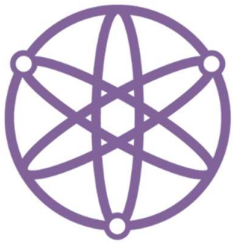


WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ
POLITECHNIKI RZESZOWSKIEJ

Dariusz Strojny

Algorytmy i Struktury Danych

Projekt zaliczeniowy nr 3

Rzeszów, 2023

1 Spis treści

2 Spis ilustracji

3 Temat

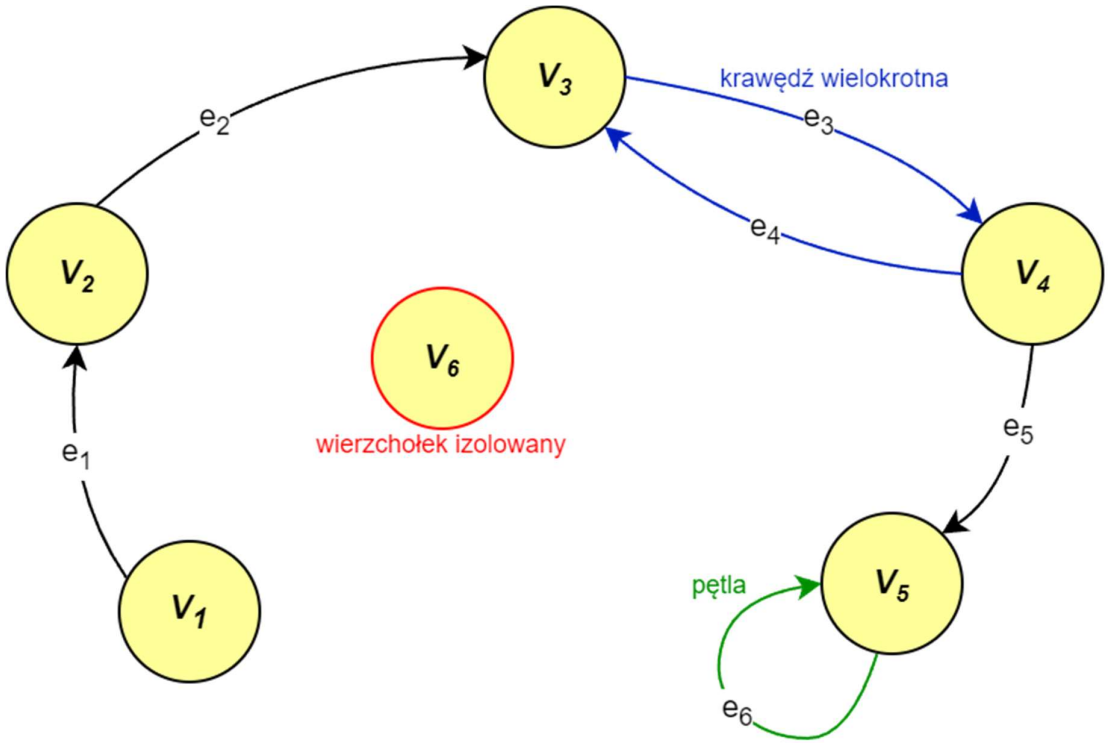
Zadanie 9. Napisz program, który dla zadanego grafu skierowanego reprezentowanego przy pomocy macierzy sąsiedztwa wyznaczy i wypisze następujące informacje:

1) Wszystkich sąsiadów dla każdego wierzchołka grafu (sąsiad wierzchołka w_i to ten wierzchołek, do którego prowadzi krawędź z w_i)
2) Wszystkie wierzchołki, które są sąsiadami każdego wierzchołka
3) Stopnie wychodzące wszystkich wierzchołków
4) Stopnie wchodzące wszystkich wierzchołków
5) Wszystkie wierzchołki izolowane
6) Wszystkie pętle
7) Wszystkie krawędzie dwukierunkowe

Każdy z powyższych podpunktów powinien być realizowany jako oddzielna funkcja. W funkcji main() należy przedstawić działanie napisanej przez siebie biblioteki na reprezentatywnym przykładzie. Kod powinien być opatrzony stosownymi komentarzami.

4 Analiza, projektowanie

4.1 Wstęp teoretyczny



Rysunek 1 - Przykładowy graf skierowany

4.1.1 Graf

Graf to matematyczny model reprezentujący zbiór obiektów (wierzchołków lub nodów) oraz relacji między nimi (krawędzi lub połączeń).

4.1.2 Graf skierowany

Graf skierowany jest to rodzaj grafu, w którym krawędzie posiadają kierunek. Oznacza to, że krawędzie nie są symetryczne i nie można przejść od wierzchołka A do wierzchołka B tą samą krawędzią, co z powrotem. Grafy skierowane są często używane do modelowania różnych procesów, w których istnieją jasno określone kierunki przepływu, np. w sieciach komunikacyjnych lub w procesach biznesowych.

4.1.3 Wierzchołki grafu

Wierzchołki (ang. vertices) to elementy grafu, które reprezentują obiekty lub relacje między obiektami. Mogą one reprezentować różne rzeczy, w zależności od kontekstu, w jakim jest używany graf. Na przykład, w grafie opisującym sieć drogową, wierzchołki mogą reprezentować miasta, a krawędzie drogi między nimi.

4.1.4 Krawędzie w grafie

Krawędzie (ang. edges) to połączenia między wierzchołkami, które określają relacje między nimi. Krawędzie mogą być oznaczone różnymi atrybutami, takimi jak waga lub przepustowość, w zależności od kontekstu. W przypadku grafu opisującego sieć drogową, krawędzie mogą mieć atrybut "długość" lub "przepustowość"

4.1.5 Sąsiedzi w grafie

Sąsiedzi (ang. neighbors) to wierzchołki, z którymi dany wierzchołek jest połączony krawędzią. W grafie skierowanym, sąsiedzi to tylko te wierzchołki, do których krawędź jest skierowana.

4.1.6 Stopień wierzchołka

Stopień wierzchołka (ang. vertex degree) to liczba krawędzi, które do niego prowadzą lub od niego wychodzą. W grafie skierowanym rozróżniamy stopie

4.1.7 Pętle

Pętle (ang. loops) to krawędzie, które łączą dany wierzchołek z samym sobą. Są one szczególnym przypadkiem krawędzi skierowanych, ponieważ kończą się one i zaczynają w tym samym wierzchołku. Pętle są często używane w różnych modelach matematycznych, w których jest wymagane przejście przez dany wierzchołek wiele razy.

W grafach skierowanych, stopień wierzchołka oznacza liczbę krawędzi, które do niego prowadzą lub od niego wychodzą. Pętle są wliczane do stopnia wierzchołka, co oznacza, że jeśli wierzchołek posiada pętlę, jego stopień jest o jeden większy niż liczba krawędzi, które do niego prowadzą lub od niego wychodzą.

