WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

SPRAWOZDANIE

Magdalena Błażej

Listopad 28, 2023

Spis treści

1 Treść zadania

Dla tablicy NxM generowanej losowo dla N i M, wypełnionej wartościami 0 i 1, znajdź liczbę znaków z jedynek ("plusem"jest krzyżyk z jedynek otoczony zerami) Przykładowe szukanie sekwencji :
0 0 0 0 0
0 0 1 0 0
0 1 1 1 0
0 0 1 0 0
0 0 0 0 0

2 Etapy rozwiązywania

2.1 Rozwiązanie 1: brute force

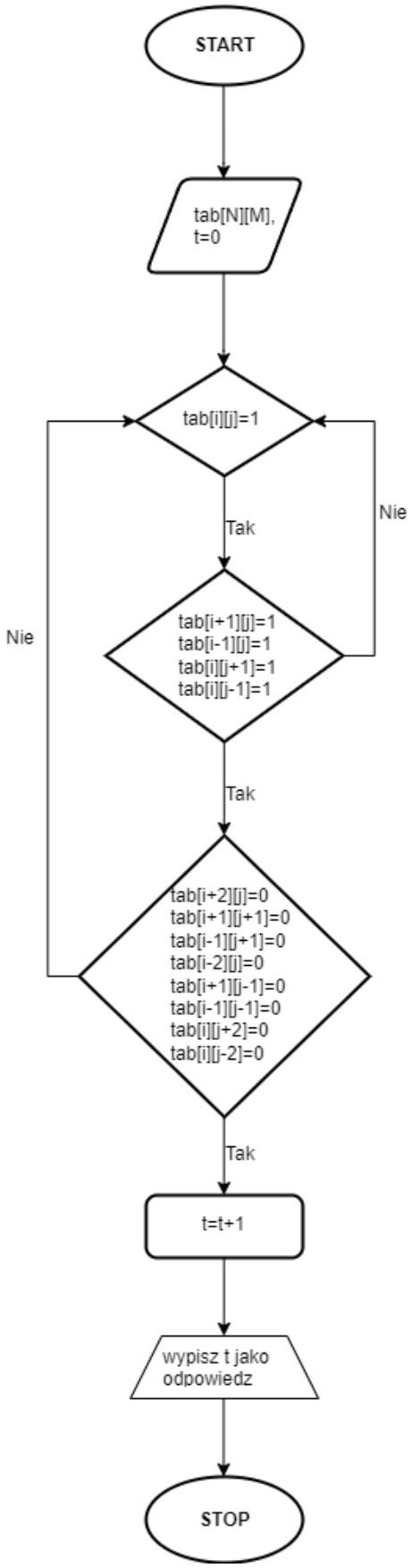
2.1.1 Analiza problemu

Aby rozwiązać to zadanie, musimy utworzyć dwuwymiarową tablicę o wymiarach NxM, wypełnioną jedynie wartościami 0 i 1, których położenie będzie losowe. Następnie piszemy algorytm, który będzie sprawdzał wygenerowaną tablicę w poszukiwaniu sekwencji. Jako że jest to najprostszy sposób
rozwiązania, algorytm będzie sprawdzał wszystkie koordynaty tabeli NxM w poszukiwaniu sekwencji zgodnej z kryteriami "plusa"([i+1][j]=1, [i][j-2]=0, [i][j-1]=1, [i-1][j-1]=0, [i+1][j-1]=0, [i-1][j]=1, [i+1][j]=1, [i-2][j]=0, [i+2][j]=0, [i-1][j+1]=0, [i+1][j+1]=0, [i][j+1]=1, [i][j+2]=0).

