

POLITECHNIKA WARSZAWSKA

**WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH**

ANALIZA ALGORYTMÓW MST II

Autorzy:

Joanna Maciak

Kacper Dominik

Prowadzący: dr inż. Sebastian Kozłowski

Warszawa 2018

Spis treści

1. Sprawozdanie 1	3
2. Sprawozdanie 2	4
2.1. Algorytm Prima	4
2.1.1. Pseudokod zaimplementowanego algorytmu	4
2.1.2. Zasada działania zaimplementowanego algorytmu na przykładzie . . .	6
2.2. Składowe implementacji algorytmu	15
2.3. Algorytm Kruskala	17
Spis załączników	18
1. Kod źródłowy algorytmu Prima	18
2. Kod źródłowy modułu odpowiedzialnego za odczyt macierzy z pliku	18
3. Link do repozytorium z kodem źródłowym generatora macierzy sąsiedztwa . .	18
4. Kod źródłowy algorytmu Kruskala	18
Bibliografia	19
Spis rysunków	20
Spis tabel	20

1. Założenia projektowe

Tematem projektu jest **przeprowadzenie eksperymentalnej analizy czasu wykonywania dwóch algorytmów wyznaczania najłżejszego drzewa rozpinającego dla różnych klas grafów.**

Wybranymi algorytmami, będącymi przedmiotem tematu niniejszego projektu, są:

- Algorytm Kruskala
- Algorytm Prima

Językiem programowania, wybranym do implementacji powyższych algorytmów jest język Java.

2. Opis i implementacja algorytmów

Problem (zdefiniowany na podstawie [2]):

Znalezienie takiego podzbioru krawędzi spójnego grafu nieskierowanego ważonego, który zapewnia połączenie każdego wierzchołka grafu z dowolnym innym i ponadto posiada najmniejszą możliwą sumę wag krawędzi.

Taki podzbiór nie może zawierać żadnego cyklu, a zatem jest drzewem i musi zawierać dokładnie $n-1$ krawędzi dla grafu o n wierzchołkach. Drzewo takie ze względu na minimalną sumę wag nazywane jest minimalnym drzewem rozpinającym.

2.1. Algorytm Prima

Dane wejściowe:

Spójny ważony graf nieskierowany, zawierający n wierzchołków, gdzie $n \geq 2$.

Graf jest reprezentowany przez macierz sąsiedztwa *graph*, gdzie element *graph[i][j]* reprezentuje wagę krawędzi łączącej wierzchołki *i* oraz *j*.

2.1.1. Pseudokod zaimplementowanego algorytmu

1. Inicjalizacja

- Utwórz pusty zbiór zawierający krawędzie minimalnego drzewa rozpinającego: *minSpanningTree*.
- Zainicjalizuj pustą listę odwiedzonych wierzchołków: *visitedVertices*.
- Utwórz pustą listę krawędzi incydentnych do odwiedzonych wierzchołków: *adjacencyEdges*.

2. Losuj wierzchołek początkowy: *startV*.

3. Oznacz wierzchołek początkowy jako odwiedzony poprzez dodanie go do listy wierzchołków odwiedzonych *visitedVertices*.
4. Pobierz do zmiennej *adjacencyEdges* posortowaną względem wag listę wszystkich krawędzi incydentnych do wierzchołka *startV*.
5. Dopóki lista *visitedVertices* nie zawiera wszystkich wierzchołków grafu:
 - a. Zainicjalizuj zmienną aktualnie rozpatrywanej krawędzi – *minEdge* – poprzez pobranie najmniejszego elementu listy *adjacencyEdges*.
 - b.1. Jeśli lista odwiedzonych wierzchołków *visitedVertices* nie zawiera jeszcze wierzchołka końcowego aktualnie rozpatrywanej krawędzi *minEdge*:
 - dodaj krawędź *minEdge* do zbioru krawędzi minimalnego drzewa rozpinającego *minSpanningTree*
 - przypisz zmiennej *startV* wartość wierzchołka końcowego krawędzi *minEdge*
 - oznacz wierzchołek końcowy krawędzi *minEdge* jako odwiedzony
 - usuń krawędź uwzględnioną w *minSpanningTree* – *minEdge* – z listy krawędzi incydentnych do odwiedzonych wierzchołków – ustal aktualną listę nietworzących cyklu krawędzi incydentnych do wierzchołków odwiedzonych
 - b.2. Jeśli lista odwiedzonych wierzchołków *visitedVertices* zawiera już wierzchołek końcowy aktualnie rozpatrywanej krawędzi *minEdge*:
 - usuń krawędź tworzącą cykl z listy krawędzi incydentnych do odwiedzonych wierzchołków