4.2 Zasada działania programu

Program ten ma za zadanie wykonanie zbioru określonych akcji na grafie skierowanym reprezentowanym przy pomocy macierzy sąsiedztwa. Macierz ta ma zostać wypełniona wartościami określającymi istnienie krawędzi pomiędzy poszczególnymi wierzchołkami tego grafu.

4.3 Struktury danych

Najważniejszą strukturą danych w programie jest macierz sąsiedztwa reprezentująca wprowadzany do niej graf skierowany. Macierz sąsiedztwa posiada wymiary n x n, gdzie n jest ilością wierzchołków w danym grafie. Każde pole w macierzy sąsiedztwa reprezentuje informację o istnieniu pewnej krawędzi poprzez wartość 1 lub jej brak przez wartość 0. Pola znajdujące się na przekątnej reprezentują istnienie krawędzi tworzących pętle na poszczególnych wierzchołkach. Pozostałe pola definiują krawędzie między wierzchołkami, których numery są odpowiednio zależne od numeru wiersza i kolumny, na których to te pola się znajdują.

Na przykład: Pole o numerze wiersza 3 i kolumny 2 w macierzy sąsiedztwa, reprezentuje istnienie krawędzi skierowanej z wierzchołka 3 do wierzchołka 2, natomiast pole o numerze wiersza 1 i kolumny 1 definiuje istnienie pętli na tym wierzchołku.

Dodatkowo utworzoną przeze mnie strukturą danych jest struktura grafu, przechowująca w kodzie tablicę będącą macierzą sąsiedztwa oraz ilość wierzchołków w grafie. Jej definicja wygląda następująco:

```
// struktura grafu przechowująca macierz sąsiedztwa oraz ilość
wierzchołków grafu
struct {
    // tablica dwuwymiarowa przechowująca informacje o krawędziach grafu
    int** array;
    // ilość wierzchołków grafu
    int size;
} typedef graph;
```

4.4 Metodyka

4.4.1 Funkcje użyte w programie

- `graph new_graph(int size)`

  Tworzy nową strukturę typu graph i inicjuje w niej pamięć dla macierzy sąsiedztwa

  Parametry

  size – ilość wierzchołków w tworzonym grafie

  Zwraca

  Zainicjowany obiekt typu graph

- `void free_graph(graph &g)`

  Zwalnia pamięć struktury typu graph

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_add_edge(graph& g, int a, int b)`

  Ustawia w macierzy sąsiedztwa grafu skierowanego informację o obecności krawędzi od wierzchołka a do wierzchołka b

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

  a – numer wierzchołka, z którego wychodzi krawędź

  b – numer wierzchołka, do którego wchodzi krawędź

- `void graph_add_edge(graph& g, const std::vector<std::pair<int, int>>& edges)`

  Ustawia w macierzy sąsiedztwa grafu skierowanego informację o obecności krawędzi pomiędzy każdą z par przekazanych w tablicy

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

  edges – tablica zawierająca pary numerów wierzchołków, między którymi mają powstać krawędzie

- `void print_graph(const graph& g)`

  Wypisuje zawartość macierzy sąsiedztwa grafu

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_neighbors_of_each_vertex(const graph& g)`

  Wypisuje wszystkich sąsiadów dla każdego wierzchołka grafu

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_each_vertex_that_is_neighbor(const graph& g)`

  Wypisuje wszystkie wierzchołki, które są sąsiadami każdego wierzchołkaParametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_vertex_outdegrees(const graph& g)`

  Wypisuje stopnie wychodzące wszystkich wierzchołków

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_vertex_indegrees(const graph& g)`

  Wypisuje stopnie wchodzące wszystkich wierzchołków

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_isolated_vertexes(const graph& g);`

  Wypisuje wszystkie wierzchołki izolowane

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_loops(const graph& g)`

  Wypisuje wszystkie pętle

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

- `void graph_print_multi_edges(const graph& g)`

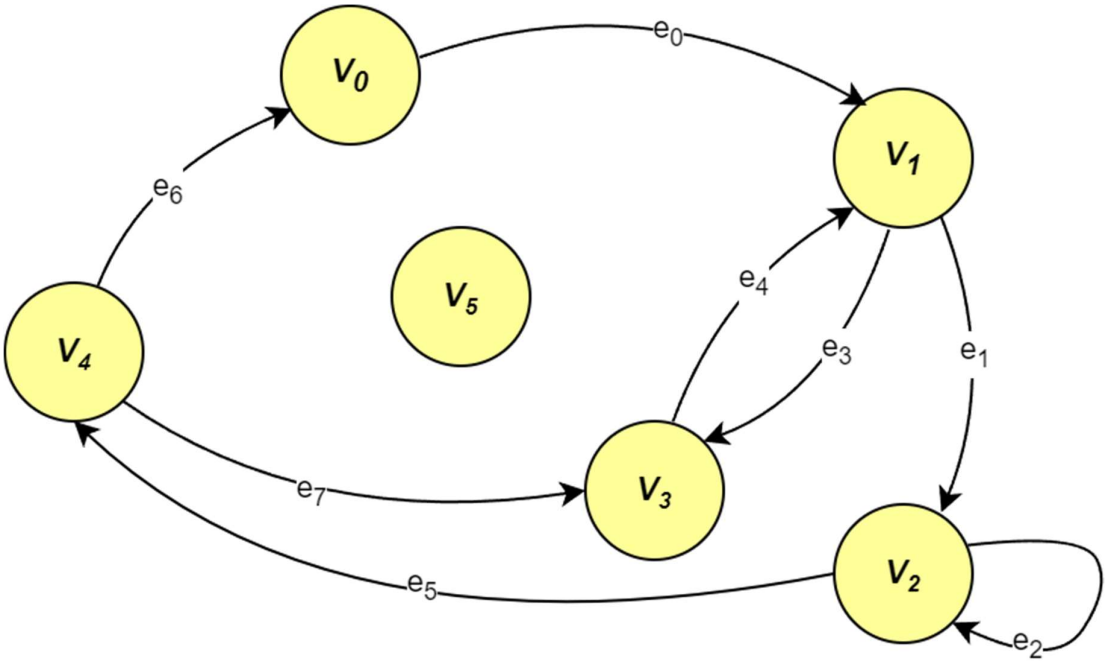
  Wypisuje wszystkie krawędzie dwukierunkowe

  Parametry

  g – obiekt typu graph przechowujący macierz sąsiedztwa i ilość wierzchołków grafu skierowanego

5 Dane wejściowe

Do przedstawienia działania napisanych funkcji do programu wprowadzony zostanie poniższy graf skierowany:



Rysunek 2 - Wejściowy graf skierowany

Jak możemy odczytać z grafu, posiada on:

Jeden wierzchołek izolowany V_5 ,

Jedną pętlę e_2 przy wierzchołku V_2 ,

Jedną krawędź dwukierunkową pomiędzy wierzchołkami V_1 $oraz$ V_3

6 Opisy i działanie funkcji programu

6.1 Wypisywanie wszystkich sąsiadów dla każdego wierzchołka grafu

6.1.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Sasiedzi dla kazdego wierzcholka grafu:
 0:  1
 1:  2,  3
 2:  2,  4
 3:  1
 4:  0,  3
 5:  brak
```

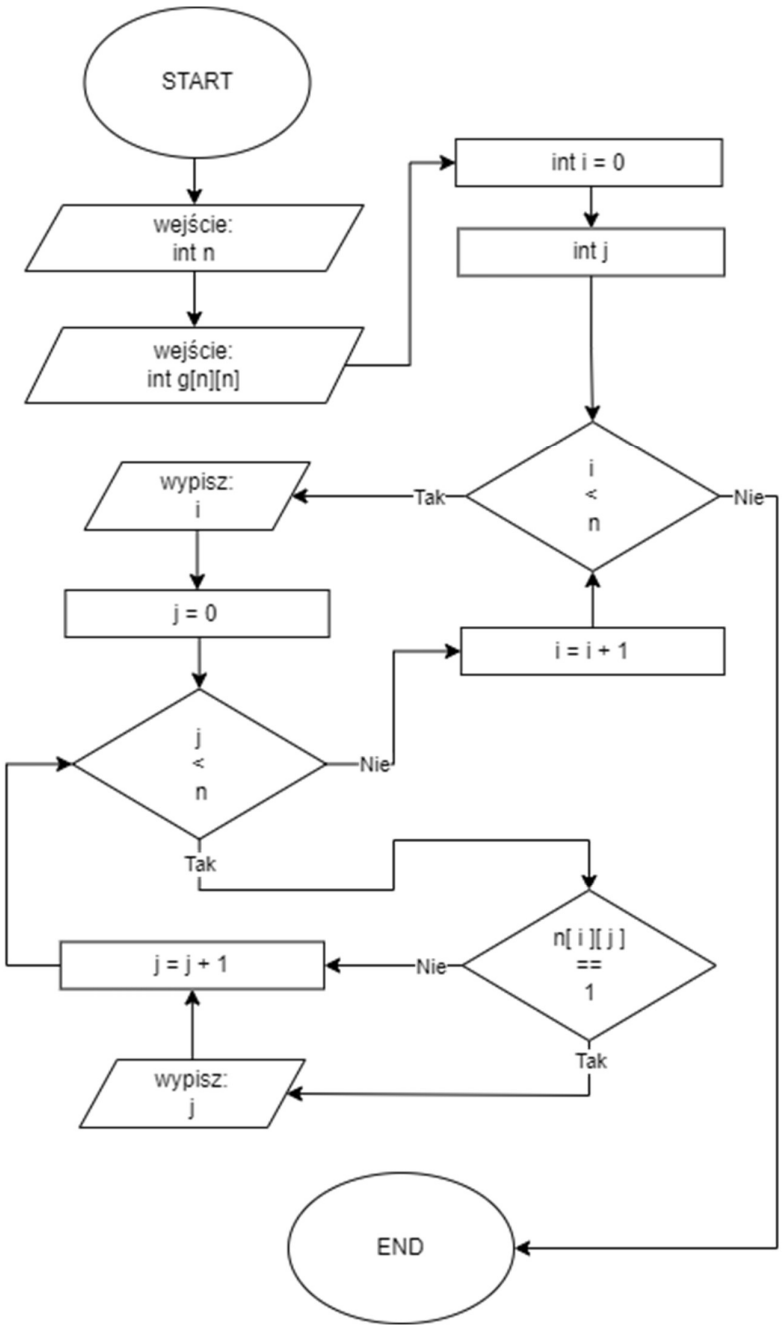
Rysunek 3 - Wynik działania funkcji 1

6.1.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków grafu
g[n][n]           - macierz sąsiedztwa grafu

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa


algorytm:

i <- 0
dopóki i < n:
      wypisz i
      j <- 0
      dopóki j < n:
            jeżeli g[i][j] == 1:
                  wypisz j
            j <- j + 1
      i <- i + 1
```

Rysunek 4 - Pseudokod funkcji 1

6.1.3 Schemat blokowy



Rysunek 5 - Schemat blokowy funkcji 1

6.2 Wypisywanie wszystkich wierzchołków, które są sąsiadami każdego wierzchołka

6.2.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Wierzcholki, ktore sa sasiadami kazdego wierzcholka:
 0:  4
 1:  0,  3
 2:  1,  2
 3:  1,  4
 4:  2
 5:  brak
```

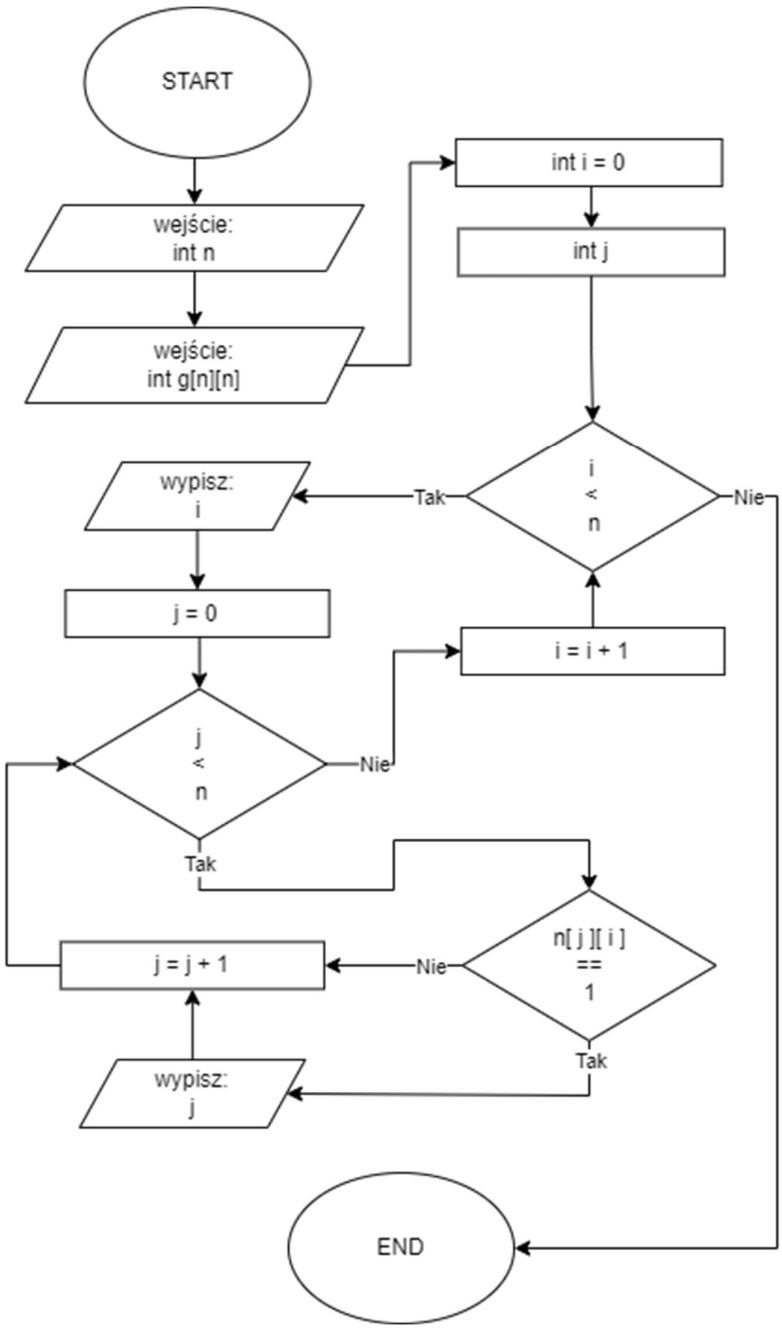
Rysunek 6 - Wynik działania funkcji 2

6.2.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków grafu
g[n][n]           - macierz sąsiedztwa grafu

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa


algorytm:

i <- 0
dopóki i < n:
      wypisz i
      j = 0
      dopóki j < n:
            jeżeli g[j][i] == 1:
                  wypisz j
            j <- j + 1
      i <- i + 1
```

