



POLITECHNIKA RZESZOWSKA  
im. Ignacego Łukasiewicza  
WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

**Filip Walkowicz**  
Grupa 8

**Projekt inżynierski**

Opiekun pracy:  
dr inż. Mariusz Borkowski, prof. PRz

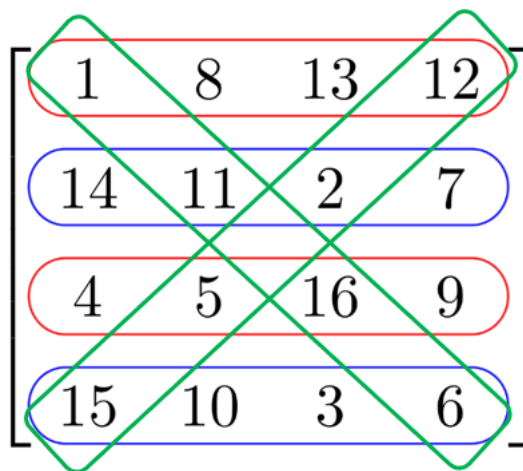
Rzeszów, 2021

# 1. Wstęp

W niniejszym sprawozdaniu z zadania projektowego wykonałem analizę wręzonego tematu. Przygotowałem schemat blokowy, pseudokod, program oraz analizę go pod kątem złożoności czasowej, obliczeniowej. Kod programu został napisany w środowisku Code::Blocks za pomocą języka C++.

## 1.1. Opis problemu oraz szczegóły implementacji

Tematem zadania: Wypisz indeksy tych wierszy macierzy kwadratowej, dla których suma elementów stojących na przekątnej i antyprzekątnej (w danym wierszu) jest większa od sumy pozostałych elementów stojących w tym wierszu.



