

WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI  
POLITECHNIKI RZESZOWSKIEJ

# Algorytmy i struktury danych.

**Jakub Wójcik 179992**

Zadanie 1 Inżynieria i analiza danych L8

## Spis treści

1. Temat zadania.....	2
2. Analiza problemu.....	2
2.1. Pseudokod:.....	2
2.2. Złożoność obliczeniowa algorytmu:.....	3
2.3. Algorytm:.....	3
2.4. Schemat blokowy.....	8
2.5. Przykładowe wyniki:.....	9
2.6. Analiza pomiarów czasowych .....	11
2.7. Ołówkowe sprawdzenie poprawności algorytmu. ....	11
3. Materiały przydatne do stworzenia sprawozdania .....	11

## 1. Temat zadania

Dla tablicy  $M \times N$  wypełnionej zerami lub jedynkami, znajdź pole największego prostokąta zbudowanego z jedynek.

Przykład:

**Wejście:**

[ 0, 1, 0, 0, 0 ]

[ 1, 1, 1, 0, 0 ]

[ 1, 1, 1, 1, 0 ]

[ 1, 1, 1, 1, 1 ]

[ 0, 1, 0, 1, 0 ]

**Wyjście:**

9 (Jedynki w wierszach 2-4, kolumnach 1-3)

## 2. Analiza problemu

### 2.1. Pseudokod:

FUNKCJA `znajdzNajwiekszyProstokat(macierz)`:

JEŚLI macierz jest pusta:

ZWRÓĆ 0

```

wiersze = liczba wierszy w macierzy
kolumny = liczba kolumn w macierzy
maxPole = 0
// Inicjalizacja DP tablicy do przechowywania szerokości jedynek
UTWÓRZ dp[wiersze][kolumny] wypełnioną zerami

DLA każdego wiersza i w kolumnie w macierzy:
JEŚLI macierz[i][j] == 1:
JEŚLI to pierwsza kolumna:
dp[i][j] = 1
W PRZECIWNYM PRZYPADKU:
dp[i][j] = dp[i][j-1] + 1
minSzerokosc = dp[i][j]
// Przejście wstecz w wierszach, aby znaleźć maksymalny prostokąt
DLA każdego k OD i DO 0:
    minSzerokosc = min(minSzerokosc, dp[k][j])
wysokosc = i - k + 1
pole = minSzerokosc * wysokosc
JEŚLI pole > maxPole:
    maxPole = pole
ZAPISZ współrzędne prostokąta (k, j - minSzerokosc + 1) do (i, j)
ZWRÓĆ maxPole oraz współrzędne prostokąta

```

## 2.2. Złożoność obliczeniowa algorytmu:

Iterujemy przez każdą komórkę macierzy w poszukiwaniu jedynek co daje nam złożoność obliczeniową na poziomie  $O(\text{wiersze} * \text{kolumny})$ . Musimy jednak jeszcze przeszukać wiersze znajdujące się nad przeszukiwanym obecnie wierszem więc złożoność obliczeniowa w najgorszym wypadku gdzie jedynki ciągnęłyby się do samej góry wynosiłaby  $O(\text{wiersze} * \text{wiersze} * \text{kolumny})$  gdzie każda z nich jest jakąś liczbą więc złożoność czytelnie jest na poziomie  $O(N^3)$ .

## 2.3. Algorytm:

```

#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <iomanip> // Do ustawiania precyzji
using namespace std;

// Funkcja do generowania losowej macierzy 0 i 1
vector<vector<int>> generujLosowaMacierz(int wiersze, int kolumny) {
    vector<vector<int>> macierz(wiersze, vector<int>(kolumny));
    srand(time(0));
    for (int i = 0; i < wiersze; i++) {
        for (int j = 0; j < kolumny; j++) {
            macierz[i][j] = rand() % 2; // Losuj 0 lub 1
        }
    }
}