2.1.2 Złożoność obliczeniowa

$$O(N * M) + O((N * M)^2) + O(N * M) = O((N * M)^2)$$

2.1.3 Schemat blokowy



2.1.4 Pseudokod

Pętla if jest niezwykle długa, ale jako że jest to próba najłatwiejszego rozwiązania zadania (brute force), najprostszym rozwiązaniem było stworzenie pętli szukającej, która sprawdzi i wszystkie koordynaty dookoła znalezionej wartości 1

Algorithm 1 Pseudokod: 1 podejścia

```
input: tab // tablica dwuwymiarowa wypełniona 0 i 1 w której szukamy "plusów"
output: t // ilość zliczonych "plusów"

tab[N][M]

i, j, t=0 //t zmienna w której zapisuje się liczba znalezionych plusów
if tab[i][j] równe 1 then

    if tab[i+1][j]=1, tab[i-1][j]=1, tab[i][j+1]=1, tab[i][j-1]=1 then

        if tab[i+2][j]=0, tab[i+1][j+1]=0, tab[i-1][j+1]=0, tab[i][j+2]=0,
tab[i][j-2]=0, tab[i-1][j+1]=0, tab[i-1][j-1]=0, tab[i-2][j]=0 then
            zwiększ t o 1 (t++)

            Wypisz t jako odpowiedź
```

2.1.5 Napotkane problemy

Znalezienie sekwencji "plus"w tabeli z wartościami rzędu milionów na pewno jest bardziej prawdopodobne niż znalezienie jej w tablicach z wartościami mniejszymi niż 100, na których pracuję.
Dlatego wprowadzam element pseudolosowy.
1)Losowane wartości wypełniające tabelę są od 0 do 5. Wartości większe od 1 zamieniam na 0, zmniejszając ilość wartości 1 pojawiających się w tabeli.
2)Dodatkowa pętla szuka "samotnych"wartości 1 (otoczonych zerami na koordynatach warunku "plusa"), tworząc wokół niej szukaną sekwencję, która jest zliczana przez algorytm jako plus.

Algorithm 2 Pseudokod: Pętla tworząca "plusy"

```
1: for (i równe 0; i mniejsze od N; ++i) do
2:     for (j równe 0; j mniejsze od M; ++j) do
3:         if tab[i][j]=1 then
4:             if ([i+1][j]=0 oraz [i][j-2]=0 oraz
5:                 [i][j-1]=0 oraz [i-1][j-1]=0 oraz
6:                 [i+1][j-1]=0 oraz [i-1][j]=0 oraz
7:                 [i+1][j]=0 oraz [i-2][j]=0 oraz
8:                 [i+2][j]=0 oraz [i-1][j+1]=0 oraz
9:                 [i+1][j+1]=0 oraz [i][j+1]=0 oraz [i][j+2]=0)  then
10:                tab[i - 1][j] = 1;
11:                tab[i + 1][j] = 1;
12:                tab[i][j - 1] = 1;
13:                tab[i][j + 1] = 1;
```

Dzięki tej zmianie jesteśmy w stanie fizycznie sprawdzić, czy algorytm szukający "plusy"działa. Oczywiście, ta pętla nie jest wymagana i służy jedynie upewnieniu się, czy algorytm działa na niewielkich tablicach, gdzie naturalne wygenerowanie sekwencji jest prawie niemożliwe

