

Programowanie  
matematyczne  
c.d.

# Algorytm transponowania macierzy

Translokację macierzy rozumiemy jako zamianę miejscami poszczególnych wierszy tej macierzy z jej kolumnami. Oznacza to, że wiersz staje się kolumna a kolumna wierszem. Operacja taka ma bardzo często miejsce w różnych procedurach i algorytmach, jest np. stosowana w odwracaniu macierzy czy rozwiązywaniu układów równań. Algorytm odwracania macierzy jest jednym z podstawowych algorytmów metod macierzowych i różne jego odmiany można znaleźć w literaturze [6-7, 12, 17-18, 26, 35, 43, 68, 93, 110].

Zastanówmy się teraz w wspólnie nad transpozycją macierzy. Jeśli będziemy mieli daną macierz  $A$  o  $m$  wierszach i  $n$  kolumnach zapisana wzorem:

$$A^{m \times n} = \begin{bmatrix} a_{11} & a_{12} & a_{1j} & \cdots & a_{1n} \\ & & & & \vdots \\ a_{i1} & a_{i2} & a_{ij} & \cdots & a_{in} \\ & & & & \vdots \\ a_{m1} & a_{m2} & a_{mj} & & a_{mn} \end{bmatrix}, i = 1, \dots, m, j = 1, \dots, n,$$

to transpozycja  $A^T$  takiej macierzy będzie wyglądać według wzoru :

$$\left(A^{m \times n}\right)^T = \begin{bmatrix} a_{11} & a_{i1} & \cdots & a_{m1} \\ & & & \vdots \\ a_{1j} & a_{ij} & \cdots & a_{mj} \\ & & & \vdots \\ a_{1n} & a_{in} & & a_{mn} \end{bmatrix}, i = 1, \dots, m, j = 1, \dots, n.$$

## Przykład 56

Napisać program, który wczytuje wymiar macierzy prostokątnej i elementy macierzy  $A \in R^{n \times m}$  z pliku macierz.txt. Wymiar macierzy transponowanej i elementy macierzy transponowanej  $A^T \in R^{m \times n}$  zapisuje do pliku wynik.txt.

Program realizujący algorytm transponowania macierzy matematycznie wydaje się mało skomplikowany, ponieważ jego realizacja polega na zamianie miejscami elementów w kolumnach i wierszach transponowanej macierzy. Ponieważ jest to jednak jeden z ważniejszych algorytmów nie tylko dla metod macierzowych został przedstawiony w niniejszej książce.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
int main()
{
    int i, j, n, m;
    FILE *p, *q;
    if((p = fopen( "macierz.txt", "r" )) == NULL)
    {
        printf("Nie moge otworzyc pliku macierz.txt\n");
        _getch();
    }
    if((q = fopen( "wynik.txt", "w" )) == NULL)
    {
        printf("Nie moge otworzyc pliku wynik.txt\n");
        _getch();
    }

    fscanf(p, "%d %d", &n, &m);

    double **a=new double*[n];

    for(i = 0; i < m; i++)
        a[i] = new double[m];
```

```
// Deklaracja bibliotek.
```

```
// Funkcja główna programu.
```

```
// Deklaracja plików.
```

```
/* Otwarcie pliku do odczytania
danych. */
```

```
/* Otwarcie pliku do zapisu danych. */
```

```
// Czytanie wymiaru zadania.
```

```
// Deklaracja macierzy.
```

```
for(i = 0; i < n; i++)  
    for(j = 0; j < m; j++)  
        fscanf(p, "%lf", a[i] + j);
```

```
fprintf(q, "%d %d\n", m, n);  
for(j = 0; j < m; j++)  
{  
    for(i = 0; i < n; i++)  
        fprintf(q, "%g ", a[i][j]);  
    fprintf(q, "\n");  
}
```

