Data : 14. 01 . 2022 r.

Politechnika Rzeszowska im. Ignacego Łukasiewicza
Wydział Matematyki i Fizyki Stosowanej
Kierunek Inżynieria i Analiza Danych
Gr. Laboratoryjna nr 4
Nazwa przedmiotu :
Projekt – Algorytmy i struktury Danych

Temat projektu:

Program wyznaczający informacje dla zadanego grafu skierowanego

Anna Turek

Nr albumu: 169856

1. Realizacja projektu

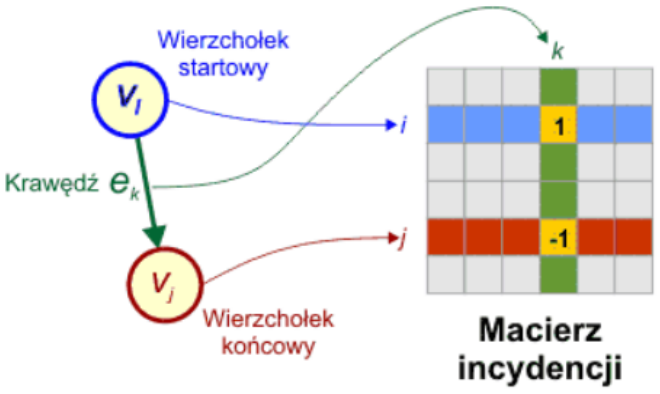
Program ma za zadanie wyznaczyć i wypisać dla zadanego grafu skierowanego reprezentowanego przy pomocy macierzy incydencji następujące informacje :

- wszystkich sąsiadów dla każdego wierzchołka grafu
- wszystkie wierzchołki, które są sąsiadami każdego wierzchołka
- stopnie wychodzące wszystkich wierzchołków
- stopnie wchodzące wszystkich wierzchołków
- wszystkie wierzchołki izolowane
- wszystkie pętle
- wszystkie krawędzie dwukierunkowe

2. Informacje o macierzy incydencji

Macierz incydencji (ang. incidence matrix) jest macierzą A o wymiarze $n \times m$, gdzie n oznacza liczbę wierzchołków grafu, a m liczbę jego krawędzi. Każdy wiersz tej macierzy odwzorowuje jeden wierzchołek grafu. Każda kolumna odwzorowuje jedną krawędź. Zawartość komórki $A[i, j]$ określa powiązanie (incydencję) wierzchołka v_i z krawędzią e_j w sposób następujący:

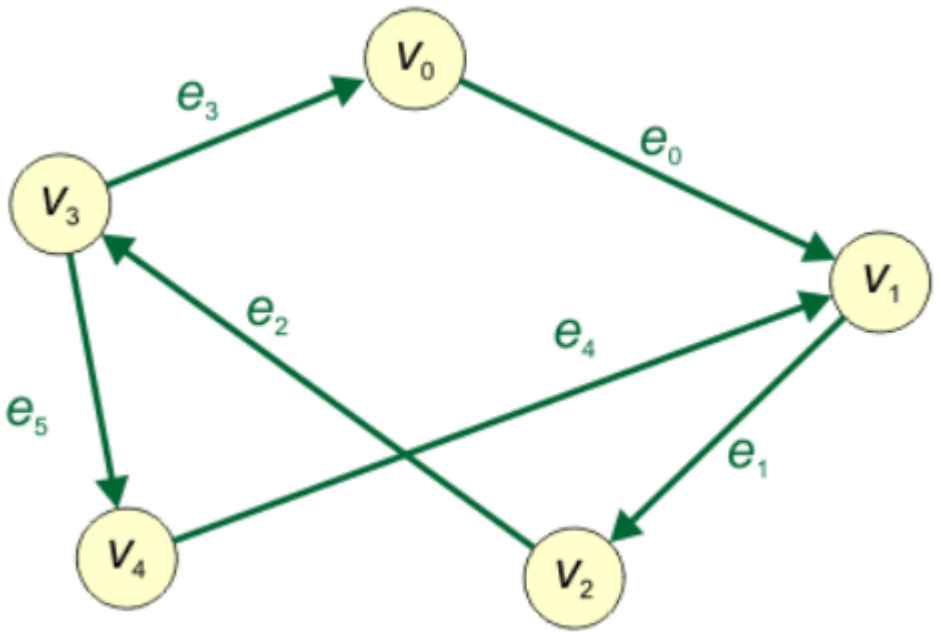
$$A[i,j] = \begin{cases} 0, \text{ jeśli } v_i \text{ nie należy do } e_j \\ 1, \text{ jeśli } v_i \text{ jest początkiem } e_j \\ -1, \text{ jeśli } v_i \text{ jest końcem } e_j \end{cases}$$



3. Informacje o grafie skierowanym

Graf skierowany (digraf)– zbiór wierzchołków i zbiór krawędzi skierowanych łączących (co najwyżej jeden raz) uporządkowane pary wierzchołków. Mówimy wtedy, że krawędź łączy pierwszy wierzchołek z drugim (albo, że prowadzi od pierwszego wierzchołka do drugiego).