2.2 Rozwiązanie 2: próba optymalizacji

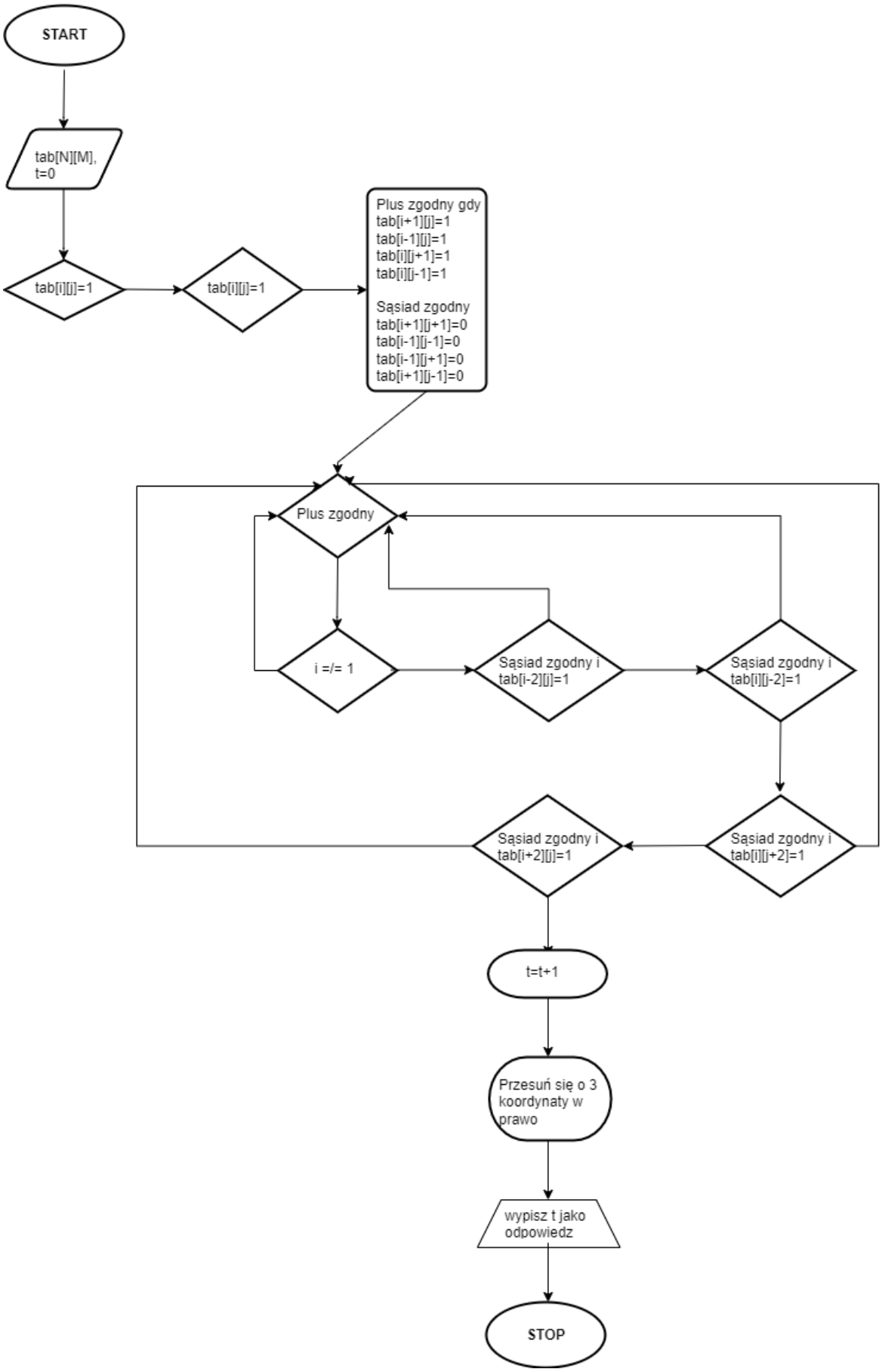
2.2.1 Ponowne przemyślenie problemu

By zoptymalizować algorytm, będziemy używać funkcji bool która będzie wykrywać "plusa"(wartość 1 znaleziona na
koordynatach [i - 1][j], [i][j], [i + 1][j], [i][j - 1], [i][j + 1]) i jeśli sekwencja jest prawdziwa przesuwa się o 3 pola, dzięki czemu przeszuka tablice szybciej, zliczone sekwencje są sprawdzane czy posiadają wartości 0 na obrzeżach "plusa"by spełniać wszystkie warunki na bycie zaliczone do ostatecznego wyniku. Kolejną zmianą będzie pomijanie "brzegów"tabeli, czyli zaczęcie przeszukiwanie od N+1 i M+1 gdyż plus nie ma szansy powstać zaraz przy nim.