```
fclose(p);  
fclose(q);  
}
```

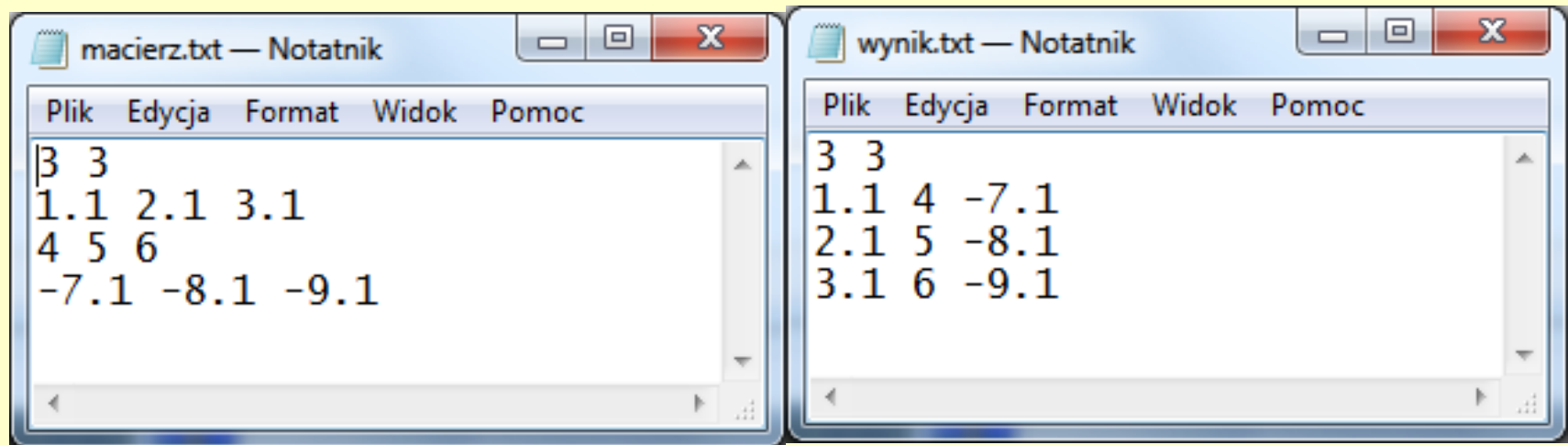
// Czytanie macierzy.

/\* Zapis do pliku wymiaru macierzy  
transponowanej.\*/

/\* Zapis do pliku macierzy  
transponowanej.\*/

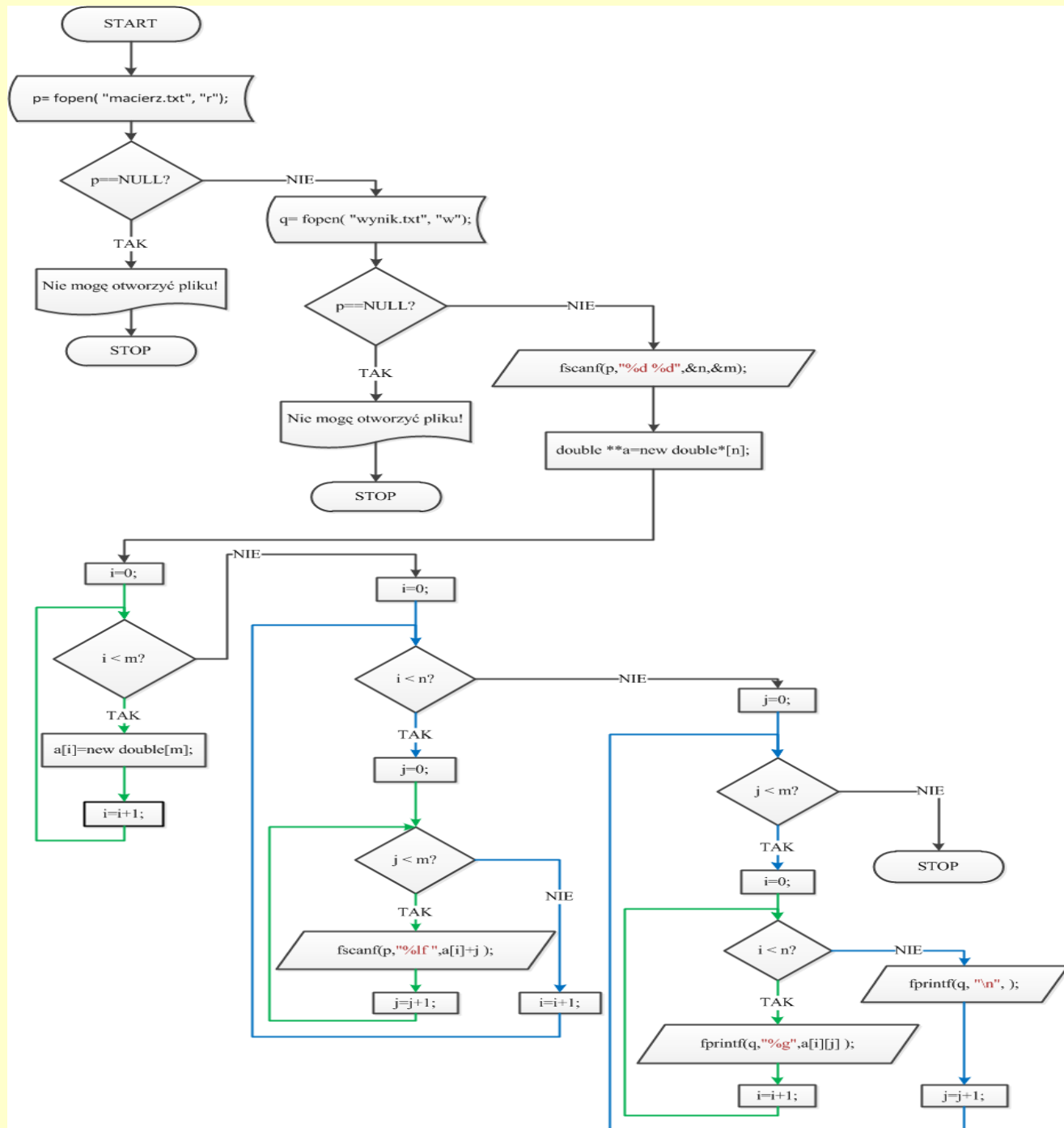
// Zamknięcie plików.

Dla obliczeń została przyjęta pokazana macierz  $A$ , natomiast program wyznaczył transpozycję  $A^T$ , co zostało pokazane na poniższej ilustracji



Realizacja takiej procedury w kodzie programu wymaga odpowiedniego połączenia ze sobą procedury pobierania danych z pliku z procedurą transponowania macierzy i zapisania jej w takiej postaci do pliku wynikowego. Czytelnik zechce prześledzić wykonanie tych operacji na poniższym kolejnym blokowym opisyującym przebieg całego programu.

# Schemat blokowy algorytmu transponow ania macierzy.





# Macierz odwrotna

## Twierdzenie

Jeśli wyznacznik macierzy  $|A| \neq 0$ , to istnieje macierz odwrotna  $A^{-1}$  do macierzy  $A$  taka, że

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

Do wyznaczenia macierzy odwrotnej i dowodu twierdzenia należy rozwiązać równanie macierzowe postaci:

$$AX = I, \quad \text{gdzie } A, X, I \in R^{n \times n}$$

Nietrudno zauważyć, że znalezienie macierzy  $X$  sprowadza się do rozwiązywania  $n$  układów równań liniowych postaci:

$$Ax_j = i_j \quad \text{dla } j = 1, \dots, n$$

Wektor kolumnowy  $x_j$  jest  $j$  – tą kolumną macierzy  $X$ , a wektor  $i_j$  jest  $j$  – tą kolumną macierzy jednostkowe  $I$ . W celu rozwiązania  $n$  równań liniowych jednocześnie za pomocą eliminacji Gaussa z wyborem elementu głównego wygodnie jest dokonać zapisu samej macierzy  $A$  i wektorów prawych stron w macierzy stopnia  $R^{n \times 2n}$  w następujący sposób:

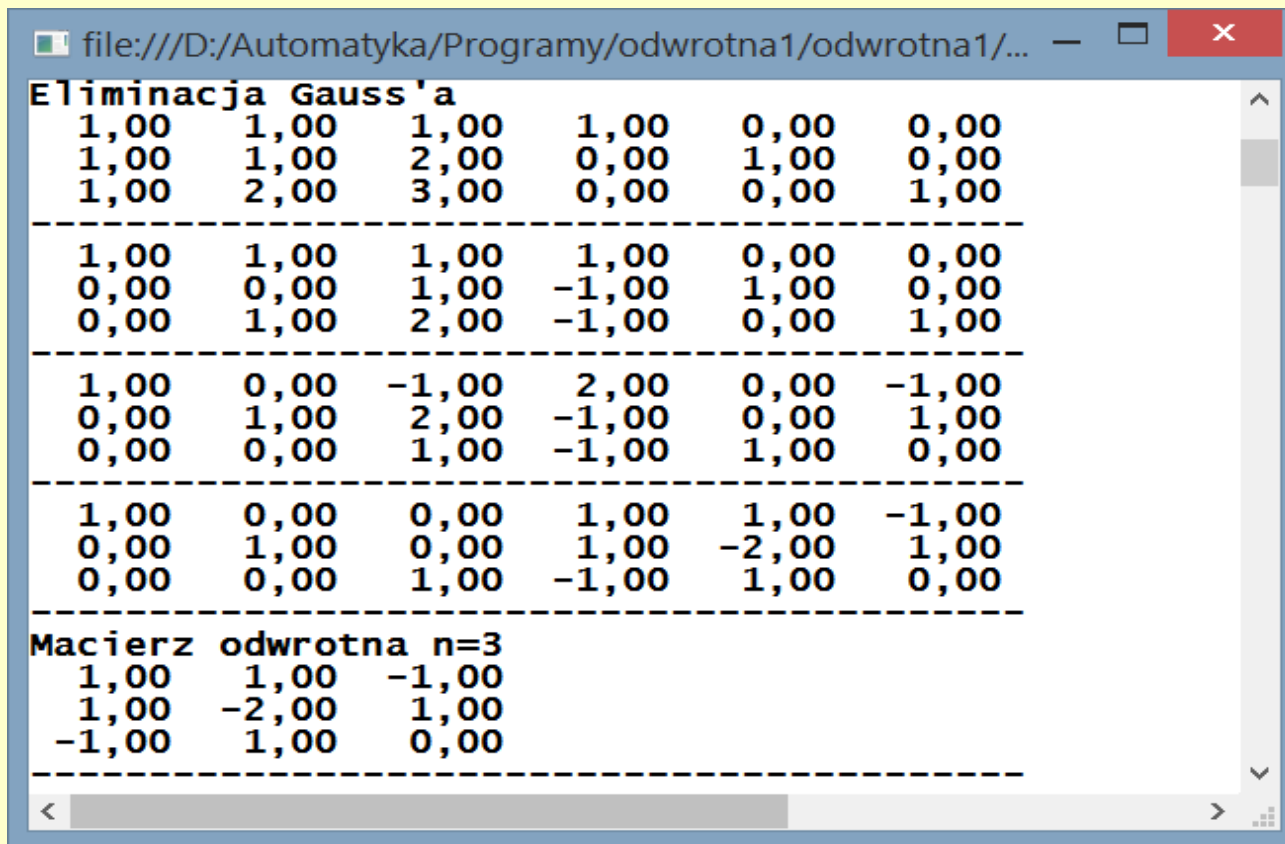
$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & a_{2n} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Za pomocą eliminacji niewiadomych pod i nad górną przekątną i dzieląc kolejne wiersze przez elementy znajdujące się na głównej przekątnej otrzymujemy rozwiązania  $n$  układów równań liniowych. W zapisie macierzowym wygląda to następująco:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 & i_{11}^n & i_{12}^n & \cdots & i_{n1}^n \\ 0 & 1 & \cdots & 0 & i_{21}^n & i_{22}^n & \cdots & i_{2n}^n \\ \vdots & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & i_{n1}^n & i_{n2}^n & \cdots & i_{nn}^n \end{bmatrix}$$

W przykładzie pokazano znalezienie macierzy odwrotnej do macierzy:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$



file:///D:/Automatyka/Programy/odwrotna1/odwrotna1/...

**Eliminacja Gauss'a**

1,00	1,00	1,00	1,00	0,00	0,00
1,00	1,00	2,00	0,00	1,00	0,00
1,00	2,00	3,00	0,00	0,00	1,00

---

1,00	1,00	1,00	1,00	0,00	0,00
0,00	0,00	1,00	-1,00	1,00	0,00
0,00	1,00	2,00	-1,00	0,00	1,00

---

1,00	0,00	-1,00	2,00	0,00	-1,00
0,00	1,00	2,00	-1,00	0,00	1,00
0,00	0,00	1,00	-1,00	1,00	0,00

---

1,00	0,00	0,00	1,00	1,00	-1,00
0,00	1,00	0,00	1,00	-2,00	1,00
0,00	0,00	1,00	-1,00	1,00	0,00

---

**Macierz odwrotna n=3**

1,00	1,00	-1,00
1,00	-2,00	1,00
-1,00	1,00	0,00