Przykład grafu skierowanego:



4. Analiza funkcji w kodzie

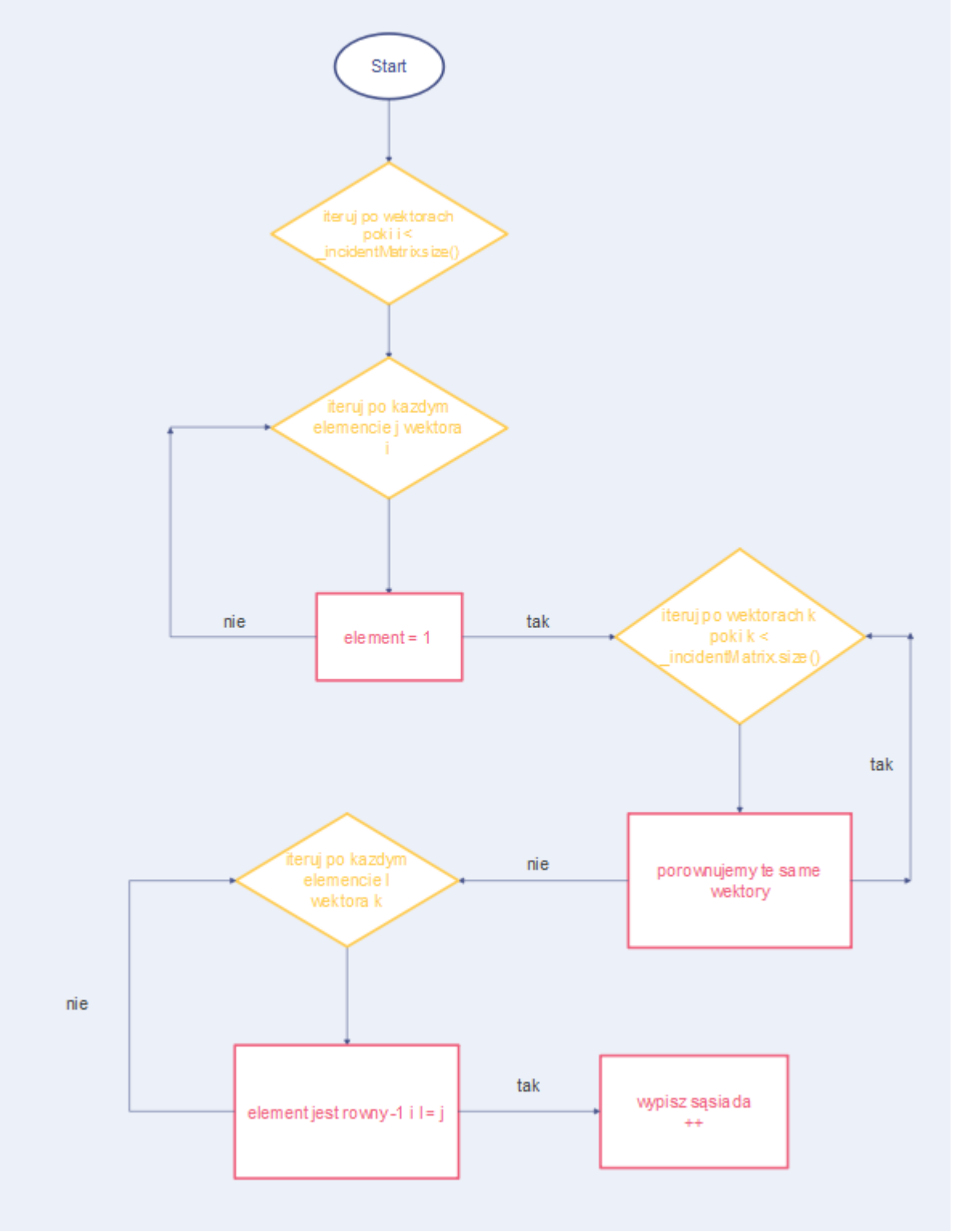
4.1 Wyszukiwanie wszystkich sąsiadów dla każdego wierzchołka grafu

Cel : wyszukujemy w każdym wektorze kolumny macierzy incydencji źródła, czyli 1 oraz destynacji, czyli -1.Gdy już znajdziemy taki przypadek mamy pewność, że istnieje w grafie wierzchołek, który ma sąsiada. Przy tym wierzchołku (wektorze wierszy macierzy), który zawiera 1 zastaniemy początek (źródło) krawędzi, a przy wierzchołku, przy którym zastaniemy -1, będzie koniec krawędzi. Analizujemy w ten sposób wektor każdej kolumny.

