tabelaGraf[j][2] + "
Od " + tabelaGraf[i][3] + tabelaGraf[i][4] + " do -> " + tabelaGraf[j][3] + tabelaGraf[j][4]);
        }
    }
}
}
}

```

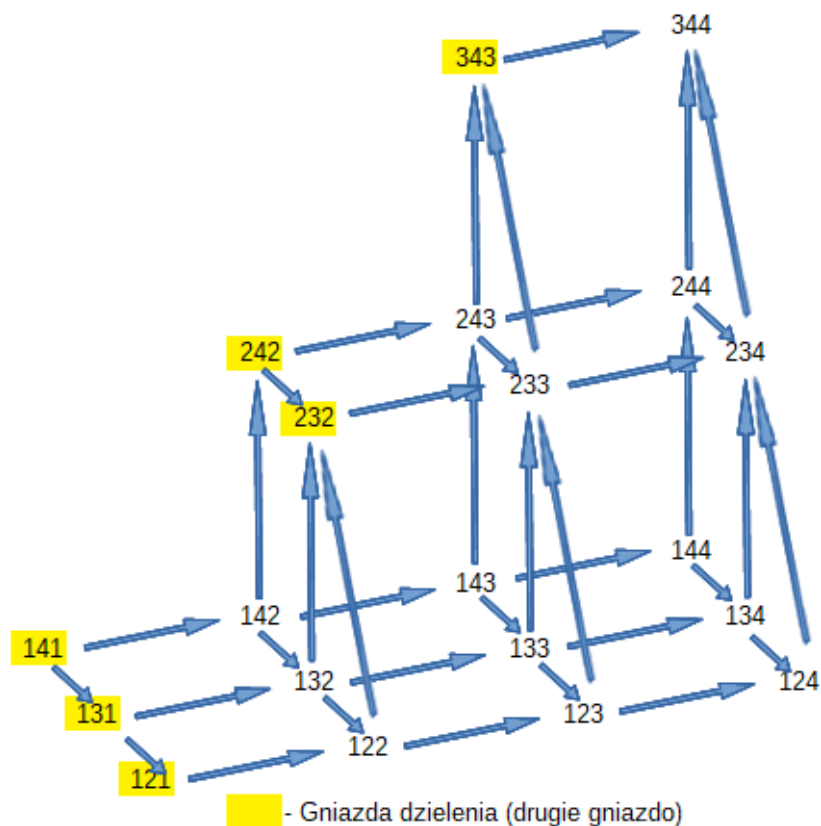
Lista otrzymanych wyników:

```

Połączenia gniazd - węzły
Węzełek 121 - > 122 Od 21 do -> 21
Węzełek 121 - > 131 Od 21 do -> 31
Węzełek 122 - > 123 Od 21 do -> 21
Węzełek 122 - > 132 Od 21 do -> 31
Węzełek 122 - > 232 Od 21 do -> 32
Węzełek 123 - > 124 Od 21 do -> 21
Węzełek 123 - > 133 Od 21 do -> 31
Węzełek 123 - > 233 Od 21 do -> 32
Węzełek 124 - > 134 Od 21 do -> 31
Węzełek 124 - > 234 Od 21 do -> 32
Węzełek 131 - > 132 Od 31 do -> 31
Węzełek 131 - > 141 Od 31 do -> 41
Węzełek 132 - > 133 Od 31 do -> 31
Węzełek 132 - > 142 Od 31 do -> 41
Węzełek 132 - > 232 Od 31 do -> 32
Węzełek 133 - > 134 Od 31 do -> 31
Węzełek 133 - > 143 Od 31 do -> 41
Węzełek 133 - > 233 Od 31 do -> 32
Węzełek 134 - > 144 Od 31 do -> 41
Węzełek 134 - > 234 Od 31 do -> 32
Węzełek 141 - > 142 Od 41 do -> 41
Węzełek 142 - > 143 Od 41 do -> 41
Węzełek 142 - > 242 Od 41 do -> 42
Węzełek 143 - > 144 Od 41 do -> 41
Węzełek 143 - > 243 Od 41 do -> 42
Węzełek 144 - > 244 Od 41 do -> 42
Węzełek 232 - > 233 Od 32 do -> 32
Węzełek 232 - > 242 Od 32 do -> 42
Węzełek 233 - > 234 Od 32 do -> 32
Węzełek 233 - > 243 Od 32 do -> 42
Węzełek 233 - > 343 Od 32 do -> 43

```

Otrzymane wyniki węzłów i łuków nanosimy na wykres 3 wymiarowy.



5. Odwzorowanie algorytmów w architektury MP.

Przyjęto następujące założenia do dalszych obliczeń:

```
int D[][] = {{ 1, 0, 0, 1 },
              { 0, 1, 0, 1 },
              { 0, 0, 1, 0 }};

int Ft[] = { 1, 1, 1};
```

Macierz $D[][]$ jest wynikiem połączeń między węzłami, tzn ich kierunku. Kolejne kolumny macierzy D oznaczają odpowiednie kierunki na osi trójwymiarowej.

Macierz Ft jest macierzą odwzorowania czasowego, jako macierz jednowymiarowa.

6. Konstruowanie macierzy zależności informacyjnych.

Tworzymy macierz $FS1$, aby odwzorować macierz trójwymiarową na macierzy dwuwymiarowej.

Sprawdzenie warunku lokalności

$FS1 * D = 1$ or 0 or -1

Jeśli wyniki różnią się od powyższych, macierz Fs1 nie może być użyta.

```
// wyświetlenie FSd1 dla kolejnych kolumn
System.out.println("Obliczona tablica FSd1");
for( i = 0 ; i < 2 ; i++) {
    for( j = 0 ; j < 4 ; j++) {
        System.out.print(FSd1[i][j]);
        if(FSd1[i][j] == -1 || FSd1[i][j] == 0 || FSd1[i][j] == 1 ) {

        }else
        {
            sprawdzenieFSd1++;
        }
    }
    System.out.println("");
}
// jeśli warunek dla FSd1 nie został spełniony, FSd1 nie nadaje się do architektury
if(sprawdzenieFSd1 > 0 ) {
    System.out.println("FSd1 nie spełnia warunku FS1 * d = -1 or 0 or 1");
}else {
    System.out.println("FSd1 spełnia warunek FS1*d = -1 or 0 or 1");
}
```

Program sprawdza poprawność macierzy Fs

```
Obliczona tablica FSd1
1011
-11-10
FSd1 spełnia warunek FS1*d = -1 or 0 or 1
```

Jeśli macierz jest Fs jest poprawna przechodzimy do obliczenia EP dla Fs

```
Architektura FS 1
x = 2 y = 0 takt - 4 Wektor - 121
x = 3 y = -1 takt - 5 Wektor - 122
x = 4 y = -2 takt - 6 Wektor - 123
x = 5 y = -3 takt - 7 Wektor - 124
x = 2 y = 1 takt - 5 Wektor - 131
x = 3 y = 0 takt - 6 Wektor - 132
x = 4 y = -1 takt - 7 Wektor - 133
x = 5 y = -2 takt - 8 Wektor - 134
x = 2 y = 2 takt - 6 Wektor - 141
x = 3 y = 1 takt - 7 Wektor - 142
x = 4 y = 0 takt - 8 Wektor - 143
x = 5 y = -1 takt - 9 Wektor - 144
x = 4 y = -1 takt - 7 Wektor - 232
x = 5 y = -2 takt - 8 Wektor - 233
x = 6 y = -3 takt - 9 Wektor - 234
x = 4 y = 0 takt - 8 Wektor - 242
x = 5 y = -1 takt - 9 Wektor - 243
x = 6 y = -2 takt - 10 Wektor - 244
x = 6 y = -2 takt - 10 Wektor - 343
x = 7 y = -3 takt - 11 Wektor - 344
```

Otrzymujemy współrzędne

EP w wartościach x i y.

Na podstawie współrzędnych EP ustalamy najdłuższą ścieżkę .

Współrzędne EP sprawdzamy dla różnych wartości N – wielkości macierzy wejściowej.

Tablica Ep dla macierzy $N = 3$