# Algorytm Gaussa – Jordana znajdowania macierzy odwrotnej

Za pomocą metody Gaussa – Jordana można obliczyć macierz odwrotną  $A^{-1}$  do macierzy  $A$ . Metoda ta jest bardziej zwarta algorytmicznie i nie wymaga dodatkowych zasobów pamięciowych. Algorytm Gaussa – Jordana dokonuje odwrócenia macierzy w deklarowanej macierzy  $A$ . Przedstawmy najpierw algorytm obliczania macierzy odwrotnej dla  $A \in R^{2 \times 2}$ , który polega na znalezieniu odwzorowania odwracającego układ równań liniowych postaci:

$$x \rightarrow Ax = y, \text{ gdzie } A \in R^{2 \times 2} \quad x, y \in R^2$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = y_1 \\ a_{21}x_1 + a_{22}x_2 = y_2 \end{cases}$$

W pierwszy kroku dokonujemy zamiany zmiennej  $x_1$  ze zmienną  $y_1$  w następujący sposób:

$$-y_1 + a_{12}x_2 = -a_{11}x_1$$

$$\frac{1}{a_{11}}y_1 - \frac{a_{12}}{a_{11}}x_2 = x_1$$

$$\left\{ \begin{array}{l} \frac{1}{a_{11}}y_1 - \frac{a_{12}}{a_{11}}x_2 = x_1 \\ a_{21} \left( \frac{1}{a_{11}}y_1 - \frac{a_{12}}{a_{11}}x_2 \right) + a_{22}x_2 = y_2 \end{array} \right.$$

$$\left\{ \begin{array}{l} \frac{1}{a_{11}}y_1 - \frac{a_{12}}{a_{11}}x_2 = x_1 \\ \frac{a_{21}}{a_{11}}y_1 + \left( a_{22} - a_{21} \frac{a_{12}}{a_{11}} \right) x_2 = y_2 \end{array} \right.$$

Oznaczmy nasz układ równań następująco:

$$\begin{cases} a_{11}^1 y_1 + a_{12}^1 x_2 = x_1 \\ a_{21}^1 y_1 + a_{22}^1 x_2 = y_2 \end{cases}$$

$$a_{21}^1 y_1 - y_2 = -a_{22}^1 x_2$$

$$-\frac{a_{21}^1}{a_{22}^1} y_1 + \frac{1}{a_{22}^1} y_2 = x_2$$

$$\begin{cases} a_{11}^1 y_1 + a_{12}^1 \left( -\frac{a_{21}^1}{a_{22}^1} y_1 + \frac{1}{a_{22}^1} y_2 \right) = x_1 \\ -\frac{a_{21}^1}{a_{22}^1} y_1 + \frac{1}{a_{22}^1} y_2 = x_2 \end{cases}$$

$$\begin{cases} \left( a_{11}^1 - a_{12}^1 \frac{a_{21}^1}{a_{22}^1} \right) y_1 + \frac{a_{12}^1}{a_{22}^1} y_2 = x_1 \\ -\frac{a_{21}^1}{a_{22}^1} y_1 + \frac{1}{a_{22}^1} y_2 = x_2 \end{cases}$$

$$\begin{cases} a_{11}^2 y_1 + a_{12}^2 y_2 = x_1 \\ a_{21}^2 y_1 + a_{22}^2 y_2 = x_2 \end{cases}$$

W wyniku otrzymujemy układ równań, której współczynniki są elementami macierzy odwrotnej  $A^{-1}$ .

Kolejne kroki algorytmu możemy przedstawić za pomocą macierzy otrzymywanych podczas przekształceń:

$$A = A^0 \rightarrow A^1 \rightarrow A^2 = A^{-1}$$



Algorytm Gaussa – Jordana dla obliczania macierzy odwrotnej znajduje odwzorowanie odwracające układ równań liniowych postaci:

$$x \rightarrow Ax = y, \text{ gdzie } A \in R^{n \times n} \quad x, y \in R^n$$
$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = y_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = y_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = y_n \end{array} \right.$$

W kroku pierwszym dokonujemy zamiany zmiennej  $x_1$  ze zmienną  $y_t$ . Indeks  $t$  jest obliczany ze wzoru:

$$|a_{t1}| = \max_i |a_{i1}| \quad \text{dla } i = 1, \dots, n$$

Następnie zamieniamy równanie  $t$  z równaniem pierwszym i otrzymujemy układ równań liniowych:

$$\begin{cases} \bar{a}_{11}x_1 + \bar{a}_{12}x_2 + \dots + \bar{a}_{1n}x_n = \bar{y}_1 \\ \bar{a}_{21}x_1 + \bar{a}_{22}x_2 + \dots + \bar{a}_{2n}x_n = \bar{y}_2 \\ \dots \\ \bar{a}_{n1}x_1 + \bar{a}_{n2}x_2 + \dots + \bar{a}_{nn}x_n = \bar{y}_n \end{cases}$$