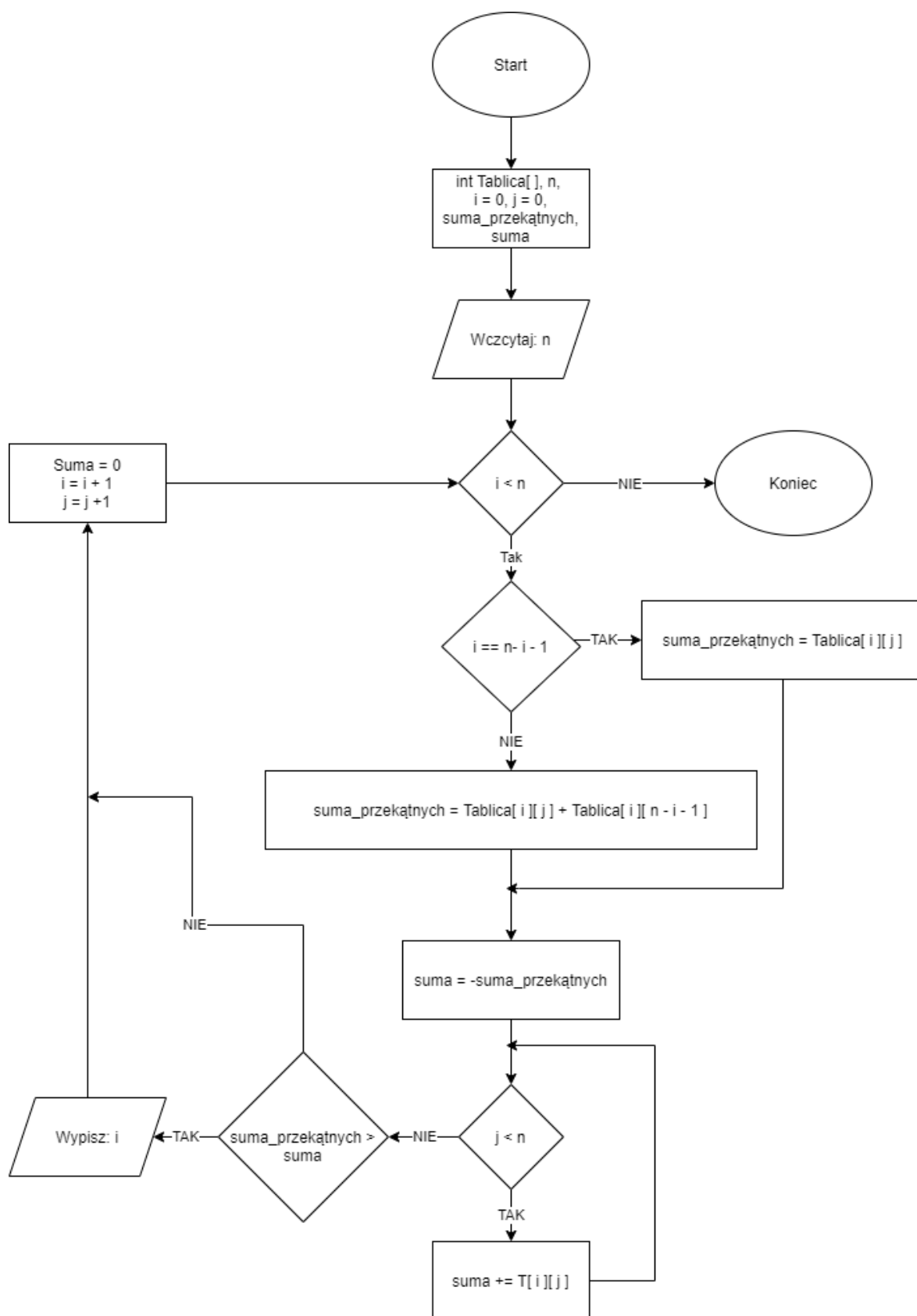
1	8	13	12
14	11	2	7
4	5	16	9
15	10	3	6

Rys 1.1. Macierz kwadratowa wraz z zaznaczonymi wierszami i przekątnymi

Z każdego wiersza musimy osobno zsumować przekątną z antyprzekątną oraz pozostałe liczby które zostały w wierszu ze sobą, następnie porównać ze sobą ich sumy a następnie wypisać indeksy wierszy w których suma przekątnych jest większa od pozostałej sumy liczb. Żeby rozwiązać problem w języku programowania będzie potrzebna tablica dwuwymiarowa która będzie reprezentowała naszą macierz, dwie zmienne w których przechowamy sumy do porównania. Za pomocą dwóch pętli for będziemy przechodzić przez dane wiersze w macierzy co pozwoli nam zapisywać zmiennych odpowiednie dane oraz za pomocą funkcji if wypisywać te wiersze które spełniają warunki. Należy uważać żeby w tablicy nie zliczać podwójnie miejsca w którym przekątne się ze sobą przecinają.