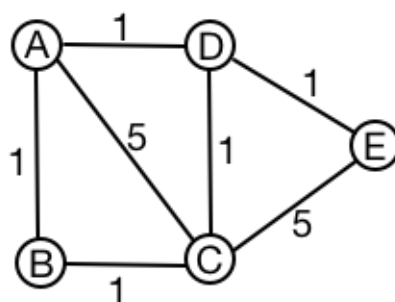
2.1.2. Zasada działania zaimplementowanego algorytmu na przykładzie

Poniżej znajduje się legenda oznaczeń wykorzystanych w opisie działania algorytmu na przykładzie.



rys. 2.1. Oznaczenia wykorzystane w opisie

Dany jest graf widoczny na rys.2.2. Wierzchołki oznaczono wielkimi literami alfabetu, natomiast krawędziom nadano wagi $w \in \{1, 5\}$.



rys. 2.2. Graf wejściowy.

Inicjalizacja

Zbiór: *minSpanningTree* = { };

Lista wierzchołków odwiedzonych: *visitedVertices* = [];

Lista krawędzi incydentnych: *adjacencyEdges* = [];

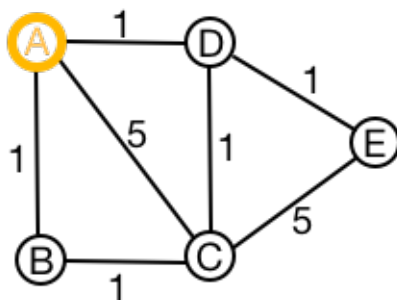
Krok 1.

Losowanie wierzchołka początkowego. W tym przypadku niech będzie to wierzchołek A.

$$startV = A$$

Krok 2.

Oznaczenie wierzchołka $startV$ jako odwiedzonego.



rys. 2.3. Prim – Krok 2.

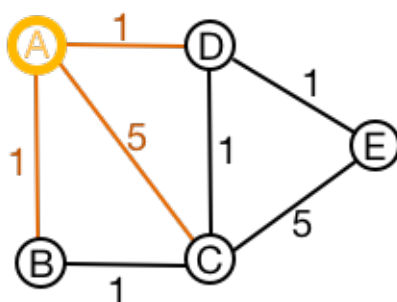
Zbiór: $minSpanningTree = \{ \}$;

Lista wierzchołków odwiedzonych: $visitedVertices = [A]$;

Lista krawędzi incydentnych: $adjacencyEdges = []$;

Krok 3.

Pobranie listy krawędzi incydentnych do wierzchołka $startV$, które nie tworzą cyklu.



rys. 2.4. Prim – Krok 3.

Zbiór: $minSpanningTree = \{ \}$;

Lista wierzchołków odwiedzonych: $visitedVertices = [A]$;

Lista krawędzi incydentnych: $adjacencyEdges = [(A - D), (A - B), (A - C)]$;

Krok 4.

Inicjalizacja $minEdge$ poprzez wybór pierwszego elementu posortowanej rosnąco listy krawędzi incydentnych $adjacencyEdges$.

$$minEdge = (A - D)$$

Krok 5.

Lista $visitedVertices$ nie zawiera jeszcze wierzchołka końcowego krawędzi $minEdge$ (D), dlatego:

- Brak cyklu z krawędzią $minEdge$
- Dodanie krawędzi $minEdge$ do zbioru krawędzi minimalnego drzewa rozpinającego $minSpanningTree$

$$\text{Zbiór: } minSpanningTree = \{ (A - D) \};$$

- przypisanie zmiennej $startV$ wartości wierzchołka końcowego krawędzi $minEdge$

$$startV = D$$

- oznaczenie wierzchołka końcowego krawędzi $minEdge$ jako odwiedzony
- usunięcie krawędzi uwzględnionej w $minSpanningTree - minEdge$ – z listy krawędzi incydentnych do odwiedzonych wierzchołków $adjacencyEdges$

