Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Семестровая работа

по Алгоритмам и структурам данных

Тема: реализация на языке программирования Java класса GraphCode, хранящего список рёбер для ориентированного графа (вариант 8).

Выполнили студенты группы

11-902 Высшей школы ИТИС

Ганиев Р. А., Евсикова А. Н.

Казань 2020

Класс **GraphCode** хранит список рёбер ориентированного графа с петлями и кратными ребрами (или псевдографа). Элемент списка представляет собой ориентированное ребро графа с номерами двух вершин, которые оно соединяет. Номера вершин принимают значения от 1 до *maxVertexNumber* (атрибут, хранящий наибольший номер среди вершин графа). То есть элемент списка – это упорядоченная пара целых чисел, в которой первое число – это номер вершины, из которой выходит ребро, а второе число – номер вершины, в которую входит ребро.

Класс **GraphCode** позволяет работать с матрицей инцидентности графа, вставлять и удалять ребра, удалять вершины и выполнять некоторые другие операции.

Атрибуты класса.

private ArrayList<Edge> edges.

Список рёбер ориентированного графа.

private int maxVertexNumber.

Атрибут, хранящий наибольший номер среди вершин графа.

private class Edge.

Внутренний приватный класс Edge хранит ориентированное ребро графа как упорядоченную пару двух целых чисел: *beginningOfEdge* и *endOfEdge*. Первое значение – номер вершины, которая является началом ориентированного ребра. Второе значение – номер вершины, которая является концом ориентированного ребра. У класса реализован конструктор Edge(int beginningOfEdge, int endOfEdge), а также переопределены методы equals, hashCode и toString.

Реализация:

```
private class Edge {
    private int beginningOfEdge;
    private int endOfEdge;
    private Edge(int beginningOfEdge, int endOfEdge) {
        this.beginningOfEdge = beginningOfEdge;
        this.endOfEdge = endOfEdge;
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Edge edge = (Edge) o;
        return beginningOfEdge == edge.beginningOfEdge &&
                endOfEdge == edge.endOfEdge;
    }
    @Override
    public int hashCode() {
        return Objects.hash(beginningOfEdge, endOfEdge);
    }
    @Override
    public String toString() {
```

```
        return "(" + beginningOfEdge + " -> " + endOfEdge + ")";
    }
}
```

Конструкторы класса.

public GraphCode().

Создаёт пустой список ориентированных рёбер графа. По умолчанию граф не имеет вершин.

Время выполнения:

O(1).

Используемая память:

O(1).

Реализация:

```
public GraphCode() {
    edges = new ArrayList<>();
    maxVertexNumber = 0;
}
```

public GraphCode(int[][] incidenceMatrix).

Создает список ориентированных рёбер графа, используя матрицу инцидентности. Матрица инцидентности ориентированного графа представляет собой матрицу *B* размером *n* * *m*, где *n* и *m* – количество вершин и рёбер соответственно. Элемент матрицы *B(i, j)* = 1, если ребро *e(j)* выходит из вершины *v(i)*; *B(i, j)* = -1, если ребро входит в вершину *v(i)*; *B(i, j)* = 2, если ребро выходит из вершины *v(i)* и в то же время входит в вершину *v(i)*, то есть является петлёй; во всех остальных случаях *B(i, j)* = 0.

Параметры:

incidenceMatrix – матрица инцидентности ориентированного графа.

Исключения:

IllegalArgumentException - если элемент матрицы инцидентности равен значениям, отличным от -1, 0, 1 или 2; если строки матрицы инцидентности различаются по длине; если каждый столбец матрицы инцидентности не содержит в точности ровно одну 1 и ровно одну -1 или в точности одну 2.

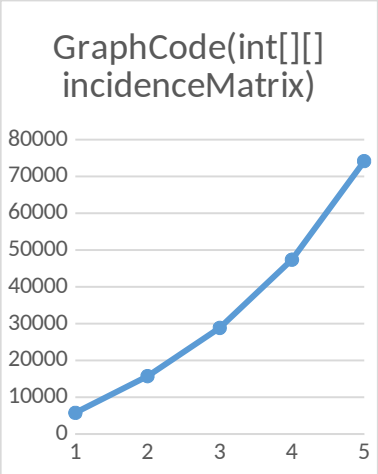
NullPointerException – если *incidenceMatrix* указывает на *null*.

Время выполнения:

$O(n^2) \sim O(e * v)$, где e и v – количество рёбер и вершин в матрице инцидентности.

Результаты тестов замера времени для изучения времени работы алгоритма:

Размер матрицы	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
2 * 2	6900	4300	7300	6300	4300	5200	5800	5729	2025	9433
4 * 4	18300	12800	12500	19000	12900	17300	17200	15714	12010	19418
6 * 6	23200	27600	25400	46100	19500	35400	24500	28814	25110	32518
8 * 8	43900	51500	58400	37100	39400	53700	47400	47343	43639	51047
10 * 10	88100	95200	61900	56000	59600	84700	73500	74143	70439	77847



Используемая память:

O(n) ~ O(e), где e – количество рёбер в матрице инцидентности.