## 2. Schemat blokowy oraz zapis w pseudokodzie

### 2.1. Schemat blokowy



Rys. 2.1. Schemat blokowy algorytmu

## 2.1. Zapis problemu w pseudokodzie

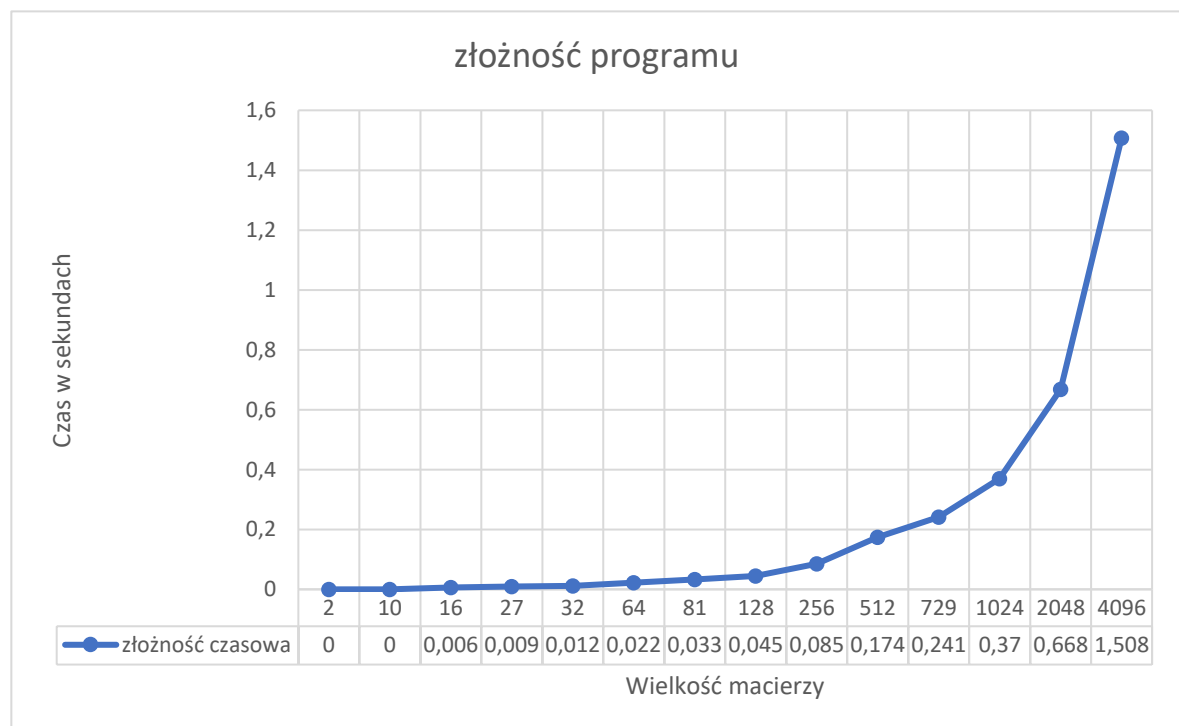
1. For  $i = 0$  to  $i < \text{wielkość\_macierzy}$
2.     If  $i == \text{wielkość\_macierzy} - i - 1$
3.          $\text{suma\_przekątnych} = T[i]$
4.     Else
5.          $\text{suma} = \text{suma\_przekątnych} = T[i] + T[\text{wielkość\_macierzy} - i - 1]$
6.      $\text{suma} = -\text{suma\_przekątnych}$
7.     For  $j = 0$  to  $j < \text{wielkość\_macierzy}$
8.          $\text{suma} = \text{suma} + T[i][j]$
9.      $\text{suma} = 0$

### 3. Testy programu

#### 3.1. Testy czasowe oraz wykres

Tabel 3.1 Złożoność czasowa

Wielkość macierzy	Czas w sekundach
<b>2</b>	0
<b>10</b>	0
<b>16</b>	0,006
<b>27</b>	0,009
<b>32</b>	0,012
<b>64</b>	0,022
<b>81</b>	0,033
<b>128</b>	0,045
<b>256</b>	0,085
<b>512</b>	0,174
<b>729</b>	0,241
<b>1024</b>	0,37
<b>2048</b>	0,668
<b>4096</b>	1,508



Rys 3.1 Wykres złożoności

Po przeprowadzeniu testów złożoności czasowej oraz po analizie kodu można wywnioskować, że algorytm ma złożoność  $O(n^2)$ . Wraz ze zwiększającą się macierzą wykres układa się w parabolę.

### 3.2. Testy programu

1) Macierz o rozmiarach 5x5:

```
4 2 8 8 2
9 5 8 9 8
9 7 9 8 0
9 0 5 8 6
8 1 2 2 6
4
```

Problem z macierzami nieparzystymi mógł polegać na tym że w miejscu przecięcia się przekątnych program mógł zliczać podwójnie wartość. Na przykładzie widać że algorytm jest na to odporny.