```

```

    }
    return macierz;
}

// Funkcja do wyświetlania macierzy
void wypiszMacierz(const vector<vector<int>>& macierz) {
    for (const auto& wiersz : macierz) {
        for (int komorka : wiersz) {
            cout << komorka << " ";
        }
        cout << endl;
    }
}

// Funkcja obliczająca największe pole prostokąta z jedynek (dynamiczne programowanie)
int znajdzNajwiekszyProstokat(const vector<vector<int>>& macierz, int& najlepszy_w1, int&
najlepszy_k1, int& najlepszy_w2, int& najlepszy_k2) {
    if (macierz.empty() || macierz[0].empty()) return 0;

    int wiersze = macierz.size();
    int kolumny = macierz[0].size();
    int maxPole = 0;

    // DP tablica do przechowywania szerokości jedynek dla każdego punktu
    vector<vector<int>> dp(wiersze, vector<int>(kolumny, 0));

    for (int i = 0; i < wiersze; i++) {
        for (int j = 0; j < kolumny; j++) {
            if (macierz[i][j] == 1) {
                // Obliczamy szerokość jedynek w bieżącym wierszu
                dp[i][j] = (j == 0 ? 1 : dp[i][j - 1] + 1);

                // Minimalna szerokość w pionie (wysokość prostokąta)
                int minSzerokosc = dp[i][j];
                for (int k = i; k >= 0; k--) {
                    minSzerokosc = min(minSzerokosc, dp[k][j]);
                    int wysokosc = i - k + 1;
                    int pole = minSzerokosc * wysokosc;
                    if (pole > maxPole) {
                        maxPole = pole;
                        najlepszy_w1 = k;
                        najlepszy_k1 = j - minSzerokosc + 1;
                        najlepszy_w2 = i;
                        najlepszy_k2 = j;
                    }
                }
            }
        }
    }
}

```

```

    return maxPole;
}

// Funkcja do wypisywania wyników
void wypiszWynik(int maxPole, int najlepszy_w1, int najlepszy_k1, int najlepszy_w2, int
najlepszy_k2) {
    if(maxPole!=0){
        cout << "Najwieksze pole prostokata: " << maxPole << endl;

        // Wypisanie wierszy i kolumn tworzących największy prostokąt
        cout << "Jedynki tworzące największy prostokąt znajdują się w:" << endl;
        cout << "Wiersze: " << (najlepszy_w1 + 1) << "-" << (najlepszy_w2 + 1) << endl; // Zakres
wierszy
        cout << "Kolumny: " << (najlepszy_k1 + 1) << "-" << (najlepszy_k2 + 1) << endl; // Zakres
kolumn
    }
    else cout<<"Utworzona macierz nie posiada jedynek z których można obliczyć pole
prostokąta"<<endl;
}
int main() {
    int wiersze, kolumny;
    cout << "Podaj liczbę wierszy: ";
    cin >> wiersze;
    cout << "Podaj liczbę kolumn: ";
    cin >> kolumny;

    vector<vector<int>> macierz = generujLosowaMacierz(wiersze, kolumny);

    cout << "Wygenerowana macierz:" << endl;
    wypiszMacierz(macierz);

    int najlepszy_w1 = 0, najlepszy_k1 = 0, najlepszy_w2 = 0, najlepszy_k2 = 0;

    // Mierzenie czasu wykonania
    clock_t start = clock();

    int maxPole = znajdźNajwiększyProstokąt(macierz, najlepszy_w1, najlepszy_k1,
najlepszy_w2, najlepszy_k2);

    clock_t end = clock();

    wypiszWynik(maxPole, najlepszy_w1, najlepszy_k1, najlepszy_w2, najlepszy_k2);

    // Wypisanie czasu wykonania w milisekundach
    double czas_wykonania = (double)(end - start) / CLOCKS_PER_SEC * 1000.0; // Czas w
milisekundach
    cout << "Czas wykonania programu: " << fixed << setprecision(3) << czas_wykonania <<
" ms" << endl;
    return 0;}