Реализация:

```
public GraphCode(int[][] incidenceMatrix) throws IllegalArgumentException, NullPointerException {
    this();
    readIncidenceMatrix(incidenceMatrix);
}
```

Методы класса.

public int[][] getIncidenceMatrix().

Возвращает матрицу инцидентности для ориентированного графа. Матрица инцидентности строится по заданному списку ребер ориентированного графа. Матрица инцидентности ориентированного графа представляет собой матрицу *B* размером *n* * *m*, где *n* и *m* – количество вершин и рёбер соответственно. Элемент матрицы *B(i, j)* = 1, если ребро *e(j)* выходит из вершины *v(i)*; *B(i, j)* = -1, если ребро входит в вершину *v(i)*; *B(i, j)* = 2, если ребро выходит из вершины *v(i)* и в то же время входит в вершину *v(i)*, то есть является петлёй; во всех остальных случаях *B(i, j)* = 0. Если список рёбер пуст, то этот метод возвращает *null*.

Возвращаемое значение:

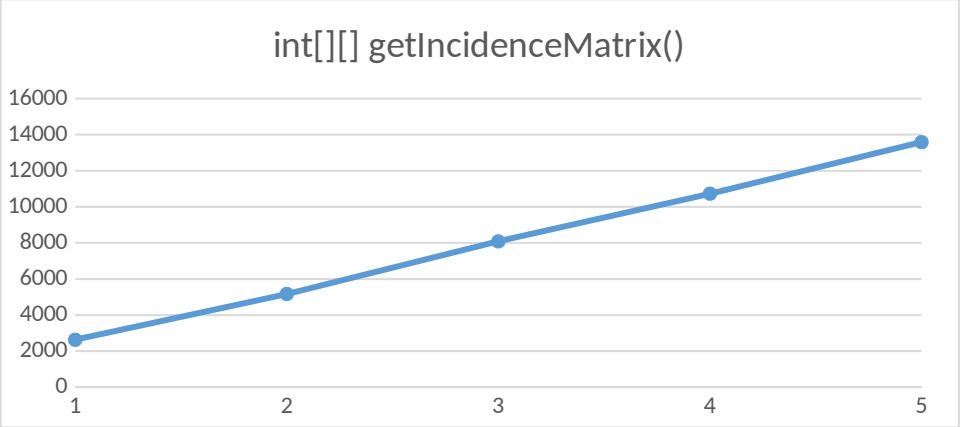
Матрицу инцидентности ориентированного графа в виде двумерного массива целых чисел или *null*, если список ребер графа пуст.

Время выполнения:

O(n) ~ O(e), где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Размер матрицы	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
2 * 2	3200	2700	2600	2400	2400	2600	2500	2629	1147	4110
4 * 4	6300	4100	3800	4600	4500	7300	5600	5171	3690	6653
6 * 6	9700	8200	8500	6700	7800	7400	8300	8086	6604	9567
8 * 8	12300	9300	10300	12700	10100	9600	10800	10729	9247	12210
10 * 10	15500	10600	13300	14600	13700	12300	15100	13586	12104	15067



Используемая память:

$O(n^2)$ ~ O(e * v), где e и v – количество рёбер и вершин в матрице инцидентности.

Реализация:

```
public int[][] getIncidenceMatrix() {
    if (maxVertexNumber == 0) {
        return null;
    }
    int[][] incidenceMatrix = new int[maxVertexNumber][edges.size()];
    for (int i = 0; i < edges.size(); i++) {
        int beginningOfEdge = edges.get(i).beginningOfEdge;
        int endOfEdge = edges.get(i).endOfEdge;
        if (beginningOfEdge == endOfEdge) {
            incidenceMatrix[beginningOfEdge - 1][i] = 2;
        }
        else {
            incidenceMatrix[beginningOfEdge - 1][i] = 1;
            incidenceMatrix[endOfEdge - 1][i] = -1;
        }
    }
    return incidenceMatrix;
}
```

public void insert(int beginningOfEdge, int endOfEdge).

Вставляет новое ориентированное ребро (*beginningOfEdge*, *endOfEdge*) в список рёбер ориентированного графа. Вершины ориентированного графа имеют номера от 1 до *maxVertexNumber* (атрибут, хранящий наибольший номер среди вершин графа), поэтому вставляемое ребро должно иметь номера своих начала и конца от 1 до *maxVertexNumber* + 1, если оно инцидентно существующим вершинам графа, или (*maxVertexNumber* + 1, *maxVertexNumber* + 2), (*maxVertexNumber* + 2, *maxVertexNumber* + 1), если оно не инцидентно существующим вершинам графа.

Параметры:

beginningOfEdge – номер вершины, которая является началом вставляемого ориентированного ребра.

endOfEdge – номер вершины, которая является концом вставляемого ребра.