2) Macierz o wymiarach 20x20: 9 5 4 5 0 9 2 6 3 2 9 2 0 3 5

```
9 2 9 2 2 5 8 2 5 1 1 5 3 1 1
3 9 5 5 8 0 7 1 9 2 5 6 4 5 2
4 6 0 2 7 8 7 5 6 0 6 5 1 3 9
6 5 2 3 2 1 9 0 5 1 5 5 7 5 8
3 6 3 7 9 2 7 8 2 3 8 3 2 8 3
3 4 1 5 1 8 4 1 8 1 3 4 9 0 2
6 6 4 2 1 5 0 3 8 1 1 7 0 6 7
1 3 5 0 5 0 9 6 0 9 6 6 2 1 1
7 4 5 5 2 3 9 2 5 9 8 8 4 6 3
8 3 8 1 7 7 0 2 2 1 9 4 7 1 2
7 6 6 5 9 7 4 8 4 6 0 0 7 2 7
0 3 0 8 7 1 9 0 6 4 9 5 4 0 3
8 3 6 3 7 5 1 7 4 1 3 6 9 8 0
1 0 1 8 2 0 3 4 4 9 8 8 7 2 7
```

3) Macierz o rozmiarach 3x3:

3 7 9

8 6 0

8 2 9

0

2

#### **4. Wnioski oraz podsumowanie**

Realizowany przez mnie temat dotyczył operacji na tablicy dwuwymiarowej/ macierzy którego celem było sumowanie ze sobą przekątnej oraz antyprzekątnej, zsumowanie pozostałych elementów w wierszu macierzy a następnie porównanie ich ze sobą. W teoretycznej części pracy omówiono problem rozwiązania teoretycznego oraz możliwych komplikacji, schemat blokowy oraz pseudokod pokazał w jaki sposób należy wykonać poszczególne kroki aby poprawnie wykonać problem. Za pomocą wykonanego programu postawione potwierdziły początkowo postawione założenia. Przeprowadzone testy oraz analiza kodu udowodniły złożoność programu na  $O(n^2)$ .

```

#include <iostream>
#include <time.h>
#include <cstdlib>
#include <bits/stdc++.h>

using namespace std;

// złożoność to  $n^2$ 
void WypelnijMacierz(int *T[],int a,int x, int y)
{
    //Wypelnienie macierzy liczbami
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<a;j++)
        {
            //wygenerowanie liczb z zakresu [-9; 9]
            T[i][j]= x+rand() % (y-x+1);
            //wyswietlenie wylosowanej liczby
            cout<<T[i][j]<<" ";
        }
        cout << endl;
    }
}

```



```

void Wypisz(int *T[], int a)
{
    int suma = 0;
    int suma_przekatnych = 0;
    //przekatka macierzy
        for(int i=0; i<a; i++)
        {
            // sprawdzam zeby 2 razy nie brac przekatnych
            if(i == a-i-1)
            {
                // sumuje przekatne
                suma_przekatnych = T[i][i];
            }else{
                suma_przekatnych = T[i][i] + T[i][a-i-1];
            }

            // sumuje liczby w wierszu pomijajac przekatne
            suma = -suma_przekatnych;
            for(int j=0; j<a; j++)
            {
                suma += T[i][j];
            }

            // porownuje sumy i wyswietlam indeks
            if(suma_przekatnych > suma){
                cout << i << endl;
            }
            // zeruje sume
            suma = 0;
        }
    }
}

```

```

int main()
{
    srand((unsigned)time (NULL));

    //zmienne
    int **T, n, a, b;

    //przedzial liczb z macierzy
    a=0;
    b=9;

    cout << "podaj wielkosc macierzy kwadratowej(nxn): ";
    cin >> n;

    //Przydzielanie pamieci na tablice dwuwymiarowe(macierz)
    T = new int *[n];
    for(int i=0;i<n;i++)
        T[i] = new int[n];

    WypelnijMacierz(T,n,a,b);
    // obliczanie czasu trwania programu
    auto begin = std::chrono::high_resolution_clock::now();
    Wypisz(T,n);

    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end -
begin);
    printf("Time measured: %.3f seconds.\n", elapsed.count() * 1e-9);
    // zwolnienie pamieci
    for(int i=0;i<n;i++)
        delete [] T[i];
    delete []*T;
}

```