Lista wierzchołków odwiedzonych: $visitedVertices = [A, D]$;

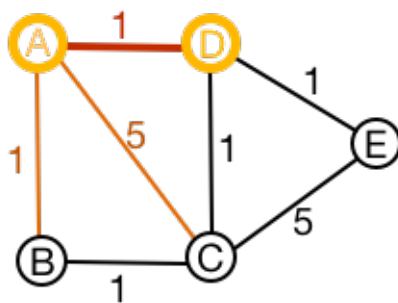
Lista krawędzi incydentnych: $adjacencyEdges = [(A - B), (A - C)]$;

- aktualizacja listy krawędzi incydentnych do odwiedzonych wierzchołków $adjacencyEdges$

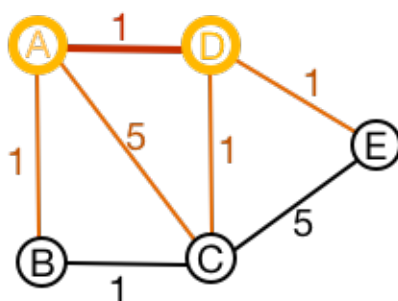
Lista krawędzi incydentnych: $adjacencyEdges = [(A - B), (D - C), (D - E), (A - C)]$;

Krok 6.

Aktualizacja $minEdge$ poprzez wybór pierwszego elementu posortowanej rosnąco listy krawędzi incydentnych $adjacencyEdges$.



rys. 2.5. Prim – Krok 5.



rys. 2.6. Prim – Krok 5 - aktualizacja listy krawędzi.

$$\text{minEdge} = (A - B)$$

Krok 7.

Lista *visitedVertices* nie zawiera jeszcze wierzchołka końcowego krawędzi *minEdge* (B), dlatego:

- Brak cyklu z krawędzią *minEdge*
- Dodanie krawędzi *minEdge* do zbioru krawędzi minimalnego drzewa rozpinającego *minSpanningTree*

$$\text{Zbiór: } \text{minSpanningTree} = \{ (A - D), (A - B) \};$$

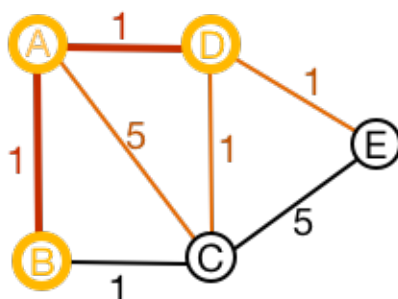
- przypisanie zmiennej *startV* wartości wierzchołka końcowego krawędzi *minEdge*

$$\text{startV} = B$$

- oznaczenie wierzchołka końcowego krawędzi *minEdge* jako odwiedzony
- usunięcie krawędzi uwzględnionej w *minSpanningTree* – *minEdge* – z listy krawędzi incyden-
nych do odwiedzonych wierzchołków *adjacencyEdges*

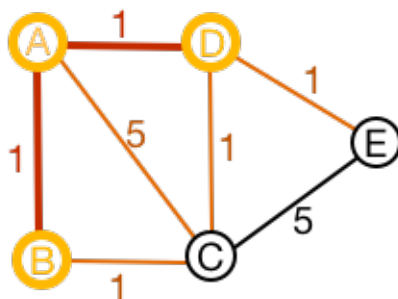
Lista wierzchołków odwiedzonych: *visitedVertices* = [A , D , B];

Lista krawędzi incyden-nych: *adjacencyEdges* = [(D – C), (D – E), (A – C)];



rys. 2.7. Prim – Krok 7.

- aktualizacja listy krawędzi incyden-nych do odwiedzonych wierzchołków *adjacencyEdges*



rys. 2.8. Prim – Krok 7 - aktualizacja listy krawędzi.

Lista krawędzi incyden-nych: *adjacencyEdges* = [(B – C), (D – C), (D – E), (A – C)];

Krok 8.

Aktualizacja *minEdge* poprzez wybór pierwszego elementu posortowanej rosnąco listy krawędzi incyden-nych *adjacencyEdges*.

$$\text{minEdge} = (B - C)$$

Krok 9.