Исключения:

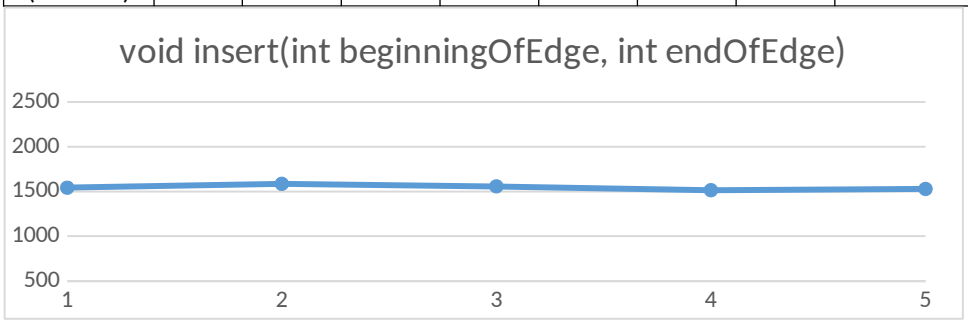
IllegalArgumentException – если начало или конец вставляемого ребра ориентированного графа содержат числа вне диапазона от 1 до *maxVertexNumber* + 1 и вставляемое ребро не равны (*maxVertexNumber* + 1, *maxVertexNumber* + 2) или (*maxVertexNumber* + 2, *maxVertexNumber* + 1).

Время выполнения:

O(1).

Результаты тестов замера времени для изучения времени работы алгоритма:

Ребро	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
(2 -> 7)	2400	1600	1100	1800	1200	1500	1200	1543	802	2284
(8 -> 1)	2500	1100	900	1000	2300	1600	1700	1586	845	2327
(5 -> 5)	2400	2100	1200	900	2600	900	800	1557	816	2298
(10 -> 11)	2400	1200	1000	800	1900	1800	1500	1514	773	2255
(12 -> 11)	2200	1100	1000	800	2800	1000	1800	1529	788	2269



Используемая память:

O(1).

Реализация:

```
public void insert(int beginningOfEdge, int endOfEdge) throws IllegalArgumentException {
    if (beginningOfEdge < 1 || endOfEdge < 1 ||
            beginningOfEdge > maxVertexNumber + 2 || endOfEdge > maxVertexNumber + 2 ||
            (beginningOfEdge == maxVertexNumber + 2 && endOfEdge == maxVertexNumber + 2)) {
        if (maxVertexNumber == 0) {
            throw new IllegalArgumentException("The graph has no vertices yet. " +
                    "You can insert only the edges: 1-1, 1-2 or 2-1");
        }
        else {
            throw new IllegalArgumentException("The graph has vertices with numbers from " + 1 + " to "
+
                    maxVertexNumber + ". You can insert edges connecting vertices with numbers from 1 to
" +
                    (maxVertexNumber + 1) + " or edges: " + (maxVertexNumber + 1) + "-" +
(maxVertexNumber + 2) +
                    ", " + (maxVertexNumber + 2) + "-" + (maxVertexNumber + 1));
        }
    }
    else {
        if (Integer.max(beginningOfEdge, endOfEdge) > maxVertexNumber) {
            maxVertexNumber = Integer.max(beginningOfEdge, endOfEdge);
```

```
        }
        edges.add(new Edge(beginningOfEdge, endOfEdge));
    }
}
```

public void delete(int beginningOfEdge, int endOfEdge).

Удаляет ориентированное ребро (*beginningOfEdge*, *endOfEdge*) из списка ребер ориентированного графа. Если указанное ребро графа отсутствует в списке рёбер или были введены некорректные данные, то метод не осуществляет каких-либо действий. Вершины, которые являются началом или концом ребра графа, не удаляются при удалении ребра.

Параметры:

beginningOfEdge – номер вершины, которая является началом удаляемого ориентированного ребра.

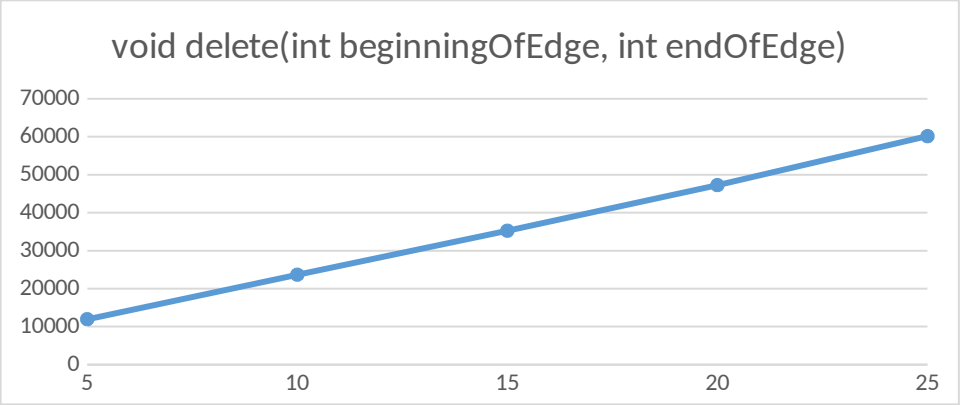
endOfEdge – номер вершины, которая является концом удаляемого ребра.

Время выполнения:

O(n) ~ O(e), где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

№ ребра	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	16400	12700	9300	14200	7300	15400	8500	11971	9749	14194
10	28900	22500	20500	16800	28100	29900	19100	23686	21463	25908
15	39300	28100	34200	33300	31800	36100	44000	35257	33035	37480
20	55400	43200	47000	39100	48600	51300	46200	47257	45035	49480
25	57400	56700	63700	63600	59800	60100	59700	60143	57920	62365



Используемая память:

O(1).