Rysunek 7 - Pseudokod funkcji 2

6.2.3 Schemat blokowy



Rysunek 8 - Schemat blokowy funkcji 2

6.3 Wypisywanie stopni wychodzących wszystkich wierzchołków

6.3.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Stopnie wychodzace:
 0: 1
 1: 2
 2: 2
 3: 1
 4: 2
 5: 0
```

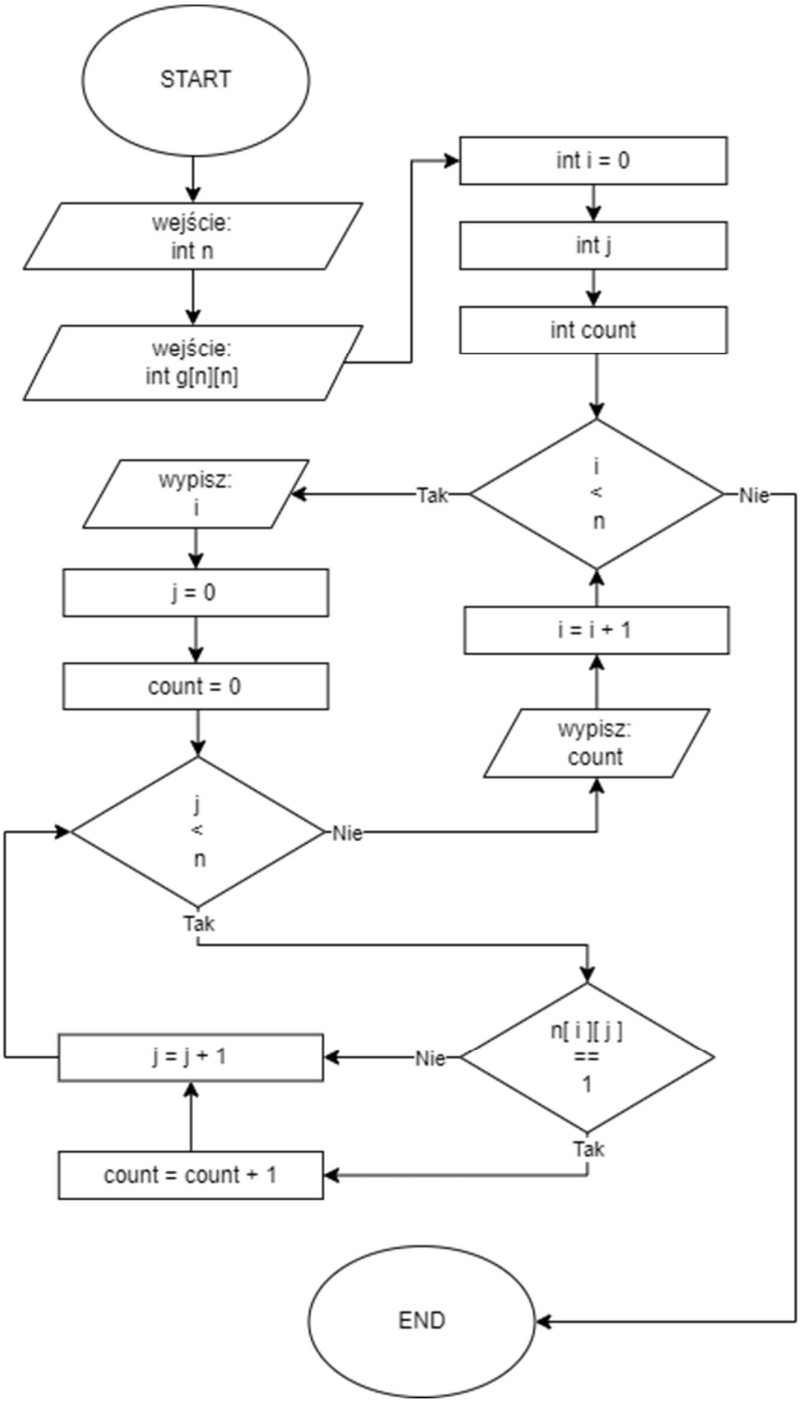
Rysunek 9 - Wynik działania funkcji 3

6.3.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków grafu
g[n][n]           - macierz sąsiedztwa grafu

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa
count             - licznik wierzchołków, których dany wierzchołek jest
                  sąsiadem


algorytm:

i <- 0
dopóki i < n:
      wypisz i
      count <- 0
      j <- 0
      dopóki j < n:
            jeżeli g[i][j] == 1:
                  count <- count + 1
            j <- j + 1
      wypisz count
      i <- i + 1
```