Lista *visitedVertices* nie zawiera jeszcze wierzchołka końcowego krawędzi *minEdge* (C), dlatego:

- Brak cyklu z krawędzią *minEdge*
- Dodanie krawędzi *minEdge* do zbioru krawędzi minimalnego drzewa rozpinającego *minSpanningTree*

$$\text{Zbiór: } \text{minSpanningTree} = \{ (A - D), (A - B), (B - C) \};$$

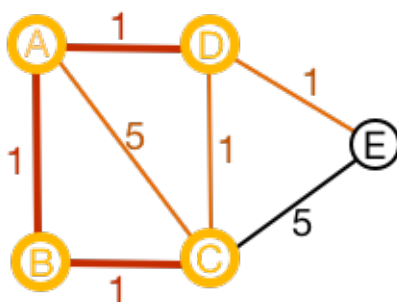
- przypisanie zmiennej *startV* wartości wierzchołka końcowego krawędzi *minEdge*

$$\text{startV} = C$$

- oznaczenie wierzchołka końcowego krawędzi *minEdge* jako odwiedzony
- usunięcie krawędzi uwzględnionej w *minSpanningTree* – *minEdge* – z listy krawędzi incydentnych do odwiedzonych wierzchołków *adjacencyEdges*

Lista wierzchołków odwiedzonych: *visitedVertices* = [A , D , B, C];

Lista krawędzi incydentnych: *adjacencyEdges* = [(D - C), (D - E), (A - C)];



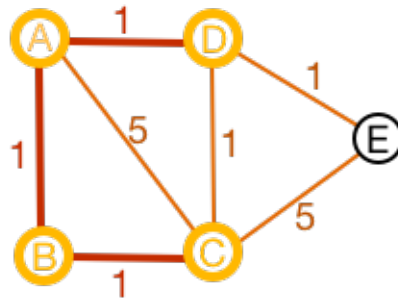
rys. 2.9. Prim – Krok 9.

- aktualizacja listy krawędzi incydentnych do odwiedzonych wierzchołków *adjacencyEdges*

Lista krawędzi incydentnych: *adjacencyEdges* = [(D - C), (D - E), (A - C), (C - E)];

Krok 10.

Aktualizacja *minEdge* poprzez wybór pierwszego elementu posortowanej rosnąco listy krawędzi



rys. 2.10. Prim – Krok 9 - aktualizacja listy krawędzi.

incydentnych *adjacencyEdges*.

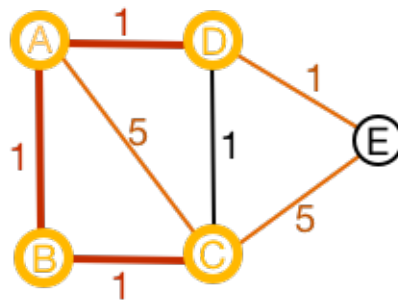
$$\text{minEdge} = (D - C)$$

Krok 11.

Lista *visitedVertices* zawiera wierzchołek krawędzi *minEdge* (C), dlatego:

– Usunięcie krawędzi *minEdge* z listy krawędzi incydentnych do odwiedzonych wierzchołków

Lista krawędzi incydentnych: *adjacencyEdges* = [(D – E), (A – C), (C – E)];



rys. 2.11. Prim – Krok 11.

Krok 12.

Aktualizacja *minEdge* poprzez wybór pierwszego elementu posortowanej rosnąco listy krawędzi incydentnych *adjacencyEdges*.

$$\text{minEdge} = (D - E)$$

Krok 13.