Реализация:

```
public void delete(int beginningOfEdge, int endOfEdge) {
    if (beginningOfEdge > 0 && endOfEdge > 0 &&
            beginningOfEdge <= maxVertexNumber && endOfEdge <= maxVertexNumber) {
        edges.remove(new Edge(beginningOfEdge, endOfEdge));
    }
}
```

public GraphCode getEdgesWithNode(int vertex).

Возвращает список ориентированных рёбер, инцидентных вершине *vertex* ориентированного графа. Список ориентированных рёбер возвращается в виде нового **GraphCode**, хранящего этот список. Если нет рёбер графа, инцидентных вершине *vertex*, или если *vertex* не существует в графе, то возвращается **GraphCode** с пустым списком рёбер.

Параметры:

vertex – вершина ориентированного графа, для которой возвращается список рёбер, инцидентных ей.

Возвращаемое значение:

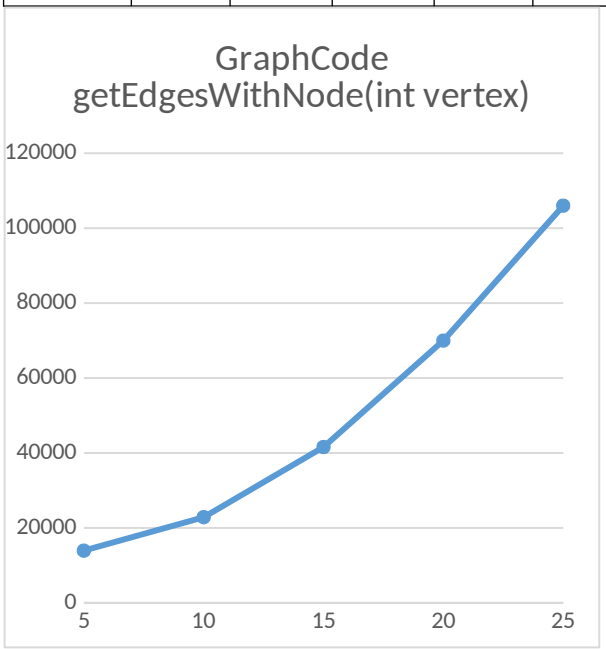
Новый **GraphCode**, хранящий список рёбер ориентированного графа, которые инцидентны указанной вершине *vertex*, или пустой список, если нет ребер графа, инцидентных вершине *vertex*, или если *vertex* не существует в графе.

Время выполнения:

$O(n^2) \sim O(e^2)$, где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	14600	16100	14500	16400	12800	11200	12100	13957	8031	19884
10	20700	24100	22400	24600	18700	20700	29000	22886	16959	28812
15	34700	40100	37300	36600	31200	55500	55900	41614	35688	47541
20	58700	63200	82300	83300	79100	59900	63700	70029	64102	75955
25	94900	112300	109900	95800	119110	103300	107000	106044	100118	111971



Используемая память:

O(n) ~ O(e), где e – количество рёбер в графе.

Реализация:

```
public GraphCode getEdgesWithNode(int vertex) {
    GraphCode graph = new GraphCode();
    if (vertex > 0 && vertex <= maxVertexNumber) {
        graph.edges = new ArrayList<>(edges);
        graph.maxVertexNumber = maxVertexNumber;
        for (Edge edge : edges) {
            if (vertex != edge.beginningOfEdge && vertex != edge.endOfEdge) {
                graph.delete(edge.beginningOfEdge, edge.endOfEdge);
            }
        }
    }
    return graph;
}
```

public void modify(int vertex).

Модифицирует список рёбер ориентированного графа удалением из него вершины *vertex*. Удаляет из списка рёбер все рёбра, инцидентные данной вершине *vertex*. Если вершины *vertex* не существует в ориентированном графе, тогда этот метод не осуществляет каких-либо действий.

Параметры:

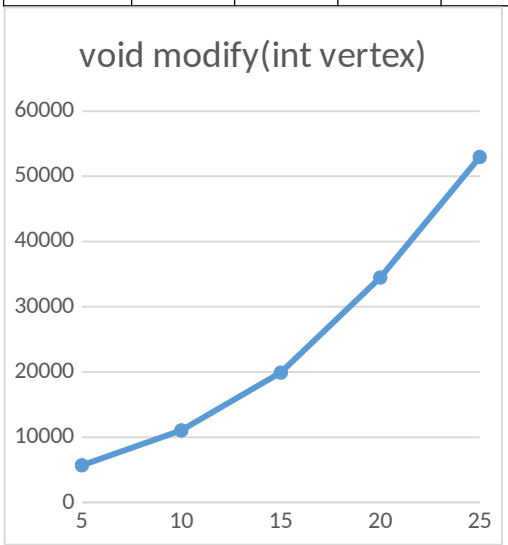
vertex – вершина, которая удаляется из ориентированного графа со всеми рёбрами, которые инцидентны данной вершине.