4.1.1 Pseudokod:

```
wczytaj (_incidentMatrix)
void GetAllNeighboursOfSpecificVertex()
wczytaj (counter)
      i < _incidentMatrix.size()
          counter = 0
      j < _incidentMatrix[i].size()
          jeżeli _incidentMatrix[i][j] == 1
    k < _incidentMatrix.size();
          jeżeli k==1
                kontynuuj
          w innym razie
                counter == 0
                 j == _incidentMatrix[i].size() - 1
    l < _incidentMatrix[k].size()
          jeżeli _incidentMatrix[k][l] == -1 && j == l
                inkrementuj counter
```

4.1.2 Schemat blokowy dla każdego wierzchołka grafu:



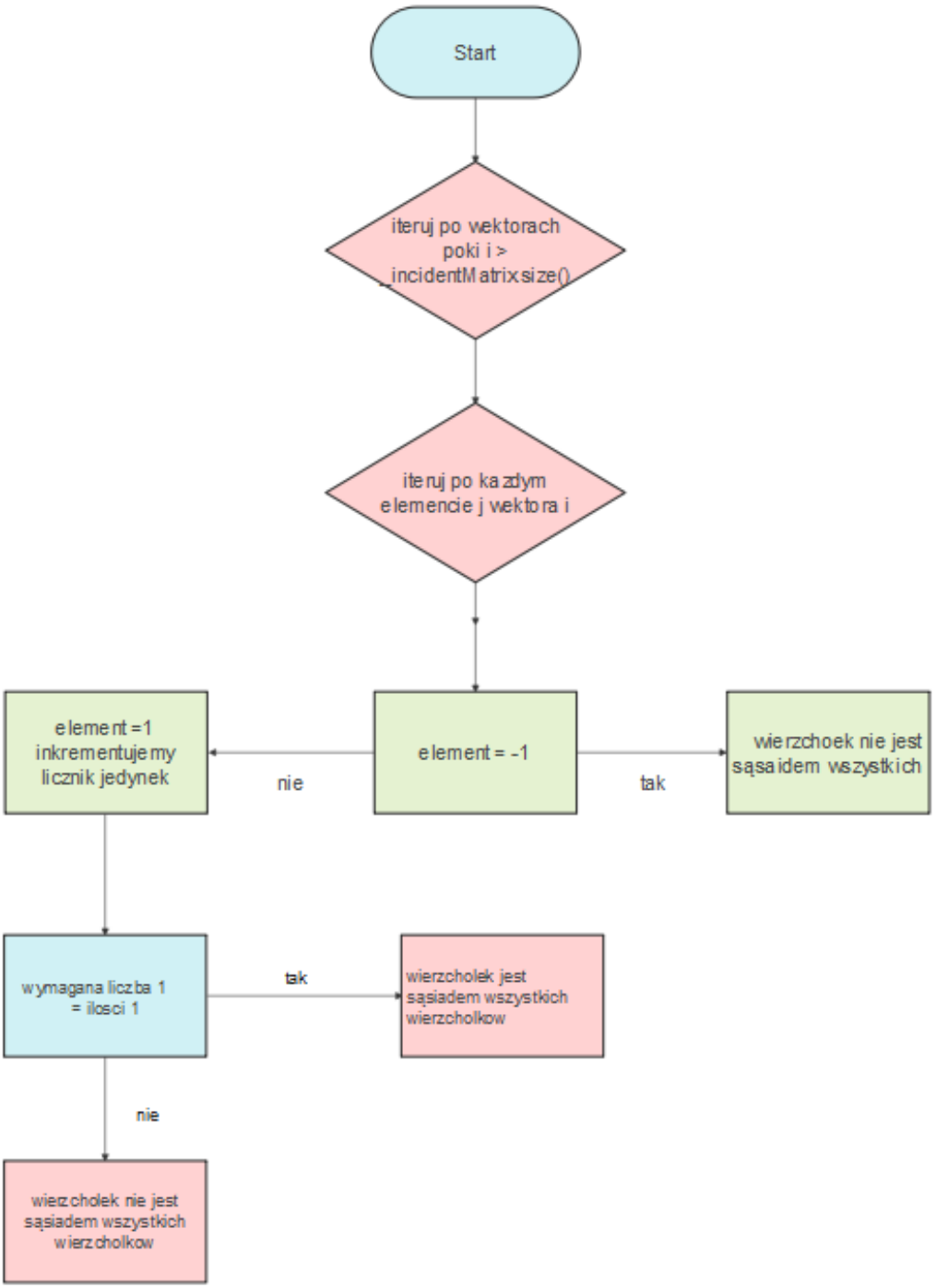
4.2 Wyszukiwanie wszystkich wierzchołków, które są sąsiadami każdego wierzchołka

Cel: Wyszukujemy wektora wierzchołka takiego, żeby miał tyle 1, ile jest pozostałych wierzchołków. Co więcej w wektorze kolumny do każdej z tej 1 musi być przypisana -1 każdego innego wierzchołka. W podanej macierzy -1 przyjmują formę przekątnej, co dobrze pokazuje, że jeden z wierzchołków jest sąsiadem wszystkich wierzchołków.