Lista *visitedVertices* nie zawiera jeszcze wierzchołka końcowego krawędzi *minEdge* (E), dlatego:

- Brak cyklu z krawędzią *minEdge*
- Dodanie krawędzi *minEdge* do zbioru krawędzi minimalnego drzewa rozpinającego *minSpanningTree*

$$\text{Zbiór: } \text{minSpanningTree} = \{ (A - D), (A - B), (B - C), (D - E) \};$$

- przypisanie zmiennej *startV* wartości wierzchołka końcowego krawędzi *minEdge*

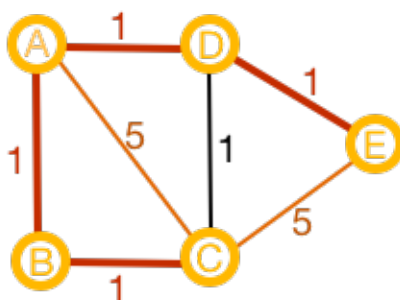
$$\text{startV} = E$$

- oznaczenie wierzchołka końcowego krawędzi *minEdge* jako odwiedzony
- usunięcie krawędzi uwzględnionej w *minSpanningTree* – *minEdge* – z listy krawędzi incydentnych do odwiedzonych wierzchołków *adjacencyEdges*

Lista wierzchołków odwiedzonych: *visitedVertices* = [A , D , B , C , E];

Lista krawędzi incydentnych: *adjacencyEdges* = [(C - E), (A - C)];

- aktualizacja listy krawędzi incydentnych do odwiedzonych wierzchołków *adjacencyEdges*

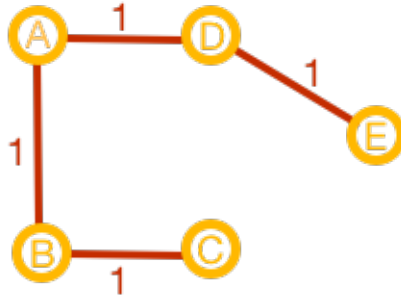


rys. 2.12. Prim – Krok 13.

Lista krawędzi incydentnych: *adjacencyEdges* = [(C - E), (A - C)];

Krok 14.

Lista *visitedVertices* zawiera wszystkie wierzchołki grafu wejściowego. W związku z tym otrzymano następujące drzewo rozpinające:



rys. 2.13. Prim – Krok 14 – otrzymane MST.

2.2. Składowe implementacji algorytmu

Składowa	Rodzaj	Opis
rand	Zmienna pomocnicza	Element odpowiedzialny za losowy wybór wierzchołka początkowego
startV	Zmienna typu int	Wierzchołek początkowy minimalnej krawędzi z listy <i>adjacencyEdgesList</i> . Na początku przyjmuje wartość losową.
visitedVertices	Lista elementów typu Integer	Lista odwiedzonych wierzchołków grafu
adjacencyEdgesList	Lista elementów typu Edge ArrayList<Edge>	Lista krawędzi incydentnych do odwiedzonych wierzchołków grafu
minSpanningTree	Zbiór Set<Edge>	Zbiór krawędzi, tworzących minimalne drzewo rozpinające
minEdge	Zmienna typu Edge	Obiekt klasy <i>Edge</i> (krawędź), zawierający minimalną krawędź z listy <i>adjacencyEdgesList</i>
setVertexVisited(int vertex, ArrayList<Integer> list)	metoda	Metoda dodająca wierzchołek <i>vertex</i> do listy wierzchołków odwiedzonych <i>list</i> jeśli lista ta nie zawiera wierzchołka <i>vertex</i>
getAllAdjacencyEdges(int vertex, int[][] graph)	Metoda zwracająca listę obiektów Edge	Metoda odpowiedzialna za ustalenie listy krawędzi incydentnych do odwiedzonych punktów <i>adjacencyEdgesList</i> . Dla każdego wierzchołka $i \subseteq V$ grafu weryfikuje czy dany wierzchołek <i>i</i> tworzy krawędź z wierzchołkiem <i>vertex</i> . Jeśli krawędź taka istnieje i lista <i>visitedVertices</i> nie zawiera wierzchołka <i>i</i> , to do listy <i>adjacencyEdgesList</i> zostaje dodana krawędź (<i>vertex</i> – <i>i</i>)
minSpanning-Tree.add(minEdge)	Operacja na zbiorze minSpanningTree	Operacja polegająca na dodaniu do zbioru <i>minSpanningTree</i> minimalnej krawędzi <i>minEdge</i>

tab. 2.1. Składowe implementacji algorytmu Prima

Składowa	Rodzaj	Opis
adjacencyEdges.remove(minEdge)	Operacja na liście <i>adjacencyEdges</i>	Operacja polegająca na usunięciu z listy krawędzi <i>adjacencyEdges</i> krawędzi <i>minEdge</i>
Edge	Klasa	Jest to klasa, której obiekty reprezentują krawędzie grafu. Ich atrybutami są: wierzchołek początkowy i końcowy krawędzi oraz jej waga. Klasa implementuje również metodę odpowiedzialną za sortowanie obiektów Edge znajdujących się w liście (metoda compareTo(Edge arg)) oraz metodę odpowiedzialną za wypisanie danej krawędzi (metoda toString()).

tab. 2.2. Składowe implementacji algorytmu Prima c.d.