```

```

1  #include <iostream>
2  #include <vector>
3  #include <cstdlib>
4  #include <ctime>
5  #include <iomanip> // Do ustawiania precyzji
6  using namespace std;
7  // Funkcja do generowania losowej macierzy 0 i 1
8  vector<vector<int>> generujLosowaMacierz(int wiersze, int kolumny) {
9      vector<vector<int>> macierz(wiersze, vector<int>(kolumny));
10     srand(time(0));
11     for (int i = 0; i < wiersze; i++) {
12         for (int j = 0; j < kolumny; j++) {
13             macierz[i][j] = rand() % 2; // Losuj 0 lub 1
14         }
15     }
16     return macierz;
17 }
18 // Funkcja do wyświetlania macierzy
19 void wypiszMacierz(const vector<vector<int>>& macierz) {
20     for (const auto& wiersz : macierz) {
21         for (int komorka : wiersz) {
22             cout << komorka << " ";
23         }
24         cout << endl;
25     }
26 }
27 // Funkcja obliczająca największe pole prostokata z jedynek (dynamiczne programowanie)
28 int znajdzNajwiekszyProstokat(const vector<vector<int>>& macierz, int& najlepszy_w1, int& najlepszy_k1, int& najlepszy_w2, int& najlepszy_k2) {
29     if (macierz.empty() || macierz[0].empty()) return 0;
30     int wiersze = macierz.size();
31     int kolumny = macierz[0].size();
32     int maxPole = 0;
33     // DP tablica do przechowywania szerokości jedynek dla każdego punktu
34     vector<vector<int>> dp(wiersze, vector<int>(kolumny, 0));
35
36     for (int i = 0; i < wiersze; i++) {
37         for (int j = 0; j < kolumny; j++) {
38             if (macierz[i][j] == 1) {
39                 // Obliczamy szerokość jedynek w bieżącym wierszu
40                 dp[i][j] = (j == 0 ? 1 : dp[i][j - 1] + 1);
41
42                 // Minimalna szerokość w pionie (wysokość prostokata)
43                 int minSzerokosc = dp[i][j];
44                 for (int k = i; k >= 0; k--) {
45                     minSzerokosc = min(minSzerokosc, dp[k][j]);
46                     int wysokosc = i - k + 1;
47                     int pole = minSzerokosc * wysokosc;
48                     if (pole > maxPole) {
49                         maxPole = pole;