2.2.2 Złożoność obliczeniowa

$$O(N * M) + O(N * M) + O(N * M) = 3O(N * M)$$

2.2.3 Schemat blokowy



2.2.4 Pseudokod

Algorithm 3 Pseudokod: 2 podejście

```
1: input: tab // tablica dwuwymiarowa wypełniona 0 i 1 w której szukamy "plusów"
2: output: t // ilość zliczonych "plusów"
3:
4: tab[N][M]
5:
6: i, j, t=0 //t to zmienna w której liczymy ilość plusów
7: Jeśli tab[i - 1][j] i tab[i][j] i tab[i + 1][j] i tab[i][j - 1] i tab[i][j + 1] = 1
8: - zapamiętaj Plus zgodny
9: Jeśli tab[i - 1][j - 1] i tab[i + 1][j + 1]) i tab[i - 1][j + 1] i tab[i + 1][j - 1] = 0
10: - zapamiętaj Sąsiad zgodny
11: Sprawdź czy tab[i - 2][j] i tab[i + 2][j] i tab[i][j - 2] i tab[i][j + 2] = 0
12: Jeśli powyższe oświadczenia są prawdziwe = t + 1
13: Przesuń się o 3 koordynaty w prawo
14: Wyświetl t jako odpowiedź =0
```

2.3 Implementacje wymyślonych algorytmów w języku C++ oraz eksperymentowanie z wydajnością

2.3.1 Prosta implementacja wymyślonych algorytmów

Kod 1 (przed optymalizacją):

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main(){

int N, M;
cout<<"wpisz N: ";
cin>>N;
cout<<"wpisz M: ";
cin>>M;
int tab[N][M];

srand(static_cast<unsigned int>(time(0)));
for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j) {
        tab[i][j] = rand() % 6;
        if (tab[i][j]>1){
            tab[i][j]=0;}}}
int i, j, t=0;
//Generowanie plusa
for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j) {
        if (tab[i][j] == 1 ){
            if(tab[i][j-2] == 0 && tab[i][j-1] == 0
                && tab[i-1][j-1] == 0 && tab[i+1][j-1] == 0
                && tab[i-1][j] == 0 && tab[i+1][j] == 0
                && tab[i-2][j] == 0 && tab[i+2][j] == 0
                && tab[i-1][j+1] == 0 && tab[i+1][j+1] == 0
                && tab[i][j+1] == 0 && tab[i][j+2] == 0){
                if (i - 1 >= 0) tab[i - 1][j] = 1;
                if (i + 1 < N) tab[i + 1][j] = 1;
                if (j - 1 >= 0) tab[i][j - 1] = 1;
                if (j + 1 < M) tab[i][j + 1] = 1; }}}}
//Szukanie Plusa
for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j) {
        if (tab[i][j] == 1) {
            if (i+1<M && tab[i+1][j]==1){
            { if (tab[i+1][j]==1 && tab[i][j-2] == 0 && tab[i][j-1] == 1
                && tab[i-1][j-1] == 0 && tab[i+1][j-1] == 0
                && tab[i-1][j] == 1 && tab[i+1][j] == 1
                && tab[i-2][j] == 0 && tab[i+2][j] == 0
```

```
                    && tab[i-1][j+1] == 0 && tab[i+1][j+1] == 0
                    && tab[i][j+1] == 1 && tab[i][j+2] == 0){
                    t++;
                    }}}}}}