4.2.1 Pseudokod:

```
void GetAllVertexesWhereInNeighoudOfAllVertexes()
wczytaj ( counter = 0 )
wczytaj ( requiredNumberOfOnes = _incidentMatrix.size() - 1)
       wczytaj( counterOfOnesInRow = 0)
bool minusValueHappened = false;
     i < _incidentMatrix.size()
          minusValueHappened = false
          minusValueHappened = false
     j < _incidentMatrix[i].size()
          jeżeli _incidentMatrix[i][j] == -1
                          minusValueHappened = true
                                          break
                    w innym razie _incidentMatrix[i][j] == 1
               inkrementuj counterOfOnesInRow
     jeżeli counterOfOnesInRow == requiredNumberOfOnes &&
     !minusValueHappened
          inkrementuj counter
     jeżeli counter == 0
          wypisz “There is no vertex which is neighbour to
     all…”
```

4.2.2. Schemat blokowy do sąsiadów każdego wierzchołka:



4.3 Wyszukiwanie stopni wychodzących i wchodzących wszystkich wierzchołków

Cel: Stopień wejściowy to liczba krawędzi wchodzących do wierzchołka. Liczymy go obliczając ilość elementów wiersza równych -1. Podobnie stopień wyjściowy to liczba krawędzi wychodzących z wierzchołka, którą znajdziemy obliczając ilość elementów równych 1.

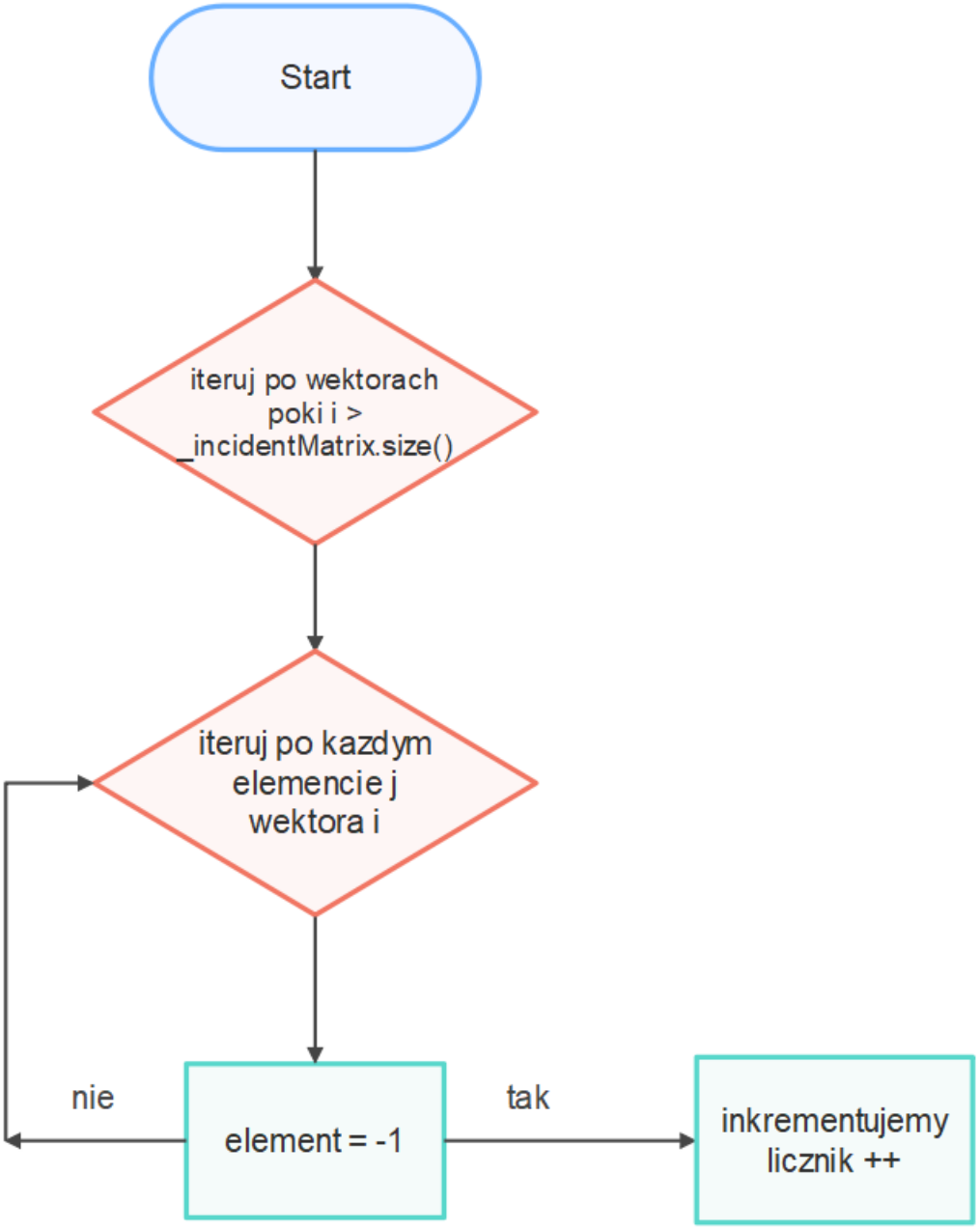
4.3.1 Pseudokod do stopni wychodzących:

```
void GetDegreesOutFromAllVertexes()
wczytaj (counter = 0)
      i < _incidentMatrix.size()
            counter = 0
      j < _incidentMatrix[i].size()
            jeżeli _incidentMatrix[i][j] == -1
                  inkrementuj counter
```