Przy czym zmienne  $\bar{y}_1, \dots, \bar{y}_n$  są permutacją zmiennych  $y_0, \dots, y_n$  tak samo jak odpowiadające tym zmiennym wiersze macierzy  $A$ . Element  $\bar{a}_{11} \neq 0$ , gdyż jeżeli byłby równy zero to macierz byłaby osobliwa i należałoby przerwać obliczenia macierzy odwrotnej. Rozwiązujemy powyższe równanie względem zmiennej  $x_1$  i wstawiamy wynik do pozostałych równań:

$$\begin{cases} a_{11}^1 \bar{y}_1 + a_{12}^1 x_2 + \dots + a_{1n}^1 x_n = x_1 \\ a_{21}^1 \bar{y}_1 + a_{22}^1 x_2 + \dots + a_{2n}^1 x_n = \bar{y}_2 \\ \dots \\ a_{n1}^1 \bar{y}_1 + a_{n2}^1 x_2 + \dots + a_{nn}^1 x_n = \bar{y}_n \end{cases}$$

gdzie:

$$a_{11}^1 = \frac{1}{\bar{a}_{11}}, \quad a_{1j}^1 = -\frac{\bar{a}_{1j}}{\bar{a}_{11}}, \quad a_{i1}^1 = \frac{\bar{a}_{i1}}{\bar{a}_{11}}$$

$$a_{ij}^1 = \bar{a}_{ij} - \frac{\bar{a}_{i1} \bar{a}_{1j}}{\bar{a}_{11}} \quad i, j = 2, \dots, n$$

W następnych krokach zamieniamy kolejne zmienne  $x$  ze zmiennymi  $y$ . Kolejne kroki algorytmu możemy zapisać jako przekształcenia dokonywane na całej macierzy i oznaczyć indeksami:

$$A = A^0 \rightarrow A^1 \rightarrow \dots \rightarrow A^n = A^{-1}$$

Macierz  $A^k = [a_{ij}^k]$  odpowiada następującym układowi równań:

$$\begin{cases} a_{11}^k \bar{y}_1 + \cdots + a_{1k}^k x_k + \cdots + a_{1n}^k x_n = x_1 \\ \quad \quad \quad \cdots \\ a_{k1}^k \bar{y}_1 + \cdots + a_{kk}^k x_k + \cdots + a_{kn}^k x_n = \bar{y}_k \\ \quad \quad \quad \cdots \\ a_{n1}^k \bar{y}_1 + \cdots + a_{nk}^k x_k + \cdots + a_{nn}^k x_n = \bar{y}_n \end{cases}$$

Przy przejściu  $A^k \rightarrow A^{k+1}$  zmienna  $x_k$  zostanie zamieniona ze zmienną  $\bar{y}_t$  za pomocą następującego algorytmu:

➤ Wybór elementu podstawowego w kolumnie:

$$|a_{tk}| = \max_{i \geq k} |a_{ik}| \quad \text{dla } i = k, \dots, n$$

Jeśli  $a_{tk} = 0$  to macierz jest osobliwa  
(przerwanie algorytmu).

- Zamieniamy  $k$  – ty wiersz macierzy  $A^k$  z  $t$  – tym wierszem i wynik oznaczamy  $\bar{A} = [\bar{a}_{ij}]$ .
- Obliczamy macierz  $A^{k+1}$  wg następujących podstawień:

$$a_{kk}^k = \frac{1}{\bar{a}_{kk}}, \quad a_{kj}^k = -\frac{\bar{a}_{kj}}{\bar{a}_{kk}}, \quad a_{ik}^k = \frac{\bar{a}_{ik}}{\bar{a}_{kk}}$$

$$a_{ij}^k = \bar{a}_{ij} - \frac{\bar{a}_{ik}\bar{a}_{kj}}{\bar{a}_{kk}} \quad i, j = 1, \dots, n, \quad i \neq k, \quad j \neq k$$

Ponieważ  $\bar{y}_1, \dots, \bar{y}_n$  są permutacją zmiennych  $y_0, \dots, y_n$  które można określić za pomocą macierzy  $P$  dokonywanych zmian wierszy w kolejnych krokach odwracania macierzy  $A$ . Co możemy zapisać:

$$(A^n P)y = x$$

Stąd

$$A^{-1} = A^n P$$

Przykład odwracania macierzy:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 3 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & -1 & -1 \\ 1 & 1 & 2 \\ 1 & 2 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & -1 & 1 \\ -1 & 1 & -2 \\ -1 & 2 & -2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1,5 & 0 & -0,5 \\ 0 & -1 & 1 \\ -0,5 & 1 & -0,5 \end{bmatrix}$$