Rysunek 10 - Pseudokod funkcji 3

6.3.3 Schemat blokowy



Rysunek 11 - Schemat blokowy funkcji 3

6.4 Wypisywanie stopni wchodzących wszystkich wierzchołków

6.4.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Stopnie wchodzace:
 0: 1
 1: 2
 2: 2
 3: 2
 4: 1
 5: 0
```

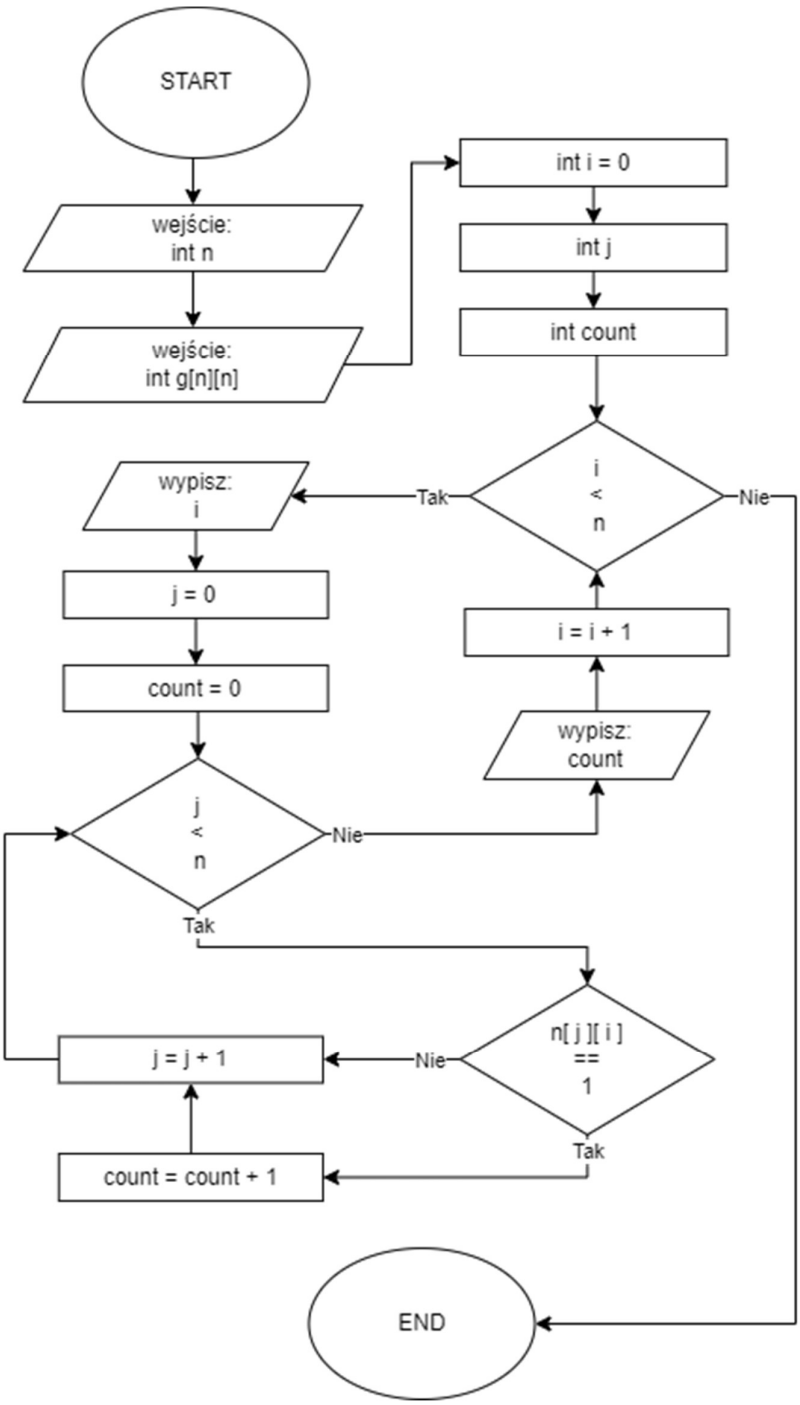
Rysunek 12 - Wynik działania funkcji 4

6.4.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków grafu
g[n][n]           - macierz sąsiedztwa grafu

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa
count             - licznik wierzchołków będących sąsiadami danego
                  wierzchołka


algorytm:

i <- 0
dopóki i < n:
      wypisz i
      count <- 0
      j <- 0
      dopóki j < n:
            jeżeli g[j][i] == 1:
                  count <- count + 1
            j <- j + 1
      wypisz count
      i <- i + 1
```

Rysunek 13 - Pseudokod funkcji 4

6.4.3 Schemat blokowy



Rysunek 14 - Schemat blokowy funkcji 4

6.5 Wypisywanie wszystkich wierzchołków izolowanych

6.5.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Wierzcholki izolowane:
 5
```

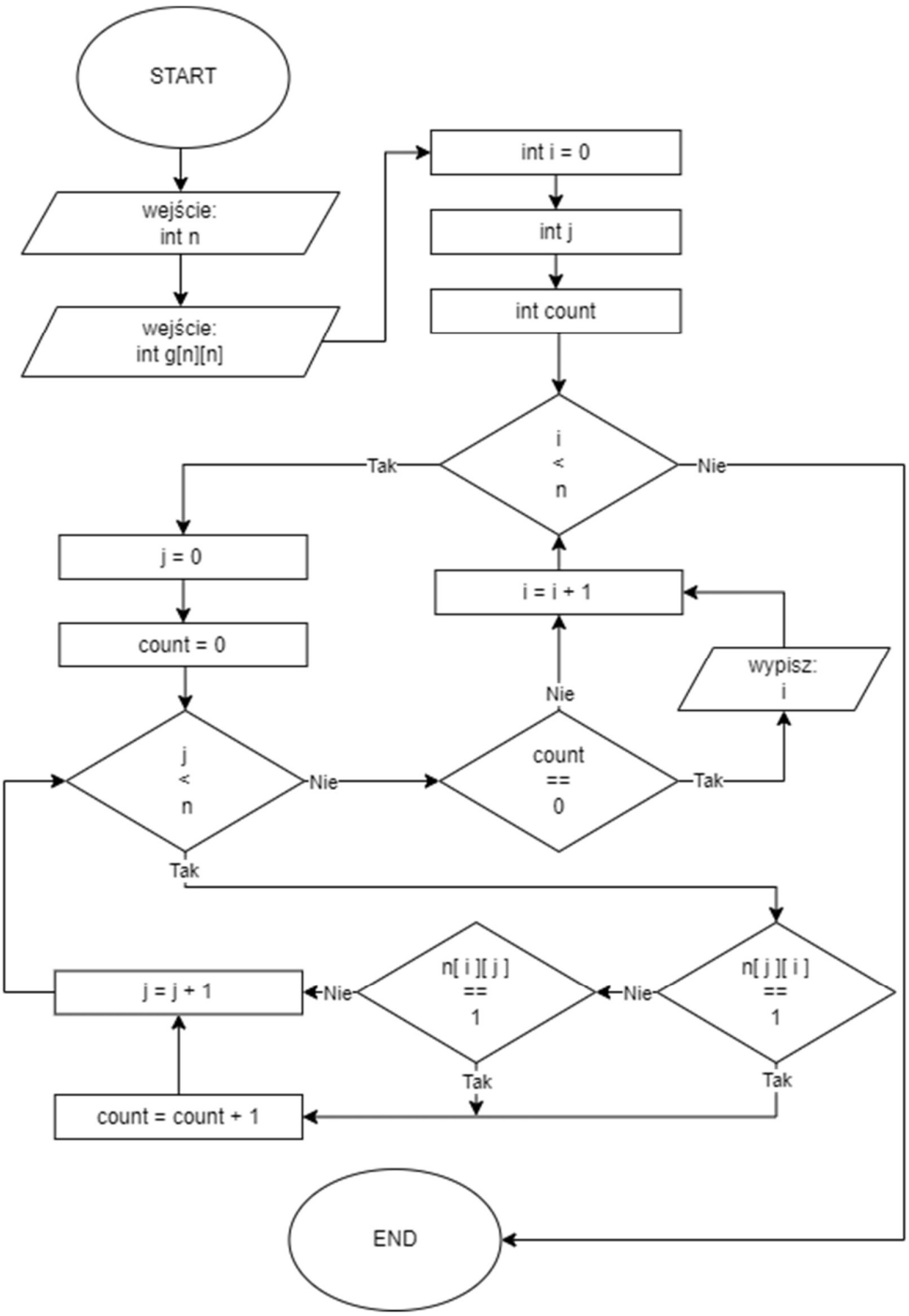
Rysunek 15 - Wynik działania funkcji 5

6.5.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków grafu
g[n][n]           - macierz sąsiedztwa grafu

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa
count             - licznik krawędzi wchodzących bądź wychodzących z
                  danego wierzchołka


algorytm:

i <- 0
dopóki i < n:
      count <- 0
      j <- 0
      dopóki j < n:
            jeżeli g[j][i] == 1 lub g[i][j] == 1:
                  count <- count + 1
            j <- j + 1
      jeżeli count > 0:
            wypisz i
      i <- i + 1
```