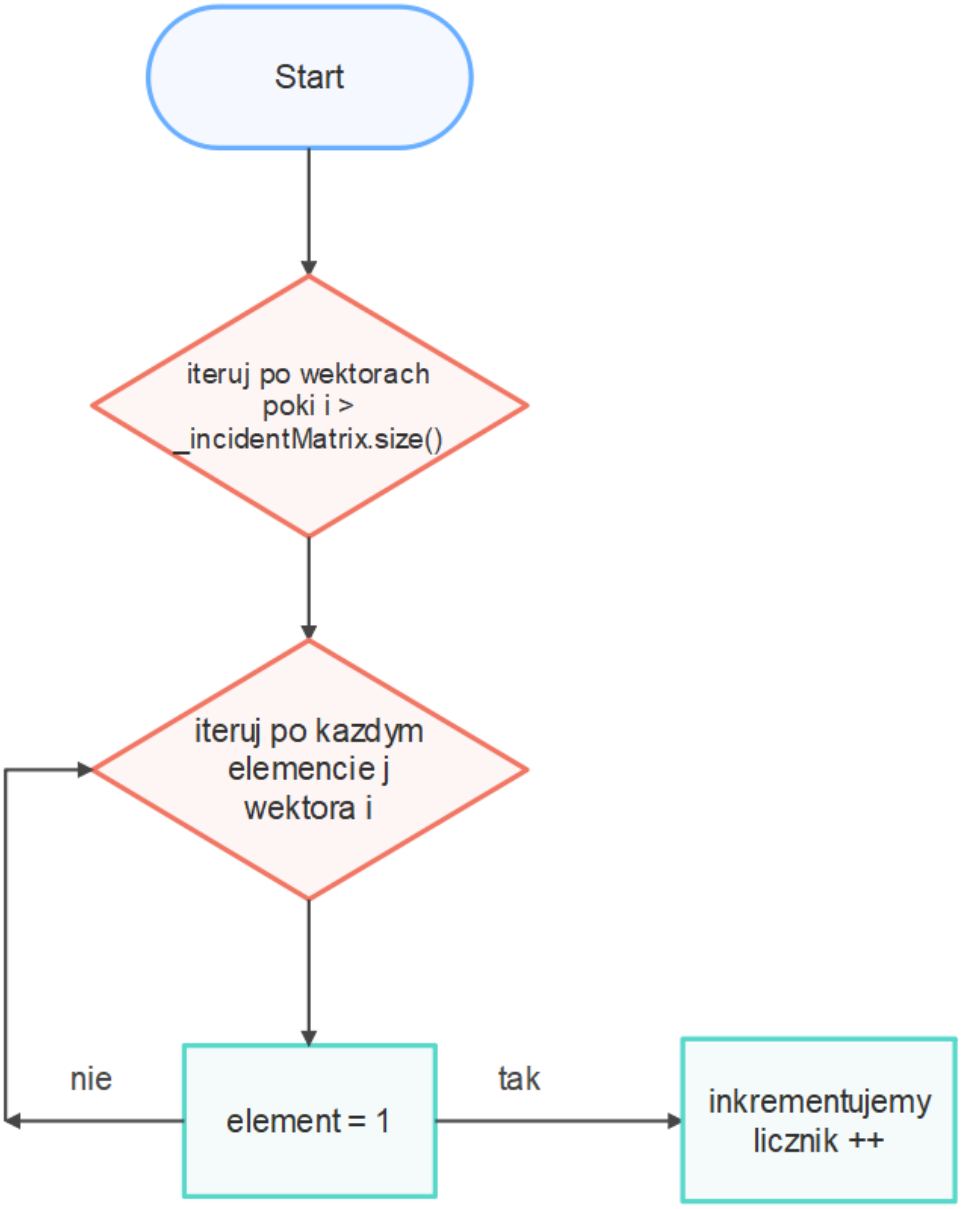
4.3.2 Pseudokod do stopni wchodzących:

```
void GetDegreesInToAllVertexes()
wczytaj (counter = 0)
      i < _incidentMatrix.size()
            counter = 0
      j < _incidentMatrix[i].size()
            jeżeli _incidentMatrix[i][j] == 1
                  inkremetnuj counter
```