```
#include<stdio.h>
```

```
#include<math.h>
```

```
#include<conio.h>
```

```
int odw( int n, double **c)
```

```
{
```

```
    double f, h, s;
```

```
    int i, j, k, t, t1;
```

```
    double *w=new double[n];
```

```
    int *p=new int[n];
```

```
    for(j = 0;j < n; j++)
```

```
        p[j] = j;
```

```
    for(k = 0; k < n; k++)
```

```
    {
```

```
        h = fabs(c[k][k]);
```

```
        t = k;
```

```
        for(j = k + 1; j < n; j++)
```

```
            if(fabs(c[j][k]) > h)
```

```
                { h = fabs(c[j][k]);
```

```
                  t = j; }
```

```
        if(h < 1e-14) return k;
```

```
        if(t>k)
```

```
        {
```

```
            for(j=0;j<n;j++)
```

```
            { s=c[k][j];
```

```
//Deklaracja bibliotek.
```

```
/* Konstrukcja wektora do przestawienie kolumn  
macierzy odwrotnej. */
```

```
/* Konstrukcja i nadanie wartości początkowych  
wektora permutacji, */
```

```
/* Kolejne kroki tworzenia macierzy  
odwrotnej.*/
```

```
// Wybór elementu głównego.
```

```
// Przerwanie obliczeń na kroku k.
```

```
/* Jeśli konieczne, przestawienie wierszy  
macierzy i wektora permutacji. */
```