for(i=0;i<N;i++){
     for(j=0;j<M;j++){
          cout<<tab[i][j]<<" ";}
          cout<<endl;}
cout<<endl<<"wielkosc tabeli NxM: "<<N<<"x"<<M;
cout<<endl<<"znalezione plusy: "<<t;
 return 0;
 }
```

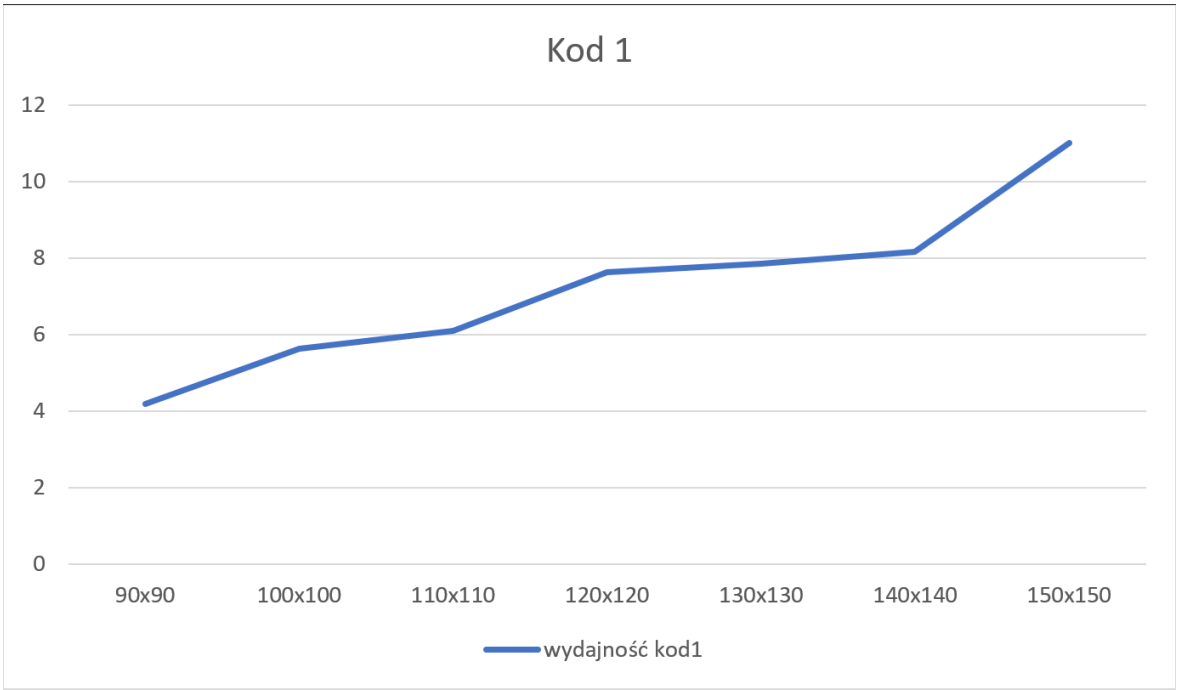
Kod 2 (po pierwszej optymalizacji):

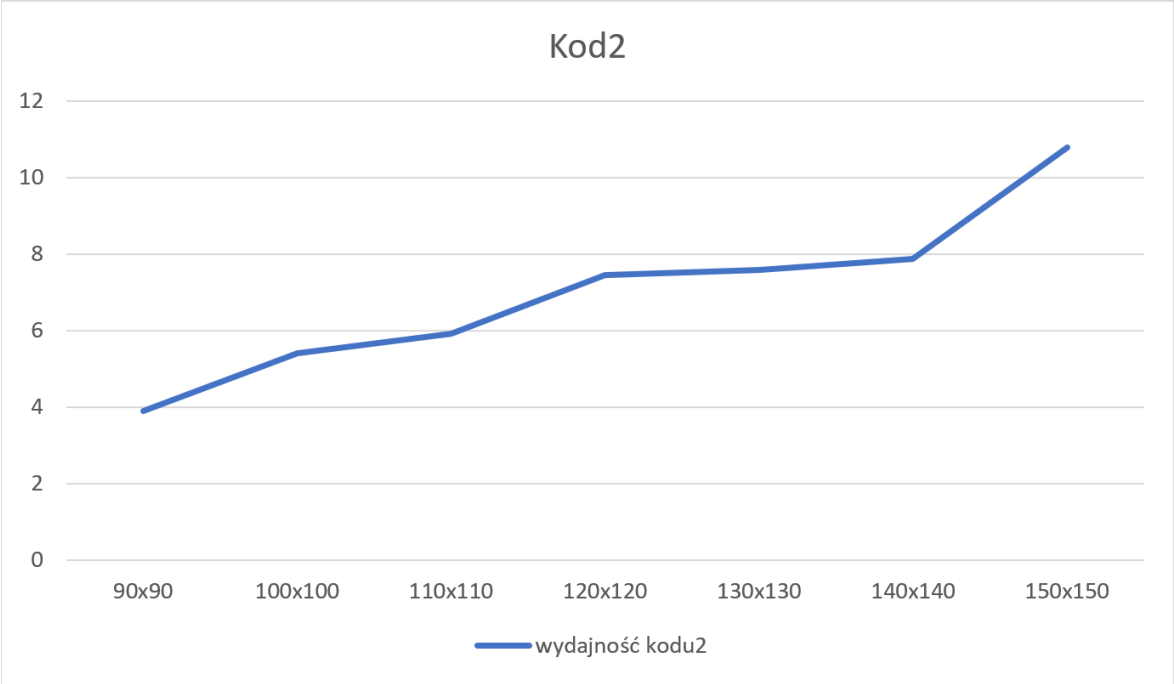
```
#include <cstdlib>
#include <ctime>
#include <iostream>
using namespace std;