Время выполнения:

$O(n^2) \sim O(e^2)$, где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	7400	5400	5000	4500	4600	7800	5100	5686	1982	9390
10	14500	10800	9300	11200	9600	11400	10500	11043	7339	14747
15	18800	24000	17500	23900	23000	15400	16700	19900	16196	23604
20	29500	32200	36800	35700	36200	38700	32300	34486	30782	38190
25	38700	43000	45900	44300	50800	48500	39600	44400	40696	48104



Используемая память:

O(1).

Реализация:

```
public void modify(int vertex) {
    if (vertex > 0 && vertex <= maxVertexNumber) {
        for (int i = edges.size() - 1; i >= 0; i--) {
            if (vertex == edges.get(i).beginningOfEdge || vertex == edges.get(i).endOfEdge) {
                edges.remove(i);
            }
            else {
                if (edges.get(i).beginningOfEdge > vertex) {
                    edges.get(i).beginningOfEdge--;
                }
                if (edges.get(i).endOfEdge > vertex) {
                    edges.get(i).endOfEdge--;
                }
            }
        }
        maxVertexNumber--;
    }
}
```

public ArrayList<Integer> outdegreeShow(int m).

Возвращает список вершин ориентированного графа, полустепень исхода которых больше, чем *m*. Если список рёбер ориентированного графа пуст, то метод возвращает *null*.

Параметры:

m – целое число такое, что в возвращаемый список вершин добавлены вершины ориентированного графа с полустепенью исхода больше, чем это число *m*.

Возвращаемое значение:

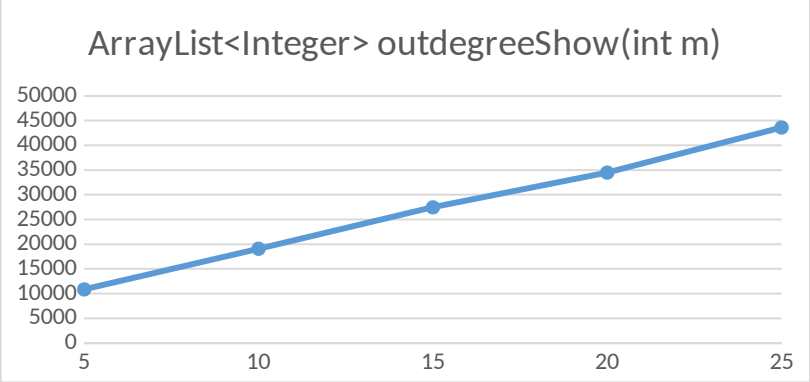
Список вершин ориентированного графа, полустепень исхода которых больше, чем *m*, или *null*, если список рёбер ориентированного графа пуст.

Время выполнения:

O(n) ~ O(e + v), где e и v – количество рёбер и вершин в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	14000	10300	9100	8200	8300	12500	13600	10857	8635	13080
10	20500	15500	16800	21800	23000	17200	18800	19086	16863	21308
15	28400	31300	23400	26600	22500	34300	25700	27457	25235	29680
20	41900	31700	34300	31400	30700	37300	34100	34486	32263	36708
25	52800	42600	37500	44200	48200	36900	43100	43614	41392	45837



Используемая память:

O(n) ~ O(v), где v – количество вершин в графе.

Реализация:

```
public ArrayList<Integer> outdegreeShow(int m) {
    if (edges.isEmpty()) {
        return null;
    }
    ArrayList<Integer> vertices = new ArrayList<>();
    if (m < 0) {
        for (int i = 1; i <= maxVertexNumber; i++) {
            vertices.add(i);
        }
        return vertices;
    }
    int[] outdegrees = new int[maxVertexNumber];
    for (Edge edge : edges) {
        outdegrees[edge.beginningOfEdge - 1]++;
    }
    for (int i = 0; i < maxVertexNumber; i++) {
        if (outdegrees[i] > m) {
            vertices.add(i + 1);
        }
    }
    return vertices;
}
```

public ArrayList<Integer> indegreeShow(int m).

Возвращает список вершин ориентированного графа, полустепень захода которых больше, чем *m*. Если список рёбер ориентированного графа пуст, то метод возвращает *null*.

Параметры:

m – целое число такое, что в возвращаемый список вершин добавлены вершины ориентированного графа с полустепенью захода больше, чем это число *m*.

Возвращаемое значение:

Список вершин ориентированного графа, полустепень захода которых больше, чем *m*, или *null*, если список рёбер ориентированного графа пуст.

Время выполнения:

O(n) ~ O(e + v), где e и v – количество рёбер и вершин в графе.

Аналогично методу ArrayList<Integer> outdegreeShow(int m).

Используемая память:

O(n) ~ O(v), где v – количество вершин в графе.

Реализация:

```
public ArrayList<Integer> indegreeShow(int m) {
    if (edges.isEmpty()) {
        return null;
    }
    ArrayList<Integer> vertices = new ArrayList<>();
    if (m < 0) {
        for (int i = 1; i <= maxVertexNumber; i++) {
            vertices.add(i);
        }
        return vertices;
    }
```

```
    int[] indegrees = new int[maxVertexNumber];
    for (Edge edge : edges) {
        indegrees[edge.endOfEdge - 1]++;
    }
    for (int i = 0; i < maxVertexNumber; i++) {
        if (indegrees[i] > m) {
            vertices.add(i + 1);
        }
    }
    return vertices;
}
```

private void readIncidenceMatrix(int[][] incidenceMatrix).