```

    c[k][j]=c[t][j];
    c[t][j]=s;
}
l=p[k]; p[k]=p[t]; p[t]=l;
}
h=c[k][k];
for(j = 0; j < n; j++)
    if(k != j) c[k][j] = -c[k][j]/h;
    c[k][k]=1.0/h;
for(i = 0; i < n; i++)
    if(k!=i)
    {
        f = c[i][k];
        c[i][k] = f/h;
        for(j = 0; j < n; j++)
            if(k != j) c[i][j] += f*c[k][j];
    }
}
for(i=0;i<n;i++)
{ for( k=0; k<n; k++) w[p[k]]=c[i][k];
  for(k=0; k<n; k++)c[i][k]=w[k];}
return n;
}

```

/\* Jeśli konieczne, przestawienie wierszy macierzy i wektora permutacji. \*/

// Algorytm odwracania macierzy,

// Permutacja macierzy odwrotnej.



```
int main()
{
    int n, i, j, t;
    double s;
    FILE *f;
    if((f=fopen("we.txt","r"))==NULL)
    {
        printf("Nie moge otworzyc pliku we.txt\n");
        _getch();
        return 0;
    }
    fscanf(f, "%d", &n);
    double **a = new double *[n];
    for(i=0;i<n;i++)
        a[i]= new double[ n ];
    printf("Obliczenia prowadzone sa dla n=%d\n",n);
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {   fscanf(f,"%lf",a[i]+j);
            printf("%g ",a[i][j]);
        }
        printf("\n");
    }
}
```

// Funkcja główna programu.

// Otwarcie pliku wejściowego.

// Czytanie wymiaru zadania.

// Konstrukcja macierzy.

// Wydruk przeczytanej macierzy.

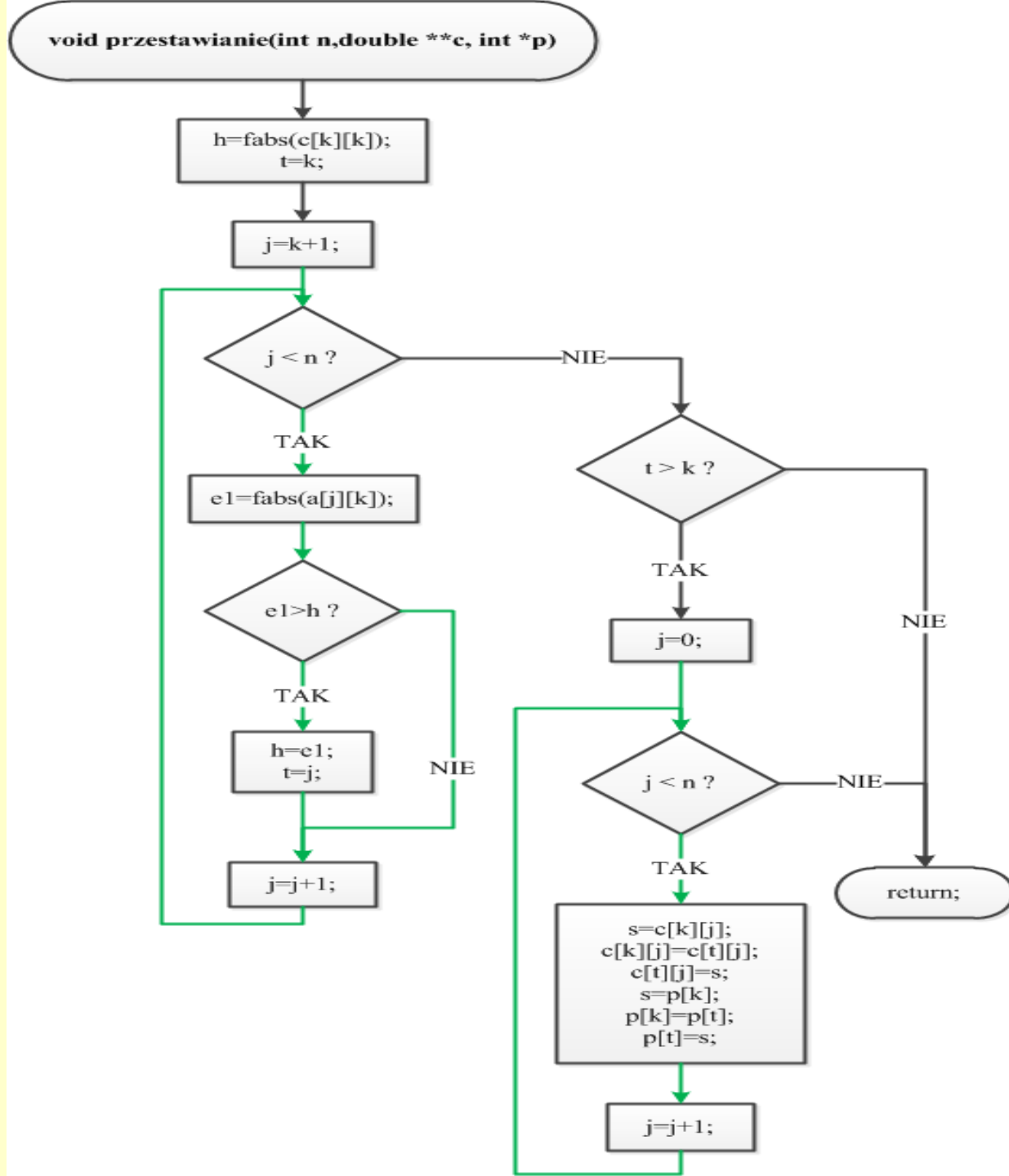
```
printf("\n");
t = odw(n,a);
if(t == n)
{
    printf("Macierz odwrotna:\n");
    for(i = 0; i < n; i++)
    {
        for( j = 0; j < n; j++)
        {
            if(fabs((s = a[i][j])) < 1e-14) s = 0;
            printf("%g ", s);
        }
        printf("\n");
    }
}
else printf("Macierz osobliwa\n");