int main() {

int N, M;
cout<<"wpisz N: ";
cin>>N;
cout<<"wpisz M: ";
cin>>M;
int tab[N][M];

srand(static_cast<unsigned int>(time(0)));
for (int i = 0; i < N; ++i) {
for (int j = 0; j < M; ++j) {
    tab[i][j] = rand() % 6;
    if (tab[i][j] > 1) {
        tab[i][j] = 0;
                }}}
//Generowanie plusa
for (int i = 1; i < N + 1; ++i) {
for (int j = 1; j < M + 1; ++j) {
    if (tab[i][j] == 1) {
    if (rand() % 12 > 9) {
        tab[i - 1][j] = 1;
        tab[i - 1][j - 1] = 0;
        tab[i - 1][j + 1] = 0;
        tab[i][j - 1] = 1;
        tab[i][j + 1] = 1;
        tab[i][j] = 1;
        tab[i + 1][j - 1] = 0;
        tab[i + 1][j] = 1;
        tab[i + 1][j + 1] = 0;
}}}}
```

```
int t=0;

//Szukanie plusa
for (int i = 1; i < N - 1; ++i) {
for (int j = 1; j < M - 1; ++j) {
    if (tab[i][j] == 1) {
```

```
    bool isValidPlus = (tab[i - 1][j]) &&
                       (tab[i][j]) &&
                       (tab[i + 1][j]) &&
                       (tab[i][j - 1]) &&
                       (tab[i][j + 1]);
    bool doesNeighboursValid = !(tab[i - 1][j - 1]) &&
                               !(tab[i + 1][j + 1]) &&
                               !(tab[i - 1][j + 1]) &&
                               !(tab[i + 1][j - 1]);
if (isValidPlus) {
if (i != 1)
doesNeighboursValid &= !(tab[i - 2][j]);
if (j != 1)
doesNeighboursValid &= !(tab[i][j - 2]);
if (j != M - 2)
doesNeighboursValid &= !(tab[i + 2][j]);
if (i != N - 2)
doesNeighboursValid &= !(tab[i][j + 2]);
if (doesNeighboursValid) {
t++;
i += 3;
}}}}}
int i, j;
for (i = 0; i < N; i++) {
for (j = 0; j < M; j++) {
cout << tab[i][j] << " ";}
cout << endl;}
cout << endl << "wielkosc tabeli NxM: " << N << "x" << M;
cout << endl << "znalezione plusy: " << t;
return 0;
}
```

2.3.2 Testy wydajności - eksperymentalne sprawdzenie złożoności czasowej





Mimo tego, że wykresy obu kodów wyglądają podobnie wydajność programu zmieniła się średnio o 0,2 sekundy. Wydajność sprawdzałam na tych samych wielkościach tabel wyciągając średni czas realizacji algorytmu (10 prób na każdą tablicę).