Приватный метод для создания списка ориентированных рёбер графа с использованием заданной матрицы инцидентности этого графа. Метод используется в конструкторе и может быть использован в других методах класса. Матрица инцидентности ориентированного графа представляет собой матрицу *B* размером *n* * *m*, где *n* и *m* – количество вершин и рёбер соответственно. Элемент матрицы *B(i, j)* = 1, если ребро *e(j)* выходит из вершины *v(i)*; *B(i, j)* = -1, если ребро входит в вершину *v(i)*; *B(i, j)* = 2, если ребро выходит из вершины *v(i)* и в то же время входит в вершину *v(i)*, то есть является петлёй; во всех остальных случаях *B(i, j)* = 0.

Параметры:

incidenceMatrix – матрица инцидентности ориентированного графа.

Исключения:

IllegalArgumentException - если элемент матрицы инцидентности равен значениям, отличным от -1, 0, 1 или 2; если строки матрицы инцидентности различаются по длине; если каждый столбец матрицы инцидентности не содержит в точности ровно одну 1 и ровно одну -1 или в точности одну 2.

NullPointerException – если *incidenceMatrix* указывает на *null*.

Время выполнения:

$O(n^2)$ ~ O(e * v), где e и v – количество рёбер и вершин в матрице инцидентности.

Результаты тестов замера времени для изучения времени работы алгоритма можно посмотреть в описании конструктора GraphCode(int[][] incidenceMatrix, в котором используется данный метод.

Используемая память:

O(n) ~ O(e), где e – количество рёбер в матрице инцидентности.

Реализация:

```
private void readIncidenceMatrix(int[][] incidenceMatrix) throws IllegalArgumentException,
NullPointerException {
    if (incidenceMatrix != null) {
        int numberOfVertices = incidenceMatrix.length;
        int numberOfEdges = incidenceMatrix[0].length;
        for (int i = 1; i < numberOfVertices; i++) {
            if (incidenceMatrix[i].length != numberOfEdges) {
                throw new IllegalArgumentException("Incorrect incidence matrix");
            }
        }
        for (int j = 0; j < numberOfEdges; j++) {
            int beginningOfEdge = 0;
            int endOfEdge = 0;
```

```
            for (int i = 0; i < numberOfVertices; i++) {
                if (incidenceMatrix[i][j] == 2) {
                    if (beginningOfEdge != 0) {
                        throw new IllegalArgumentException("Incorrect incidence matrix");
                    }
                    beginningOfEdge = i + 1;
                    endOfEdge = i + 1;
                }
                else if (incidenceMatrix[i][j] == 1) {
                    if (beginningOfEdge != 0) {
                        throw new IllegalArgumentException("Incorrect incidence matrix");
                    }
                    beginningOfEdge = i + 1;
                }
                else if (incidenceMatrix[i][j] == -1) {
                    if (endOfEdge != 0) {
                        throw new IllegalArgumentException("Incorrect incidence matrix");
                    }
                    endOfEdge = i + 1;
                }
                else if (incidenceMatrix[i][j] != 0) {
                    throw new IllegalArgumentException("Incorrect incidence matrix");
                }
            }
            if (beginningOfEdge != 0 && endOfEdge != 0) {
                if (Integer.max(beginningOfEdge, endOfEdge) > maxVertexNumber) {
                    maxVertexNumber = Integer.max(beginningOfEdge, endOfEdge);
                }
                this.insert(beginningOfEdge, endOfEdge);
            }
            else if (!(beginningOfEdge == 0 && endOfEdge == 0)) {
                throw new IllegalArgumentException("Incorrect incidence matrix");
            }
        }
    }
    else {
        throw new NullPointerException("Argument points to null");
    }
}
```

public void printIncidenceMatrix().

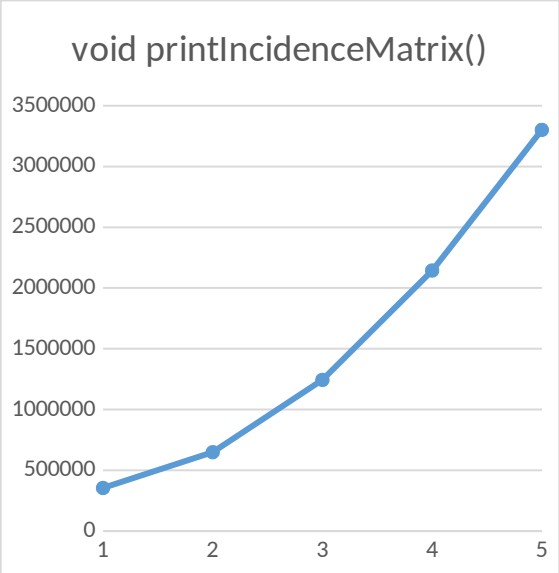
Печатает матрицу инцидентности ориентированного графа. Метод ничего не печатает, если список рёбер ориентированного графа пуст. Матрица инцидентности ориентированного графа представляет собой матрицу *B* размером *n* * *m*, где *n* и *m* – количество вершин и рёбер соответственно. Элемент матрицы *B(i, j)* = 1, если ребро *e(j)* выходит из вершины *v(i)*; *B(i, j)* = -1, если ребро входит в вершину *v(i)*; *B(i, j)* = 2, если ребро выходит из вершины *v(i)* и в то же время входит в вершину *v(i)*, то есть является петлёй; во всех остальных случаях *B(i, j)* = 0.