Rysunek 16 - Pseudokod funkcji 5

6.5.3 Schemat blokowy



Rysunek 17 - Schemat blokowy funkcji 5

6.6 Wypisywanie wszystkich pętli

6.6.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Petle:
 2
```

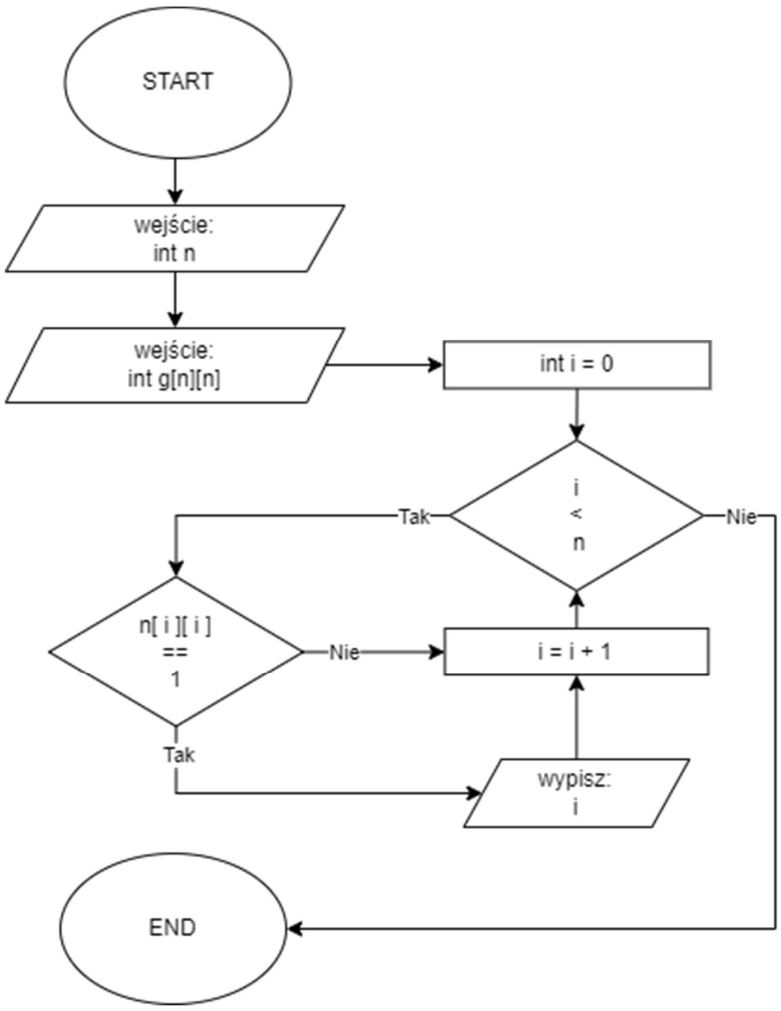
Rysunek 18 - Wynik działania funkcji 6

6.6.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków macierzy sąsiedztwa
g[n][n]           - macierz sąsiedztwa macierzy sąsiedztwa

dane:

i                 - iterator numerów wierzchołków

algorytm:

i <- 0
dopóki i < n:
      jeżeli g[i][i] == 1:
            wypisz i
      i <- i + 1
```

Rysunek 19 - Pseudokod funkcji 6

6.6.3 Schemat blokowy



Rysunek 20 - Schemat blokowy funkcji 6

6.7 Wypisywanie wszystkich krawędzi dwukierunkowych

6.7.1 Wynik działania programu

Wywołanie programu dla przykładowego grafu zwróciło następujące wyniki:

```
Krawedzie dwukierunkowe:
1 <-> 3
```