Uwagi

- W celu realizacji projektu zaistniała również konieczność implementacji modułu generującego macierz sąsiedztwa reprezentującą graf oraz zapisującego ją do pliku tekstowego na pulpicie. Ponadto należało zaimplementować mechanizm odpowiedzialny za odczyt wygenerowanej macierzy z pliku.
- W procesie implementacji algorytmu Prima bazowano na następujących źródłach: [1] oraz [3]. Źródła te miały wpływ na wybór odpowiednich struktur danych oraz na samo zrozumienie istoty algorytmu.

2.3. Algorytm Kruskala

Spis załączników

1. Kod źródłowy algorytmu Prima
2. Kod źródłowy modułu odpowiedzialnego za odczyt macierzy z pliku
3. Link do repozytorium z kodem źródłowym generatora macierzy sąsiedztwa

<https://github.com/joannasia9/Gis-pro>

4. Kod źródłowy algorytmu Kruskala

Bibliografia

- [1] *<http://www.algorytm.org/algorytmy-grafowe/algorytm-prima.html>*
- [2] *<http://algorytmika.wikidot.com/mst>*
- [3] *http://eduinf.waw.pl/inf/alg/001_search/0141.php*

Spis rysunków

2.1. Oznaczenia wykorzystane w opisie	6
2.2. Graf wejściowy.	6
2.3. Prim – Krok 2.	7
2.4. Prim – Krok 3.	7
2.5. Prim – Krok 5.	9
2.6. Prim – Krok 5 - aktualizacja listy krawędzi.	9
2.7. Prim – Krok 7.	10
2.8. Prim – Krok 7 - aktualizacja listy krawędzi.	10
2.9. Prim – Krok 9.	11
2.10. Prim – Krok 9 - aktualizacja listy krawędzi.	12
2.11. Prim – Krok 11.	12
2.12. Prim – Krok 13.	13
2.13. Prim – Krok 14 – otrzymane MST.	14