4.3.3 Schemat blokowy do wychodzących stopni wierzchołków:



4.3.4 Schemat blokowy do wchodzących stopni wierzchołków:



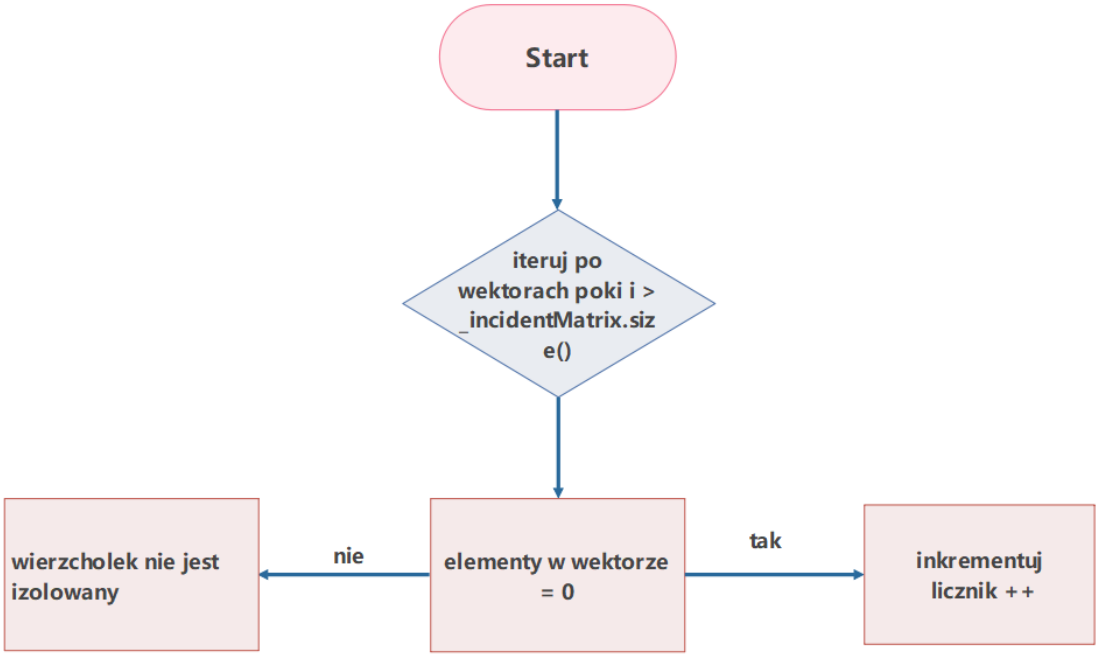
4.4 Wyszukiwanie wszystkich wierzchołków izolowanych

Cel: Wyszukujemy wektor wierzchołka macierzy, który nie ma styczności poprzez krawędź z żadnym innym wierzchołkiem. W tym celu musimy znaleźć wiersz, który składa się z samych 0.

4.4.1 Pseudokod:

```
void GetAllIsolatedVertexes()
wczytaj (counter = 0)
      i < _incidentMatrix.size()
            bool allElementsAreZeros
            jeżeli (allElementsAreZeros)
            inkrementuj counter
      jeżeli (counter == 0)
```