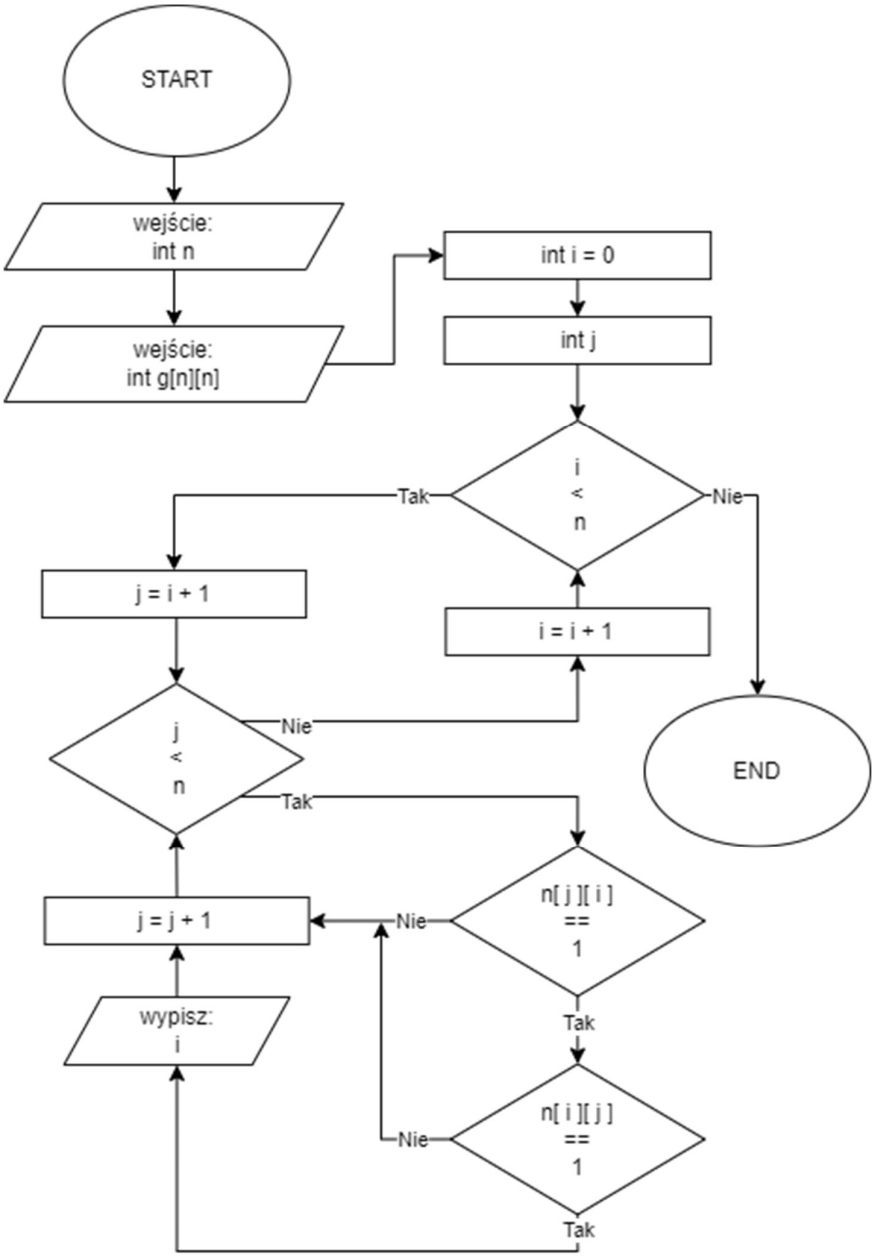
Rysunek 21 - Wynik działania funkcji 7

6.7.2 Pseudokod

```
wejście:

n                 - ilość wierzchołków macierzy sąsiedztwa
g[n][n]           - macierz sąsiedztwa macierzy sąsiedztwa

dane:

i                 - iterator wierszy macierzy sąsiedztwa
j                 - iterator kolumn macierzy sąsiedztwa

algorytm:

i <- 0
dopóki i < n:
      j <- i + 1
      dopóki j < n:
            jeżeli g[i][j] == 1 oraz g[j][i] == 1:
                  wypisz i
                  wypisz j
            j <- j + 1
      i <- i + 1
```

Rysunek 22 - Pseudokod funkcji 7

6.7.3 Schemat blokowy



Rysunek 23 - Schemat blokowy funkcji 7

7 Kod programu

7.1 Definicja struktur

```
// struktura grafu przechowująca macierz sąsiedztwa oraz ilość wierzchołków grafu
struct {
    // tablica dwuwymiarowa przechowująca informacje o krawędziach grafu
    int** array;
    // ilość wierzchołków grafu
    int size;
} typedef graph;
```

7.2 Deklaracje funkcji

```
graph new_graph(int size);
void free_graph(graph& g);
void graph_add_edge(graph& g, int a, int b);
void graph_add_edge(graph& g, const std::vector<std::pair<int, int>>& edges);
void print_graph(const graph& g);
void graph_print_neighbors_of_each_vertex(const graph& g);
void graph_print_each_vertex_that_is_neighbor(const graph& g);
void graph_print_vertex_outdegrees(const graph& g);
void graph_print_vertex_indegrees(const graph& g);
void graph_print_isolated_vertexes(const graph& g);
void graph_print_loops(const graph& g);
void graph_print_multi_edges(const graph& g);
```

7.3 Funkcja main

```
int main()
{
    // utworzenie grafu zawierającego 6 wierzchołków
    graph g = new_graph(6);

    // wpisanie krawędzi do grafu
    graph_add_edge(g, {
        {0, 1},
        {1, 2},
        {2, 2},
        {1, 3},
        {3, 1},
        {2, 4},
        {4, 0},
        {4, 3}
        });

    // wypisanie macierzy sąsiedztwa znajdującej się w grafie
    print_graph(g);

    // 1) wypisanie wszystkich sąsiadów dla każdego wierzchołka grafu.
    std::cout << "Sasiedzi dla kazdego wierzcholka grafu: \n";
    graph_print_neighbors_of_each_vertex(g);

    // 2) wypisanie wszyskich wierzchołków, które są sąsiadami każdego wierzchołka
    std::cout << "Wierzcholki, ktore sa sasiadami kazdego wierzcholka: \n";
    graph_print_each_vertex_that_is_neighbor(g);

    // 3) wypisanie stopni wychodzących wszystkich wierzchołków
    std::cout << "Stopnie wychodzace: \n";
    graph_print_vertex_outdegrees(g);

    // 4) wypisanie stopni wchodzących wszystkich wierzchołków
    std::cout << "Stopnie wchodzace:\n";
    graph_print_vertex_indegrees(g);

    // wypisanie wszystkich wierzchołków izolowanych
    std::cout << "Wierzcholki izolowane: \n";
    graph_print_isolated_vertexes(g);

    // wypisanie wysztkich pętli
    std::cout << "Petle: \n";
    graph_print_loops(g);

    // wypisanie wszystkich krawędzi dwukierunkowych
    std::cout << "Krawedzie dwukierunkowe: \n";
    graph_print_multi_edges(g);

    // zwolnienie pamięci struktury grafu
    free_graph(g);
    return 0;
}
```