Załącznik 1. Kod źródłowy algorytmu Prima

```
1 public class PrimAlgorithm{
2     /**
3      * Element losowosci
4      */
5     Random rand;
6     /**
7      * Lista odwiedzonych wierzchołkow
8      */
9     private ArrayList<Integer> visitedVertices;
10
11     /**
12      * Lista wszystkich wierzchołkow
13      */
14     private ArrayList<Edge> adjacencyEdgesList;
15
16
17     /**
18      * Konstruktor
19      * Odpowiada za wykonanie poszczególnych krokow algorytmu
20      *
21      * @param graph macierz sasiedztwa reprezentujaca graf
22      *
23      */
24     public PrimAlgorithm(final int [][] graph){
25         this.adjacencyEdgesList = new ArrayList<>();
26
27         this.visitedVertices = new ArrayList<>();
28
29         /**
30          * Zbior krawedzi
31          * Reprezentuje minimalne drzewo rozpinajace
32          */
33         Set<Edge> minSpanningTree = new HashSet<>();
34
35         /**
36          * Zmienna pomocnicza
37          * Lista krawedzi
38          * Reprezentuje krawedzie incydentne do odwiedzonych wierzchołkow grafu
39          */
40         ArrayList<Edge> adjacencyEdges;
41
42         /**
43          * Zmienna pomocnicza
44          * Element odpowiedzialny za losowy wybor wierzchołka początkowego
45          */
46         this.rand = new Random();
47
48         /**
49          * Losowy wybor wierzchołka początkowego
50          * Z zakresu od 0 do n-1,
51          * gdzie n-liczba wierzchołkow grafu
```

```
52 */
53 int startV = rand.nextInt(graph.length - 1);
54
55 /**
56  * Oznaczenie wierzchołka startowego jako odwiedzony
57  * Dodanie wierzchołka startV do listy odwiedzonych wierzchołków
58  */
59 setVertexVisited(startV, visitedVertices);
60
61
62 /**
63  * Pobranie do zmiennej adjacencyEdges listy krawędzi incydentnych do startV
64  */
65 adjacencyEdges = getAllAdjacencyEdges(startV, graph, adjacencyEdgesList);
66
67 /**
68  * Petla warunkowa while()
69  * Wykonuje się dopóki lista odwiedzonych wierzchołków visitedVertices
70  * nie zawiera wszystkich wierzchołków grafu
71  */
72 while(visitedVertices.size() != graph.length) {
73 /**
74  * Zmienna reprezentująca krawędź z listy krawędzi incydentnych
75  * do punktów odwiedzonych adjacencyEdges, która ma minimalną wagę
76  */
77 Edge minEdge = adjacencyEdges.get(0);
78 /**
79  * Instrukcja warunkowa if()
80  * Pod warunkiem, że lista odwiedzonych wierzchołków visitedVertices
81  * nie zawiera wierzchołka końcowego krawędzi minEdge:
82  *
83  * Do MST zostaje dodana krawędź minEdge
84  * Wierzchołek startV zmienia wartość na wierzchołek końcowy minEdge
85  * Krawędź minEdge zostaje usunięta z listy krawędzi incydentnych
86  * do rozpatrzenia
87  * Lista krawędzi incydentnych adjacencyEdges zostaje zaktualizowana
88  * o krawędź
89  * incydentne do wierzchołka startV
90  *
91  * W przeciwnym wypadku:
92  * Krawędź minEdge zostaje usunięta z listy krawędzi do rozpatrzenia
93  */
94 if(!visitedVertices.contains(minEdge.getEnd())){
95 minSpanningTree.add(minEdge);
96 setVertexVisited(minEdge.getEnd(), visitedVertices);
97 startV = minEdge.getEnd();
98 adjacencyEdges.remove(minEdge);
99 adjacencyEdges = getAllAdjacencyEdges(startV, graph, adjacencyEdgesList);
100 } else adjacencyEdges.remove(minEdge); }
101 /**
102  * Wypisanie otrzymanego MST w konsoli
103  */
104 printMST(minSpanningTree); }
```

```

105  /**
106   * @param vertex
107   * @param list
108   * Oznacza wierzcholek jako odwiedzony poprzez dodanie wierzchołka vertex
109   * do listy wierzchołków odwiedzonych list
110   */
111  private void setVertexVisited(int vertex , ArrayList<Integer> list){
112  /**
113   * Instrukcja warunkowa if()
114   * Blok instrukcji zostaje wykonany, gdy lista wierzchołków odwiedzonych
115   * nie zawiera jeszcze wierzchołka vertex
116   */
117   if(! list.contains(vertex))
118   list.add(vertex);
119  }
120
121  /**
122   *
123   * @param vertex
124   * @param graph
125   * @return
126   *
127   * Dodaje do listy adjacencyEdgesList krawędzie incydentne
128   * do wierzchołka vertex, które nie tworzą cyklu
129   */
130  private ArrayList<Edge> getAllAdjacencyEdges(int vertex ,
131      int [][] graph, ArrayList<Edge> adjacencyEdgesList){
132
133  /**
134   * Instrukcja iteracyjna for()
135   * Blok instrukcji jest wykonywany dla każdego wierzchołka grafu
136   */
137  for(int i=0; i<graph.length; i++){
138  /**
139   * Instrukcja warunkowa if()
140   * Operacja dodania krawędzi incydentnej do wierzchołka vertex
141   * zostaje wykonana pod warunkiem, że:
142   * Krawędź (vertex-i) istnieje, czyli jej waga jest różna od 0 oraz
143   * Lista odwiedzonych wierzchołków visitedVertices
144   * nie zawiera jeszcze i-tego wierzchołka
145   */
146   if(graph[vertex][i]!= 0 && !visitedVertices.contains(i))
147     adjacencyEdgesList.add(new Edge(vertex,i,graph[vertex][i]));
148   }
149
150  /**
151   * Sortowanie otrzymanej listy krawędzi względem wag
152   */
153  Collections.sort(adjacencyEdgesList);
154
155  return adjacencyEdgesList;
156  }
157