4.4.2 Schemat blokowy do wierzchołków izolowanych:



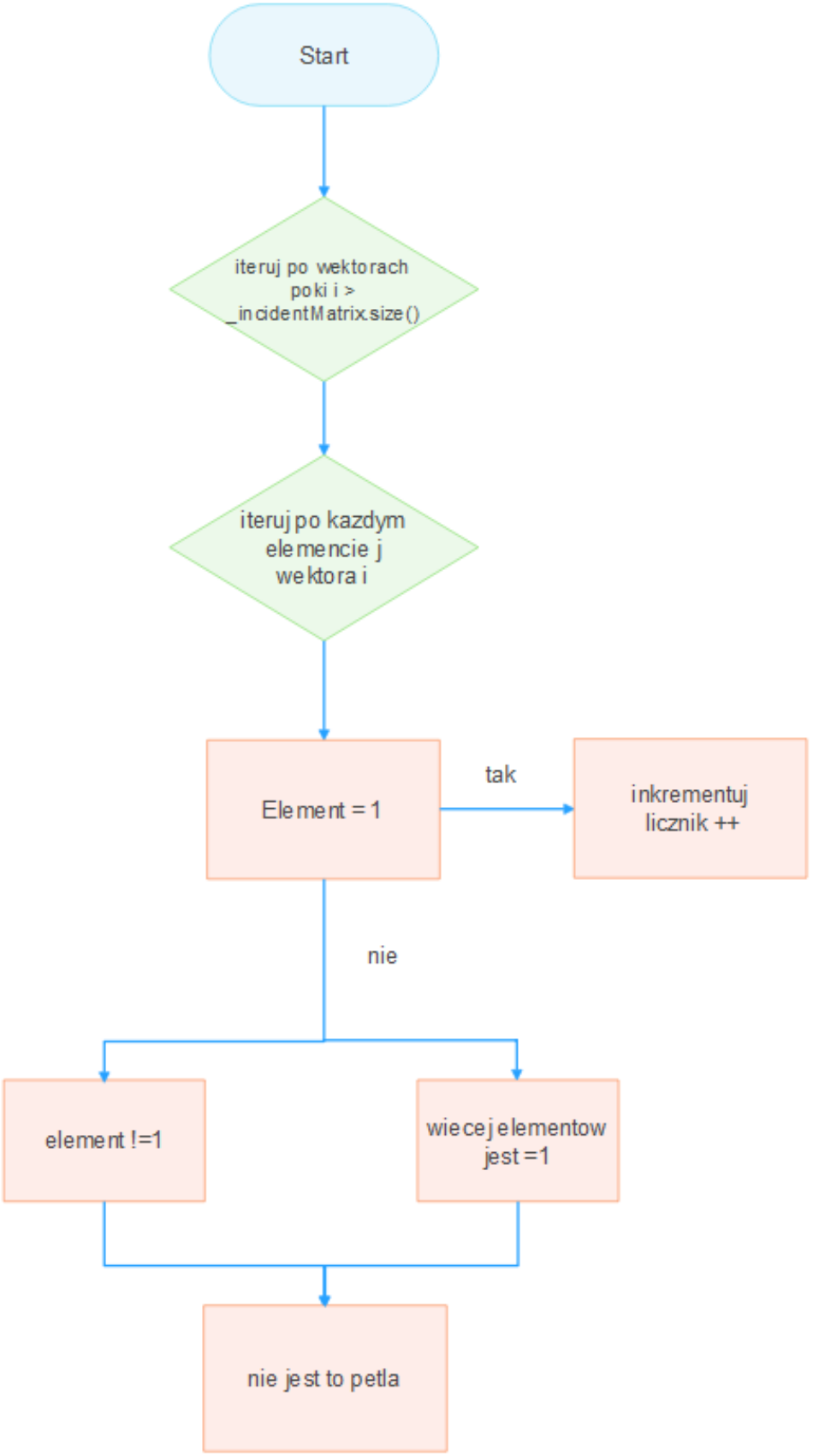
4.5 Wyszukiwanie wszystkich pętli

Cel: Wyszukujemy w kolumnie 1, która nie będzie miała wektora kolumny odpowiadającego za destynacje krawędzi.

4.5.1 Pseudokod:

```
wczytaj  (counter = 0)
wczytaj ( counterOfOnesInColumn = 0 )
bool isLooped = true
wczytaj ( numberOfColumns = _incidentMatrix[0].size() )
      i < numberOfColumns
            isLooped = true
            counterOfOnesInColumn = 0
      j < _incidentMatrix.size()
            jeżeli _incidentMatrix[j][i] == 1
                  inkrementuj counterOfOnesInColumn
            w innym razie
                        _incidentMatrix[j][i] == -1 lub
                  counterOfOnesInColumn > 1
                        isLooped = false
                        break
      jeżeli (isLooped)
                  inkrementuj counter
      jeżeli (counter == 0)
            wypisz "There is not any loop..."
```

4.5.2 Schemat blokowy do pętli:



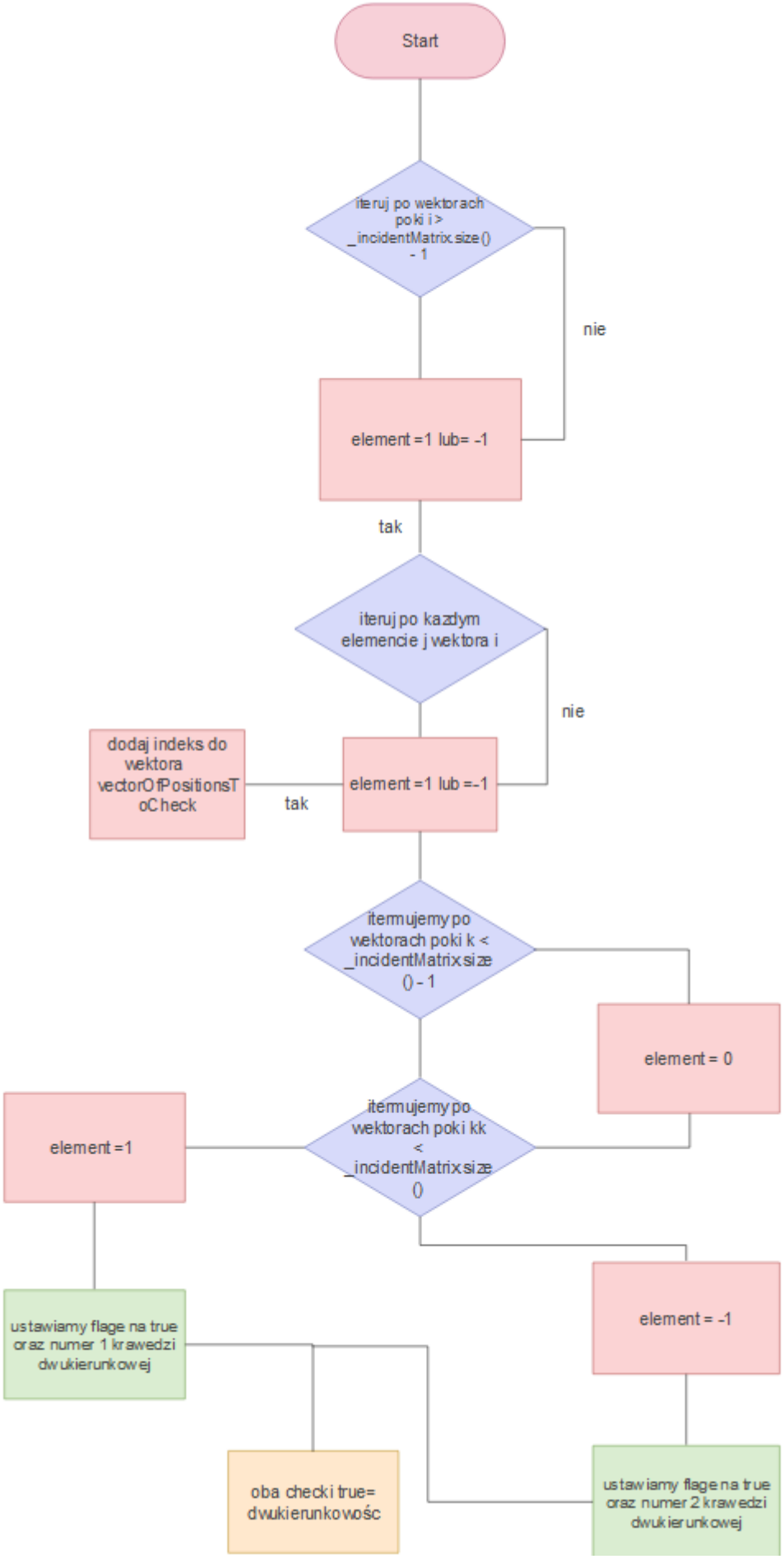
4.6 Wyszukiwanie wszystkich krawędzi dwukierunkowych