7.4 Funkcje pomocnicze

```
// inicjalizacja struktury grafu tablicą wypełnioną zerami
graph new_graph(int size)
{
    if (size < 0) throw "Rozmiar grafu nie moze byc ujemny!\n";
    // stworzenie nowej struktury typu graf
    graph g;
    // przypisanie wymiarów grafu do struktury
    g.size = size;

    // zainicjowanie tablicy wskaźników
    g.array = new int* [size];

    int i, j;
    // dla każdego elementu tablicy
    for (int i = 0; i < size; i++) {
        // zainicjowanie pamięci dla wiersza
        g.array[i] = new int[size];
        // zapełnienie zainicjowanej pamięci zerami
        memset(g.array[i], 0, size * sizeof(int));
    }

    // zwrócenie utworzonej struktury
    return g;
}

// zwolnienie pamięci zajętej przez strukturę grafu
void free_graph(graph& g)
{
    // dla każdego elementu w tablicy
    for (int i = 0; i < g.size; i++)
    {
        // zwolnienie pamięci dla wiersza
        delete[] g.array[i];
    }
    // zwolnienie pamięci tablicy
    delete[] g.array;

    // wyzerowanie elementów w strukturze
    g.array = NULL;
    g.size = 0;
}

// dodanie krawędzi między wierzchołkami a i b do grafu
void graph_add_edge(graph& g, int a, int b)
{
    // jeżeli podane wierzchołki znajduja się w zakresie grafu
    if (a < 0 || a >= g.size || b < 0 || b >= g.size) throw "Podane wierzcholki nie
znajduja sie w grafie!";
    // wpisanie istnienia krawędzi do macierzy sąsiedztwa
    g.array[a][b] = 1;
}

// przeciążenie funkcji graph_add_edge przyjmujące tablicę par numerów wierzchołków
tworzących krawędzie
void graph_add_edge(graph& g, const std::vector<std::pair<int, int>>& edges)
{
    // dla każdej pary wierzchołków
    for (const std::pair<int, int>& edge : edges)
    {
        // utworzenie krawędzi między nimi
        graph_add_edge(g, edge.first, edge.second);
    }
}
```

```
// wypisanie zawartości tablicy relacji między wierzchołkami grafu
void print_graph(const graph& g)
{
    int i, j;
    // wypisanie numerów kolumn grafu
    printf("   ");
    for (i = 0; i < g.size; i++) printf("%3d ", i);
    printf("\n\n");

    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // wypisanie numeru wiersza
        printf("%2d ", i);

        // wypisanie zawartości wiersza
        for (j = 0; j < g.size; j++)
            printf("%3d ", g.array[i][j]);
        printf("\n");
    }
}
```

7.5 Funkcje z treści zadania

```
// wypisanie sąsiadów każdego wierzchołka
void graph_print_neighbors_of_each_vertex(const graph& g)
{
    int i, j, print_count;

    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        print_count = 0;

        // wypisanie numeru wierzchołka
        printf("%2d: ", i);

        // dla każdej kolumny w tablicy
        for (j = 0; j < g.size; j++)
        {
            // jeżeli istnieje krawędź w grafie od wierzchołka i do j
            if (g.array[i][j] == 1)
            {
                // wypisanie numeru wierzchołka j
                printf("%s%2d", print_count++ ? ", " : "", j);
            }
        }
        // wypisanie informacji o braku elementów jeżeli
licznik wypisań jest równy 0
        if (print_count == 0) printf(" brak");
        printf("\n");
    }
}
```

```
// wypisanie każdego wierzchołka, który jest sąsiadem każdego wierzchołka
void graph_print_each_vertex_that_is_neighbor(const graph& g)
{
    int i, j, print_count;

    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        print_count = 0;

        // wypisanie numeru wierzchołka
        printf("%2d: ", i);

        // dla każdej kolumny w tablicy
        for (j = 0; j < g.size; j++)
        {
            // jeżeli istnieje krawędź w grafie od wierzchołka j do i
            if (g.array[j][i] == 1)
            {
                // wypisanie numeru wierzchołka j
                printf("%s%2d", print_count++ ? ", " : "", j);
            }
        }
        // wypisanie informacji o braku elementów jeżeli
licznik wypisań jest równy 0
        if (print_count == 0) printf(" brak");
        printf("\n");
    }
}

// wypisanie stopni wychodzących każdego z wierzchołków
void graph_print_vertex_outdegrees(const graph& g)
{
    int i, j, count;
    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // wypisanie numeru wierzchołka
        printf("%2d: ", i);
        // zerowanie licznika
        count = 0;

        // dla każdej kolumny w tablicy
        for (j = 0; j < g.size; j++)
        {
            // jeżeli istnieje krawędź w grafie od wierzchołka i do j
            if (g.array[i][j] == 1)
                // zwiększenie licznika krawędzi wychodzących z wierzchołka i
                count++;
        }
        // wypisanie licznika
        printf("%d\n", count);
    }
}
```

```
// wypisanie stopni wchodzących każdego z wierzchołków
void graph_print_vertex_indegrees(const graph& g)
{
    int i, j, count;
    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // wypisanie numeru wierzchołka
        printf("%2d: ", i);
        // zerowanie licznika
        count = 0;

        // dla każdej kolumny w tablicy
        for (j = 0; j < g.size; j++)
        {
            // jeżeli istnieje krawędź w grafie od wierzchołka j do i
            if (g.array[j][i] == 1)
                // zwiększenie licznika krawędzi wchodzących do wierzchołka i
                count++;
        }
        // wypisanie licznika
        printf("%d\n", count);
    }
}

// wypisanie wszystkich izolowanych wierzchołków
void graph_print_isolated_vertexes(const graph& g)
{
    int i, j, count, print_count = 0;
    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // zerowanie licznika
        count = 0;

        // dla każdej kolumny w tablicy
        for (j = 0; j < g.size; j++)
        {
            // jeżeli istnieje krawędź w grafie między wierzchołkami i oraz j
            if (g.array[j][i] == 1 || g.array[i][j] == 1)
                // zwiększenie licznika krawędzi wchodzących lub wychodzących z
wierzchołka i
                count++;
        }
        // jeżeli licznik jest równy 0 to dany wierzchołek nie posiada żadnych
krawędzi i jest izolowany
        if (count == 0) printf("%s%2d", print_count++ ? ", " : "", i);
    }
    printf("\n");
}

// wypisanie wszystkich pętli w grafie
void graph_print_loops(const graph& g)
{
    int i, print_count = 0;
    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // jeżeli wierzchołek i łączy się z krawędzią z samym sobą to jest to pętla
        if (g.array[i][i] == 1) printf("%s%2d", print_count++ ? ", " : "", i);
    }

    if (print_count == 0) printf(" brak");
    printf("\n");
}
```

```
// wypisanie wszystkich krawędzi dwukierunkowych w grafie
void graph_print_multi_edges(const graph& g)
{
    int i, j, print_count = 0;
    // dla każdego elementu w tablicy
    for (i = 0; i < g.size; i++)
    {
        // dla każdej kolumny w tablicy pomijając elementy już sprawdzone przez
poprzednie iteracje oraz wierzchołek i
        for (j = i + 1; j < g.size; j++)
        {
            // jeżeli istnieją krawędzie w obu kierunkach między wierzchołkami i
oraz j
            if (g.array[i][j] == 1 && g.array[j][i] == 1)
                // wypisanie tej pary wierzchołków
                printf("%s%d <-> %d", print_count++ ? ", " : "", i, j);
        }
    }

    if (print_count == 0) printf(" brak");
    printf("\n");
}
```