_getch();
}
```

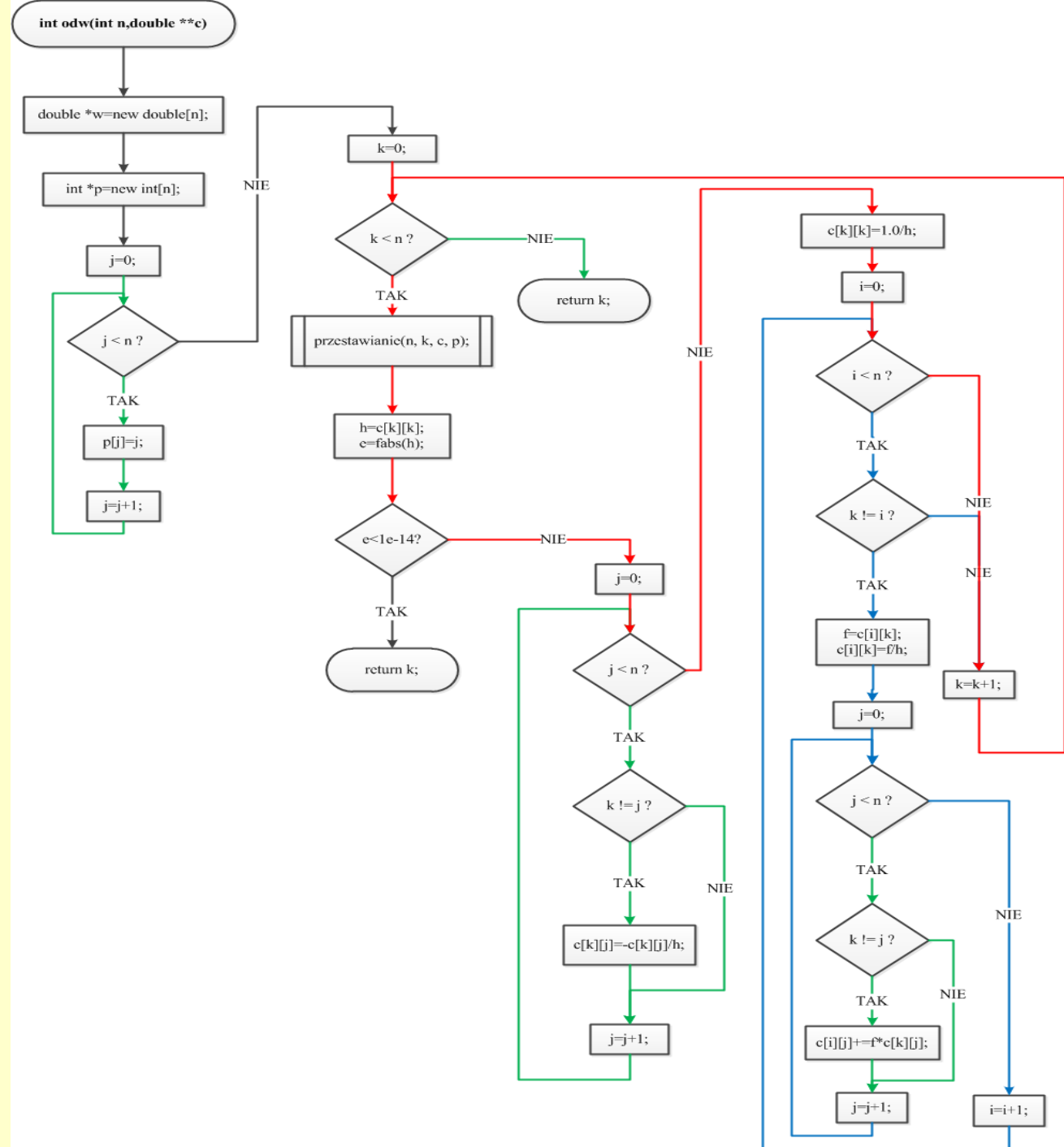
/\* Wywołanie funkcji odwracania macierz.  
Funkcja zwraca wymiar odwróconej macierzy  
albo krok, na którym nastąpiło przerwanie. \*/

/\* Obcięcie błędu zaokrąglenia przy zerze.\*/  
// Wydruk macierzy odwrotnej.

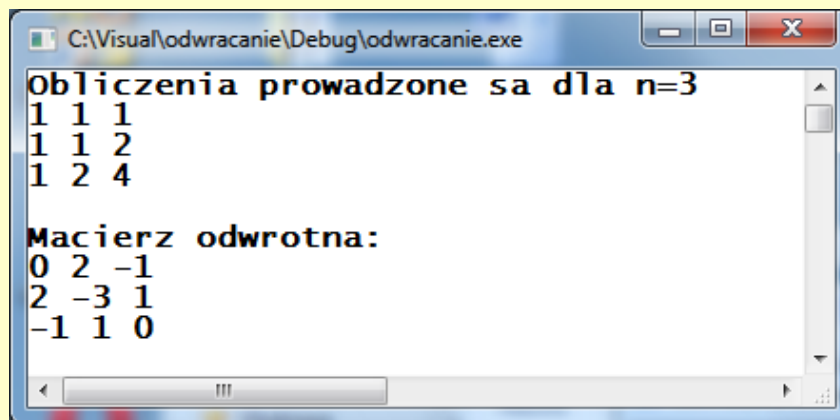
Schemat  
blokowy funkcji  
przestawienia  
elementów  
macierzy w  
trakcie  
realizacji  
algorytmu  
odwracania  
macierzy  
Gaussa-  
Jordana



# Schemat blokowy algorytmu odwracania macierzy Gaussa- Jordana



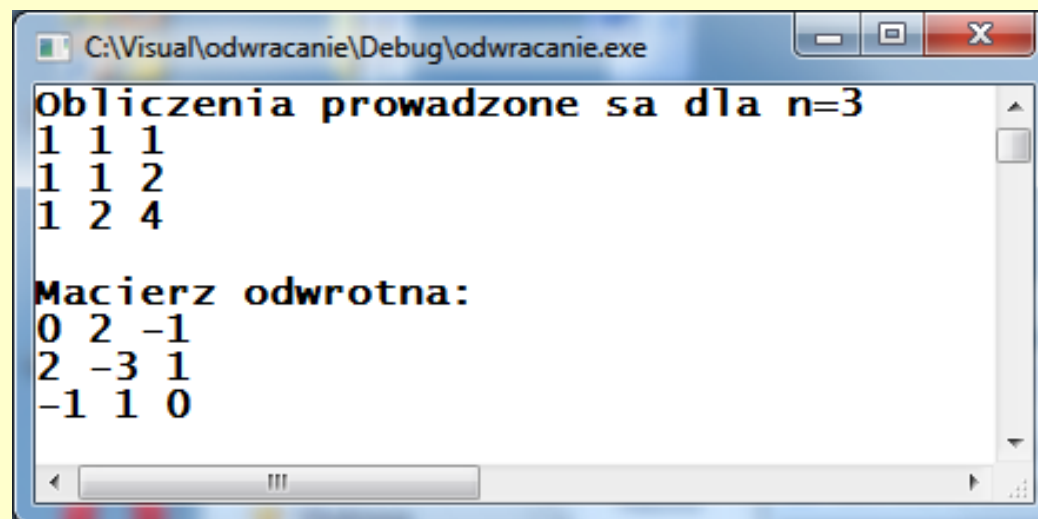
Przeprowadźmy test działania powyższego programu dla różnych macierzy. Dla macierzy, dla której algorytm nie dokonuje przestawień wierszy