Время выполнения:

$O(n^2) \sim O(e * v)$, где e и v – количество рёбер и вершин в матрице инцидентности.

Результаты тестов замера времени для изучения времени работы алгоритма:

Размер матрицы	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
2 * 2	362100	343900	361300	355100	387300	352400	323100	355029	301325	408733
4 * 4	640800	688200	649100	672100	584500	651900	659100	649386	595682	703090
6 * 6	1203400	1242000	1247900	1279600	1246700	1239700	1251100	1244343	1190639	1298047
8 * 8	2275300	2312500	2111600	2119500	2130900	2042500	2015200	2143929	2090225	2197633
10 * 10	3427900	3168300	3227300	3365000	3272800	3375800	3268400	3300786	3247082	3354490



Используемая память:

$O(n^2) \sim O(e * v)$, где e и v – количество рёбер и вершин в матрице инцидентности.

Реализация:

```
public void printIncidenceMatrix() {
    int[][] incidenceMatrix = this.getIncidenceMatrix();
    if (incidenceMatrix != null) {
        for (int i = 0; i < maxVertexNumber; i++) {
            for (int j = 0; j < edges.size(); j++) {
                System.out.printf("%" + 2 + "d", incidenceMatrix[i][j]);
                System.out.print(" ");
            }
            System.out.println();
        }
    }
}
```

public int[][] get().

Возвращает список рёбер ориентированного графа в виде двумерного массива целых чисел. Каждая строка двумерного массива состоит из двух целых чисел: первое значение – номер вершины, которая является началом ориентированного ребра; второе значение – номер вершины, которая является концом ориентированного ребра. Если список рёбер ориентированного графа пуст, то метод возвращает *null*.

Возвращаемое значение:

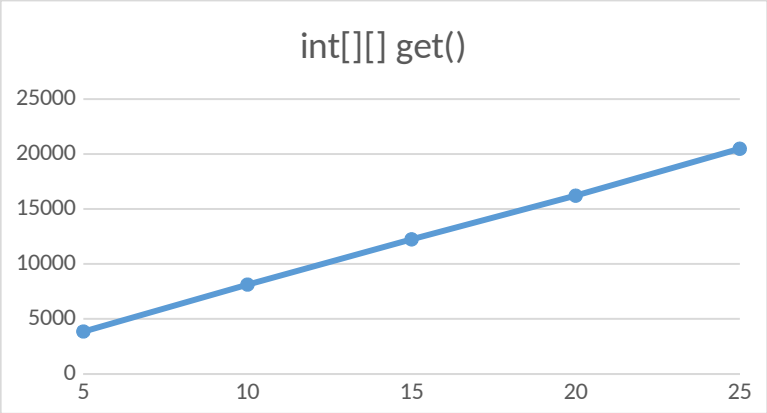
Список рёбер ориентированного графа в виде двумерного массива целых чисел или *null*, если список рёбер графа пуст.

Время выполнения:

O(n) ~ O(e), где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	4700	3900	3600	3500	3400	3900	4000	3857	3116	4598
10	8200	6600	9200	7100	8700	8200	8900	8129	7388	8869
15	13500	12600	11600	11300	11300	13100	12300	12243	11502	12984
20	16600	14900	16000	15500	18000	16200	16300	16214	15473	16955
25	22400	19700	18500	20600	23900	19500	18800	20486	19745	21227



Используемая память:

O(n) ~ O(e), где e – количество рёбер в графе.

Реализация:

```
public int[][] get() {
    if (maxVertexNumber == 0) {
        return null;
    }
    int[][] edgesOfGraph = new int[edges.size()][2];
    for (int i = 0; i < edges.size(); i++) {
        edgesOfGraph[i][0] = edges.get(i).beginningOfEdge;
        edgesOfGraph[i][1] = edges.get(i).endOfEdge;
    }
    return edgesOfGraph;
}
```

public boolean equals(Object o).

Сравнивает заданный объект с данным **GraphCode** на равенство. Возвращает *true*, если заданный объект совпадает с **GraphCode**: списки рёбер двух графов имеют одинаковый размер, число вершин в двух графах совпадает и каждое ориентированное ребро заданного объекта содержится в данном **GraphCode**.

Параметры:

o – объект, который будет сравниваться на равенство с данным **GraphCode**.