50                         najlepszy_w1 = k;
51                         najlepszy_k1 = j - minSzerokosc + 1;
52                         najlepszy_w2 = i;
53                         najlepszy_k2 = j;
54                     }
55                 }
56             }
57         }
58     }
59 }

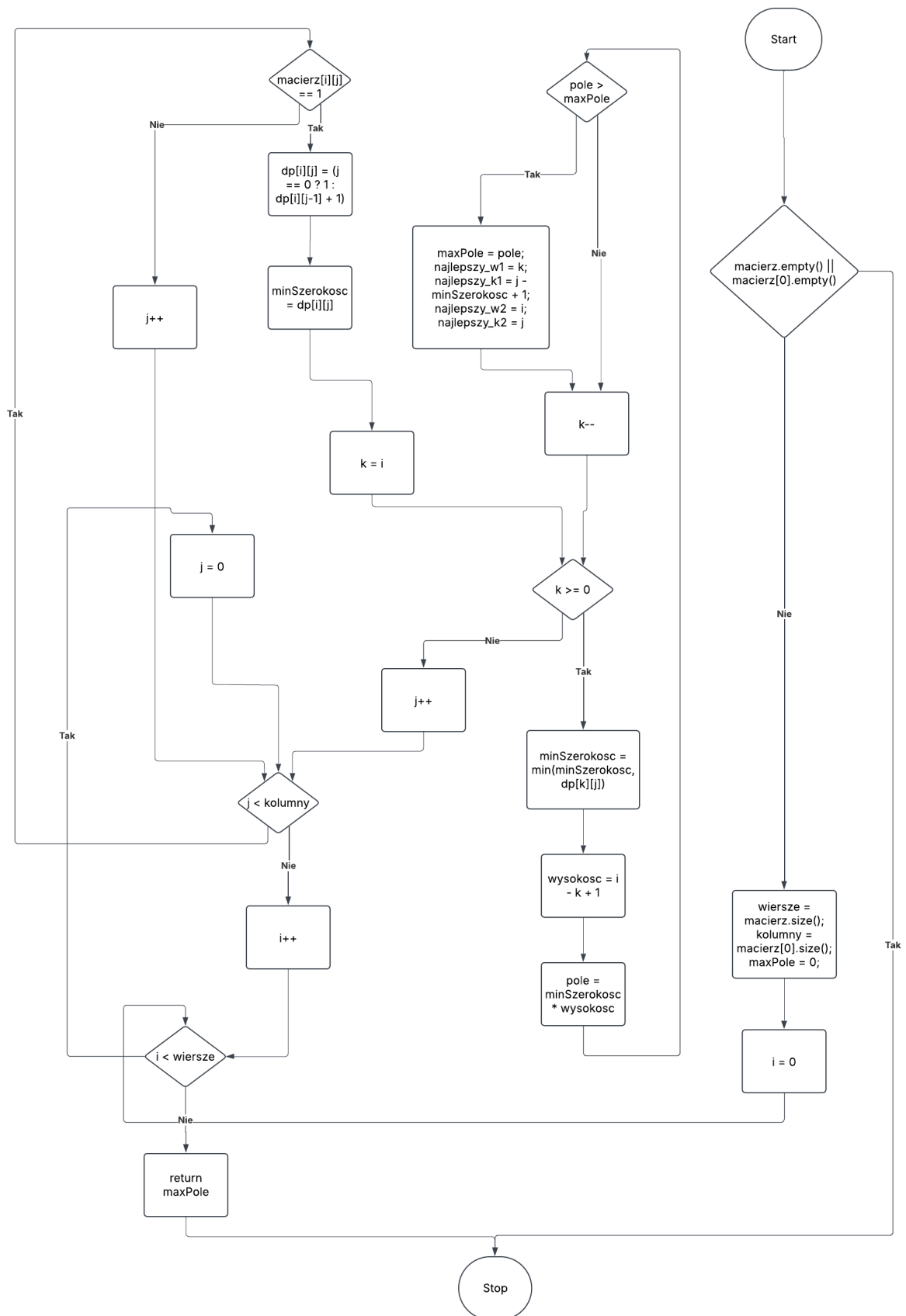
```

```

59     return maxPole;
60 }
61 // Funkcja do wypisywania wyników
62 void wypiszWynik(int maxPole, int najlepszy_w1, int najlepszy_k1, int najlepszy_w2, int najlepszy_k2) {
63     if(maxPole!=0){
64         cout << "Największe pole prostokata: " << maxPole << endl;
65
66         // Wypisanie wierszy i kolumn tworzących największy prostokąt
67         cout << "Jedynki tworzące największy prostokąt znajdują się w:" << endl;
68         cout << "Wiersze: " << (najlepszy_w1 + 1) << "-" << (najlepszy_w2 + 1) << endl; // Zakres wierszy
69         cout << "Kolumny: " << (najlepszy_k1 + 1) << "-" << (najlepszy_k2 + 1) << endl; // Zakres kolumn
70     }
71     else cout<<"Utworzona macierz nie posiada jedynek z których można obliczyć pole prostokata"<<endl;}
72
73 int main() {
74     int wiersze, kolumny;
75     cout << "Podaj liczbę wierszy: ";
76     cin >> wiersze;
77     cout << "Podaj liczbę kolumn: ";
78     cin >> kolumny;
79
80     vector<vector<int>> macierz = generujLosowaMacierz(wiersze, kolumny);
81
82     cout << "Wygenerowana macierz:" << endl;
83     wypiszMacierz(macierz);
84
85     int najlepszy_w1 = 0, najlepszy_k1 = 0, najlepszy_w2 = 0, najlepszy_k2 = 0;
86     // Mierzenie czasu wykonania
87     clock_t start = clock();
88
89     int maxPole = znajdźNajwiększyProstokąt(macierz, najlepszy_w1, najlepszy_k1, najlepszy_w2, najlepszy_k2);
90
91     clock_t end = clock();
92
93     wypiszWynik(maxPole, najlepszy_w1, najlepszy_k1, najlepszy_w2, najlepszy_k2);
94
95     // Wypisanie czasu wykonania w milisekundach
96     double czas_wykonania = (double)(end - start) / CLOCKS_PER_SEC * 1000.0; // Czas w milisekundach
97     cout << "Czas wykonania programu: " << fixed << setprecision(3) << czas_wykonania << " ms" << endl;
98
99     return 0;
100 }
101
102