Cel: Wyszukujemy dwóch wierzchołków, które są swoimi wzajemnymi sąsiadami. W tym celu szukamy wierzchołków

4.6.1 Pseudokod:

```
void GetAllTwoWayEdges()
wczytaj( numberOfRowsToCompare = _incidentMatrix.size() - 1)
wczytaj (numberOfColumns = _incidentMatrix[0].size())
bool checkOfOnes = false
bool checkOfMinusOnes = false
wczytaj wektor (vectorOfPositionsToCheck)
     i < _incidentMatrix.size() - 1
          jeżeli  (vectorOfPositionsToCheck) ->! 1 lub !-1
               kontynuuj
     j < _incidentMatrix[i].size()
          jeżeli _incidentMatrix[i][j] == 1 lub
          _incidentMatrix[i][j] == -1
wczytaj (twoDirectionalEdge1 = -1)
wczytaj  (twoDirectionalEdge2 = -1)
     k < _incidentMatrix.size() - 1
          twoDirectionalEdge1 = -1
          twoDirectionalEdge2 = -1
          checkOfOnes = false
          checkOfMinusOnes = false
     kk < vectorOfPositionsToCheck.size()
          jeżeli
incidentMatrix[i][vectorOfPositionsToCheck[kk]] ==
-_incidentMatrix[k + 1][vectorOfPositionsToCheck[kk]
               jeżeli _incidentMatrix[k +
          1][vectorOfPositionsToCheck[kk]] == 1
                    checkOfOnes = true;
                         twoDirectionalEdge1 =
                    vectorOfPositionsToCheck[kk]
               w innym razie _incidentMatrix[k +
          1][vectorOfPositionsToCheck[kk]] == -1
               checkOfMinusOnes = true
               twoDirectionalEdge2 =
          vectorOfPositionsToCheck[kk]
          jeżeli (checkOfOnes && checkOfMinusOnes)
               wypisz “ Between V i and V k + 1 edges: E and E
          are two-directional
```

4.6.2 Schemat blokowy dwukierunkowości:



5. Użyte w programie macierze incydencji

{1, 1, 0, 0, 0, 0, -1, 0, -1, 0, 0, 0, 0},
{0, -1, 1, 1, -1, 0, 0, 0, 0, -1, 0, 0, 0},
{0, 0, -1, 0, 0, 1, 0, 0, 0, 0, -1, 0, 0},
{0, 0, 0, -1, 1, 0, 0, -1, 0, 0, 0, -1, 0},
{0, 0, 0, 0, 0, -1, 1, 1, 0, 0, 0, 0, -1},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1}

{1, 1, 0, 0, 0, 0, -1, 0, -1, 0, 0, 0, 0},
{0, -1, 1, 1, -1, 0, 0, 0, 0, -1, 0, 0, 0},
{0, 0, -1, 0, 0, 1, 0, 0, 0, 0, -1, 0, 0},
{0, 0, 0, -1, 1, 0, 0, -1, 0, 0, 0, -1, 0},
{0, 0, 0, 0, 0, -1, 1, 1, 0, 0, 0, 0, -1},
{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1}

Pierwsza macierz zawiera w sobie wierzchołek izolowany, a druga wierzchołek, który jest sąsiadem wszystkich wierzchołków.

Żródła:

https://eduinf.waw.pl/inf/alg/001_search/0124.php

Link do repozytorium:

https://github.com/AnnaTurek123/Projekt3/tree/main/Projekt3/Projekt3