Возвращаемое значение:

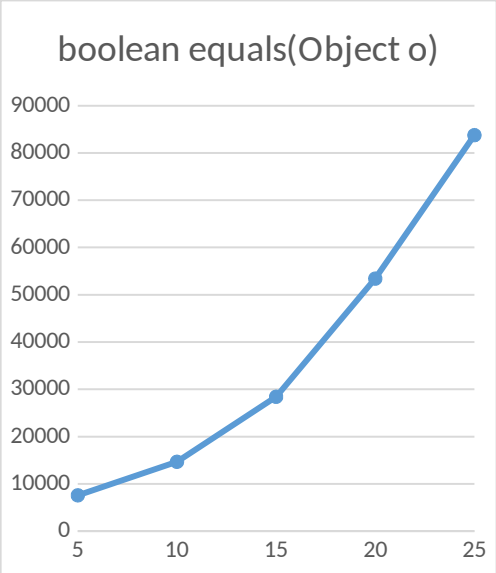
true, если заданный объект равен данному **GraphCode**.

Время выполнения:

$O(n^2) \sim O(e^2)$, где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	9100	8600	7000	5800	6400	8700	7400	7571	5349	9794
10	14600	15300	13900	14500	13800	15200	15400	14671	12449	16894
15	29100	29900	30300	28000	25200	29100	27400	28429	26206	30651
20	55100	53700	55100	49800	54800	52100	53300	53414	51192	55637
25	84400	87600	80200	79200	90300	83600	81000	83757	81535	85980



Используемая память:

O(n) ~ O(e), где e – количество рёбер в графе.

Реализация:

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    GraphCode graphCode = (GraphCode) o;
    if (maxVertexNumber == graphCode.maxVertexNumber && edges.size() == graphCode.edges.size()) {
        boolean[] flags = new boolean[edges.size()];
        for (int i = 0; i < edges.size(); i++) {
            for (int j = 0; j < edges.size(); j++) {
                if (edges.get(i).equals(graphCode.edges.get(j)) && !flags[j]) {
                    flags[j] = true;
                    break;
                }
            }
        }
        for (int i = 0; i < edges.size(); i++) {
            if (!flags[i]) {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

public int hashCode().

Возвращает хэш-код для данного **GraphCode**. Хэш-код **GraphCode** определяется как сумма хэш-кодов рёбер графа и хэш-кода вершины ориентированного графа с наибольшим номером. Хэш-код элемента *null* определяется как 0.

Возвращаемое значение:

Значение хэш-кода для данного **GraphCode**.

Время выполнения:

O(n) ~ O(e), где e – количество рёбер в графе.

Используемая память:

O(1).

Реализация:

```
@Override
public int hashCode() {
    return Objects.hash(edges, maxVertexNumber);
}
```

public String toString().

Возвращает строковое представление данного GraphCode. Строковое представление, заключенное в фигурные скобки “{ }”, состоит из списка всех рёбер ориентированного графа.

Возвращаемое значение:

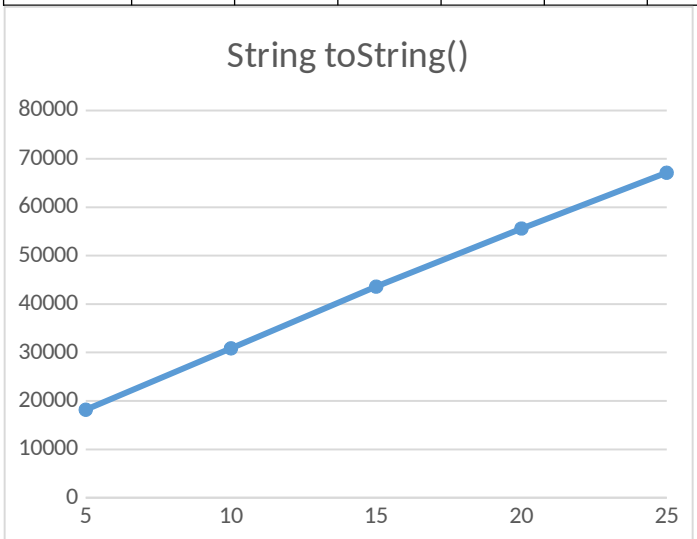
Строковое представление данного **GraphCode**.

Время выполнения:

O(n) ~ O(e), где e – количество рёбер в графе.

Результаты тестов замера времени для изучения времени работы алгоритма:

Кол-во рёбер	№1	№2	№3	№4	№5	№6	№7	Среднее значение	Доверительный интервал	
5	16300	17600	19200	18400	19000	17200	19600	18186	15963	20408
10	33300	28500	32000	30400	32800	28600	30400	30857	28635	33080
15	48400	41300	46000	41800	42700	40600	44400	43600	41378	45822
20	50600	55600	53700	55100	56700	57200	60300	55600	53378	57822
25	65700	73900	59400	61400	67000	72900	69500	67114	64892	69337



Используемая память:

O(n) ~ O(e), где e – количество рёбер в графе.

Реализация:

```
@Override
public String toString() {
    StringBuilder str = new StringBuilder("Graph edges = {");
    for (int i = 0; i < edges.size() - 1; i++) {
        str.append(edges.get(i));
        str.append(", ");
    }
    str.append(edges.get(edges.size()- 1));
    str.append('}');
    return str.toString();
}
```