```

## 2.4. Schemat blokowy





## 2.5. Przykładowe wyniki:

```
"C:\Users\Kuba\Desktop\Proj" x + v
Podaj liczbe wierszy: 3
Podaj liczbe kolumn: 3
Wygenerowana macierz:
1 1 0
0 1 0
0 1 1
Najwieksze pole prostokata: 3
Jedynki tworzące największy prostokąt znajdują się w:
Wiersze: 1-3
Kolumny: 2-2
Czas wykonania programu: 0.000 ms

Process returned 0 (0x0)   execution time : 0.880 s
Press any key to continue.

"C:\Users\Kuba\Desktop\Proj" x + v
Podaj liczbe wierszy: 5
Podaj liczbe kolumn: 5
Wygenerowana macierz:
1 1 0 0 1
1 1 0 0 0
1 0 1 0 0
1 0 0 1 0
1 1 0 0 1
Najwieksze pole prostokata: 5
Jedynki tworzące największy prostokąt znajdują się w:
Wiersze: 1-5
Kolumny: 1-1
Czas wykonania programu: 0.000 ms

Process returned 0 (0x0)   execution time : 2.820 s
Press any key to continue.
```

Macierz 50x50

```
"C:\Users\Kuba\Desktop\Proj" x + v
1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 1 1 0
1 0 1 1 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 1 0
0 1 1 1 0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 0 1 0 1 0 0 1 1 0 0 1 1 1 1 0
0 1 0 1 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1
1 0 1 1 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 0 0 0
0 1 0 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0
0 1 0 1 0 1 0 1 1 0 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0
0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 1 1
0 0 1 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 0 1 0 1 1 0 1 1 0 1 1
0 1 0 0 0 0 0 1 1 1 0 1 1 0 0 1 1 0 1 1 1 0 0 0 1 1 0 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0
0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 1 0 0 1 1 0 1 0 0 1 0
0 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1
1 0 1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 1 0 1 0 0 1
1 0 1 1 1 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1 1 1 1 0 0 0 0 1 0 1 0 0 1 0 1 1 0 1 0
0 1 0 1 1 0 0 1 1 1 1 1 0 0 1 0 0 0 0 0 1 1 0 1 1 0 1 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 0 0
1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0
0 1 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0
1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0
0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 1 0 1 1 0 0 0 1 0 1 0 1
1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 1
0 0 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1
Najwieksze pole prostokata: 16
Jedynki tworzące największy prostokąt znajdują się w:
Wiersze: 2-5
Kolumny: 10-13
Czas wykonania programu: 1.000 ms

Process returned 0 (0x0)   execution time : 2.948 s
Press any key to continue.
```

## Macierz 100x100

```
"C:\Users\Kuba\Desktop\Proj X" + - □ X
0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 1 1 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 1
1 0 1 1 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 1
0 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1 1 0 1
1 1 0 1 0 1 1 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 0 0 0 1 0 1
0 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 0 1 1 0 0 1 0 0 1 1 1 1
1 1 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 0 1 0
0 1 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0
1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0
1 1 1 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0
0 0 0 0 1 1 0 1 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0
0 0 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0
0 1 1 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 1
1 0 1 0 1 0 0 1 0 0 1 1 1 0 1 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 0
0 1 1 0 1 1 1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 1
0 0 1 0 0 1 1 1 0 0 0 1 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0
0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 1 1 1
0 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 0 0 1 0 0 1 0 0 0 1 0 1
1 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 0 1 0 1 0
0 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1
1 1 0 1 0 1 1 1 0 0 0 0 0 0 1 1 0 1 1 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0
1 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 1 1 0 1
Największe pole prostokąta: 16
Jedyńki tworzące największy prostokąt znajdują się w:
Wiersze: 76-83
Kolumny: 4-5
Czas wykonania programu: 3.000 ms

Process returned 0 (0x0)   execution time : 4.766 s
Press any key to continue.
```