```

```
158  /**
159  *
160  * @param set
161  *
162  * Wyświetla w konsoli liste krawedzi otrzymanego drzewa
163  * wraz z suma wag krawedzi
164  */
165  private void printMST(Set<Edge> set){
166  int sum = 0;
167  for(Edge item : set){
168  System.out.println(item);
169  sum+=item.getWeight();
170  }
171  System.out.println("SUMA WAG WYNOSI: " + sum);
172  }
```

Załącznik 1.a. Kod źródłowy klasy Edge

```
1  /**
2  * Klasa Edge, ktorej obiekty reprezentuja krawedzie grafu
3  */
4  private class Edge implements Comparable<Edge> {
5  /**
6  * Wierzcholek poczatkowy krawedzi
7  */
8  private int start;
9  /**
10  * Wierzcholek koncowy krawedzi
11  */
12  private int end;
13  /**
14  * Waga krawedzi (start - end)
15  */
16  private int weight;
17
18  /**
19  *
20  * @param s
21  * @param e
22  * @param w
23  *
24  * Konstruktor krawedzi
25  */
26  Edge(int s, int e, int w){
27  setStart(s);
28  setEnd(e);
29  setWeight(w);
30  }
31
32  private int getWeight(){
33  return weight;
34  }
```



```
35
36 private int getStart() {
37     return start;
38 }
39
40 private void setStart(int start) {
41     this.start = start;
42 }
43
44 private int getEnd() {
45     return end;
46 }
47
48 private void setEnd(int end) {
49     this.end = end;
50 }
51
52 private void setWeight(int weight) {
53     this.weight = weight;
54 }
55
56 /**
57  *
58  * @param e
59  * @return
60  * Metoda odpowiedzialna za porównywanie krawedzi
61  * Wykorzystywana podczas sortowania krawedzi w liście
62  * @see Collections
63  */
64 @Override
65 public int compareTo(Edge e) {
66     if (getWeight() < e.getWeight()){
67         return -1;
68     } else if (getWeight() > e.getWeight()) {
69         return 1;
70     }
71     return 0;
72 }
73
74 /**
75  * @return
76  *
77  * Metoda odpowiedzialna za postać wypisaną w konsoli krawedzi
78  */
79 @Override
80 public String toString() {
81     return String.format("%d--%d\tw:%d ", start, end, weight);
82 }
83 }
```

Załącznik 2. Kod źródłowy modułu odpowiedzialnego za odczyt macierzy z pliku

```
1 import java.io.*;
2
3 public class GraphFileExtractor {
4     private Graph graph;
5     private int[][] adjacencyMatrix;
6
7     public void getFileContent(int iterator) {
8         String home = System.getProperty("user.home");
9         File file = new File(home + File.separator + "Desktop"
10             + File.separator + "G" + iterator + ".txt");
11
12         try {
13             BufferedReader br = new BufferedReader(new FileReader(file.getPath()));
14             String line = br.readLine();
15             initiateMatrix(line);
16
17             int i = 0;
18             while (line != null) {
19                 parseLine(i, line);
20                 line = br.readLine();
21                 i++;
22             }
23
24             } catch (IOException e) {
25                 e.printStackTrace();
26             }
27         }
28
29     private void parseLine(int i, String line){
30         String delim = "\t";
31         String[] tokens = line.split(delim);
32
33         for(int j=0; j<tokens.length;j++){
34             if(Integer.parseInt(tokens[j])!=0)
35                 graph.addEdge(new Edge(i,j,Integer.parseInt(tokens[j])));
36             adjacencyMatrix[i][j] = Integer.parseInt(tokens[j]);
37         }
38
39     }
40
41     private void initiateMatrix(String line){
42         String delim = "\t";
43         String[] tokens = line.split(delim);
44         graph = new Graph(tokens.length);
45         adjacencyMatrix = new int[tokens.length][tokens.length];
46     }
47
48     public Graph getGraph(){
49         return graph;
50     }
51 }
```

```
52 public int [][] getAdjacencyMatrix () {  
53     return adjacencyMatrix ;  
54 }  
55 }
```