```
FSd1 spełnia warunek FS1*d = -1 or 0 or 1
Architektura FS 1
x = 2 y = 0 takt - 4 Wektor - 121
x = 3 y = -1 takt - 5 Wektor - 122
x = 4 y = -2 takt - 6 Wektor - 123
x = 2 y = 1 takt - 5 Wektor - 131
x = 3 y = 0 takt - 6 Wektor - 132
x = 4 y = -1 takt - 7 Wektor - 133
x = 4 y = -1 takt - 7 Wektor - 232
x = 5 y = -2 takt - 8 Wektor - 233
```

Tablica Ep dla macierzy $N = 4$

```
Architektura FS 1
x = 2 y = 0 takt - 4 Wektor - 121
x = 3 y = -1 takt - 5 Wektor - 122
x = 4 y = -2 takt - 6 Wektor - 123
x = 5 y = -3 takt - 7 Wektor - 124
x = 2 y = 1 takt - 5 Wektor - 131
x = 3 y = 0 takt - 6 Wektor - 132
x = 4 y = -1 takt - 7 Wektor - 133
x = 5 y = -2 takt - 8 Wektor - 134
x = 2 y = 2 takt - 6 Wektor - 141
x = 3 y = 1 takt - 7 Wektor - 142
x = 4 y = 0 takt - 8 Wektor - 143
x = 5 y = -1 takt - 9 Wektor - 144
x = 4 y = -1 takt - 7 Wektor - 232
x = 5 y = -2 takt - 8 Wektor - 233
x = 6 y = -3 takt - 9 Wektor - 234
x = 4 y = 0 takt - 8 Wektor - 242
x = 5 y = -1 takt - 9 Wektor - 243
x = 6 y = -2 takt - 10 Wektor - 244
x = 6 y = -2 takt - 10 Wektor - 343
x = 7 y = -3 takt - 11 Wektor - 344
```


Tablica Ep dla macierzy N = 5

```
Architektura FS 1
x = 2 y = 0 takt - 4 Wektor - 121
x = 3 y = -1 takt - 5 Wektor - 122
x = 4 y = -2 takt - 6 Wektor - 123
x = 5 y = -3 takt - 7 Wektor - 124
x = 6 y = -4 takt - 8 Wektor - 125
x = 2 y = 1 takt - 5 Wektor - 131
x = 3 y = 0 takt - 6 Wektor - 132
x = 4 y = -1 takt - 7 Wektor - 133
x = 5 y = -2 takt - 8 Wektor - 134
x = 6 y = -3 takt - 9 Wektor - 135
x = 2 y = 2 takt - 6 Wektor - 141
x = 3 y = 1 takt - 7 Wektor - 142
x = 4 y = 0 takt - 8 Wektor - 143
x = 5 y = -1 takt - 9 Wektor - 144
x = 6 y = -2 takt - 10 Wektor - 145
x = 2 y = 3 takt - 7 Wektor - 151
x = 3 y = 2 takt - 8 Wektor - 152
x = 4 y = 1 takt - 9 Wektor - 153
x = 5 y = 0 takt - 10 Wektor - 154
x = 6 y = -1 takt - 11 Wektor - 155
x = 4 y = -1 takt - 7 Wektor - 232
x = 5 y = -2 takt - 8 Wektor - 233
x = 6 y = -3 takt - 9 Wektor - 234
x = 7 y = -4 takt - 10 Wektor - 235
x = 4 y = 0 takt - 8 Wektor - 242
x = 5 y = -1 takt - 9 Wektor - 243
x = 6 y = -2 takt - 10 Wektor - 244
x = 7 y = -3 takt - 11 Wektor - 245
x = 4 y = 1 takt - 9 Wektor - 252
x = 5 y = 0 takt - 10 Wektor - 253
x = 6 y = -1 takt - 11 Wektor - 254
x = 7 y = -2 takt - 12 Wektor - 255
x = 6 y = -2 takt - 10 Wektor - 343
x = 7 y = -3 takt - 11 Wektor - 344
x = 8 y = -4 takt - 12 Wektor - 345
x = 6 y = -1 takt - 11 Wektor - 353
x = 7 y = -2 takt - 12 Wektor - 354
x = 8 y = -3 takt - 13 Wektor - 355
x = 8 y = -3 takt - 13 Wektor - 454
x = 9 y = -4 takt - 14 Wektor - 455
```

7. Pomiar czasów i kroków.

Na podstawie zależności oraz zmian wartości ustalamy wzór T latency. W naszym przypadku wzór ma postać:

$$T_{lat} = (N * 2 + (N - 4)) * 15$$

Czas w krokach

$$T_k = T_{max} - T_{min} + 1$$

Czas latencji dla N – zależność analityczna między kolejnymi N, przyjęto największą wartość latencji „15” dla dzielenia, które występuje przy rozkładzie Lu Gaussa.

$$T_{lat} (N) = (N * 2 (N - 4)) * 15$$

$$T_{clk} = T_k + T_{lat}$$

$$F = 1 / T$$

$T = T_{clk} / F_{max} (500 \text{ Mhz})$ - obliczenie czasu w ms działania FPG

Otrzymane wyniki dla macierzy $N=3$

$$T_{min} = 4$$

$$T_{max} = 8$$

Czas w taktach

$$T_k = 5$$

$$T_{clk} = 0,16 \text{ ms}$$