## Macierz 1000x800

```
"C:\Users\Kuba\Desktop\Proj X" + - □ X
0 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 0 1 1 1
0 0 0 0 0 1 1 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 0
1 0 1 0 1 0 1 0 0 0 1 1 0 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 1 1 1 0 0 1 0 0
0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 1 0 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1
0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 1 0 1 1 0 1 1 0 1 0
1 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0 1 0 0
1 0 0 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 1
0 0 0 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 1 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 1 0 0 1
1 0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 0 1 0 1 1 0 0 0 1 1 0 0 1 0 1 0 0
0 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1
0 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 0 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1
0 0 1 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 0 0 0 1 0 0 1 0 1 0
0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 0 0 0 1 0 0 0 1 0 0 1
0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 1 1 1 1 0 0
1 0 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 1 0 1 0 0
1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 1 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1 0 1 1 0
0 0 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 1 1 1 0 0 1 0 1 0 0 1 1 1 0 1
0 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1 0 0 0 1 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 1 1 1 0 1 1 1 1
0 0 0 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 0 1
0 1 0 1 0 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 0 0 1 1 1 0 1
0 1 0 1 0 0 0 0 1 1 1 0 1 1 1 1 0 1 1
Największe pole prostokąta: 18
Jedyńki tworzące największy prostokąt znajdują się w:
Wiersze: 51-56
Kolumny: 533-535
Czas wykonania programu: 1102.000 ms

Process returned 0 (0x0)   execution time : 51.284 s
Press any key to continue.
```

## 2.6. Analiza pomiarów czasowych

Tablica 3x3 – 0ms

Tablica 5x5 – 0ms

Tablica 50x50 – 1ms

Tablica 100x100 – 4ms

Tablica 1000x800 – 1102ms

Po wynikach widać, że czas wzrasta dosyć szybko przy czy coraz większych tablicach ale nie przesadnie więc złożoność obliczeniowa jest zgodna z zakładaną.

## 2.7. Ołówkowe sprawdzenie poprawności algorytmu.

Na przykładzie tablicy 3x3 wygenerowanej w przykładzie można sprawdzić poprawność wyniku oraz jak algorytm to policzył. Algorytm najpierw sprawdza wiersz i najdłuższy ciąg jedynek. W pierwszym wierszu sprawdzając znajduje największe pole 2 licząc w prawo [1 2 0]. Po napotkaniu 0 nie dodaje już kolejnej liczby czyli 0 bo nie jest jedynką. Aktualizuje więc największe pole, które teraz równe jest 2. W drugim wierszu licząc wygląda to tak:[0 2 0]. Jest tak, ponieważ w drugim wierszu sprawdza on również ilość jedynek znajdujących się w poprzednich wierszach nad tymi które są ciągiem jedynek w sprawdzanym aktualnie wierszu. Ponieważ największe pole w tym wierszu wynosi tyle samo nie aktualizujemy największego pola. W 3 wierszu krok po kroku wygląda to tak:[0 1 2], [0 2 2], [0 3 2]. W pierwszej iteracji sprawdził cały wiersz a następnie zaczął sprawdzać do góry. Ponieważ w trzeciej kolumnie nad jedynką jest zero i nie wyszło pole równe 4, algorytm w drugiej kolumnie zaczął sprawdzać dalej do góry, aż do pierwszego wiersza gdzie kończy, ponieważ tutaj kończą się wiersze zadanej macierzy. Wychodzi mu po przemnożeniu pole równe 3 czyli do tej pory największe więc aktualizuje największe pole. Ponieważ został sprawdzony ostatni wiersz program zwraca największe znalezione pole czyli w tym wypadku równe 3.

## 3. Materiały przydatne do stworzenia sprawozdania

[https://lucid.app/documents#/home?folder\\_id=recent](https://lucid.app/documents#/home?folder_id=recent)

<https://www.algorytm.edu.pl/matura-informatyka/zlozonosc-algorytmu>

[http://marbor.strony.prz.edu.pl/files/AiSD/jak\\_poprawnie\\_rozwiazac\\_zadanie.pdf](http://marbor.strony.prz.edu.pl/files/AiSD/jak_poprawnie_rozwiazac_zadanie.pdf)

<https://github.com>

<https://www.algorytm.edu.pl/matura-informatyka/algorytm/pseudokod>