```
C:\Visual\odwracanie\Debug\odwracanie.exe
Obliczenia prowadzone sa dla n=3
1 1 1
1 1 2
1 2 4

Macierz odwrotna:
0 2 -1
2 -3 1
-1 1 0
```

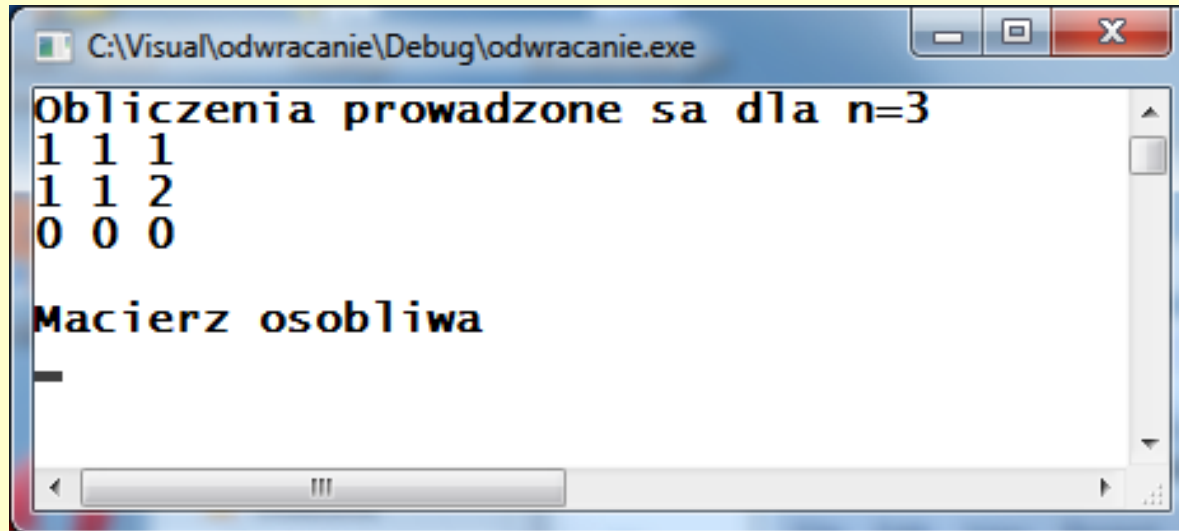
Dla macierzy, dla której jest konieczne przestawienie wierszy.



```
C:\Visual\odwracanie\Debug\odwracanie.exe
Obliczenia prowadzone sa dla n=3
1 1 1
1 1 2
1 2 4

Macierz odwrotna:
0 2 -1
2 -3 1
-1 1 0
```

# Macierzy osobliwej



```
C:\Visual\odwracanie\Debug\odwracanie.exe
Obliczenia prowadzone sa dla n=3
1 1 1
1 1 2
0 0 0

Macierz osobliwa
```

Z przeprowadzonych testów wynika, że program działa poprawnie.