```
Takt Min - 4 Takt Max - 8
Czas w taktach: 5
Czas Tk : 5.0
Czas Tclk : 0.16 ms
Czas (komp) - 1.10042 ms
```

Otrzymane wyniki dla macierzy $N=4$

$$T_{min} = 4$$

$$T_{max} = 11$$

Czas w taktach

$$T_k = 8$$

$$T_{clk} = 0,256 \text{ ms}$$

```
Takt Min - 4 Takt Max - 11
Czas w taktach: 8
Czas Tk : 8.0
Czas Tclk : 0.256 ms
Czas (komp) - 2.880847 ms
```

Otrzymane wyniki dla macierzy $N=5$

$$T_{min} = 4$$

$$T_{max} = 14$$

Czas w taktach

$$T_k = 11$$

$$T_{clk} = 0,352 \text{ ms}$$

```
Takt Min - 4 Takt Max - 14
Czas w taktach: 11
Czas Tk : 11.0
Czas Tclk : 0.352 ms
Czas (komp) - 3.129775 ms
```

$$P [\%] = L_w(\text{liczba węzłów}) / (Lep(\text{liczba elementów przetwarzających} * T_k))$$

Przykładowe obliczenia dla $N = 3$

$$P [\%] = 20 / (8 * 5) = 0,5$$

Przykładowe obliczenia dla $N = 26$

$$P [\%] = 20 / (8 * 74) = 4,62$$

```
Takt Min - 4 Takt Max - 8
Czas w taktach: 5
Czas Tk : 5.0
Czas Tclk : 0.16 ms
Takt Min - 4 Takt Max - 8
Czas w taktach: 5
Czas Tk : 5.0
Czas Tclk : 0.16 ms
Czas (komp) - 1.10042 ms
P [ \% ] - obciążenie procesora - 0.5
```

Tabela dla różnych wartości N

N - >	10	15	26
CPU	23,02	70,79	197,21
MP1	0,832	1,312	2,38
MP2	0,832	1,312	2,38

8. Podsumowanie

Pomiary dokonano dla trzech różnych macierzy N. Zauważono, że dla małych macierzy, zysk zastosowania metody konstruowania grafów zależności informacyjnych dla algorytmów regularnych mają niewielkie zastosowanie. Zysk czasowy jest niewiele większy niż rząd wielkości.

Znaczące zyski czasowe pojawiają się dla większych macierzy N. Największa macierz jaką udało się zaimplementować miała $N = 26$. Już przy tej wielkości widać znaczący zysk czasowy, sięgający prawie 100 razy szybsze wykonanie tego samego rozkładu Lu Gaussa. Z doświadczenia wynika, że metody konstruowania grafów zależności informacyjnych dla algorytmów regularnych mają zastosowanie dla większych macierzy. Znacząco przyspieszają wykonanie operacji przekształcania macierzy.

Przy macierzach o niewielkich rozmiarach zysk czasowy jest niewielki ze względu na latencję mającą duży wpływ przy niewielkiej ilości operacji.